AMOS: Using the Cloud for On-Demand Execution of e-Science Applications

[1,2]Rudolf Strijkers, [1]Willem Toorop, [1]Alain van Hoof, [1]Paola Grosso, [1]Adam Belloum, [1]Dmitry Vasuining, [1]Cees de Laat, [1,2]Robert Meijer

[1]Universiteit van Amsterdam, Amsterdam, The Netherlands

[2]TNO, Groningen, The Netherlands

Abstract—**The amount of computing resources currently available on Clouds is large and easily available with pay per use cost model. E-Science applications that need to be executed on-demand can benefit from the Cloud, because no permanent computing resources to support peak demand has to be acquired. In this paper, we present AMOS, a system that automates creation and management of temporary Grids on a Cloud to execute (parts of) application workflows. We performed experiments with AMOS and a representative e-Science application on a research Grid and on the Amazon EC2 Cloud. The results show that AMOS is a viable approach to manage and execute e-Science applications in a flexible Grid environment and to explore novel mechanisms that allow optimal utilization of Cloud resources. Furthermore, we consider AMOS as a step towards an operating system for (virtual) infrastructures that enables Grid applications to control their computational resources at run-time.**

Keywords: Grid, Cloud, Workflows, Urgent Computing

I. Introduction

An increasing number of e-Science applications [1, 2] need on-demand access to a large amount of computational resources or need to tune their demands at runtime. Grid technology enables resource sharing amongst various organizations and the use of a large amount of shared computational resources. Grid resource management, however, is optimized towards fair usage. The drawback is that on-demand access to Grid resources is difficult to achieve in practice, even if multiple Grids are available.

E-Science applications can be composed of a workflow of smaller applications that solve parts of a larger problem set. The execution of the workflow becomes particularly important for applications in which execution time is bound to time constraints. For example, in cardiovascular research, wave propagation models play an important role [3]. The development of these models towards patient-specific simulations involves many model parameters based on data measured in-vivo and subject to uncertainties. One potential application of the wave propagation model is to help doctors to locate the position of the hemodialysis fistula for patients with renal failure. Because the time needed to complete wave propagation simulations vary with the state of the patient, the required computing resources to vary from one run to another. In practice, medical doctors might need the results within hours for urgent cases.

Multiple simulations can be executed concurrently to shorten the total execution time. This would, however, require a large amount of computational resources at once. Although Grid technology enables access to computing and storage resources, it does not guarantee that the computing resources will be available within a given time. In practice, the resource broker can delay a job with a large claim on the Grid resources so long that it leads to starvation. So, even though a Grid might have enough resources to execute a number of concurrent simulations, users may be better off submitting many small jobs rather than one large job.

To support on-demand execution of e-Science applications we propose AMOS, a system that extends a workflow management system and besides Grids, utilizes the Amazon EC2 Cloud [4] (from now on referred to as Amazon Cloud) to offload computation. Rather than extending Grid technologies to support prioritization or to support Clouds, AMOS adds an additional abstraction layer, and manages (virtual) infrastructure on behalf of the application. For example, AMOS can instantiate a transient Grid for each application or add and remove Virtual Machines (VM) to a transient Grid when applications demand more resources at run-time.

Our research shows (Section III and Section IV) that when applications require more resources at run-time Clouds are more flexible than Grids. While one could write an application that requests additional Grid resources by submitting jobs to the resource broker, the time at which these resources become available remains uncertain. By using Clouds, AMOS can instantiate additional VMs to meet new application demands. Additional jobs can either be submitted to the existing cluster, in which it will be scheduled to execute immediately, or create a new broker with an empty queue to manage the additional VMs.

Before we describe in detail our implementation (Section III) and the results of our work (Section IV), we provide a short overview of limitations of Grids, which we address by utilizing Clouds and how our solution leads to the AMOS architecture (Section II). There are some open issues with our implementation and we plan to address them in our future work (Section V and Section VII). But, there is certainly interest and need in the e-Science community for this type of solution as is shown by the amount of existing work in this area (Section VI).

II. Design Considerations and Architecture

Grid computing enables large-scale resource sharing to support e-Science applications with large computational demands [5]. Most of the effort in Grid research has been focused on the federation aspects, i.e. how to efficiently and securely share resources of different owners. In the Grid, users are part of a Virtual Organization (VO), which is associated with policies regarding resource access and usage. With the correct credentials, users can submit jobs to a resource broker. The resource broker queues the job until it can schedule it on the resources under its control.

Resource management in Grids is optimized to ensure fair usage. Resource brokers use queues to schedule jobs on the computational resources. The consequence is that on-demand access to computing resources cannot be supported without prioritization [6]. Although technically feasible, prioritization presents administrative (political) problems, which prevent it from being implemented and supported by Grids.

