Interfaces for Placement, Migration, and Monitoring of Virtual Machines in Federated Clouds

Erik Elmroth and Lars Larsson
Department of Computing Science
Umeå University
Umeå, Sweden
Email: {elmroth, larsson}@cs.umu.se

***Abstract*—Current cloud computing infrastructure offerings are lacking in interoperability, which is a hindrance to the advancement and adoption of the cloud computing paradigm. As clouds are made interoperable, federations of clouds may be formed. Such federations are from the point of view of the user not burdened by vendor lock-in, and opens for business possibilities where a market place of cloud computing infrastructure can be formed. Federated clouds require unified management interfaces regarding the virtual machines (VMs) that comprise the services running in the cloud federation. Standardization efforts for the required management interfaces have so far focused on definition of description formats regarding VMs, and the control of already deployed VMs. We propose technology-neutral interfaces and architectural additions for handling placement, migration, and monitoring of VMs in federated cloud environments, the latter as an extension of current monitoring architectures used in Grid computing. The interfaces presented adhere to the general requirements of scalability, efficiency, and security in addition to specific requirements related to the particular issues of interoperability and business relationships between competing cloud computing infrastructure providers. In addition, they may be used equally well locally and remotely, creating a layer of abstraction that simplifies management of virtualized service components.**

I. Introduction

Cloud computing is growing increasingly popular in the IT business, and industry leaders such as Bill Joy of Sun Microsystems fame estimated that utility and pervasive computing such as cloud computing may be a trillion dollar business (as quoted in [1]). Implementations and architectures vary widely, and the cloud computing offerings of one vendor are often not guaranteed to be compatible with those of some other vendor, thus creating vendor lock-in.

The US National Institute of Standards and Technology is currently working on a definition of cloud computing [2], where cloud computing is stated as having five key characteristics: (a) on-demand self-service; (b) ubiquitous network access; (c) location independent resource pooling; (d) rapid elasticity; and (e) pay per use. There are also three delivery models: (i) Software as a Service; (ii) Platform as a Service; and (iii) Infrastructure as a Service. These delivery models differ substantially in scope. Our focus is cloud infrastructure, which we use to denote the infrastructure required for hosting virtualized services, and thus Infrastructure as a Service (IaaS). In particular, the infrastructure provided should be flexible and adapt to the dynamics in demand for a deployed service in a cost-efficient way. This includes optimizing resource usage at the infrastructure provider's site locally (e.g. reducing the number of powered machines and consolidation of load), as well using bidirectional contracts with other infrastructure provider sites. A federated cloud is one where (competing) infrastructure providers can reach cross-site agreements of cooperation regarding the deployment of service components in a way similar to how electrical power providers provision capacity from each other to cope with variations in demand. Such collaboration increases location independence. To achieve this vision, the cloud sites in the federation must conform to common interfaces regarding virtualized service components and employ compatible virtualization platforms.

Cloud computing leverages technologies such as Grid computing and virtualization to provide the basic infrastructure and platform for the cloud computing stack. Services (or components thereof) are generally deployed in virtual machines (VMs) that can be defined in terms of the required virtual hardware resources and network connectivity. As demand dynamically fluctuates, the resources can be scaled up or down to allow the customer to pay only for the capacity needed and to reduce the costs for the cloud provider. Rules for this type of dynamic elasticity in resource demands are formulated as Service Level Agreements (SLAs), either in terms of virtual hardware resources and the utilization thereof, or as Key Performance Indicators (KPIs). KPIs include e.g. application-specific terms such as the number of jobs in a work queue.

The contribution of this article is two-fold. First, we analyze the current state of the art of (proposed) standards for VM management interfaces to find where enhancements are needed, and based on usage scenarios, determine what the requirements of such enhancements are. Second, we propose interfaces for supporting the additional required functionality, adding placement, migration, and monitoring interfaces to the interfaces already defined in (proposed) standards. We argue that placement is a special case of migration, and thus can be supported by the same homogeneous interface operations. We define these interface operations, and introduce a component called Transfer proxy that is used to carry out the file transfers. An algorithm that utilizes Transfer proxies for such transfer is presented. With regard to monitoring interfaces, we present

additions to the descriptors of VMs that configure a general monitoring system. In addition to discussing how monitoring data should be transferred in a general cloud computing platform and presenting our solution, we introduce a novel approach for supporting application-level measurements in virtualized cloud service components while requiring a minimal change to the application itself.

The remainder of the article is structured as follows. In Section II, we discuss the rationale and requirements regarding migration and monitoring interfaces. Section III presents a scalable and decentralized solution to handling migration of VMs that fulfills these requirements. In Section IV, we present additions to current standardized descriptors of VMs that relate to monitoring of both the infrastructure itself and of the applications that run in the VMs. These suggestions are later discussed in Section V. Related work in some currently active projects is presented in Section VI. Section VII contains a summary and conclusions.

