2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

Controlling the deployment of virtual machines on clusters and clouds for scientific computing in CBRAIN

Tristan Glatard*†, Marc-Etienne Rousseau*, Pierre Rioux*, Reza Adalat*, Alan C. Evans*
*McConnell Brain Imaging Centre, Montreal Neurological Institute, McGill University, Canada.
†University of Lyon, CNRS, INSERM, CREATIS, Villeurbanne, France.
Email: {first.last}@mcgill.ca

***Abstract*—The emergence of hardware virtualization, notably exploited by cloud infrastructures, led to a paradigm shift in distributed computing by enabling complete software customization and elastic scaling of resources. However, new software architectures and deployment algorithms are still required to fully exploit virtualization in web platforms used for scientific computing, commonly called science gateways. We propose a software architecture and an algorithm to enable and optimize the deployment of virtual machines on clusters and clouds in science gateways. Our architecture is based on 3 design principles: (i) separation between resource provisioning and task scheduling (ii) encapsulation of VMs in regular computing tasks (iii) association of a virtual computing site to each disk image. Our algorithm submits and removes VMs on clusters and clouds based on the current system workload, the number of available job slots in active VMs, the cost and current performance of clouds clusters, and a parameter quantifying the performance-cost trade-off. To cope with variable queuing and booting times, it replicates VMs on independent computing sites selected from a minimization of a makespan-cost linear combination in the Pareto set of non-dominated solutions. Makespan and cost are estimated from the last measured queuing, booting, and task execution times, using an exponential model of the gain yielded by VM replication. We implement this algorithm in CBRAIN, a science gateway widely used for neuroimaging, and we evaluate it on an infrastructure of 2 clusters and 1 cloud. Results show that it is able to reach some points of the performance-cost trade-off associated to VM deployment.**

I. Introduction

Cloud technologies, in particular hardware virtualization, are revolutionizing scientific computing but new architectures and algorithms are needed to mingle them with clusters and grids for which tremendous efforts were invested during the last 15 years. Science gateways, the suites of tools, services and web portals used to leverage massive computing resources for data analyses and simulations [1], are currently not able to efficiently exploit virtual machines uniformly on clusters and clouds while this would have several advantages. It would facilitate application deployment on multiple infrastructures, it would facilitate the exchange of applications among platforms, it would reduce software errors, and it would ensure the reproducibility of computations which is sometimes challenged by hardware and software heterogeneity [2].

In this paper, we study the use of virtual machines (VMs) for distributed computing in CBRAIN [3], [4], a science gateway largely used for brain image analysis. Our ultimate goal is to build a science gateway where application developers install applications in virtual disk images, users transparently launch application tasks on the homogeneous software environment provided by these disk images, and the system dynamically and autonomously deploys and shutdowns virtual machines on batch processing clusters and on clouds to support the computing workload.

Adjusting the deployment of VMs to the computing workload requires to solve a performance-cost trade-off: while enough virtual machines must be deployed to execute application tasks concurrently, overheads also have to be controlled to limit costs. Performance is considered here as the time difference between the start and finish of the workload execution, i.e. the makespan, and the cost is the cumulative walltime consumed on all the resources multiplied by a site-specific factor expressed in arbitrary units (a.u.). Controlling this trade-off is challenging because (i) there is generally no reliable task duration estimates on the computing resources (ii) the queuing and booting time of VMs on the resources are notoriously hard to predict (iii) the problem is online, i.e. the characteristics of the workload to execute vary over time and are not predictable.

The main contributions of this paper are the following:

- We propose software design principles to extend existing science gateways to deploy VMs on clusters and clouds and schedule tasks in these VMs.
- We propose an algorithm to control the performance-cost trade-off associated to VM deployment on clusters and clouds.
- We implement and demonstrate the relevance of our VM control algorithm in the CBRAIN platform.

The rest of the paper is organized as follows. Section II reviews the related work, Section III describes our software design principles and illustrates them in CBRAIN, Section IV introduces our VM control algorithm, and Section V presents experiments and results.

II. Related work

The relevance of cloud computing for scientific applications has been demonstrated several times [5], [6], and it is now starting to be exploited by science gateways. For instance, WS-PGRADE/gUSE [7] and Apache Airavata [8] can submit application tasks to clouds, the WeNMR science gateway for structural biology [9] recently announced that it can exploit clouds provided by the European Grid Infrastructure

978-1-4799-2784-5/14 $31.00 © 2014 IEEE
DOI 10.1109/CCGrid.2014.42

(EGI)[1], and the Galaxy web-based platform for biomedical research can exploit clusters of Amazon EC2 instances using its CloudMan [10] manager. Among the science gateways used in neuroimaging, the primary application field of CBRAIN, LONI [11] provides a virtual machine packaging neuroimaging tools, and A-BRAIN [12] exploits Microsoft Azure for brain image analysis. To the best of our knowledge, none of these science gateways can at the same time allow users to select a particular disk image to run their analyses, deploy VMs uniformly and automatically on clusters and clouds, and control the performance-cost trade-off associated to the deployment. CBRAIN would be the first science gateway to provide these features.