Users might be motivated to run applications on multiple Grids to shorten execution time. First, it may take longer to execute a large job on one Grid than executing a number of smaller jobs over multiple Grids. Large jobs can be delayed indefinitely, because a scheduler might need to give exclusive access to the Grid resources. Second, the application might need more resources than a single Grid supports. One problem with this approach is that the user has to explicitly deal with issues regarding distributing the application over multiple Grids, such as transferring data sets, submitting jobs to different resource brokers and merging results. If applications are composed of multiple processing steps, this process becomes even more cumbersome. Another problem is that even with profiling the Grid performance, users have no control when the jobs are scheduled for execution.

One solution to run applications over multiple Grids is to automate their execution and management using a meta-scheduler, i.e. a scheduler that monitors multiple Grids and manages job submission and execution. Our system, WS-VLAM [7] combines the ability to use multiple Grids with a flexible high-level rapid prototyping environment to compose workflows of applications that involve many processing steps. For example, WS-VLAM automates setting application parameters and job submission, e.g. as part of a parameter sweep study. When available resources on one Grid are occupied or the queuing time is too long, WS-VLAM can switch to another Grid to achieve shorter execution time.

An elegant solution that provides scientists who need computing resources in a burst manner is to create and destroy a Grid only when needed, i.e. a transient Grid. A newly instantiated transient Grid is like any other Grid, but it starts with an empty queue, thus executing the first job immediately. Because (IaaS) Clouds support on-demand access to VMs on a pay-per-use basis, we can realize transient Grids by automatically installing and configuring Grid middleware on the purchased resources. We implemented in AMOS the necessary components to create and manage transient Grids on the Amazon Cloud, and to schedule and execute application workflows.

Figure 1 shows the architecture of AMOS. Applications are composed in the workflow editor of WS-VLAM. The WS-VLAM workflow engine takes the workflow of applications (1) as input and creates a schedule to execute the workflow with the available resources. Resources are registered in the Resource Manager, which is extended to contain references to transient Grids. A profiler maintains a history of performance statistics of each Grid, based on execution information of scheduled jobs. The scheduler uses profiler statistics to determine which Grid to use for which job, e.g. Grid A might have a better history with accepting large jobs than Grid B. We added a Transient Grid Manager (TGM) to WS-VLAM, which takes care of creation, adaptation and removal of transient Grids (3) and which registers transient Grids with the Resource Manager. The actual creation, adaptation and removal of a transient Grid is done by the Elastic Site Manager (ESM), which runs on the first instance of the transient Grid in the Amazon Cloud. Although transient Grids can be created with any kind of Grid middleware, we implemented automated installation and configuration of the Globus Toolkit (GT) [8].

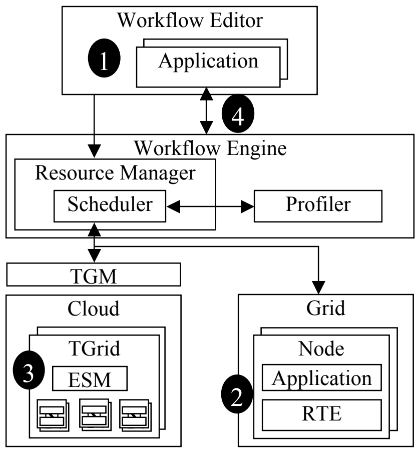

Figure 1. AMOS architecture.

Grid applications are encapsulated by the WS-VLAM Run-Time Environment (RTE) (2), which connects to the Resource Manager at job execution. This allows the Resource Manager to transfer or load input data, configure application parameters, setup communication networks, and to capture output data before it executes the application. Using the WS-VLAM API (4), applications can adjust their resource requirements at run-time. In previous work, for example, we showed how to control Grid networks from Grid applications in this manner [9].

III. Implementation

We discuss our implementation by describing the various stages of creating, administering and using a transient Grid in AMOS. AMOS utilizes Amazon Machine Images (AMI) to create preconfigured systems ready for immediate use in the Amazon Cloud. We created an AMI that contains all the necessary software to instantiate a transient Grid. Unlike similar approaches [10, 11], our implementation runs off-the-shelf and does not need external support services or other environmental parameters. Moreover, as of July 2010, we made the AMI, including the code, publicly available on Amazon's community AMI web site for 32-bit, 64-bit and Cluster instance types[1].

A. Creating a Transient Grid

Our AMI is based on Ubuntu Linux and is configured with an installation of the Torque Resource Manager [12]. The first instance of the AMI operates as the transient Grid's head node. This instance is also configured as a compute node. At instantiation, a new Certificate Authority (CA) and initial host certificate, which are needed by GT is created. The CA then generates an End Entity Certificate (EEC) for a preconfigured user account on the instance. Because the AMI is already setup with Grid software and a user account for Grid usage, the launched instance is ready to be included as a Grid resource in AMOS. However, further configuration of the instance may be performed to authorize other Grid EECs to use the resource and to enable and set the parameters for the elastic sizing of the transient Grid.

