Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH Open Access

An adaptive framework for utility-based optimization of scientific applications in the cloud

Martin Koehler

Abstract

Cloud computing plays an increasingly important role in realizing scientific applications by offering virtualized compute and storage infrastructures that can scale on demand. This paper presents a self-configuring adaptive framework optimizing resource utilization for scientific applications on top of Cloud technologies. The proposed approach relies on the concept of utility, i.e., measuring the usefulness, and leverages the well-established principle from autonomic computing, namely the MAPE-K loop, in order to adaptively configure scientific applications. Therein, the process of maximizing the utility of specific configurations takes into account the Cloud stack: the application layer, the execution environment layer, and the resource layer, which is supported by the defined Cloud stack configuration model. The proposed framework self-configures the layers by evaluating monitored resources, analyzing their state, and generating an execution plan on a per job basis. Evaluating configurations is based on historical data and a utility function that ranks them according to the costs incurred. The proposed adaptive framework has been integrated into the Vienna Cloud Environment (VCE) and the evaluation by means of a data-intensive application is presented herein.

Keywords: Cloud; Cloud stack; Adaptive; Autonomic computing; Utility

Introduction

Executing scientific applications in a Cloud-based environment requires dynamic allocation of computing resources, provisioning of the underlying programming environments and the applications themselves. In addition, these applications are often Cloud-enabled by following the Software as a Service approach. Cloud computing [1,2] offers researchers the illusion of virtually infinite resources that can be allocated on demand and are accessible via the Internet. Nevertheless, researchers usually have to pay for the utilized resources when using a public Cloud environment [3,4] or, in case of a private Cloud, resources are not disposable for other experiments. Consequently, a shared goal of service providers and clients is the optimization of resource utilization while keeping costs and runtime of potentially time-consuming applications low. In general, researchers want to obtain results in a given period of time and they want to spend as little money as possible on compute resources. Cloud providers aim at serving as many researchers as possible in order to increase earnings and thus strive to optimize the utilization of resources.

This work presents an adaptive framework optimizing the utilization of Cloud computing resources as well as the runtime of an application. Within this context, the two main and at the same time contradicting objectives are the allocation of as little computing resources as possible and the minimization of runtime. The adaptive framework tackling this challenge on a per-job-basis relies on well-known concepts from autonomic computing [5,6], particularly on the MAPE-K loop containing a monitor, analyzer, planner, executor, and knowledge component. By accessing knowledge about historical jobs, the adaptive framework is able to effect the configuration of a specific job.

This approach represents knowledge by means of the concept of *utility* [7] known from economics, which measures the usefulness from the researchers' and the service providers' perspective. Utility takes into account the utilization of Cloud computing resources as well as the runtime of an application. For maximizing the utility of a

Correspondence: martin.koehler@ait.ac.at
Mobility Department, Austrian Institute of Technology (AIT), Giefinggasse 2, 1210 Vienna, Austria

specific job it is necessary to consider the configuration of all three layers of the Cloud service model: (1) the application layer where applications are provisioned based on the Software-as-a-Service (SaaS) concept, (2) the programming and execution environment layer, also referred to as Platform-as-a-Service (PaaS) layer, and (3) the resource layer, also known as Infrastructure-as-a-Service (IaaS) layer. Therefore, a comprehensive and generic model enabling the configuration of different implementations of the Cloud service model layers is needed. This paper presents a Cloud stack configuration model tackling this challenge by supporting the specification of layer-specific parameters to be taken into account during the utility optimization process.

The runtime of an application depends on the hardware characteristics and the amount of computing resources available or allocated (e.g. number of processors used). The programming environment layer (e.g. MapReduce programming model [8]) provides multiple configuration parameter sets which may effect the runtime. Additionally, the configuration of an application may impact the runtime as well. Optimizing the configuration of resources, the programming environment, or the application itself, are non-trivial problems on their own and a lot of research has been done in these areas.

A prototype of the proposed framework has been implemented and integrated in the Vienna Cloud Environment (VCE) [9], a versatile cloud environment for scientific applications, data sources, and scientific workflows. The VCE follows the Software as a Service (SaaS) model and relies on the concept of virtual appliances to provide a common set of generic interfaces to the user while hiding the details of the underlying software and hardware infrastructure. The adaptive framework is evaluated on top of a data-intensive application [10] from the field of high-throughput molecular systems biology [11], which has been Cloud-enabled with the VCE.

The remainder of this paper is organized as follows: The next section discusses related work followed by a section introducing the model for describing the different layers of the Cloud stack and presenting challenges regarding the configuration of these layers. Afterwards, the design of the proposed adaptive framework based on utility functions and autonomic computing concepts is delineated and the MAPE-K loop components are presented in detail. Subsequently, the adaptive utility-optimization process is described on the basis of a cloudified data-intensive application. Finally, a conclusion of the work including future work completes this paper.

Related work

The paper investigates how utility functions and adaptive technologies, namely the MAPE-K loop, can be utilized to configure the Cloud stack towards optimizing runtime and resource utilization for specific jobs. To place this in context, this section reviews work on utility-based optimization and adaptive methods for scheduling and resource allocation.

In adaptive systems relying on feedback loops, e.g. the MAPE-K loop, various concepts for managing knowledge are established and could be utilized (e.g., Concept of Utility, Reinforcement Learning, Bayesian Techniques) [6]. The basic idea is the provisioning of knowledge about the system and to use it in the process of decision-making. In autonomic computing decision-making has been classified in action policies, goal policies and utility function policies [12]. While action policies define specific actions taken in response to sensed information, goal policies identify actions potentially leading to the desired state of the system. Utility function policies are based on the concept of utility from economics and are used to measure and quantify the relative satisfaction of customers with goods.

The principle of utility functions has been applied to diverse problem statements for resource allocation in multiple domains. In [13], utility functions in autonomic systems are used to continually optimize the utilization of managed computational resources in a dynamic, heterogeneous environment. The authors describe a system that is able to self-optimize the allocation of resources to different application environments. In contrast to their approach, we try to reduce the costs (resource usage, runtime) for a single application by automatically configuring the environment. In [14], a utility-based resource allocation and scheduling process for wireless broadband networks is described. The approach uses utility-based resource management and QoS architecture enabling an optimization system where only the utility function has to be changed for new applications. In this sense, their approach is quite similar to the approach described herein. In [15] the authors present an architecture for the implementation of self-manageable Cloud services which in case of failures or environmental changes manage themselves to fulfill the guaranteed quality of service. Whereas they focus on the quality of service, our paper uses a similar approach to optimize the resource utilization and the runtime of applications in the Cloud. Research on QoS-aware scheduling for heterogeneous datacenters is presented in [16]. Their work is not based on utility functions but likewise our work, their approach leverages from information the system already has about applications.