At the infrastructure level, hardware virtualization has been used to provision virtual clusters on grids as early as 2003 [13], and was later on exploited in systems such as VMPlant [14], In-VIGO [15], Condor [16] and GridWay [17]. This last system encapsulates virtual machines in grid jobs to reuse parts of the grid middleware, a solution that we reuse in CBRAIN (see our second design principle in Section III). Nowadays, virtualized grid sites are commonly used on production infrastructures, for instance on EGI using StratusLab[2]. The ATLAS High-Energy Physics experiment at CERN also jointly exploits grids and clouds with its Panda scheduler [18]. Finally, cloud infrastructures are also used to dynamically extend grid sites when demands grows [19], [20].

An efficient way to optimize multiple conflicting criteria simultaneously, performance and cost in our case, is to optimize a weighted sum of objectives in the Pareto set. The Pareto set, also called Pareto frontier, contains all the feasible solutions that are not dominated by any other solution [21], i.e. for which all the other solutions have either a worse performance or a higher cost. Elements of the Pareto set are called Pareto-optimal solutions. Optimization in the Pareto set was e.g. used in [22] for workflow scheduling in heterogeneous environments. We use this approach to multi-objective optimization in our VM control algorithm presented in Section IV.

Online problems are commonly handled by MAPE-K loops (Monitoring, Analysis, Planning, Execution – with a-priori Knowledge) [23] which have been popularized by autonomic computing to create self-managing systems. An example of such loops in cloud computing are the auto-scaling rules proposed by Amazon to start and stop EC2 instances. These rules constantly monitor the load of the infrastructure and start or stop VMs when the system load remains beyond a threshold for some time. This time, the lower and upper thresholds, and the number of VMs to submit or stop at each iteration are configurable. In another approach, adopted by Rightscale[3], scaling is decided based on votes formulated by the deployed VMs when their load becomes too high. A detailed review of auto-scaling techniques is provided in [24]. Our VM control algorithm extends Amazon's auto-scaling rules to deploy VMs on multiple clusters and clouds.

In addition to determining when VMs have to be started or stopped, our algorithm must also decide on which cluster or cloud sites they should be submitted or removed. The work in [25] describes an optimal VM placement algorithm on multiple clouds to minimize cost. In a multi-cluster-cloud environment, unpredictable resource access times strongly impact performance. A practical and efficient solution to deal with this issue is to submit redundant task copies, also called replicas, on various sites and, when one of them reaches a computing resources, to cancel the others [26], [27]. This strategy guarantees that the resource access time is minimized, but it also generates overheads. The actions selected by our algorithm in the Pareto set correspond to different replication configurations of VM tasks.

Finally, reliable estimations of the task execution times are required in order to estimate the cost associated to a VM submission strategy. Research on the estimation of task duration and resource consumption has been very active, but existing approaches require specific knowledge of the applications, acquired e.g. by machine learning, and still have limited accuracy [28], [29]. Estimations provided by task developers are a practical and reasonable alternate to these techniques, which we use in our algorithm.

III. System Architecture

A. Design principles

We identified three software architecture principles to extend science gateways to deploy VMs on clusters and clouds, and to schedule application tasks on these VMs. The first principle is to separate the provisioning of VMs from the execution of tasks in these VMs, and to schedule tasks to VMs as late as possible, i.e. only once they have booted. Therefore, VMs may execute several tasks and tasks may run in several VMs, for instance in case of resubmission after a failure. This separation and late binding have the following advantages: (i) failures impacting VMs before they have booted do not impact task execution (ii) the application makespan depends on the average rather than the maximal VM queuing time (see equations 4 and 8 in [30]) (iii) the load can be dynamically balanced among VMs depending on their actual performance to execute the application tasks. Naturally, separating VM provisioning from task execution does not prevent tasks from specifying requirements on VMs (e.g. a particular disk image, RAM, disk space and CPU), and neither does it forbid VM provisioning to take workload characteristics into account to deploy VMs. In grid computing, the late task-to-resource binding and the separation between resource provisioning and task execution have been very successfully demonstrated by pilot-job systems [31], [32], [33] which are now used by the vast majority of systems exploiting large grids. Thus, we believe that this model should be reused to deploy VMs on clusters and clouds.

Our second principle is to encapsulate VMs as regular tasks of the science gateway, which allows to reuse to a large extent the task execution mechanisms used for grids and clusters. For instance, interfaces to batch systems and pilot jobs can be reused to deploy VMs on various clusters, data management systems may handle disk images, and workflow systems may

[1] http://wiki.egi.eu/wiki/Fedcloud-tf:FedCloudWeNMR
[2] http://stratuslab.eu
[3] http://support.rightscale.com/

manage the life cycle of VMs, including submission, monitoring, and error recovery.

Our third principle is to associate a virtual computing site to each disk image, and to reuse site management features of existing systems for scheduling tasks on VMs. For instance, access control policies defined for physical sites can be used to specify permissions on disk images, accounting systems collecting statistics for physical sites may also track information per disk image, requirement description languages may be used for task-to-disk-image matchmaking, and of course, existing schedulers can be used to dispatch tasks among disk images and among VMs of a given disk image.