The transient Grid provides a web interface to configure and monitor its resources at run-time. For basic security, the web interface only listens on a local port. The local port is then forwarded to the local http port of the instance with Secure Shell (SSH), because SSH is Amazon's default method to connect securely to Linux instances. Because Amazon instances use public key authentication, the advantage of this approach is that the web interface is only accessible to the user that created the instance. The web-interface can also be used to authorize Grid users with a Grid EEC signed by a real Grid Certificate Authority (CA). All systems that want to use the transient Grid, however, need to acknowledge its CA by downloading and adding the certificate to their list of certified CAs.

B. Transient Grid Elasticity

Figure 2 shows the components and interactions in the transient Grid. The ESM runs on the initial instance (head node) and parameters, such as minimum and maximum size of the transient Grid can be set using the web interface. Once ESM is provided with Amazon Access Credentials (AAC), it is able to launch new compute instances, which in our case use the same AMI. Dynamic scaling of the Grid is implemented by monitoring Torque's Job-queue. When the queue reaches a threshold ESM launches new compute instances or kills instances that are no longer in use.

The ESM uses the following retention formula (1) to determine the number of nodes (L_t) to launch at time t to prevent over-provisioning of compute instances. The ESM maintains a history of the number of pending jobs (W_t) over the last l seconds. The startup window length l can be configured in the web interface.

- C_t, number of cores;

[1] The work, documentation, and instructions to use the public AMI can be downloaded from https://www.os3.nl/gridondemand/start.

- C_t^p, number of cores of compute nodes that have already been started but have not joined the cluster yet (pending cores);
- R_t, number of running nodes;
- $S_{\min}$ and $S_{\max}$, minimum and maximum cluster size that can be configured with the web interface.

$$L_t = \min\left\{\max\left\{\max\left\{\frac{\min\left\{W_t\right\}-C_t^p}{C_t},0\right\}+R_t,S_{\min}\right\},S_{\max}\right\}-R_t \quad (1)$$

Amazon's payment model for VMs is per hour. Therefore, the ESM automatically terminates any node that is idle for almost an hour to save costs.

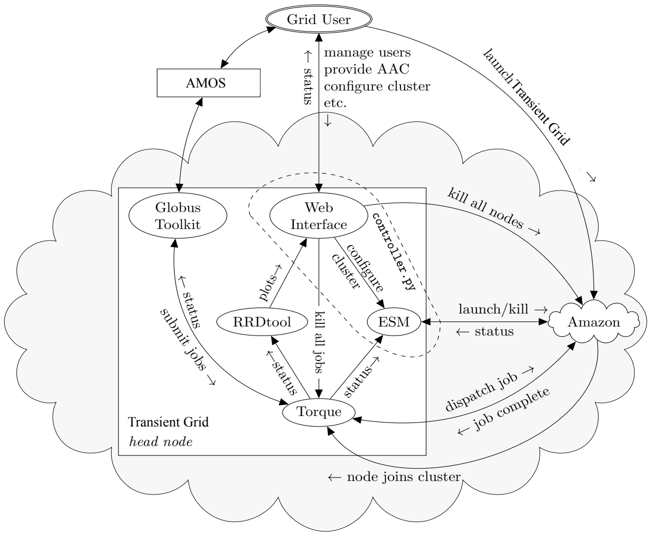

Figure 2. Components and interactions in the transient Grid.

C. Administering a Transient Grid

Once a transient Grid is setup, it will contain a number of services and certificates, such as a CA, AAC, which are necessary to operate and interact with the Grid. To ease the administration process and to accommodate reuse, we use Amazon's Elastic Block Storage (EBS) to store the transient Grid settings. With EBS, persistent storage images (called *volumes*) may be attached to running instances. Volumes will not be destroyed on termination of the instance and they can be copied to snapshots, which in turn can be copied to new volumes for new instances. Our implementation incorporates this feature to administrate the transient Grid in different stages of configuration. Here, the most important properties related to starting and stopping the transient Grid is explained. For a more in-depth description of the process, we refer to [13].

The head node can be shutdown for reuse or for sharing, which can be controlled from the web interface. When the instance is shutdown for reuse, all configuration parameters will be preserved for the next instantiation, e.g. the CA, AAC, and EEC of the local and externally signed Grid users. The advantage is that no new CA or EEC of a local Grid user needs to be created on instantiation anymore. Only a new host EEC will be generated to match the new hostname the instance will get. Another advantage is that applications, which are installed on the image, will also be saved. This allows a database of transient Grids with specific applications/configurations to be created.

When the instance is shutdown for sharing, all user-related information, such the CA, EEC and AAC, will be removed before halting the instance. The resulting AMI may then be shared with the public again. This might be useful to branch versions of a transient Grid with specific Grid services to accommodate Virtual Organizations or to save transient Grids with specific applications preconfigured.