In [12], the authors apply autonomic computing concepts and utility functions on adaptive resource allocation for concurrent workflows. They define utility based on response time and profit, realize dynamic and adaptive workflow optimization on top of Pegasus and DAGMan and present the feasibility of the approach on top of five workflow-based scenarios. In contrast to our approach, they focus on the assignment of specific workflow tasks to execution sites. Additional related work on utility-directed resource allocation has been published in [17]. They apply this approach in virtual desktop clouds with the objective of improving the user experience. Their approach deals with resource scheduling while the work presented herein tries to optimize the utility for single jobs by configuring the Cloud stack.

In [18], an adaptive resource control system is presented. It adjusts the resources to individual tiers in order to meet application-level Quality of Service (QoS) goals. That is, to increase the resource utilization in a data center by taking into account the application level QoS. The premier objective of our framework is to reduce the amount of utilized resources for individual jobs.

The Automat toolkit [19] is a community testbed architecture that targets research into mechanisms and policies for autonomic computing that are under closed-loop control. The toolkit enables researchers to instantiate self-managed virtual data centers and to define the controllers that govern resource allocation, selection, and dynamic migration.

Cloud stack configuration model

To achieve the goal of maximizing the utility for a specific job, the adaptive framework has to take into account the resource utilization as well as the runtime of the application. This approach assumes that both, utilization and runtime, depend on the configuration of the application, the execution environment (platform), as well as the resources. This assumption raises the need for a generic model enabling the specification of layer-specific parameters to be taken into account during the utility-optimization process. Thus, we defined a model of the Cloud stack (see Figure 1) comprising three layers: the application layer (SaaS), the execution environment layer (PaaS), and the resource layer (IaaS). For each layer a declarative descriptor captures a set of configuration parameters that might impact the resource requirements and the runtime of an application by assuming that a finite parameter set is sufficient for optimizing the resource utilization. The definition of the concrete set of parameters at each layer, which should be configured adaptively, should be hidden from the end user but has to be done by experts on the specific layer (e.g. application or service provider). Specifically, defining a concrete set of parameters spanning the whole Cloud stack requires deep knowledge about resource allocation, resource and environment configuration, and application behaviour. Thus, this approach provides a flexible model enabling the definition of a variety of parameters, but promotes the minimization of the parameters defined with the objective to reduce the overall complexity.

Representation of descriptors

The application descriptor, the environment descriptor, as well as the resource descriptor are defined in a generic manner by enabling the definition of element-specific parameters. All descriptors are defined on the basis of XSD schemes which include generic key/value pairs for defining parameters. Additionally, the XSD schema supports the definition of the scope of each parameter to

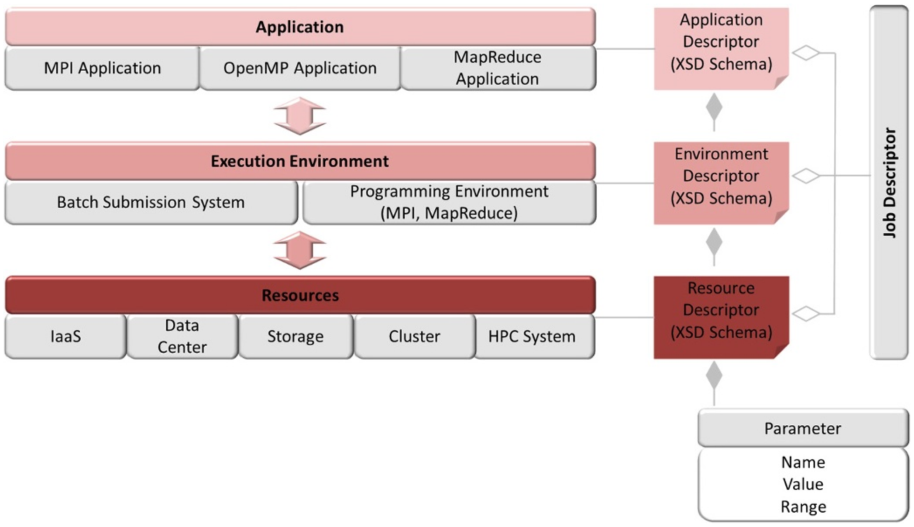

Figure 1 Layered Cloud stack model. Each layer (application, execution environment, resources) can include multiple elements, each describable by an XML descriptor. Each XML descriptor includes a list of parameters with a name, a value, and a range. A job descriptor consists of one or more application, environment, and resource descriptors.

be considered during job configuration. By following this approach, different applications, execution environments, and resources can be easily supported.

Application descriptors

The purpose of the application descriptor is to explicitly describe a set of application-specific parameters that should be tuned by the adaptive framework. Many applications provide a large set of configuration parameters enabling the tuning of an application regarding specific hardware resources. Moreover, experiments often rely on the execution of parameter studies which can be configured in different ways. Depending on the input parameters and the available resources, it may be appropriate to change the configuration of an application. The application descriptor has to be specified by the application provider by defining a specific set of parameters.

Environment descriptors

The execution of an application may require different execution environments depending on the application characteristics and underlying resources. On HPC systems and clusters, usually batch submission systems are utilized for allocating the computing resources. In case of a virtual cluster in the Cloud, a batch submission system can be configured on demand for scheduling the jobs dependent on the virtual execution appliances available. Thus, scheduling system-specific parameters that have to be set at job submission time can be defined via the environment descriptor.

For scientific applications usually a parallel execution environment such as MPI, OpenMP, or MapReduce is utilized. Most of these environments provide a huge set of configuration parameters that may impact the runtime of an application. For example, the Apache Hadoop framework supports an enormous set of parameters enabling to specify the number of map and reduce tasks, the configuration of the Java Virtual Machines, or how many tasks can be executed in parallel (on a single machine). The provisioning of a "good" configuration for such a system can be very complex. With our approach, those parameters that should be taken into account by the adaptive framework are defined in the environment descriptor and set upon job submission time to improve the behavior of the system for a specific job.