II. Cross-site Management in Federated Cloud Environments

The previous section referred to the working definition of cloud computing by NIST, which states a number of key characteristics for cloud computing. As the cloud computing paradigm has become increasingly popular, various cloud infrastructure architectures have been developed. For the purpose of this work, we assume that a general cloud computing architecture requires functionality that may well be implemented by components divided into a number of layers with clear *separation of concern* and responsibilities. Although the number of layers vary among cloud computing implementations, we assume that a general architecture is divided in at least the following two conceptual layers: (a) the *Management layer*, responsible for overall management, such as admission control, decisions regarding where to deploy VEEs in the cloud (referred to as *placement* of VMs), control of resources, accounting for usage, etc; and (b) the *Implementation layer*, responsible for hosting the VMs, maintaining virtual networks across sites, etc.

To note the difference in responsibilities of the sites in the federation, we let *primary site* denote the site that has been contractually bound by the service provider (customer) to manage the VMs that comprise a service and other sites be denoted as *remote sites*.

The layered architecture clearly marks a separation of concern between the Management and Implementation layers. There is also a separation of concern between cloud sites, as sites may avoid disclosing exact information regarding the placement of VMs on physical host machines. We call this the *principle of location unawareness*. The principle is one of the pillars of the RESERVOIR project [3], motivated by use cases where cloud infrastructure providers do not wish to disclose information regarding the size or utilization of the infrastructure, due to the sensitivity of such information. Also, it is more convenient for a site to make use of the infrastructure offered as a service by the remote site, relying on SLA compliance rather than minute resource management of VMs running remotely. This also enables infrastructure-less resource brokers to act in the federated cloud.

We use the term *primary placement* for denoting the selection of, and transfer to, a site for future deployment of a VM that has not been run yet, or one that has been shut down after having run previously. A VM is regarded as *deployed* when all its files have been transferred to the host that will run it, it has been activated (or "booted up"), and is currently running. Transferring a VM from one VM host to another, possibly with either one or both hosts running at remote sites, is called *VM Migration*. Note that both local (across local hosts) and remote (cross-site) migration is included in this definition. The process of migration may be initiated reactively due to the submission of new VMs to deploy, or proactively to continuously optimize placement to avoid SLA violations, evenly distribute system load, consolidate VMs to few hosts for power savings, etc.

We claim that placement of a VM that has not yet been run, or one that has been shut down, may be regarded as a special case of cold migration, as the only conceptual difference is that there is no serialized runtime state to transfer. We therefore, for the purposes of this article, consider primary placement of a VM to be no different from migration. This lets us employ the same protocol in both contexts, and as a result, simplifies management operations.

We consider three general usage scenarios related to migration of VMs. In the first (Usage scenario 1), the primary site initiates the VM migration due to lack of local resources, whereas in the second (Usage scenario 2), a remote site currently running a VM on behalf of the primary site is unable to continue doing so and requires that the primary site finds new placement for the VM immediately. The third (Usage scenario 3) is the case when a VM running locally at a certain host must be migrated to another host, running at the same site. Such migration may be used for e.g. energy conservation (consolidation of VMs to a low number of hosts) or for load-balancing (maximizing distribution of VMs among local hosts).

Requests to deploy a VM on behalf of some other site are to be considered as offers of contracts. These contracts may contain SLAs that stipulate the requirements on both parties, and from the Infrastructure Provider's point of view, they could be costly to violate. Thus, in a migration scenario, the involved sites must be able to verify the identities of each other in a secure manner. If a site is unable to deploy a VM due to lack of resources, it should be able to delegate the responsibility to some partner site. Some of the aforementioned SLAs are expressed in terms of VM resource allocation (e.g. "5 Mb/second bandwidth 95% of the time over some time period"), whereas others are expressed in application-specific terms (e.g. "no more than 1000 concurrent database users at any time"). In order to avoid violating the SLAs, the Management layer is responsible for scaling the size and amount of the VMs allocated to the service up and down, as required. For reasons such as VM host maintenance, a site responsible for running a VM may require that a VM is migrated away from it. Such requests cannot be ignored by the site that has delegated responsibility to the site.

A. State of the Art Technologies and Standards

The state of the art technologies and standards include support for some of the features mentioned in the usage scenarios presented in Section II. In the following sections, we briefly consider the topics of two of these sets of standards: VM descriptors and VM control interfaces.

1) VM Descriptors : In order to be deployable in the federated cloud, each VM must be described in terms of required resources, its network membership and configuration, have a unique identifier, etc. One standard dealing with this issue is the Open Virtualization Format (OVF) [4] by the Distributed Management Task Force (DMTF). OVF is supported by leading technologies such as the open source Xen [5] and the proprietary VMware [6].