D. *Grid Participation*

A newly created transient Grid has a dynamically assigned IP-address and a DNS name. The process of creating an EEC for the transient Grid requires actions from a user. For DutchGrid [14], for example, the user has to apply for a host certificate in person. This process is unpractical for transient Grids, in terms of cost (the transient Grid needs to exist while the application is in progress) and because it requires user intervention. To avoid waiting for an ECC, we store the private key of the CA in the persistent storage accessible by the instance. This allows the instance to sign its own EEC request. A drawback is that a self-signed EEC will not be able to collaborate in all the possible ways as a static resource with an EEC signed by a widely accepted CA would.

E. *Using a Transient Grid*

Once the Grid is instantiated, it is added as a Grid resource to the resource manager. Because the instantiated Grid runs Globus, which is supported by AMOS, there is no difference with respect to managing jobs and monitoring performance over other Globus-based Grids.

IV. Experiments and Results

An ongoing experiments at the TU Eindhoven and TU Leiden aim at improving the estimate of arterial mechanical properties, a reverse method combining experiments with a wave propagation model (WAVE). To validate the proposed method, a study including 6 volunteers, a total number of 60000 simulations of approximately 30 minutes are required. During the validation phase researchers are willing to wait a couple of days for results of the simulations. In practice, however, medical doctors might need the results within hours if a case is urgent. Because the WAVE application illustrates a practical scenario for on-demand computing, we use this application in our experiments.

Another reason to use the WAVE application is that the data set is divided over the nodes and processed independently. This has two advantages. One, by using a uniform data set we can gain insight in potential differences in execution time on physical resources or on the Amazon Cloud. A large variance in the execution time the nodes will indicate that computational resources are shared. In practice, however, differences in the dataset lead to different execution times of the nodes. Two, because the data chunks are processed independently, network performance will not impact execution time. From our own experience [15] and recent results [16] of measuring network performance in the Amazon Cloud, network communication might lead to an unfair advantage of Grid, which typically utilize dedicated high-performance networks.

For our experiments we had access to the Distributed ASCI Supercomputer 3 (DAS3) [17], a five-cluster wide-area distributed system designed by Advanced School for Computing and Imaging (ASCI). We used the 32-node DAS3 cluster of the University of Amsterdam (UvA). The transient Grid in the Amazon Cloud was configured similarly, using *m1.large* and *c1.xlarge* instances for the head node and compute nodes respectively. TABLE I summarizes the properties of the machines and instances used.

TABLE I. Configuration of the Grids on DAS-3 and Amazon EC2

	Head Node		**Compute Nodes**	
	Amazon	***DAS3***	***Amazon***	***DAS3***
Cores	2	2	8	2
CPU		2 (2.2 GHz AMD Opteron DP280)		2[b] (2.2 GHz AMD Opteron DP275)
CU[a]	4		20	
Mem. (GB)	7.5	8	7	4

a. One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.

b. Only one CPU was used in the experiments.

As opposed to transient Grids, physical Grids have fixed amounts of resources. If the resource requirements of the WAVE application exceed the available resources on the Grid, the jobs will need to time-share resources (2).

$$W_t = J_t \times \left(W_j \bmod C_a \right) \tag{2}$$

In (2), W_t is the execution time of the workload, W_j is the number of jobs in the workload, J_t is the total execution time of the job and C_a the available cores. Because the ESM dynamically scales resources, the number of available cores in a transient Grid is always equal or more than the number of jobs. Therefore the total execution time of the workload is expected to equal the total execution (running + pending) time of the slowest job.

We created two workloads, one smaller (30 jobs) and one larger (100 jobs) than the amount of nodes on the UvA cluster, to test the resource elasticity of the transient Grid as opposed to the static resources on the DAS3 UvA cluster. During the experiments three quantities were monitored in AMOS:

- *Total execution time,* the time it takes for the complete workload to finish,
- *Job pending time*, the time a job is pending,
- *Job running time*, the time a job is running until it finishes.