Resource descriptors

The purpose of the resource descriptor is to explicitly describe a set of computing and storage resource-specific parameters that should be taken into account by the adaptive framework. Within Cloud environments, often virtual clusters consisting of a variable set of computing resources with different CPUs and memory sizes are utilized during job execution. HPC systems provide different hardware architectures to consumers (e.g. multicore CPUs, GPUs) suitable for the execution of different applications. Resource descriptors enable an explicit description of the compute resources to be considered for the execution of a specific job.

Additionally, many applications require processing of large data sets, especially in the area of data-intensive computing. Storage environments, such as for example, the Hadoop Distributed File System (HDFS), provide a huge set of configuration parameters effecting the systems behavior. For example, the HDFS file system enables the definition of the block size of files (how a file is split across multiple distributed storage nodes) and the replication factor (how often the file is replicated). Adjusting these parameters is often not feasible for single jobs because their configuration is time-consuming. Nevertheless, these parameters effect the runtime of the application and have to be considered during the job configuration (e.g. changing the replication of a huge file in a distributed file system may require prohibitive data transfers), but they may impact the job configuration process. The resource descriptor supports the specification of these parameters.

Often, Cloud-based applications necessitate the configuration of a specific network stack, for instance a private virtual network provided by the Cloud offering. Currently, networking parameters are not in the scope of this work, but, by the generic design, resource descriptors are capable of representing different networking requirement.

Job descriptors

The purpose of the adaptive framework is to adaptively configure a job upon submission time on the basis of the application, the environment, and the resource descriptor(s). Therefore, a job descriptor comprises application, environment, and resource descriptors which consist all job-specific parameters to be configured. The current implementation of the Cloud stack configuration model is quite simple but very flexible due to the fact that any parameter can be represented and different types of descriptors are available to define diverse components of the Cloud stack including the application, multiple platforms, different resources (storage, compute, networking). Currently, consistency between the different descriptors is not assured by the system itself, but has to be taken into account by carefully defining the parameter set for all descriptors.

Design of the adaptive framework

On top of the Cloud stack model representing the configuration of all three layers, an adaptive framework for optimizing the utility of a specific job regarding these layers has been designed. The main objective of the adaptive framework is the optimization of the utility for a specific job which is achieved by adaptive configuration of these layers.

The adaptive framework is utilized for configuring the resource layers and for self-optimizing a specific job regarding runtime and resource utilization. Therefore, the design of the framework follows the MAPE-K loop [6], which is a well-established concept in autonomic computing. The MAPE-K loop comprises a monitoring, an analyzing, a planning, an execution, and a knowledge element and has the objective of designing systems with self-* characteristics (self-configuration, self-healing, self-protection, and self-optimization) [20]. The adaptive framework itself acts as autonomic manager of the different layers of the Cloud stack and the planning component relies on the utility function. The generic nature of the framework enables the adaption of the objective without changing the framework itself, but by defining different utility functions supporting varying target functions. The design of the framework is shown in Figure 2.

Managed resources

The adaptive framework has been designed to manage resources at all three Cloud stack layers involved. The framework provides sensor and effector interfaces, following the definition of the MAPE-K loop, for gaining actual information about resources and their utilization and for changing the state of the resources.

Multiple execution environments may be involved during the job execution including the scheduling system and the programming environment (e.g. MPI, MapReduce). Currently, the Hadoop Framework can be configured by changing the configuration files or by setting parameters at job execution time.

The management of the resource layer provides an interface to the computing and storage resources. Computing resources may be HPC resources and clusters managed by a batch scheduling system. In case of the Oracle Grid Engine (OGE), information about the allocatable resources can be retrieved via the Distributed Resource Management Application API (DRMAA) [21]. In case of private or public Cloud environments, the management can be done over the Cloud environment's API. Storage resources include distributed file systems, such as for example, the HDFS, and Cloud storage solutions as provided by Amazon (Amazon S3).

Knowledge

To realize a framework capable of adaptively configuring the application, the execution environment, and the resources, there is a need to integrate knowledge gained from previous configurations of the system. Following the concept of the MAPE-K loop, this knowledge is made available to the framework via the knowledge component and used in the process of decision-making. The knowledge is automatically revised during the execution by the

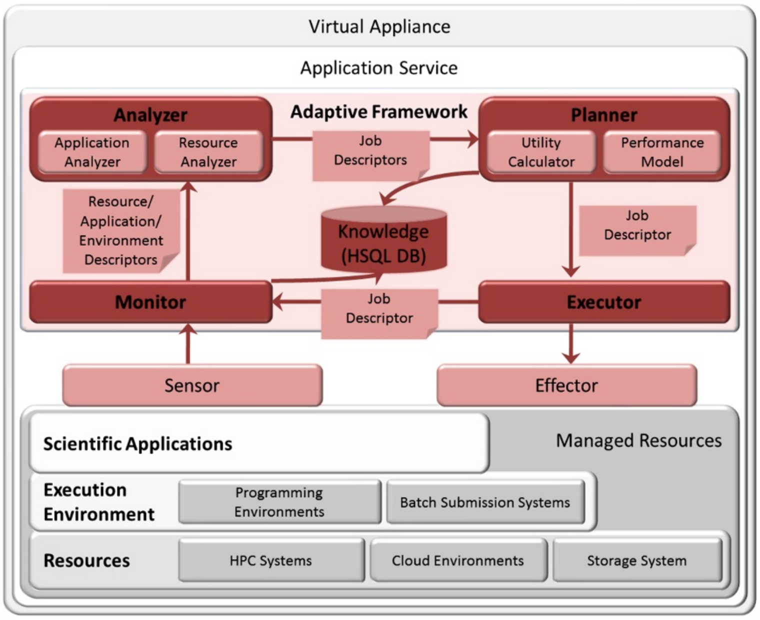

Figure 2 Design of adaptive framework. The design follows the concept of the MAPE-K loop for managing the configuration of the Cloud stack for a specific job. The framework monitors the managed resources, analyzes the state of the resources, generates an execution plan based on knowledge, and executes this plan on the resources.

adaptive framework by integrating new insights gained from additional runs of the application.

In this approach, the concept of utility, describing the usefulness of a specific configuration, is utilized for representing knowledge. This enables the representation of varying goals from different stakeholders by defining different utility functions. For example, a specific configuration may have a different utility for the researcher or for the service provider depending on their goals.