B. Implementation in CBRAIN

Figure 1 presents the application to CBRAIN of the 3 principles described above. Figure 1(a) shows the main components of the initial CBRAIN system, and their interactions involved in task execution. The CBRAIN portal is the user front-end, where task executions and file transfers can be initiated. The portal communicates with the CBRAIN database which centralizes information about applications, tasks, files and their location, computing and storage sites, users and groups. Associations between applications and sites are also declared in the portal and database. A worker is deployed on the head node of each site, and it periodically pulls tasks from the database for execution. Once it selected a task for execution, the worker downloads the task input files from CBRAIN data providers, submits and monitors the task to the site's batch manager, and uploads the output files back to data providers. This data transfer architecture avoids scalability issues at the data providers since the number of concurrent connections is proportional to the number of sites rather than the number of active jobs. The worker also has a data cache to limit file transfers. All entities communicate through ssh tunnels authenticated using the portal's key pair. A more comprehensive description of CBRAIN and its positioning with respect to other grid systems is available in [3].

Figure 1(b) shows CBRAIN's extension to deploy VMs. As a result, VMs can be started, monitored and terminated from the CBRAIN portal on clusters and clouds. Consistently with our first principle, this process, handled by our VM control algorithm presented in Section IV, is separated from task scheduling. In application of our second principle, a VM deployment task was added to CBRAIN. This task has a single input file, the disk image, and produces no output file. If the worker processing this task runs on a cluster (as in the left part of Figure 1(b)), the disk image is transferred from a data provider to the worker cache, and the VM deployment task is enqueued in the batch system. When it reaches a computing node, the task boots a virtual machine from the disk image and redirects a high port to the VM's ssh port to allow communications. Once the VM is booted, the worker uses this port to initiate an ssh tunnel with the VM, and to mount a shared directory in the VM using sshfs. No other communication between the VM and external hosts is required, which simplifies network configuration as VMs don't need a dedicated IP address. We use KVM and its QEMU layer, for which the only required site administrator intervention is to set the proper permissions on `/dev/kvm` to enable full virtualization. In case this is not available, pure emulation can be used, for prototyping only since it remains very slow compared to full virtualization. The only requirements for disk images are to start an ssh daemon at startup, to authorize the portal's public key (as a normal user), and to have sshfs installed. The parameters of a VM are: RAM size, number of CPUs, and number of task slots that CBRAIN could schedule in it. The CBRAIN worker was also extended to start, monitor and terminate VMs on OpenStack clouds (see right part of Figure 1(b)). The worker is deployed inside the cloud, so that only 1 public IP is required. In our current implementation, VMs are manually pre-registered in the cloud, and associated to the disk images declared in CBRAIN. In the future, we could also extend the worker to register disk images and store their cloud id in the CBRAIN database.

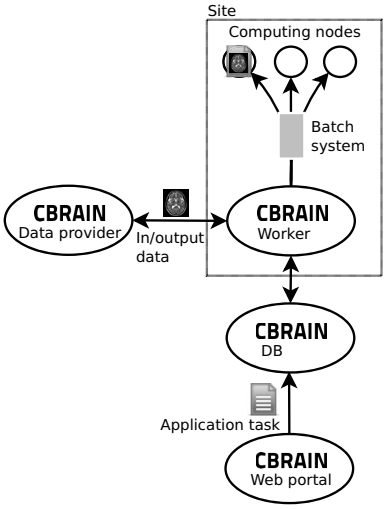
Figure 1(c) shows task scheduling on VMs. In application of our third principle, a virtual site is associated to each disk image. In CBRAIN, this allows to associate applications to this virtual site, and to submit tasks to a disk image with no extra development. The parameters of a virtual site are the disk image file and the user name used to connect to this disk image. We modified the CBRAIN worker so that it considers pulling tasks submitted to disk images for which at least 1 VM with free task slots is available on its resources. Workers handle input and output files of tasks submitted to VMs as in the initial system. Task submission and monitoring is done through the ssh tunnel between the worker and the VM. The next section presents the VM factory controlling when and on which sites VMs are deployed in response to the workload generated by users.

IV. VM CONTROL ALGORITHM

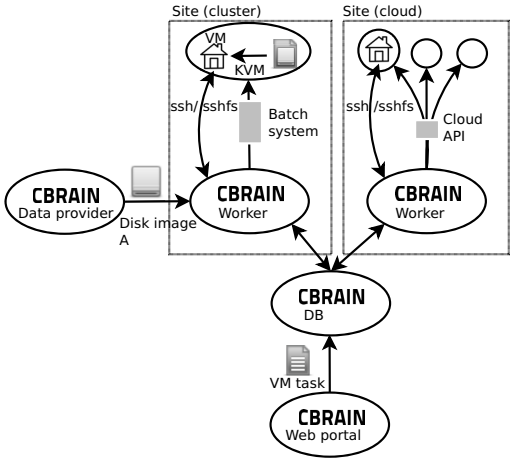
Our VM control algorithm is based on a MAPE-K loop that periodically evaluates auto-scaling rules based on the load of the system and fixed thresholds. When the load is too high, a VM is submitted and replicated on multiple sites. Site selection is based on an estimation of the task execution cost and round-trip time on each site, computed from the last measured waiting, booting and execution times on this site. The gain yielded by VM replication on the queuing and booting times is estimated using an exponential model. All possible replication strategies are then considered, and the Pareto set of non-dominated solutions is constructed iteratively. The Pareto-optimal replication strategy that minimizes the linear combination of cost and makespan is then selected and implemented. A parameter λ controls the weights associated to the makespan and cost. When the load is too low, VMs are removed starting with queuing VMs, followed by booting VMs, and finally idle VMs. The applications targeted by this algorithm are bag of tasks for which a code is iterated on independent pieces of data. This matches several use cases encountered in CBRAIN and other science gateways.