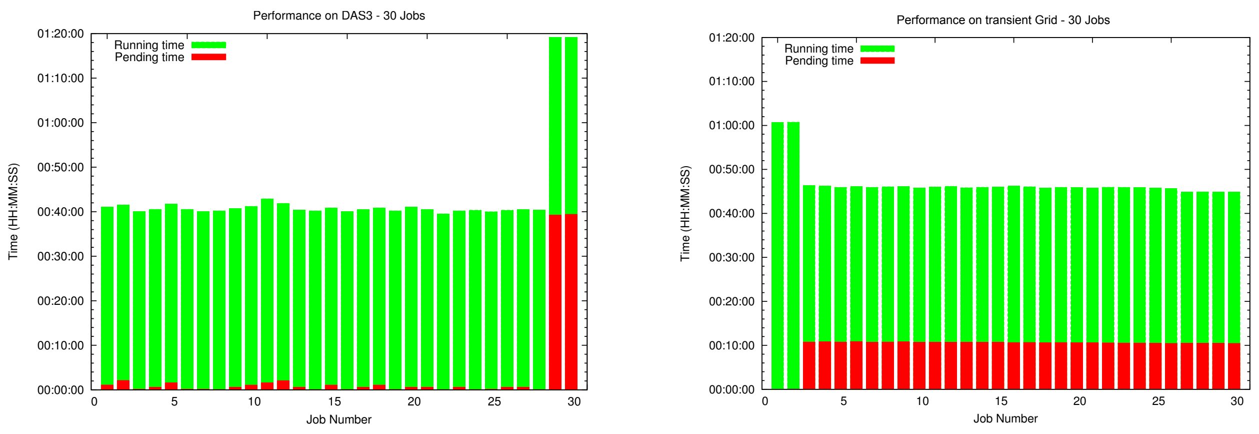

Figure 3. 30-job workload: pending and running time on DAS3 and on transient Grid.

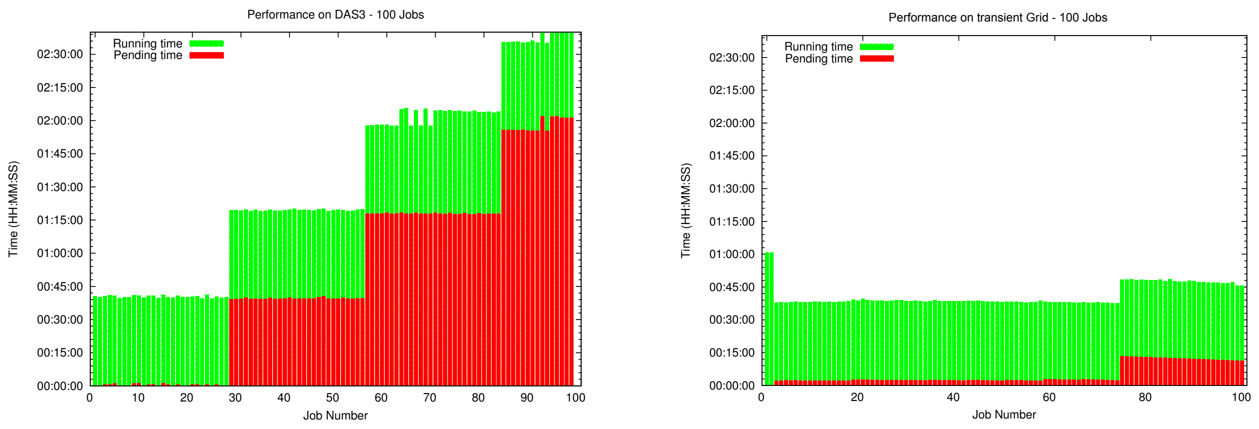

Figure 4. 100-job workload: pending and running time on DAS3 and on transient Grid.

In the transient Grid, the first two jobs in both the 30 and the 100-job workload runs (Figure 3 and Figure 4), have a different running time compared to the other jobs. The head node participates in the transient Grid as compute node, but runs on a difference instance type. Because the instance type has different performance characteristics than the other compute node instances, we removed the head node measurements from our results.

We also observe that in the 30-job test on DAS3, the last two jobs show a different job pending time compared to the other jobs. This is due to the (unexpected) availability of only 28 nodes in the cluster. The two jobs have to wait for other two jobs to finish before they can run. We also removed these results from our results.

For the 30-job workload, the execution time on the DAS3 cluster almost equals the job execution time on the transient Grid (TABLE II). Because the workload has to be time-shared in DAS-3 for the 100-job workload, execution time in DAS3 becomes four times larger than the workload execution time in the transient Grid. The transient Grid scales with the number of jobs, which keeps the total workload execution time of the 100-job workload close to the 30-job execution time (Figure 3 and Figure 4). In the transient Grid the job pending time suddenly increases around job 70. Why this is, needs further investigation in more elaborate scaling experiments on the Amazon Cloud. We were unable to obtain more resources from Amazon during our research, however (Section V).

TABLE II. TOTAL EXECUTION TIME IN THE 30-JOB AND 100-JOB WORKLOAD TESTS

Cluster	Number of nodes	Total execution time
DAS3	28	00:42:58
transient Grid	28	00:46:22
DAS3	98	02:46:36
transient Grid	98	00:52:26

Compared to the 30-job workload in the transient Grid, the average job execution time increases for the 100-job workload with more than 15 per cent (TABLE III and Figure 4). The increase in execution time can also be observed in the minimum and maximum job execution time. Again, more scaling experiments will be needed to determine the cause of the performance decrease.