The utility U of a job [22] is defined as $U(A, E, R)$, where $A = \{a_1, \ldots, a_n\}$, $E = \{e_1, \ldots, e_m\}$, and $R = \{r_1, \ldots, r_k\}$ represent the parameter set at the application layer, the execution environment layer, and the resource layer, respectively. Different configurations are ranked on the basis of their utility. If $U_c(A_c, E_c, R_c) > U_{c'}(A_{c'}, E_{c'}, R_{c'})$, then configuration C is ranked higher than configuration C'. The configuration with the highest utility is chosen for job execution. The utility of a configuration is normalized in the range [0, 1]. The utility function itself is defined as Sigmoid function depending on a function $f(A, E, R)$. This function f is scenario-specific and has to be defined by domain experts. The equation of the utility function is depicted in the following:

$$U(f) = -e^{-5e^{-0.5*f}} + 1 \tag{1}$$

A Sigmoid function has been chosen because it highlights a small set of "good" configurations, slopes promptly, and settles down on a low value. Hence, the function (1) fits the problem of accentuating good job configurations.

The knowledge itself is captured within a database system which stores application-, environment-, and resource-descriptors of previous application executions. In our framework we utilize a HSQL database system for the knowledge base following a generic database schema. For each job, the runtime of the job, the utility of the job, and estimated values for runtime and utility (during planning phase) are stored and made available to the framework. A `parameter` table is used to store parameters specific to the managed resources. After a job has been finished, the utility of this job is calculated based on the runtime of the job, and both values are added to the knowledge base.

Monitor

The monitoring component is responsible for sensing the involved managed resources and for providing this information to the autonomic manager. Sensing the resources results in the generation of actual application, environment, and resource descriptors. These descriptors refer to the actual configuration of the managed resource (e.g. OGE). The adaptive framework has to monitor multiple resources at the different layers (IaaS, PaaS, SaaS). Therefore, the monitor relies on a component-oriented architecture, which enables simple integration of new monitor components for monitoring specific resources (e.g., different set of resources).

The realization of the monitor relies on the DRMAA API [21] for communicating with OGE. By utilizing the Java API, the monitor retrieves information about free computing resources and their configuration (e.g. number of nodes). Additionally, information about free computing resources in the private Cloud environment are provided via the KVM API [23] on the Cloud nodes. Information about the HDFS storage resource, including the block size, replication factor, and the size of the file, is retrieved by utilizing the Hadoop API. The configuration of the Hadoop framework is available via the Hadoop configuration files. Application and job specific information is supplied by the user, including the number of input files and the needed database.

Analyzer

The analyzer is responsible for evaluating the actual configuration of all layers involved. The analyzer adopts a component-based architecture, as depicted in Figure 3, and can be composed of multiple specific analyzers, for analyzing the configuration of specific resources. The analyzer executes all specific analyzer components sequentially. The basic execution chain of analyzer components is bottom-up, starting from the resource layer, next the execution environment, and finally the application layer. The execution chain can be changed by the service provider if required. The analysis phase of the adaptive framework results in the provisioning of job descriptors depicting the possible configurations on all layers that should be taken into account by the planner.

Each analyzer component provides the same generic interface retrieving a set of actual job descriptions and the resource specific description and examines the layer-specific parameters to generate a set of corresponding job descriptors. For example, if the resource descriptor includes a virtual cluster consisting of ten virtual appliances, the resource analyzer component creates ten different job configurations describing the utilization of one up to ten virtual appliances. In order to limit the number of different configurations, the range of the different parameters has to be restricted by the service provider. The analyzing phase results in a set of feasible job configurations, each specifying a different configuration of the resources on each layer. One aspect with this approach is to balance the amount of parameters with the accuracy of the approach. On the one hand, we try to minimize the amount of generated job configurations by utilizing as less parameters as possible at each layer. On the other hand, the utilized parameters have to be chosen carefully to retrieve appropriate and accurate results. The parameter set has high impact on the complexity and the accuracy

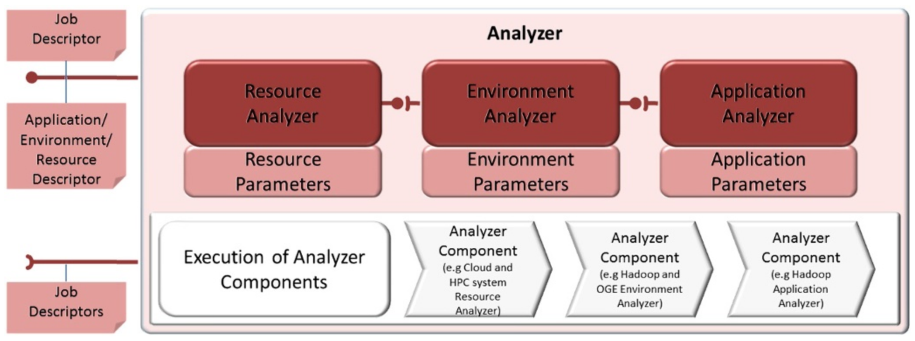
Job Descriptor
Application/ Environment/ Resource Descriptor
Job Descriptors
Analyzer
Resource Analyzer
Environment Analyzer
Application Analyzer
Resource Parameters
Environment Parameters
Application Parameters
Execution of Analyzer Components
Analyzer Component (e.g Cloud and HPC system Resource Analyzer)
Analyzer Component (e.g Hadoop and OGE Environment Analyzer)
Analyzer Component (e.g Hadoop Application Analyzer)

Figure 3 MAPE-K analyzer. The analyzer is comprised of multiple analyzer components, each being aware of the configuration of one managed resource. The execution of analyzer components is chained and has to be defined by the service provider. The default execution chain is depicted.

of the solution. Thus, we propose that parameters should be defined by experts in the whole system (Cloud stack model) and think about future extensions towards expert recommendation systems enabling automatic decisions on the parameters to use.

The complexity of the configuration scope can be explained as follows: n, m, k define the number of parameters $|A| = n$, $|E| = m$, $|R| = k$ and $a_i : u_i$, $e_i : v_i$, $r_i : w_i$ define the number of possible values per parameter. Thus, the number of possible configurations L is defined as $\prod_{i=1}^{n} u_i \prod_{j=1}^{m} v_j \prod_{j=1}^{k} w_l$. If we claim that $u_i, v_i, w_i \geq 2$ (saying that we have at least two possible values for each parameter), we can state that the number of configurations is exponential and at least $L \geq 2^n 2^m 2^k = 2^{nmk}$. Thus, reducing the problem scope is necessary for assuring acceptable runtime of the approach.