We use the term *VM Descriptor* to refer to the description of a single VM. Note that the OVF does not include a section related to monitoring of VMs, but rather the configuration of the virtual hardware and some basic configuration of this hardware, e.g. network addresses, etc.

2) VM Control Operations: Once deployed, VMs must be controlled by the management layer. A general Control interface for VMs is used to carry out two types of operations: (a) modifying the Descriptor of a VM, making it possible to alter the amount of resources allocated, or any other configurable parameter related to the VM; and (b) updating the runtime state of the VM, e.g. by shutting down a running VM or activating a hibernated one. The DMTF has proposed standards for these operations in [7] and [8], respectively. The states of VMs are described in [9].

B. Requirements

The usage scenarios in Section II require additional functionality not offered in the state of the art technologies described in Section II-A. We summarize these requirements below:

1) A cryptographically secure identification scheme must be used to authenticate the parties involved in all cross-site communication.
2) Monitoring of VM resource allocation must be provided.
3) Monitoring of application-specific values must be provided.
4) Migration of VMs must be supported.
5) Migration must be possible to delegate.
6) A remote site must be able to make a request to the site from which the delegation of responsibility came to initiate immediate migration of a VM away from the remote site. Such requests must not be ignored.

The following sections contain an approach for extending upon the state of the art and meeting these requirements.

III. Migration

Migration of a VM requires that the source and destination host machines coordinate the transfer of the VM's definition and state files. We wish to set up such a transfer without losing the separation of concern between the Management layer and the Implementation layer or, similarly, the cross-site location-unawareness, while still adhering to the overall requirements of efficiency, security, and scalability.

To this end, we propose that each site maintains at least one *Transfer proxy* component in the Management layer. A Transfer proxy is a component that provides a layer of abstraction and information hiding, while at the same time associates an upcoming transfer with a transfer negotiation process. For scalability reasons, a site may wish to deploy several Transfer proxies. On behalf of the site, the Transfer proxies maintain a mapping between a VM identifier and where (at the Implementation level) its files are to be placed or read from. This mapping is called the *Transfer token*. The Transfer token is a unique identifier whose meaning is only known within the originating Transfer proxy. In this way, only the address of the Transfer proxy at a site is disclosed, not the address (or any other internal information) of the VM host at the Implementation level.

Using an approach similar to the File Transfer Protocol (FTP [10, Section 2.3]), the system uses separate control and transfer channels (out-of-band). The control channel is entirely contained within the Migration management protocol, maintaining the separation between the Management and Implementation layers.

A. Roles

Migration of a VM can be initiated for several reasons, and by several parties. In all cases, we can identify the following roles:

- *Controller*. The site responsible for setting up the transfer of the VM is denoted the Controller. It needs not awareness of the actual current placement of the VM, and is unaware of whether it communicates directly with the Source or Destination sites or indirectly via any number of Intermediary sites.
- *Source*. The Source is where the VM currently resides.
- *Destination*. The Destination is where the VM is to be deployed.
- *Intermediary*. Between the Controller and the Source/Destination sites, there may be any number of Intermediaries. On an intuitive level, the Intermediary acts as Controller on behalf of another Controller. This offers a layer of abstraction regarding the actual placement of the VM.

A site may, depending on context, take on several of the roles, e.g. act as both Controller and Source in the case of migration of a VM from the primary site that was also initiated by the primary site. Also, it should be noted that any site already involved in the management of a VM due to a prior delegation of responsibility can take on the role of Controller if needed.

Figure 1 shows an overview of the migration process. Because management of a VM can be delegated, the Controller is unaware of the actual placement of the VM on which site or host a VM is placed. Thus, the Controller only keeps track of what Intermediary site to contact in order to control the VM.

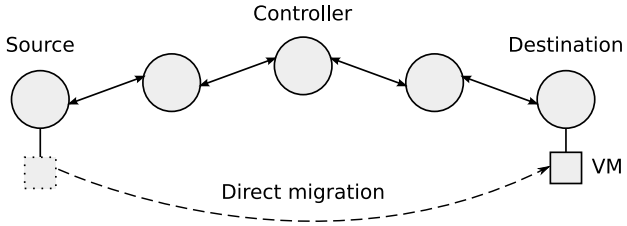

Figure 1: Migration process overview: solid lines denote control messages that are sent indirectly from the Controller to the Source site (where the VM is currently running) to the Destination site (where the VM should be migrated to), whereas dashed lines are used for direct network communication between sites. The transfer of the VM is carried out directly between the Source and Destination for efficiency reasons.