TABLE III. JOB RUNNING TIME IN THE 30-JOB AND 100-JOB WORKLOAD TESTS

Cluster	Nr. of nodes	Average	Min	Max	σ
DAS3	28	00:40:01	00:39:26	00:41:19	00:00:21
transient Grid	28	00:34:21	00:35:34	00:35:34	00:00:20
DAS3	98	00:41:09	00:39:23	00:47:47	00:02:34
transient Grid	98	00:40:03	00:39:27	00:41:19	00:00:21

Because physical resources on Clouds may be shared amongst many users, we expected a larger standard deviation in the execution time than on a physical Grid. However, standard deviation of the 100-job workload on DAS3 increased significantly, while the standard deviation on the transient Grid remained the same. A possible explanation is that in DAS3 the jobs need to share the physical resources, while in the transient Grid the VMs are isolated by the virtualization layer and scaled as necessary. Because then only one job is running per node (no processes compete for CPU time), the transient Grid performs more consistently.

V. Discussion

Although our approach shows that it is viable to use transient Grids for on-demand execution of Grid applications, we have not yet fully automated the transient Grid manager. At this moment, we manually register and remove transient Grids in AMOS. This is just an implementation issue and we expect to automate this in our next software revision.

During our research the default limitation of twenty running instances per region was extended to seventy running instances per region. We planned scaling experiments to thousand instances, which would be a realistic number for concurrent execution of WAVE simulations. When launching a thousand *c1.xlarge* instances, the theoretical capacity of a cluster would be 35.2 TFLOPS at $760.00 per hour (with data from [18]). In comparison, DAS3 has an aggregate peak performance of 3.8 TFLOPS.

During our research we were unable to negotiate launching of a thousand instances in the European region. Clearly, Clouds have resource limits. Most likely, the instantly available resources are dimensioned to support small- to medium-sized e-Science applications. We wonder if supporting on-demand execution of large-scale e-Science applications is commercially viable or if having so many resources instantly available is just a matter of time.

Studies have been done to quantify the cost aspect of doing experiments in Clouds [19]. Unfortunately, we could not estimate the cost of running our application on DAS-3, because there are no cost estimates available for CPU time. This might be a common problem, as the price tag of computation on already deployed and purchased clusters is usually not provided to end users of a Grid. By using AMOS, scientists have on-demand access to Grid resources while keeping a clear overview of the costs of their experiments.

VI. Related Work

In the past years, Cloud computing has become a commercial reality and its application to scientific computing has become an area of interest [20]. Here, we review the literature related to various aspects of our work.

A. Grid on Clouds

Nimbus [10, 21] is an open source toolkit developed by the Globus Alliance, which enables a users to turn their cluster into an Infrastructure-as-a-Service (IaaS) Cloud. The Nimbus Cloud client software contains components to instantiate virtual clusters on Nimbus Clouds or Amazon Clouds using a broker service located in Chicago. Their solution allows on-demand instantiation of preconfigured infrastructures and software installations and *contextualization* to the user's deployment environment. Moreover, it offers a structured approach to maintain a database of transient Grids with specific applications/configurations.

The Elastic Site Manager, a service that monitors the job queues and launches or kills VMs depending on the workload is introduced in [11] and we included this component in our architecture. The authors evaluate different scheduling algorithms, and their effectiveness with different jobs submission patterns typical for specific applications. We believe that AMOS would be better at scheduling, because it can optimize the scheduling of a complete workflow and take into account specific requirements (or preferences). Because AMOS is not limited to transient Grids, it can submit jobs to Grids with specific hardware, such as GRAPE [22], or save costs and submit jobs to regular Grids.

B. Workflows

The authors of [23] aimed to quantify the performance differences of running workflows on various physical and virtual platforms. They focused on a specific workflow to compare the executing time and overhead associated with running the workflow on a specific platform. Their main conclusions are that with short job runtimes, the Amazon Cloud provides good performance. However, in their experiments with the Amazon Cloud the workflow suffered from resource scheduling delays and wide-area communications problems. To execute their experiments, they had to setup a suitable execution environment on the Cloud resources for their workflow management system. We provide automated management and execution of workflows including instantiation and management of transient Grids. Compared to this work, however, it would be useful to measure overhead of our software. This is a topic for future experiments.

In [24] the pros and cons of executing workflows in the Clouds are reviewed. One of the issues raised in this paper, which we did not analyze in our case, is the cost of transferring and storing large amounts of data. More importantly, the paper presents the issues, possible applications, and related work of virtualization in scientific applications and on-demand computing. The view that we present is that workflow management systems provide enough structure to include Clouds for on-demand access to Grid computing. Therefore, the AMOS architecture is largely based on WS-VLAM.

C. Applications

A number of scientific applications have been ported to the Amazon Cloud. In the Gaia project of the European Space Agency, for example, the in-house developed distributed computing software was configured on Amazon's AMIs [25]. In their case, using the Cloud is a cost effective approach, because the computations only need to run two weeks of every six months. They estimated that by using the Amazon Cloud, rather than the in-house solution, they could cut costs at least by half. They also found that Clouds would be useful in finding scalability issues. In our case, however, we were unable to get enough resources from Amazon to perform scalability tests.