A. Notations

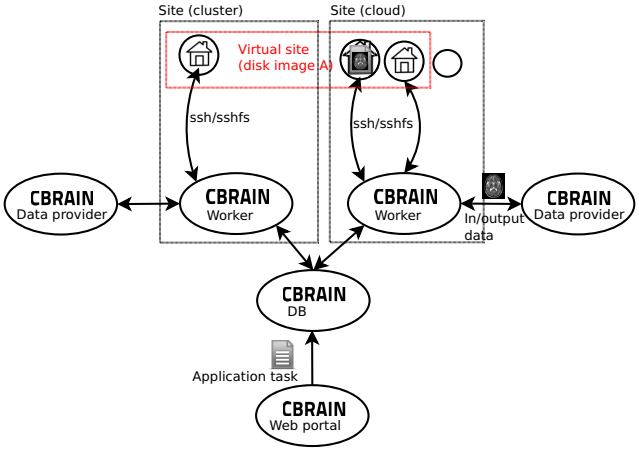
The following notations describe the entities involved in our VM control algorithm. In the following, $\#$ denotes the cardinal



(a) Initial system



(b) VM deployment



(c) Task scheduling on VMs

Fig. 1. Evolution of the CBRAIN system to deploy VMs on clusters and clouds for task execution.

operator, uppercase letters denote sets, bold lowercase letters are vectors, and plain lowercase letters are scalars.

- VMs:
  - Set of all active VMs: V
  - Number of active VMs: $\#V$
  - Indices of queuing VMs: Q
  - Indices of booting VMs: B
  - Indices of booted VMs: C
  - Indices of idle VMs: I
  - Number of job slots in VM i: p_i
  - Indices of the replicas of VM i: R_i
  - Timestamp of last status change of VM i: t_i
  - Site where VM i was submitted: $\mathbf{c_i} \in S$

  The number of job slots p_i in the VM, and other VM characteristics not used by the control algorithm, mainly RAM and number of CPUs, are fixed for each site.
- Tasks:
  - Set of all active tasks: T
  - Number of active tasks: $\#T$
  - Estimated walltime of task i: e_i
  - Median estimated walltime of all queued tasks: $\bar{e}$

  The walltime denotes the total time spent by a task on a computing resource. In the following, we will also refer to the round-trip time of a task (i.e. the sum of its queuing time and walltime), and to the makespan of a set of tasks (i.e. the max of all task round-trip times).
- Sites:
  - Set of all sites: S
  - Number of sites: $\#S$
  - Max number of active VMs on site i: m_i
  - Estimated queuing time: q_i
  - Estimated booting time: b_i
  - Cost factor: α_i
  - Performance factor on site i: ϕ_i
  - Indices of VMs queuing on site i: Q_i
  - Indices of VMs booting on site i: B_i
  - Indices of VMs idle on site i: I_i

  The max number of active VMs on site i is set by the site or science-gateway administrator. The estimated queuing and booting times may be produced by any method available in the literature. In CBRAIN, we use their last measured values since this is the only available information so far. The cost factor α_i, measured in arbitrary cost units a.u., is applicable both to cloud and to cluster sites. For cloud sites, it may correspond to the monetary cost charged by the cloud providers per time unit; for clusters, it may be linked to the time allocation β_i granted to the platform, e.g. $\alpha_i = 1/\beta_i$, and updated accordingly. A performance factor ϕ_i models performance heterogeneity among computing sites. The actual duration of task j on site i is estimated as $e_j\phi_i$. Performance factors are considered independent from the application, but they could easily be extended to describe specific application-site interactions. VM submission strategies are defined by actions $A \subset S$ where elements of A are the sites where the VM will be replicated.

B. Control loop

Algorithm 1 controls VM submission and removal, implementing auto-scaling rules in a MAPE-K loop. At a period τ, the system load L is measured as the ratio between the number of active tasks in the system and the number of job slots of active VMs. When L goes beyond a threshold value μ^+ for a duration ν^+, k^+ new VMs are submitted by procedure SUBMITVM. And when L stays under μ^- for ν^-, k^- VMs are removed by procedure REMOVEVM. The performance-cost trade-off is controlled by λ: if $\lambda = 1$ (resp. $\lambda = 0$), then the Pareto-optimal solution with the lowest makespan (resp. cost) is selected at each iteration. Finally, VM replicas are handled by procedure HANDLEREPLICAS. This control loop is started for every disk image in the platform by a process called VM factory.