The prototype implementation of the adaptive framework includes an application-analyzer (application specific parameters), an environment-analyzer (Hadoop specific parameters), and a resource-analyzer (computing resource parameters). The resource-analyzer generates a set of job descriptors by utilizing information about the available computing resources (cluster and Cloud resources). Either cluster or Cloud resources are utilized by the system. The environment-analyzer provides job descriptors including different Hadoop configuration parameters. The basic implementation takes into account if compression should be utilized within Hadoop, and the number of Map and Reduce tasks which can be executed in parallel on one node. The application-analyzer configures the execution of parameter studies by evaluating the number of input files to be matched. The application-analyzer evaluates the possibility of splitting the input files to multiple jobs and generates a set of job descriptors including different configurations.

Planner

Within the MAPE-K loop, the planning component is responsible for acting on top of the information gained by the analyzer component. Herein, the planner is in charge of choosing the job configuration with the highest utility. This is done by means of knowledge and a planning strategy on the basis of the concept of utility.

According to the design, different planners could be implemented following different approaches for ranking the set of configurations. The approach utilized within this work is based upon a utility function, which enables ranking of different configurations based on the concept of utility, which is used in economics to describe the measurement of 'useful-ness' that a consumer obtains from any good [24].

In this approach, the planner uses internally a utility calculator component for estimating the utility of a specific job configuration. The utility calculator calculates the utility for a specific job configuration on the basis of the utility function. The utility function takes into account a set of parameters included in the job description and the runtime of the job. Thus, an application-specific performance model is needed to estimate the utility. Additionally, the parameter set changes due to different types of involved resources, execution environments, or applications. Therefore, the utility function has to be adapted according to the specific application scenario.

Regarding to the estimated runtime and job-specific parameters the utility for a job description is calculated and the planner ranks all job descriptions on this basis. The job configuration with the highest utility is chosen for execution.

The basic design of the planner including a utility calculator and an application-specific performance model is depicted in Figure 4.

Utility calculator

The utility calculator computes the utility of a job descriptor taking into account parameters within the application, environment, and resource descriptors as well as the estimated runtime as obtained with the performance model. The utility function depends on parameters from

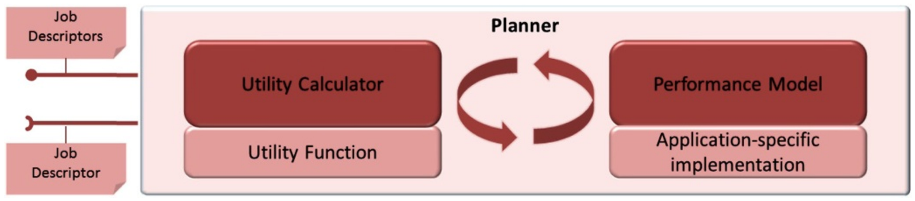

Figure 4 MAPE-K planner. The planner component consists of an application specific performance model and a utility calculator. The utility calculation is done on the basis of a utility function. The planner calculates the utility of each provided job description and ranks them afterwards by means of the utility.

the application A, the environment E and the resource R layer.

As specified, a scenario dependent function f considering the parameter sets available at the resource, execution environment, and application layer, has to be defined to be used in conjunction with the utility function. On the basis of the utility function (1) and the function f the utility of a job is estimated and different job configurations can be ranked. The prototype implementation of the adaptive framework focuses on a small sample set of parameters, specifically the parameters $n \in R$, the number of nodes, $d \in A$, the number of input files, $s \in R$ the database size, and $r \in A$ the estimated runtime of the job. Therefore, a scenario function $f(n, d, s, r)$, normalizing the parameters for the calculation of the utility, is defined in (2).

$$f(n, d, s, r) = w_f \frac{(w_n n)\ (w_r r)}{(w_d d)\ (w_s s)} \tag{2}$$

The function f is defined on the basis of the utilization of the resources (runtime and number of nodes) and the data transfer (database size and number of input files). Each value can be weighted with a factor $w_p(p \in f, n, d, s, r)$ for setting the importance of the parameter for calculation of the utility. According to initial experiments, these weighting factors have been set to $w_n = 0.3$, $w_d = 0.4$, $w_s = 0.6$, and $w_r = 0.7$. Additionally, the number of input files is normalized before applying the function. The size of the input files can have impact on the runtime of an application. Currently, this fact is not considered in the scenario dependent function due to the characteristics of the sample application. A scaling factor of $w_f = 20$ has been chosen to scale the function according to the utility function. The weighting factors within the function f have been chosen according to initial experimental results.

Performance model

Predicting the accurate runtime of an application usually is a complex, often not realistic task [25]. Similarly, a complete multidimensional regression including all parameters involved requires a large amount of test cases to deliver appropriate results. For these reasons, we propose the utilization of a history-based performance model. Nevertheless, the generic design of the adaptive framework supports the utilization of different performance models dependent on the application.

The history-based performance model is realized on the basis of knowledge about previously run jobs with a similar configuration and of parameter-specific regression functions. Therefore, the prototype implementation defines regression functions for a subset of parameters of R (resource) and A (application), which are considered in this approach. The prototype focuses on three specific parameters including the number of nodes allocated for a specific job n, the size of the database s, and the number of input files to be compared d and has been evaluated within a case study. Therein we retrieved accurate results on different computing resources [22]. Following this, the performance prediction is based on historical execution time and regression functions. While the approach could be easily extended with support for additional parameters, it is shown that considering only a subset of parameters can results in appropriate estimations. Different types of computing resources implicate changes in the runtime of the application. Therefore, the regression functions are not completely independent of the utilized computing resources (Cloud or cluster resources), but have to be adapted with a weighting factor regarding the allocated computing resources.

Executor

The task of the executor is to configure the three Cloud stack layers according to the chosen parameter configuration and to execute the application job. First, the executor reserves and configures the computing and storage resources (or initializes them in case of a virtual cluster within a Cloud environment) regarding the resource parameters specified in the configuration. Then, the executor configures the attached execution environment as well as the application. After all three layers have been configured, the application is executed according to the defined job description.