B. Operations

The operations of the protocol are as follows:

- *Migration request* (from the Controller and Intermediaries to possible Destinations). Migration requests are sent to possible future Destinations, to verify if the prospect site can accept a VM migration. The VM Descriptor and VM identifier are sent as input to the operation. Note that, depending on context and infrastructural/architectural policies, the remote site may in turn forward the request to another site and thus delegating the responsibility of managing the VM. The receiver of a Migration request may either accept or reject, depending on local site heuristics that take business rules and infrastructure capabilities into account. The return value contains a list of the information necessary to later issue the "Initiate transfer" operation presented below. This information includes the Transfer proxy's URI and the Transfer token.
- *Forced migration* (from either the Controller to a Source, or from the Source to the site that delegated the VM to the Source). A Forced migration request may be sent to indicate that a VM must immediately be prepared for migration away from its current placement. This call may be initiated by either a remote or the primary site, as a result of e.g. the Source site shutting down a host machine for maintenance (operation initiated remotely) or the primary site wishing to avoid SLA violations that are to be expected at the remote site (operation initiated by the primary site). Note that in addition to the Controller and Source sites, any Intermediary site involved with the VM may initiate a Forced migration as they are also stakeholders and may wish to optimize the placement of the VM in accordance with some site criteria.
- *Initiate transfer* (from the Controller to the Destination, via Intermediaries). The Initiate transfer operation is used to trigger the previously negotiated migration of the VM. The operation parameters contain the URIs and tokens of both the Source's and Destination's Transfer proxies.
- *Transfer verification* (from the Controller to the Source, via Intermediaries). This operation is issued by the Controller to the Source to verify that a given transfer was completed successfully.

C. Migration algorithm

In this section, we present the algorithms for the Controller, Destination, and Source sites. Intermediaries should only forward the requests along the path to the intended destination, and thus do not warrant an algorithm description.

1) At the Controller: The Controller migration algorithm is initiated either by the Controller being in the process of accepting a new service or VM, performing periodic placement optimization, or by a remote site requesting that a given VM should be migrated away invoking the Forced migration operation on the previous Intermediary in the chain of Intermediaries. Such an invocation may be regarded as an optional Step 0 for the Controller in the following algorithm. Note that *any* site along the path of responsibility may act as a Controller.

Name: Main migration algorithm.
Runs at: Controller.
Input: An event causing replacement has occurred.
Result: VM migrated to new location.

1) Let D be an empty list.
2) Let P denote the set of possible Destination sites, including the local site. For each $p \in P$ (performed in parallel):
   a) Call Migration request on p. The return value is a list. Add each returned tuple of $\langle p_{uri}, p_{tok} \rangle$ containing the returned Transfer proxy URI and Transfer token to the list D.
3) Sort D according to greatest benefit for the site and choose $d \in D$ to be the Destination.
4) Let s denote the Source and Invoke Forced migration on s. Store returned Transfer proxy URI as s_{uri} and Transfer token as s_{tok}.
5) Invoke Initiate migration on d, passing the tuple $\langle s_{uri}, s_{tok}, d_{uri}, d_{tok} \rangle$ as parameter. Store result as d_{stat}.
6) Invoke Transfer verification on s passing $\langle s_{uri}, s_{tok} \rangle$ as parameter. Store result as s_{stat}.
7) Unless $d_{stat} = s_{stat}$, go back to Step 5.

2) At the Destination: The Destination can be called by the Controller for two reasons, to handle migration requests and to react to the transfer initiation operation call.
Name: Migration request handler.
Runs at: Destination.
Input: VM identifier, VM descriptor.
Result: List of possible placement options.

1) Let D be an empty list.
2) If placement is possible locally, then for each possible local host h:
   a) Let T denote the set of Transfer proxies. Choose $t \in T$.
   b) From t, obtain Transfer token t_{tok} by supplying h and the VM identifier.

c) Add the tuple $\langle t_{uri}, t_{tok} \rangle$ containing the URI of the Transfer proxy and the Transfer token to D.

3) If delegation is allowed according to site policy:
   a) Act as the Controller does in Step 2. Add returned possible destinations to D.
4) Limit D to include only Destinations that should be exposed as possible Destinations, according to site policies (e.g. "only local" or "only preferred partner sites").
5) Return D.

Name: Initiate transfer handler.
Runs at: Destination.
Input: $\langle s_{uri}, s_{tok}, d_{uri}, d_{tok} \rangle$.
Result: "success" or "failure" of the transfer.

1) Forward the tuple of $\langle s_{uri}, s_{tok}, d_{uri}, d_{tok} \rangle$ to the Transfer proxy at d_{uri}.
2) At the Transfer proxy:
   a) Connect to s_{uri}, and supply s_{tok}.
   b) Begin copying VM-related files over a secure channel (e.g. scp).
   c) Return either "success" or "failure", depending on the status of the transfer.
3) Forward the return value from the Transfer proxy.

3) At the Source: The Source will be called upon to prepare the VM for migration using the Forced migration call, and to verify the transfer afterward. These operations are carried out as follows.