The high-energy physics community also shows interest in Clouds [26]. Researchers at KEK experimented with executing the simulated data analysis software of the Belle experiment on the Amazon Cloud. Despite having been able to accomplish their goal with a limited data set, the authors doubt that Clouds could really support the large-scale computations they need. Drawbacks in their case are the costs of executing such large-scale experiments on the Cloud and the possible performance bottlenecks in transferring the data back and forth conventional Grids. They also mention, however, that a possible solution would be a hybrid system in which Grids supply base-load resources and in which Cloud provide resources at peak or urgent demands.

CERNVM [27] highlights another advantage of transient Grids. They provide a software appliance with all the necessary software preconfigured, which allows scientists to do LHC experiments without spending time on complex software configurations. CERNVM is a practical example of an appliance that helps creating transient Grids for specific target applications.

D. Performance

Performance aspects of the Amazon Cloud are well discussed in literature [15, 17, 27, 28]. Clearly, there is a performance penalty when using VMs for high performance computing. Because the benchmarks indicate that network and I/O performance is most affected in Clouds, one has to consider if the type of application will run with acceptable performance, e.g. lattice calculations with many synchronization steps between nodes as opposed to an embarrassingly parallel application such as WAVE. All authors agree that within certain bounds, Clouds are well suited for on-demand scientific computing problems. Based on an application description, AMOS could make such platform choices automatically.

A point of consideration has to be made; although it is argued that transferring data between Clouds and Grids is an issue, transferring data between multiple Grids is not always an easy undertaking either. In practice, administrative or configuration issues complicate using multiple Grids for a single application or for high-performance data transfer between applications [28]. We believe that network performance and interconnectivity issues for Grids and Clouds can only be addressed by implementing more flexible protocols and switching hardware, such as OpenFlow [29], into the network fabric. AMOS anticipates on more flexibility in future networks by supporting workflows and applications to configure their network service.

VII. Future Work

AMOS can be improved and extended in a number of ways. One of the issues, for example, is how to utilize transient Grids in scheduling applications. Obviously, on-demand execution is preferred over delayed execution in a Grid. However, this disregards the cost aspects of acquiring on-demand resources. Therefore, the scheduler needs to optimize the tradeoff between on-demand computing, cost and what a user is prepared to spend.

Approaches, including ours, to dynamically scale a transient Grid are based on monitoring job queues. Why not create a transient Grid for each application? The choice is a tradeoff between cost and meeting a specific deadline. Creating one transient Grid for two on-demand jobs that run 10 minutes each might be better than creating a transient Grid for each job. If the pricing model is per hour, a lot of computing resources (or money) might be wasted in the latter case.

In our current model, workflows are static. After a workflow is created, the scheduler will take the workflow and find the best possible way to execute its components. AMOS allows Grid applications to make run-time resource requests, however. For example, an application can request new network paths or submit new jobs. Essentially, applications gain programmatic control over the resources on which they are executing. We have not yet solved how applications would utilize dynamic resource adaptation with transient Grids, nor have we determined a minimal API for such interactions. Therefore, to create applications that can dynamically adapt their resource demands and interact with the (virtual) infrastructure is a topic for future research.

VIII. Conclusion

To utilize the benefits of Clouds, researchers need to have knowledge about Cloud and Grid technology to setup a basic execution environment for their e-Science application. Moreover, after the environment is setup, application parameters need to be set and input data transferred. By automating this complex process with AMOS, we allow researches to gain easy access to elastic and on-demand computation using current Grid technologies. During our experiments, however, we ran into scaling limitations of the Amazon Cloud. We found that the Amazon Cloud is better suited for small to medium-size e-Science applications than for large-scale e-Science applications.

Both e-Science applications and their execution environments, require complex configurations of software and infrastructure resources. Clouds also enable dynamic configuration of infrastructure resources. We find that e-Science applications can only fully utilize the benefits of Clouds, Grids and specialized computing infrastructures, if an operating system manages the many resources available today. In AMOS, we have implemented the basic patterns to do more research in the development of such an operating system. Although many issues still remain to be solved, AMOS can be considered as a step toward an operating system for (virtual) infrastructure that enables Grid applications to control their computational resources at run-time.

References

[1] N. Kruithof and D. Marchal, "Real-time Software Correlation," in *INGRID Workshop*, 2008.

[2] R. J. Meijer and A. R. Koelewijn, "The Development of an Early Warning System for Dike Failures," in *1st International Conference*

and Exhibition on WATERSIDE SECURITY Copenhagen, Denmark, 2008.