After job completion, the executor evaluates the execution (runtime of job and recalculated utility) and stores the gained information in the knowledge base so that it can be utilized for planning of forthcoming job submissions.

Case study: adaptive configuration of a MapReduce application

A prototype of the adaptive framework including all introduced components has been implemented within the Vienna Cloud Environment (VCE) [9]. In the following we report on experiments with a MapReduce application from the domain of molecular systems biology [26,27].

At the application layer, support for the execution of the MoSys application [28], matching tryptic peptide fragmentation mass spectra data against the ProMEX database [11], has been implemented. The application supports the execution of parameter studies, in particular the comparison of hundreds of files against the database. The framework adaptively splits or combines these input files into multiple jobs and schedules their execution on different sets of computing resources. In the case study a private cloud environment and a cluster system have been utilized for executing the application. The cluster consists of eight compute nodes, each equipped with two Nehalem QuadCore processors, 24 GB of memory, and 6 TB of disk space, and is interconnected via Infiniband and Gigabit Ethernet. The private Cloud environment is connected via Gigabit Ethernet to the frontend and includes four computational nodes, each equipped with two 12-core AMD Opteron processors and 48 GB of main memory, 12 TB of storage and virtualized via KVM.

The job execution scenario is based on a test case executing a parameter study with 1000 input files against a 500 GB database. Following this scenario, the adaptive job configuration process within the adaptive framework is explained.

The job life cycle starts with a user who uploads the input files to the VCE application appliance via the Web service interface. Afterwards, the user starts the execution of the job via the application service's `start()` method and the service initiates the adaptive framework. Hence, the MAPE-K loop for this job is started, and the monitor queries the actual configuration of the application, the Hadoop environment, and the available computing resources. The monitor generates a *ResourceDescriptor* describing the available compute nodes on the cluster and the private Cloud. For this test case, it is assumed that eight compute nodes are allocatable on the cluster and up to twelve virtual machines, each containing eight virtual CPUs, can be started in the private Cloud. Additionally, an *ApplicationDescriptor* is generated including actual information about the job. Information stored in the *ApplicationDescriptor* includes the size of the database, and the number of input files.

Configuration of the job

The analyzer executes the chain of analyzer components consisting of a *ResourceAnalyzer*, an *EnvironmentAnalyzer*, and an *ApplicationAnalyzer*. Each component analyzes the parameters stored in one descriptor and generates a set of possible job descriptors.

The resource analyzer takes care of the resource descriptors provided by OGE and the private Cloud environment. The analyzer component generates eight different job configurations on the basis of the cluster resource descriptor setting the amount of compute nodes for this job between one and eight nodes. Additionally, the analyzer component creates twelve job configurations setting the amount of Cloud nodes to be allocated from one to twelve.

The *EnvironmentAnalyzer* sets Hadoop specific parameters according to the resource specific parameters. For example, the number of parallel map tasks on one node is set to eight, according to the number of cores per node.

Finally, the *ApplicationAnalyzer* evaluates the possibility to split the parameter study into multiple jobs. The test case job compares 1000 files against the database. In this case, the *ApplicationAnalyzer* generates job descriptors with one job matching 1000 input files, two jobs matching 500 input files, 4 jobs matching 250 input files, and so on. In order to simplify the procedure, the test case does not further discuss the job descriptors created by the *ApplicationAnalyzer* and explains the adaptive job configuration process on the basis of the generated resource descriptors.

The generated job descriptors are depicted in Table 1. Each line includes the Job ID, the number of input files, the database size (DB Size), the number of compute nodes including the type and the number of CPUs (Nodes(Type - CPU)), and the number of jobs generated (for the purpose of simplification all possible values are shown within one job descriptor).

Utility calculation

The planner ranks the job descriptors generated by the analyzer on the basis of the utility function and the underlying performance model. The planner estimates the performance of each job description following an application-specific performance model and information about previously run jobs stored in the knowledge base. Afterwards, the utility calculator is utilized for computing the utility of each job descriptor on the basis of the estimated performance. The planner ranks the job descriptors on the basis of the calculated utility and selects the descriptor with the highest utility for execution.

In the following the process of the utility calculation is explained in detail on the basis of the assumption that the knowledge base stores information about three historical jobs as shown in Table 2. The first job in the knowledge base has been a job matching 1000 input files against a 100

Table 1 Scope of job: this table depicts the job descriptors generated by the analyzer components (resource, environment, application) during the adaptive configuration of the test case

Job ID	Input Files	DB Size	Nodes (Type)	Number of Jobs
1	1000	500	8 (cluster - 8)	1,2,4,5,8,...,1000
2	1000	500	7 (cluster - 8)	1,2,4,5,8,...,1000
3	1000	500	6 (cluster - 8)	1,2,4,5,8,...,1000
4	1000	500	5 (cluster - 8)	1,2,4,5,8,...,1000
5	1000	500	4 (cluster - 8)	1,2,4,5,8,...,1000
6	1000	500	3 (cluster - 8)	1,2,4,5,8,...,1000
7	1000	500	2 (cluster - 8)	1,2,4,5,8,...,1000
8	1000	500	1 (cluster - 8)	1,2,4,5,8,...,1000
9	1000	500	12 (cloud - 8)	1,2,4,5,8,...,1000
10	1000	500	11 (cloud - 8)	1,2,4,5,8,...,1000
11	1000	500	10 (cloud - 8)	1,2,4,5,8,...,1000
12	1000	500	9 (cloud - 8)	1,2,4,5,8,...,1000
13	1000	500	8 (cloud - 8)	1,2,4,5,8,...,1000
14	1000	500	7 (cloud - 8)	1,2,4,5,8,...,1000
15	1000	500	6 (cloud - 8)	1,2,4,5,8,...,1000
16	1000	500	5 (cloud - 8)	1,2,4,5,8,...,1000
17	1000	500	4 (cloud - 8)	1,2,4,5,8,...,1000
18	1000	500	3 (cloud - 8)	1,2,4,5,8,...,1000
19	1000	500	2 (cloud - 8)	1,2,4,5,8,...,1000
20	1000	500	1 (cloud - 8)	1,2,4,5,8,...,1000

GB database and the second job compared one input file against a 500 GB database. Both jobs have been executed on cluster resources. The third job compared one input file against a 500 GB database by utilizing eight Cloud nodes.