Algorithm 1 Auto-scaling algorithm

```
procedure CONTROLVMS(τ,μ+,μ−,ν+,ν−,k+,k−,λ)
    while true do
        Measure #T, #V, p_i, q_i, b_i, φ_i
        L = #T / Σ_{i∈[1,#V]} p_i
        if L > μ+ then
            submit = true
            Start timer
            while timer < ν+ do
                if L < μ+ then
                    submit = false
                    break
            if submit then SUBMITVM(λ, k+)
        if L < μ− then
            remove = true
            Start timer
            while timer < ν− do
                if L > μ− then
                    remove = false
                    Break
            if remove then REMOVEVM(k−, #V, L)
        HANDLEREPLICAS
        Wait τ
```

VM submission, removal and replica handling are described in Algorithm 2. In procedure REMOVEVM, VM selection for removal is based on causal ordering: the algorithm first attempts to remove the latest VM that reached queuing, then the latest VM that started booting, and finally the VM that has been idle for the longest time. To avoid starvation, the last VM is not removed unless the system load is null. Procedure SUBMITVM selects a set of sites from the Pareto frontier of all possible site sets, and replicates a VM on all these sites. First, the set of non-saturated sites (A in procedure SUBMITVM) is determined. The set of all possible actions is then the power set $\mathcal{P}(A)$, from which the Pareto frontier P is extracted, and an optimum O is found using parameter λ. Although exploring all solutions in $\mathcal{P}(A)$ gives an exponential complexity to the VM submission procedure w.r.t the number of computing sites, the execution time remains largely tractable for a realistic number of sites[4]. If the number of sites grows, more specific search strategies should be used to estimate the Pareto set [34]. The algorithm finally executes the selected action, by replicating a VM to all sites in O. Replicas are handled by HANDLEREPLICAS which exploits the fact that all VMs submitted to the same site are identical: instead of removing the replicated VMs of a booted VM i, i.e. all $\mathbf{v_j} \in R_i$, the procedure removes VM $\mathbf{v_k}$ from site $\mathbf{c_j}$ using the same causal order than in REMOVEVM, implemented here in REMOVEVMFROMSITE. This switching of identical VMs saves queuing and booting time, but it requires to properly adjust replica indexes, which HANDLEREPLICAS does before returning.

Algorithm 2 VM submission, removal and replica handling

```
procedure REMOVEVM(k−,#V,L)
    for n = 0, n < k−, n++ do
        if #V > 1 || L=0 then
            if Q ≠ ∅ then
                i = argmin{t_i, i ∈ Q}
                Remove VM i
            else if B ≠ ∅ then
                i = argmin{t_i, i ∈ B}
                Remove VM i
            else if I ≠ ∅ then
                i = argmax{t_i, i ∈ I}
                Remove VM i
procedure SUBMITVM(λ,k+)
    A = S - {s_i ∈ S, #{v_j ∈ #V, c_j = s_i} ≥ m_i} ▷ Set of sites with available VM slots
    P = PARETOSET(P(A))
    O = OPTIMIZE(P,λ)
    Submit and replicate k+ VM to all sites in O
procedure REMOVEVMFROMSITE(i)
    if Q_i ≠ ∅ then
        j = argmin{t_j, j ∈ Q_i}
        Remove VM j
    else if B_i ≠ ∅ then
        j = argmin{t_j, j ∈ B_i}
        Remove VM j
    else if I_i ≠ ∅ then
        j = argmax{t_j, j ∈ I_i}
        Remove VM j
    return v_j
procedure HANDLEREPLICAS
    for i ∈ C do
        for j ∈ R_i do
            v_k = REMOVEVMFROMSITE(c_j)
            Remove i from R_j
            for l ∈ R_k do
                Replace k by j in R_l
                Add l to R_j
```

The construction of the Pareto frontier is described by procedure PARETOSET in Algorithm 3. The performance and

[4]We measured an execution time of 0.26s ± 0.13s on an Intel Core i7 2620M at 2.7GHz with 8GB of RAM for a call to PARETOSET on 15 sites with random queuing times, booting times, costs and performance factors.

cost of each possible action are determined by procedure ACTIONEXECUTIONTIMEANDCOST. First, the index k of the site where the VM is likely to boot is determined. Then, the cost c associated to this action is estimated as the cost to boot the VM on site k and to execute a median task in it $(\alpha_k(b_k + \phi_k\bar{e}))$ plus the maximal cost overhead due to VM replication, i.e. the sum of VM booting costs on all other sites $\left(\sum_{i\neq k} \alpha_i b_i\right)$. The action performance is modeled by an estimation r of the round-trip time of 1 task executed in a VM replicated to these sites. The gain yielded by VM replication is modeled by an exponential decrease lower-bounded by $\frac{q_k+b_k}{2}$. In absence of model to describe the VM queuing and booting times, such an exponential decrease is justified by the developments in [35]. Term $\sum_{s_i\in A} \frac{q_k+b_k}{q_i+b_i}$ will reduce the impact of sites with long waiting and booting times compared to the expected booting site k. Factor $\frac{q_k+b_k}{2}$ is empirical and denotes the minimum achievable queuing and booting time. Once the execution time and cost of an action are determined, it is added to the Pareto frontier using procedure ADDTOPARETOSET. This procedure is called iteratively by PARETOSET to build the Pareto set; its first parameter (P), is a set of actions verifying the Pareto property of non dominance, and its second parameter (r, c, A) is the execution time, cost and action to be added to the Pareto set. We could show by recurrence on $\#P$ that it returns the set of non dominated elements of $P \cup (r, c, A)$, i.e. its Pareto frontier.