Name: Forced migration handler.
Runs at: Source.
Input: VM identifier.
Result: $\langle t_{uri}, t_{tok} \rangle$.

1) Let T denote the set of Transfer proxies. Choose $t \in T$.
2) From t, obtain Transfer token t_{tok} by supplying h and the VM identifier.
3) Return the tuple $\langle t_{uri}, t_{tok} \rangle$ containing the URI of the Transfer proxy and the Transfer token.

Name: Transfer verification handler.
Runs at: Source.
Input: $\langle s_{uri}, s_{tok} \rangle$.
Result: "success" or "failure" of the transfer.

1) Connect to the Transfer proxy at s_{uri}, supplying s_{tok} as parameter and ask for file transfer status.
2) Forward the return value from the Transfer proxy.

D. Remarks

The algorithm does not dictate how the Controller or Destination should obtain a list of possible destinations. In some contexts, it may be most appropriate to have a central Directory Service of all sites known in the cloud and for the site looking for a partner site to opportunistically connect to them all. In others, sites may have preferred partner sites, due to existing contracts or similar. We do not specify which alternative should or must be used, since they are equally applicable.

Note that the direct transfer from Source Transfer proxy to Destination Transfer proxy (Step 2b in the Main migration algorithm) could also be relayed through the Transfer proxies of the Intermediaries to provide complete location unawareness. However, this is more costly in terms of network traffic, so the suggested approach is to allow such out-of-band transfers to occur. This trade-off is however not dictated by design, but rather by infrastructural policies that govern the use of the system.

All requests sent are signed with the private key of the sender. This makes it possible for the receivers to verify the origin of the requests. This applies not only to the endpoints (Controller and Source/Destination), but to all Intermediaries as well.

Note that the local site is included in the set of possible Destinations in Step 2 of the Main migration algorithm. Therefore, the algorithm needs no modification to be used between hosts within the local site, as eligible hosts will be found in Step 2 of the Migration request handler algorithm.

IV. Monitoring

In a cloud environment, monitoring is performed for two main reasons: (a) to ensure that VMs get the capacity stipulated in the SLAs; and (b) to collect data for system-wide accounting of the resources that have been in use on behalf of the service providers, which is required for billing. There are two complimentary types of monitoring that must be carried out by the system: (a) infrastructure measurements, including low-level measurements of the resources used by the VM, such as the amount of RAM or the network bandwidth; and (b) KPIs specific to the application, e.g. the amount of currently logged in users at a server. Both types of values may be of a sensitive nature, and they must be appropriately secured.

Grid computing forms the basis for the infrastructure of cloud computing, and thus, we shall briefly consider monitoring in Grid environments. The Global Grid Forum has defined a Grid Monitoring Architecture (GMA) [11]. In the architecture, a central Directory Service allows a Producer of monitoring events to register itself, so that Consumers can look it up and start subscribing to such events. Consumers can also register, so that Producers may perform a lookup regarding Consumers. All components can be replicated as needed, and the system is inherently scalable. Many implementations of Grid monitoring are based on GMA, notably including R-GMA [12], which is being used in large projects such as the European DataGrid project [13] and gLite [14].

There are two sides to monitoring that require consideration: *what* to monitor, and *how* to monitor it. Using the terminology of the GMA, *what* to monitor and at what frequency data is generated is determined by the configuration of the Producers, whereas *how* to monitor the data and the frequency of such measurements is determined by the configuration of the Consumers. In the general two-layer cloud architecture discussed previously, the Implementation layer is the Producer, and the Management layer is the Consumer. If more than one site is involved in the management of a VM, the Management layers of many sites may be regarded as Consumers.

Monitoring of resources is conceptually performed either continuously or at discrete times. Continuous measuring is a special case of discrete measuring, where the interval between measurements is very small and the data is delivered like a stream. The measured data can be delivered according to any of the following schemes: (a) as soon as possible; (b) at regular intervals; or (c) when a predicate evaluates to true (such as when the measured quantity falls below some threshold value). Similar to continuous vs. discrete measurements, scheme (a) may be regarded as a special case of scheme (b).

The data can be returned in *raw* or *processed* form. In raw form, all data measured during the interval is returned. In processed form, some mathematical function has been applied to the data set, e.g. maximum, minimum, mean, or median. Processing the information at the Implementation level at the Source lowers the amount of required network bandwidth, as less data needs to be sent back to the Management layer at the primary site.

All VMs require monitoring — at the very least for accounting purposes. Also, both the Implementation and Management layers require full information to configure the monitoring Producers and Consumers correctly. Thus, it is appropriate to specify the monitoring frequency, delivery scheme, and required format in the VM Descriptor. Should changes be required during the lifetime of the VM, they can be made through the Control interface, where other VM Descriptor changes are made possible.