The process is explained on the basis of the job configuration with `Job ID 1` depicted in Table 1 (1000 input files, 500 GB, 8 Cluster Nodes). Both job configurations from the knowledge base, which have been executed on the cluster, differ in one parameter from job configuration one. Thus, the application-specific performance model is utilized to calculate the runtime of the job descriptor by using the runtime of both historical jobs. As a result an estimated runtime of 8030.75 seconds is computed and used as basis for the calculation of the utility.

Table 2 Knowledge base: this table depicts the job descriptors, including the utility and the runtime, stored in the knowledge base before the job execution starts

Job ID (KB)	# Input Files	DB Size	Nodes (Type)	Time	Utility
1 kb	1000	100	8 (cluster - 8)	1722	0.5908
2 kb	1	500	8 (cluster - 8)	1123	0.6458
3 kb	1	500	8 (cloud - 8)	2525	0.5266

Afterwards, the utility function is applied to rank the job configurations regarding runtime and resource utilization. In Table 3 six assessed job configurations are shown. The planner choses the job configuration with the highest utility. In this case, the job configuration with eight cluster nodes and the utility of *0.6111* is chosen for execution. The job descriptors obtained during the planning phase, including the estimated runtime and the estimated utility, are shown in Table 3.

Job execution and results
Finally, the executor manages the execution of the job configuration. Therefore, eight nodes are allocated on the cluster and the Hadoop framework is initialized according to the chosen configuration. After the job has been finished, the information about the job in the knowledge base is updated with the actually measured runtime. The utility of this job configuration is recalculated on the basis of the runtime and stored accordingly. Table 3 shows the runtime and the utility of the executed job.

By comparing the estimated runtime with the real runtime of the job it can be seen that the best job configuration in terms of the runtime has been chosen by the framework for execution. Additionally, the behavior of the system can be changed by adapting the utility function to favor less resources instead of less runtime.

In this section, a prototype implementation of the adaptive framework has been presented. The realization has been based on MapReduce which is known for its scalability. Additionally, only three parameters have been taken into account in the performance model. Due to the characteristics of MapReduce, this approach delivers appropriate results in terms of resource utilization and job runtime. Nevertheless, utilizing this approach for different applications would require to specify application-specific performance models, which may not be possible for other applications, and a detailed analysis of how the utility is defined.

Conclusion
In this work, an approach towards an adaptive configuration of the Cloud stack regarding the optimization of the utility for a specific job is described. The utility of a job configuration is defined as its usefulness for a stakeholder with respect to the optimizing resource utilization and runtime. The delineated approach is based on the assumption that optimizing the utility for scientific applications in the Cloud relies on the configuration of all three Cloud layers: the infrastructure layer (IaaS), the execution layer (PaaS), and the application layer (SaaS). Therefore, the configuration has to consider the allocation of computing

Table 3 Planned job: this table depicts the job descriptors, including the utility and the runtime, created during the planning phase

Job ID	# Input Files	DB Size	Nodes (Type)	Time	Est. Time	Est. Utility	Utility
1	**1000**	**500**	**8 (cluster)**	**7855**	**8030.75**	**0.6111**	**0.6246**
2	1000	500	7 (cluster)	-	9178	0.608	
3	1000	500	6 (cluster)	-	10707.67	0.6038	
9	1000	500	12 (cloud - 8)	-	10683.33	0.6052	
13	1000	500	8 (cloud - 8)	-	16025	0.5905	
16	1000	500	4 (cloud - 8)	-	32050	0.547	

The job descriptor chosen for execution is highlighted and includes the runtime of the job inserted into the knowledge base after the job execution.

resources, the provisioning of the programming environment needed, and the configuration of the application itself.

To describe the configuration parameters on all three layers, a generic model representing the Cloud stack via descriptors has been defined. Therefore, declarative descriptors for the allocatable computing and storage resources (IaaS), the utilized programming and execution environments (PaaS), and the applications themselves (SaaS), have been defined.

On top of these descriptors, an adaptive framework, capable of optimizing the utility has been designed. The design of the adaptive framework has been done on the basis of well-established concepts from the domain of autonomic computing, especially the MAPE-K loop (monitor, analyzer, planner, knowledge), and a utility function. Thus, the adaptive framework manages the configuration of resources at all three layers (resource, environment, application) by utilizing the defined descriptors and generic implementations of a monitor, an analyzer, a planner, an executor, and a knowledge base component. Firstly, the monitor is utilized for retrieving information about the utilized resources, which includes the amount of allocatable computing resources, the configuration of the execution system, and the application. Secondly, the analyzer, following a component-based architecture, enables the chained execution of resource-specific analyzer components, and evaluates the possible configurations at all layers. Thirdly, the planner ranks the possible configurations on the basis of a utility function and an application specific performance model and choses the best configuration for execution. Finally, the executor component configures the application, the environment, as well as the computing resources and executes the application according to the chosen configuration.

Additionally, the adaptive framework has been evaluated within a case study on the basis of a MapReduce application from the domain of molecular systems biology. A prototype implementation of all application-specific components has been provided and described. Sample parameters at all layers, including the configuration of parameter studies and scaling on different computing resources, have been chosen to evaluate the design. Additionally, an application-specific performance model has been implemented and is needed for the utility calculation process.

Finally, the adaptive job configuration process within the prototype framework has been explained on the basis of a sample job. The utility calculation, the performance estimation as well as the ranking of different job descriptions within the adaptive framework have been described in detail.

The developed Cloud stack configuration model enables the definition of scenario- and layer-specific parameters in a generic and flexible way for a not restricted set of applications, programming environments, and resources (compute, storage, network). The adaptive framework provides a modular reference implementation for adaptively optimizing the utility with respect to differing objectives (resource utilization, runtime). Nevertheless, this approach necessitates the specification of scenario-specific parameters and functions enabling the measurement of the utility for submitted jobs. This raises the need for future work towards automating and simplification of domain-specific definitions of parameters and functions.

Future work will include the evaluation of the framework with additional applications, execution environments, and resource types regarding scale, heterogeneity, and energy consumption which results in new research objectives. For instance, the area of green Cloud computing [29], emerging from green-IT [30], tackles the impact of the wide spread utilization of Clouds on the energy consumption regarding network and computing resources. Therefore, the optimization of energy consumption will be a promising future direction. Moreover, self-managing in Clouds has to consider the optimization of Quality of Service criteria relating to trust, security and privacy [2] which are increasingly important aspects, especially in Cloud computing.