Algorithm 3 Construction of the Pareto set

```
procedure ACTIONEXECUTIONTIMEANDCOST(A)
    k = argmin{q_i + b_i, s_i ∈ A, i ∈ [1, #S]}
    c = α_k(b_k + φ_k ē) + Σ_{i≠k} α_i b_i
    r = (q_k+b_k)/2 (1 + exp(1 − Σ_{s_i∈A} (q_k+b_k)/(q_i+b_i))) + φ_k ē
    return (r,c)
procedure DOMINATE((r, c),(s, d))
    if r ≤ s and c ≤ d and (r, c) ≠ (s, d) then
        return true
    else
        return false
procedure ADDTOPARETOSET(P,(r, c, A))
    Q = {(r, c, A)}
    for (s, d, B) ∈ P do
        if DOMINATE ((s, d), (r, c)) then
            return P
        if not DOMINATE ((r, c), (s, d)) then
            Q = Q ∪ {(s, d, B)}
    return Q
procedure PARETOSET(Q)
    P=∅
    for A ∈ Q do
        (r,c) = ACTIONEXECUTIONTIMEANDCOST(A)
        P = ADDTOPARETOSET(P,(r,c,A)))
    return P
```

Finally, Algorithm 4 is used by procedure SUBMITVM in Algorithm 2 to determine the performance-cost trade-off based on λ. The execution time and cost values are combined after normalization w.r.t their min and max values in the Pareto set.

Algorithm 4 Optimization in the Pareto set

```
procedure OPTIMIZE(P,λ)
    r_min = min{r_i, (r_i, ., .) ∈ P}
    r_max = max{r_i, (r_i, ., .) ∈ P}
    c_min = min{c_i, (., c_i, .) ∈ P}
    c_max = max{c_i, (., c_i, .) ∈ P}
    (r̂, ĉ, Â) = argmin{λ (r−r_min)/(r_max−r_min) +
                (1 − λ) (c−c_min)/(c_max−c_min), (r, c, A) ∈ P}
    return Â
```

C. Implementation in CBRAIN

This algorithm was implemented as a thread of the CBRAIN portal. The task, site, and virtual-machine sets T, S and V were directly obtained from the central CBRAIN database, and parameters p_i, q_i, b_i and ϕ_i were estimated as the last measured values. For e_i, we used a method declared for each CBRAIN task which returns an estimation of the task execution time. This method can access all the system parameters, and therefore gives a lot of flexibility to the developers to provide accurate estimates.

We also implemented a round-robin alternative to procedure SUBMITVM for comparison purposes.

V. EXPERIMENTS

A. Experimental protocol

We evaluated our VM control algorithm for different values of the cost-performance parameter λ, using μ^+=1.3, μ^-=0.5, $k^+ = k^- = 1$, $\nu^+ = \nu^- = 5$s and τ=10s. Values for μ^+, μ^-, k^+, k^-, ν^+, ν^- and τ were chosen empirically, and studying their influence is out of the scope of this study. For each λ value, a distinct but identical CentOS disk image was created and associated to a VM factory. Disk images were cached in all computing sites prior to the experiment so that results were not impacted by transfer times. VM factories were tested under 9 different workloads characterized by a number of task bursts, a number of tasks per burst, and a number of processed images per task to which the task duration on a particular resource was proportional. Table I details these workload parameters. Burst workloads were chosen due to their numerous occurrences in CBRAIN. For a given workload, all VM factories ran concurrently to ensure similar execution conditions. Task bursts were submitted every 10 minutes to the randomly shuffled disk images. Tasks processed 33-MB images with the FSL bet application[5], and produced a 18.7-MB result per image. Before the workload was submitted, an initialization phase submitted a VM to each site (to measure up-to-date queuing and booting times), and an application task to each VM (to measure an up-to-date site performance factor). After the initialization phase, all VMs were terminated and the workload execution started. The makespan was computed on the application tasks, and the cost as the total cumulative walltime consumed by VMs weighted by site costs.

[5]http://fsl.fmrib.ox.ac.uk

Bursts	Tasks/burst	Images/task
1	25	1
1	50	1
1	75	1
1	25	1
2	25	1
3	25	1
1	25	1
1	25	25
1	25	55

TABLE I
WORKLOAD PARAMETERS FOR THE 9 EXPERIMENTAL RUNS.

Three sites were used: (i) `Nimbus`, an OpenStack cloud provided by Compute-Canada, located at the University of Sherbrooke, with 64 virtual CPUs, 92 GB of RAM, and 24 available VM instances (ii) `Guillimin`, a cluster provided by Compute-Canada, located at the McGill University HPC center, with KVM installed on 191 nodes, 16 cores per node and 62 GB of RAM per node (iii) `Creatis`, a cluster at the CREATIS laboratory in Lyon, with KVM installed on 100 nodes, 304 cores, and 2 GB of RAM per core. Table II summarizes the cost and VM parameters on these sites.