We therefore suggest that the VM Descriptor is augmented with the following additional configuration parameters regarding monitoring:

- A set of monitoring specifiers, one or more for each type of data point that can be monitored, including the following information:
  - The monitoring interval length in milliseconds. A value of zero indicates that monitoring should be performed as frequently as possible.
  - The delivery interval length in milliseconds. A value of zero indicates that monitoring data should be sent as soon as it is available.
  - The delivery format, i.e. raw or processed. If the data is to be processed, it should also be specified what mathematical function to process the data with (min, max, mean, or median).
- The public key of the primary site, used to enable optional encryption of monitoring data (see Section IV-B).

A. Obtaining measurements

Measurements on the infrastructure or implementation level are straight-forward to perform, as most (if not all) virtualization technologies offer monitoring support. Thus, we simply note that obtaining such values is possible using existing hypervisor-specific APIs and rather focus on obtaining KPI measurements.

Ideally, application software should not have to be altered for cloud deployment. However, without such alterations, it is impossible to allow the running service applications to report KPI values to the cloud monitoring system. Our approach is to require only that the application performs standard file I/O operations on a particular file system partition that is added to the VM. Using File System in User Space (FUSE) [15], the application is unaware that the data being written to a file is actually passed to a program, running in user space. We let this program be a wrapper for a database, where the data is actually stored. The VM host can then register triggers within the database to fire when new data arrives (alternatively, if database triggers are not supported, poll the database at the required interval), and the data is guaranteed by the database to adhere to the ACID (Atomicity, Consistency, Isolation, Durability) requirements [16]. Thus, the only modifications to the system running in the VM required for this approach is: (a) that FUSE or similar is enabled; and (b) that the application writes KPI monitoring data to a regular file on the FUSE file system partition. Using this approach, the VM host at the Implementation layer is able to extract both types of monitoring measurements, and may publish the data to the consumers at the configured intervals.

A more detailed description of the FUSE-based database-backed file system is as follows. The file system partition is created with two directories, *encrypted* and *unencrypted*. Files stored in the encrypted directory are assumed to be encrypted in their entirety, and therefore cannot be parsed by the Implementation layer. On the other hand, files stored in the unencrypted directory may be parsed by the Implementation layer, which makes it possible to process the data. Whenever a file is stored in the file system, regardless of whether it is encrypted or not, an entry is made in the database back end where the table name equals the file name. In addition to the data written to the file, a timestamp is added that marks the time at which the data was saved in the file system. Unencrypted files should be stored as comma-separated value (CSV) files, and be either one or two lines long. For a two-line file, the first line is assumed to contain the column names that shall be used in the database. These names can also be referenced in the SLAs (e.g. `current_users`). Columns without labels are given a standard name (such as "column$X$", where $X \in \{1, 2, \ldots\}$). A single-line file is regarded as a two-line file where the first line is empty, and thus the columns are all named according to the naming scheme.

If the service application is not modifiable (closed source application), but has some other type of monitoring facility, e.g. a protocol for querying a server via the network, a custom program can be written that obtains such values and writes them to the file system can be deployed alongside the main application.

B. Security

Monitoring data is sensitive, as it can be used to disclose information about the running application. We therefore suggest that there should be support for securing the data before it is transmitted over a network, although using such measures should be optional. Infrastructure monitoring data can be secured by encrypting it with the primary site's public key (as

the ultimate destination for the monitoring data is the primary site). Application-specific KPI values may on the other hand be too sensitive to disclose even at the Implementation level, since the VM may run at a cloud infrastructure provider that the customer has no trust relationship with. In that case, we suggest that it is encrypted using the primary site's public key before it is written by the application itself.

V. Discussion

The proposed protocols and interfaces conform to the requirements gathered from the Usage Scenarios defined in Section II. Usage Scenarios 1 and 3 are supported directly by the migration algorithm as it is presented, whereas Usage Scenario 2 additionally requires the optional Step 0 as described in Section III-C. The operations are carried out in a cryptographically secure manner, and the VMs can be monitored using the monitoring proposal of Section IV.

The work presented in this article has been developed in accordance with the principle of location-unawareness, as defined in Section II. The principle adds some complexity, as it requires control messages to be sent through a chain of Intermediaries rather than directly between the two endpoint sites. We argue that this is acceptable overhead, as the gains made by adhering to the principle are greater: (a) the system is distributed to a higher degree, which benefits scalability as the reliance upon a single point of failure decreases; (b) there is a more clear separation of concern as the Intermediaries may act as Controllers for a VM should their site policies dictate that placement should be altered; (c) there is less information exchange between sites, and thus less information to keep current using concurrency control schemes; and (d) adherence to the principle guarantees a more general system that can adapt to new environments and use cases, rather than requiring that all placement decisions are made at a central site. The less general case, where a VM must be placed at a site of the primary site's choosing, is merely a restriction on the approaches and architecture presented in this article — the algorithms need only be modified to disallow delegation of responsibility to Intermediaries. Such restrictions are best to make at deploy time, rather than at design time, since the generality of a design increases its applicability.