Competing interests
The author declares that he has no competing interests.

Acknowledgements
I thank Siegfried Benkner, University of Vienna, who provided contributions in the conception of the work and in the acquisition of data. Additionally, I thank Norbert Brändle, AIT for proofreading.

Received: 1 December 2013 Accepted: 14 April 2014
Published: 12 May 2014

References

1. Buyya R, Broberg J, Goscinski AM (2011) Cloud computing principles and paradigms. Wiley Publishing, New York
2. The Future Of Cloud Computing, Opportunities for European Cloud Computing Beyond (2010). http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf.
3. Waddington S, Zhang J, Knight G, Jensen J, Downing R, Ketley C (2013) Cloud repositories for research data–addressing the needs of researchers. J Cloud Comput: Adv Syst Appl 2(13):1
4. Petcu D, Martino B, Venticinque S, Rak M, Máhr T, Lopez G, Brito F, Cossu R, Stopar M, Šperka S, Stankovski V (2013) Experiences in building a mosaic of clouds. J Cloud Comput: Adv Syst Appl 2(12):1-22
5. Kephart JO, Chess DM (2003) The vision of autonomic computing. Computer 36:41–50
6. Huebscher MC, McCann JA (2008) A survey of autonomic computing - degrees, models, and applications. ACM Comput Surv 40:7–1728
7. Rappa MA (2004) The utility business model and the future of computing services. IBM Syst J 43:32–42
8. Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51:107–113
9. Köhler M, Benkner S (2011) VCE - A versatile cloud environment for scientific applications. In: Galis A, Dillenseger B (eds) The Seventh International Conference on Autonomic and Autonomous Systems (ICAS 2011), 22-27 May 2011, IARIA, Venice/Mestre, Italy, pp 81–87
10. Hey T, Tansley S, Tolle K (2009) The fourth paradigm: data-intensive scientific discovery. Microsoft Research, Redmond
11. Hummel J, Niemann M, Wienkoop S, Schulze W, Steinhauser D, Selbig J, Walther D, Weckwerth W (2007) Promex: a mass spectral reference database for proteins and protein phosphorylation sites. BMC Bioinformatics 8:216
12. Lee K, Paton NW, Sakellariou R, Fernandes AA (2011) Utility functions for adaptively executing concurrent workflows. Concurrency Comput: Pract Exp 23(6):646–666
13. Walsh WE, Tesauro G, Kephart JO, Das R (2004) Utility functions in autonomic systems. In: Werner B (ed) Proceedings of International Conference on Autonomic Computing. IEEE, New York, USA, pp 70–77
14. Song G, Li Y (2005) Utility-based resource allocation and scheduling in ofdm-based wireless broadband networks. Commun Mag IEEE 43(12):127–134
15. Maurer M, Brandic I, Sakellariou R (2013) Adaptive resource configuration for cloud infrastructure management. Future Generation Comput Syst 29(2):472–487. Special section: Recent advances in e-Science
16. Delimitrou C, Kozyrakis C (2013) Paragon: Qos-aware scheduling for heterogeneous datacenters. In: Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems, 16-20 March 2013. ACM, Houston, Texas, pp 77–88
17. Calyam P, Patali R, Berryman A, Lai AM, Ramnath R (2011) Utility-directed resource allocation in virtual desktop clouds. Comput Netw 55(18):4112–4130
18. Padala P, Shin KG, Zhu X, Uysal M, Wang Z, Singhal S, Merchant A, Salem K (2007) Adaptive control of virtualized resources in utility computing environments. ACM SIGOPS Oper Syst Rev 41:289–302
19. Yumerefendi A, Shivam P, Irwin D, Gunda P, Grit L, Demberel A, Chase J, Babu S (2007) Towards an autonomic computing testbed. In: Proceedings of the Second Workshop on Hot Topics in Autonomic Computing. ACM/IEEE, Jacksonville, FL, p 1
20. Kephart JO (2005) Research challenges of autonomic computing. In: Roman G-C (ed) ICSE '05: Proceedings of the 27th International Conference on Software Engineering, 15-21 May 2005. ACM, St. Louis, USA, pp 15–22
21. DRMAA - Distributed Resource Management Application API (2014). http://www.drmaa.org.
22. Köhler M, Kaniovskyi Y, Benkner S (2011) An adaptive framework for the execution of data-intensive MapReduce applications in the Cloud. In: Werner B (ed): The First International Workshop on Data Intensive Computing in the Clouds (DataCloud 2011). IEEE, Anchorage, Alaska, pp 1122–1131
23. Kernel Based Virtual Machine (2014). http://www.linux-kvm.org/page/Main_Page.
24. Principles of Economics/Utility (2014). http://en.wikibooks.org/wiki/Principles_of_Economics/Utility.
25. Bailey D, Snavely A (2005) Performance modeling: Understanding the past and predicting the future. In: Cunha J, Medeiros P (eds): Euro-Par 2005 Parallel Processing vol 3648, 1st edn. Springer, Berlin/Heidelberg, pp 620–620
26. Köhler M, Benkner S (2012) Design of an adaptive framework for utility-based optimization of scientific applications in the cloud. In: Bilof R (ed): The 2nd International Workshop on Intelligent Techniques and Architectures for Autonomic Clouds (ITAAC 2012), in Conjunction with The 5th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2012). IEEE/ACM, USA, pp 303–308
27. Köhler M (2012) A service-oriented framework for scientific cloud computing. PhD thesis, University of Vienna
28. Köhler M, Kaniovskyi Y, Benkner S, Egelhofer V, Weckwerth W (2011) A cloud framework for high troughput biological data processing. In: PoS (ed): International Symposium on Grids and Clouds, PoS(ISGC 2011 & OGF 31). Proceedings of Science, Taipei, Taiwan, p 69
29. Baliga J, Ayre RWA, Hinton K, Tucker RS (2011) Green cloud computing: balancing energy in processing, storage, and transport. Proc IEEE 99(1):149–167
30. Murugesan S (2008) Harnessing green it: principles and practices. IT Professional 10(1):24–33

doi:10.1186/2192-113X-3-4
Cite this article as: Koehler: **An adaptive framework for utility-based optimization of scientific applications in the cloud.** *Journal of Cloud Computing: Advances, Systems and Applications* 2014 **3**:4.