B. Results

The makespan and cost obtained by the different VM factories are shown in Figure 2 for different task numbers (with 1 burst and 1 image per task), in Figure 3 for different burst numbers (with 25 tasks per burst and 1 image per task), and in Figure 4 for different numbers of images per task (with 1 burst and 25 tasks per burst). In most cases, the measured makespan decreases when λ increases, and the cost increases when λ increases. This demonstrates that our algorithm is able to reach specific points of the performance-cost trade-off through λ. As shown by the distribution of selected strategies plotted on Figure 5, $\lambda = 1$ and $\lambda = 2/3$ achieve low makespans by replicating VMs to the non-saturated sites. Conversely, $\lambda = 0$ and $\lambda = 1/3$ achieve low costs by focusing on the cheaper sites `Guillimin` and `Creatis`.

In some cases, for instance 25 and 75 tasks on Figure 2, the performance and cost of $\lambda = 1$ are very close to the values obtained for $\lambda = 2/3$. This happens when these VM factories select the same submission strategies at most iterations. In this case, the observed differences are only coming from race conditions for resource attribution. This situation comes either from a reduced cardinality of the Pareto set, or from extreme values of α_i, p_i, ϕ_i, b_i and q_i. Figure 6 plots the distribution of the cardinality of the Pareto set among all executions. The average cardinality is 3.2, which explains why 2 of the 4 VM factories based on the Pareto set often have similar makespan and cost.

Site	α_i	m_i	p_i	vCPUs per VM	RAM per VM
`Nimbus`	100	10	2	2	8
`Creatis`	10	100	2	2	4
`Guillimin`	1	100	2	2	4

TABLE II
SITE PARAMETERS

The round-robin strategy is clearly not able to control the cost. VM factories with $\lambda = 0$ and $\lambda = 1/3$ have much lower costs than round-robin, up to 67 times for 25 images per task (Figure 4). This was expected since the round-robin factory submits VMs to sites regardless of their costs. Concerning the makespan, round-robin achieves similar performance than $\lambda = 1$ and $\lambda = 2/3$ in most cases. This is due to the fact that in our experimental conditions, replicating VMs to all sites at every iteration is similar to round-robin since all VMs are equivalent. We expect a better added value of $\lambda = 1$ and $\lambda = 2/3$ on infrastructures with more sites, where these strategies would focus on a reduced subset of fast sites whereas round-robin would spread VMs on all sites indistinctly.

For 55 images/task (Figure 4), round-robin achieves a much better makespan than $\lambda = 1$ and $\lambda = 2/3$ because the Pareto set contained only strategy (`Guillimin`) for 6 of the 8 VM submission iterations of $\lambda = 1$ and $\lambda = 2/3$. As shown on Figure 7, this is due to temporary poor performance on `Creatis` and `Nimbus` leading strategy (`Guillimin`) to dominate all the others. In practice, the measured performance of `Nimbus` was due to temporary long booting times which eventually did not impact round-robin. To address this issue, we could add tolerance margins to the DOMINATE procedure in Algorithm 3. Tolerance bounds could be based on the measured variability of q_i, b_i and p_i.

Finally, Figure 8 plots the load measured by the VM factories for 2 bursts, 25 tasks per burst, 1 image/task. Loads behaved as expected, with rapid decreases under μ^+ after task bursts, progressive decrease to μ^- as tasks complete, and a final step at μ^- before dropping to zero. For $\lambda = 1$ and $\lambda = 2/3$, the load decrease under μ^- is hardly mitigated by VM termination due to the fast pace of task completion. Increasing k^- could save cost in these cases. A similar behavior was observed for other workloads.

VI. DISCUSSION

The extensions of CBRAIN presented in Section III allow users to submit application tasks to disk images from which the system boots VMs in clouds and clusters. When applications are deployed on multiple disk images, CBRAIN users can choose their preferred image to control the execution environment. To the best of our knowledge, CBRAIN is currently the only science gateway to enable this. It provides a significant added value compared to (i) IaaS clouds that do not handle application tasks (ii) software projects deploying virtual clusters or grid sites on virtualized resources (iii) systems submitting tasks to both grid and cloud resources without using VMs for grid tasks, resulting in a heterogeneous software environment (iv) SaaS clouds where disk images cannot be selected by the users, limiting the flexibility of the execution environment.

The presented architecture allows to re-use existing clusters for cloud computing, with limited required administrative intervention. To adjust technical contexts and preferences, different hypervisors could be used on clusters to deploy VMs, with no major modification of the system. Overall, CBRAIN exploits cluster and cloud sites as a single uniform cloud