Transfer proxies, as presented in this article, make a logical chain of Intermediaries between the primary site and the Source where a given VM is being deployed. Should a site become unavailable due to e.g. a network error, such chains may be broken. Let us first note that policies of a site may prohibit it from ever acting as an Intermediary, thus, for the site, circumventing the problem of broken trust chains altogether. It is also reasonable to assume that a site may be configured to only consider acting as an Intermediary for another site that is within the same administrative domain. Since every Intermediary adheres to the terms in the SLA regarding the VM, and thus are at risk of paying for possible SLA violations, reasonable sites will not delegate responsibility to sites that cannot be trusted. Thus, these chains of Intermediaries will be only as short as the trust relationship between sites permits. The risk of a broken chain of trust must be weighed against the benefits offered by the possibility to delegate responsibility over VMs within the federated cloud. We argue that a distributed system design should be open-ended and *enabling*, rather than *restricting*. This allows the design to remain relevant as it can accommodate for more use cases and be adapted to more situations.

Monitoring data must reach all Intermediaries involved with a given VM, to ensure the sites that the VM is still running correctly — if it is not, placement has to be re-evaluated to avoid SLA violation penalties. The data may either be passed repeatedly through each Intermediary site from the Source to the primary site, or it may be placed in a common Enterprise Service Bus (ESB). Neither of these break the principle of location unawareness.

The overall design goals of any system are scalability, efficiency, and security. Let us now evaluate the suggested design from these perspectives. From a combined scalability and efficiency point of view, the suggested migration and monitoring interfaces are inherently scalable. If monitoring data is passed along path of Intermediaries, each site acts as both a Consumer and as a Producer of monitoring data. R-GMA has been developed with this type of data forwarding in mind [12], as it increases the scalability and efficiency of the system. ESB-style solutions are also inherently scalable and suitable for this type of application. The migration architecture, including the interface, is also very efficient and scalable, as it gives a high degree of autonomy to each site while at the same time requires very little network traffic overhead to perform migration. A high level of autonomy is important for scalability, as sites are more self-contained and thus the amount of information that has to be exchanged is reduced. Security is built in to both the migration and the monitoring interfaces and architectures, and the use of asymmetric encryption offers the confidentiality and integrity required.

Future work includes evaluation of the proposed solutions. Since the amount of control messages exchanged by the sites in the Transfer Proxy-supported migration is low, and the sizes of such messages is much lower than the transfer of the image files (which will be measured in gigabytes, rather than kilobytes for the messages), the added overhead in terms of network traffic must be assumed to be very low. To evaluate the monitoring proposal, a prototype is being developed as a proof of concept and its performance and usability will be tested.

VI. Related Work

Several standardization projects are in the early stages of developing interoperable cloud interfaces, such as the OCCI [17] working group at the Open Grid Forum. However, the topics KPI-aware monitoring and federation of clouds are deemed out of scope for the project. The authors of this work will be involved with developing extensions as contributions to OCCI to address these matters.

OpenNebula [18] is a open-source virtualization management software. It leverages various existing virtualization technologies and enables system administrators to administer a cluster

of hosts. It allows resources to be added dynamically, and advanced management functions such as workload distribution control and cluster partitioning. Migration within a single site is already supported, as is deploying VMs to Amazon EC2. Currently, the OpenNebula project is developed to support cross-site management functionality and cross-site migration of VMs.

Related Grid computing interfaces include WS-GRAM [19] and OGSA-BES [20]. The former is related to submission of jobs to a Grid, whereas the latter defines a state model, an informational model, and Web Service port types for management of Grid jobs. It also includes a proposed distributed monitoring of the resources where the jobs are running. The OGSA-BES allows for migration of Grid jobs, but does not specify how such migration should be implemented.

Resource scheduling and Grid brokering may be viewed as a theoretical basis for how to perform VM placement in a cloud computing environment. Relevant work in this field includes [21], [22]. The aforementioned research can be leveraged in Destination selection process in the Transfer Proxy-supported migration.

VM migration has been studied extensively in works such as [23], [24]. However, these works focus on the technical aspect of performing migration, rather than defining the interfaces for initiating and managing the migration process itself.

VII. Summary and Conclusions

We have presented two novel interface and architectural contributions, facilitating for cloud computing software to make use of inter- and intra-site VM migration and improved inter- and intra-site monitoring of VM resources, both on an infrastructural and on an application-specific level. Existing monitoring architectures may be leveraged, as the proposed monitoring solution is compatible with the Grid Monitoring Architecture [11], although it is proposed that a more highly distributed solution is used instead. The additions presented in the article adhere to a principle of location-unawareness, which increases scalability, decreases the degree of coupling between sites in the federated cloud environment, and makes a clear separation of concern between sites. The proposed additions expose a high level of generality, and are thus adaptable and usable in many scenarios, without being impractical to implement or standardize.