[3] C. A. D. Leguy, E. M. H. Bosboom, H. Gelderblom, A. P. G. Hoeks, and F. N. v. d. Vosse, "Estimation of distributed arterial mechanical properties using a wave propagation model in a reverse way," *to be published in Medical Engineering & Physics,* 2010.

[4] Amazon Inc., "Amazon Elastic Compute Cloud (Amazon EC2)". [Online]. Available: http://aws.amazon.com/ec2/ [Accessed: 26 July, 2010].

[5] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of High Performance Computing Applications,* vol. 15, pp. 200-222, Fall 2001 2001.

[6] P. Beckman, S. Nadella, N. Trebon, and I. Beschastnikh, "SPRUCE: A System for Supporting Urgent High-Performance Computing," in *Grid-Based Problem Solving Environments*, 2007, pp. 295-311.

[7] V. Korkhov, D. Vasyunin, A. Wibisono, V. Guevara-Masis, and A. Belloum, "WS-VLAM: Towards a Scalable Workflow System on the Grid " in *Workshop on workflows in Support of Large-Scale Science (WORKS 07) In conjunction with HPDC 2007* Monterey Bay, California, 2007.

[8] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," in *IFIP International Conference on Network and Parallel Computing*: Springer-Verlag LNCS 3779, 2005.

[9] R. Strijkers, et al., "Network Resource Control for Grid Workflow Management Systems," in *IEEE International Workshop on Scientific Workflows (SWF2010)* Miami, USA, 2010.

[10] K. Keahey and T. Freeman, "Contextualization: Providing One-Click Virtual Clusters," in *Proceedings of the 2008 Fourth IEEE International Conference on eScience*: IEEE Computer Society, 2008.

[11] P. Marshall, K. Keahey, and T. Freeman, "Elastic Site: Using Clouds to Elastically Extend Site Resources," in *IEEE International Symposium on Cluster Computing and the Grid*. vol. 0 Melbourne, Australia, 2010, pp. 43-52.

[12] Cluster Resources, "Torque Resource Manager". [Online]. Available: http://www.clusterresources.com/products/torque-resource-manager.php [Accessed: 26 July, 2010].

[13] A. v. Hoof and W. Toorop, "Grid on Demand," in *System and Network Engineering*. vol. MSc Amsterdam: University of Amsterdam, 2010.

[14] NIKHEF, "DutchGrid, Large-scale Distributed Computing in the Netherland". [Online]. Available: http://www.dutchgrid.nl/ [Accessed: 26 July 2010].

[15] A. Giurgiu, "Network Performance in Virtual Machine Infrastructures," in *System and Network Engineering*. vol. MSc Amsterdam: University of Amsterdam, 2010.

[16] G. Wang and T. S. Eugene Ng, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," in *IEEE INFOCOM 2010* San Diego, CA, 2010.

[17] The Distributed ASCI Supercomputer 3, 13 August 2008. [Online]. Available: http://www.cs.vu.nl/das3 [Accessed: 26 July, 2010].

[18] S. Ostermann, et al., "A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing," in *Cloudcomp 2009* Munich, Germany, 2009.

[19] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the Montage example," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing* Austin, Texas: IEEE Press, 2008.

[20] M. Armbrust, et al., "Above the Clouds: A Berkeley View of Cloud Computing," EECS Department, University of California, Berkeley UCB/EECS-2009-28, 2009.

[21] K. Keahey and T. Freeman, "Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. Cloud Computing and Applications," in *Cloud Computing and Its Applications (CCA-08)* Chicago, 2008.

[22] GRAvity PipE (GRAPE). [Online]. Available: http://www.astrogrape.org/ [Accessed: 26 July, 2010].

[23] C. Hoffa, et al., "On the Use of Cloud Computing for Scientific Workflows," in *Third International Workshop on Scientific Workflows and Business Workflow Standards in e-Science (SWBES) in conjunction with IEEE International Conference on e-Science 2008* Indianapolis, USA, 2008.

[24] E. Deelman, "Grids and Clouds: Making Workflow Applications Work in Heterogeneous Distributed Environments," *International Journal of High Performance Computing Applications,* 2009.

[25] A. Olias, "Feature - Grid in a cloud: Processing the astronomically large". [Online]. Available: http://www.isgtw.org/?pid=1001994 [Accessed: 26 July, 2010].

[26] A. Williamson, "Feature - A side of cloud with your grid, ma'am?". [Online]. Available: http://www.isgtw.org/?pid=1001800 [Accessed: 26 July, 2010].

[27] CERN, "CERNVM Software Appliance ". [Online]. Available: http://cernvm.cern.ch/ [Accessed: 26 July, 2010].

[28] S. Portegies Zwart, et al., "Simulating the universe on an intercontinental grid of supercomputers," *Submitted to IEEE Computer,* 2009.

[29] N. McKeown, et al., "OpenFlow: enabling innovation in campus networks," SIGCOMM Comput. Commun. Rev., vol. 38, pp. 69-74, 2008.