Acknowledgment

The authors are grateful to Daniel Henriksson and Johan Tordsson for their contributions to the foundation upon which the work was based. This work has been partly funded by the EU-IST-FP7-215605 (RESERVOIR) project.

References

[1] R. Buyya, C. Yeo, S. Venugopal, M. Ltd, and A. Melbourne, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC-08, IEEE CS Press, Los Alamitos, CA, USA)*, 2008.

[2] National Institute of Standards and Technology, Systems and Network Security Group, "Draft NIST Working Definition of Cloud Computing," 2009. [Online]. Available: http://csrc.ncsl.nist.gov/groups/SNS/cloud-computing/cloud-def-v12.doc

[3] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, L. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan, "The RESERVOIR Model and Architecture for Open Federated Cloud Computing," *IBM Systems Journal*, 2009, to appear.

[4] Distributed Management Task Force, Inc., "Open Virtualization Format Specification," DMTF 0243 (Standard), Feb. 2009. [Online]. Available: http://www.dmtf.org/standards/published_documents/DSP0243_1.0.0.pdf

[5] Xen community, "Xen Hypervisor Web page," 2003. [Online]. Available: http://www.xen.org/

[6] VMware Inc., "VMware Virtualization Technology Web page," Visisted March 30, 2009, 1999. [Online]. Available: http://www.vmware.com/

[7] Distributed Management Task Force, Inc., "System Virtualization Profile," DMTF 1042 (Preliminary Standard), Aug. 2007. [Online]. Available: http://www.dmtf.org/standards/published_documents/DSP1042.pdf

[8] ——, "System Virtualization Profile," DMTF 1057 (Preliminary Standard), May 2007. [Online]. Available: http://www.dmtf.org/standards/published_documents/DSP1057.pdf

[9] ——, "CIM System Virtualization White Paper," DMTF 2013 (Informational), Nov. 2007. [Online]. Available: http://www.dmtf.org/standards/published_documents/DSP2013_1.0.0.pdf

[10] J. Postel and J. Reynolds, "File Transfer Protocol," RFC 959 (Standard), Oct. 1985, updated by RFCs 2228, 2640, 2773, 3659. [Online]. Available: http://www.ietf.org/rfc/rfc959.txt

[11] B. Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski, and M. Swany, "A Grid Monitoring Architecture," GWD-I (Informational), Aug. 2002. [Online]. Available: http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-3.pdf

[12] A. C. et al., "The Relational Grid Monitoring Architecture: Mediating Information about the Grid," *Journal of Grid Computing*, vol. 2, pp. 323–339, 2004.

[13] B. Segal, L. Robertson, F. Gagliardi, and F. Carminati, "Grid computing: The European data grid project," Lyon, pp. 15–20, 2000.

[14] E. Laure, S. Fisher, A. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White *et al.*, "Programming the Grid with gLite," pp. 33–45, 2006.

[15] M. Szeredi, "Filesystem in userspace," 2004. [Online]. Available: http://fuse.sourceforge.net/

[16] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Computing Surveys*, vol. 15, no. 4, pp. 287–317, 1983.

[17] Open Grid Forum OCCI-WG, "Open Cloud Computing Interface," 2009. [Online]. Available: http://forge.ogf.org/sf/go/projects.occi-wg

[18] B. Sotomayor, R. Montero, I. Llorente, and I. Foster, "Capacity Leasing in Cloud Systems using the OpenNebula Engine," *Cloud Computing and Applications*, vol. 2008, 2008. [Online]. Available: http://www.cca08.org/papers/Paper20-Sotomayor.pdf

[19] Globus Project by the University of Chicago, "GRAM4," 2008. [Online]. Available: http://www.globus.org/toolkit/docs/4.2/4.2.1/execution/gram4/

[20] Open Grid Forum, "OGSA Basic Execution Services WG," 2005. [Online]. Available: https://forge.gridforum.org/sf/projects/ogsa-bes-wg

[21] E. Elmroth and J. Tordsson, "Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions," *Future Generation Computer Systems. The International Journal of Grid Computing: Theory, Methods and Applications*, vol. 24, no. 6, pp. 585–593, 2008.

[22] ——, "An interoperable, standards-based Grid resource broker and job submission service," in *First International Conference on e-Science and Grid Computing*, H. Stockinger *et al.*, Eds. IEEE CS Press, 2005, pp. 212–220.

[23] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer, "Xen and the Art of Virtualization," in *Proceedings of the ACM Symposium on Operating Systems Principles*, 2003, pp. 164–177.

[24] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2 table of contents*. USENIX Association Berkeley, CA, USA, 2005, pp. 273–286.