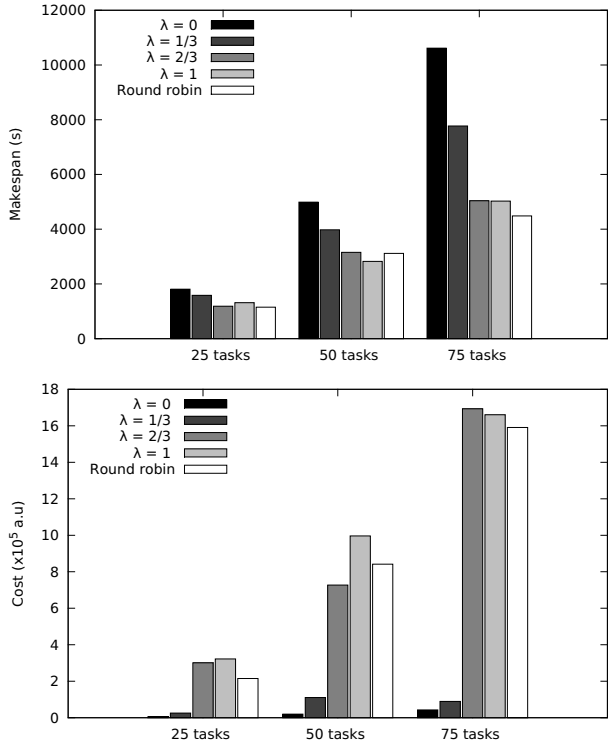


Fig. 2. Makespan and cost of the VM factories for different task numbers (1 burst, 1 image/task)

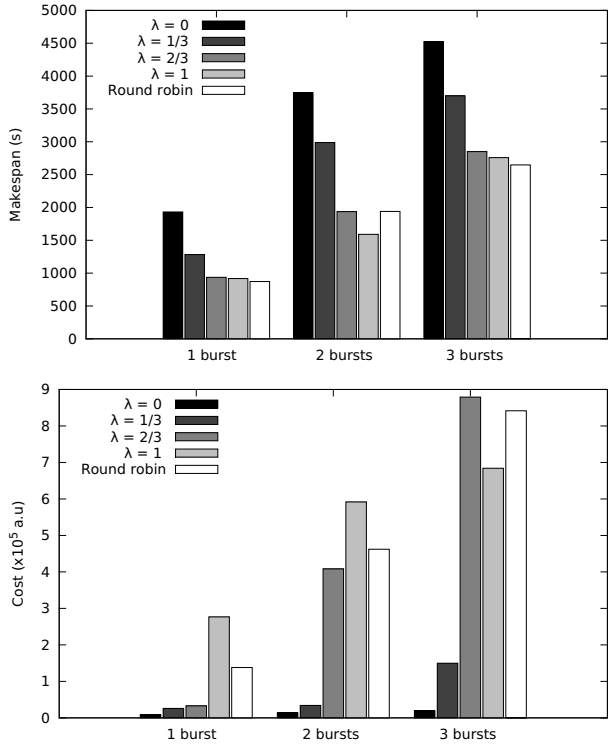


Fig. 3. Makespans and costs for different burst numbers (25 tasks/burst, 1 image/task)

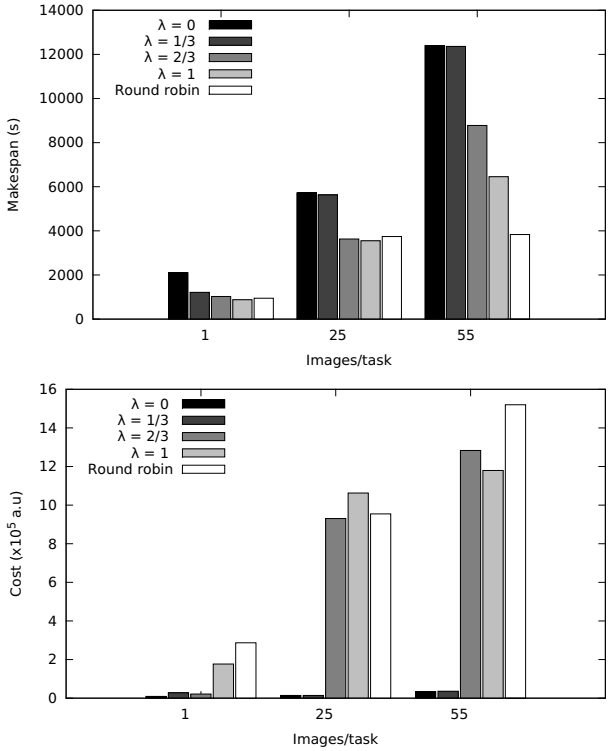


Fig. 4. Makespan and cost of the VM factories for different numbers of images per task (25 tasks/burst, 1 burst)

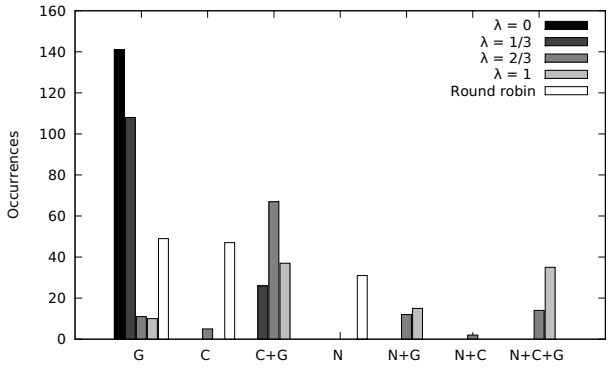


Fig. 5. VM submission strategies selected by VM factories (all workloads). C stands for `Creatis`, G for `Guillimin` and N for `Nimbus`.

where VMs can be deployed on demand. Thanks to the design principles detailed in Section III, this was achieved at a modest development effort, re-using most of the existing CBRAIN features for file and task management. We believe that these design principles are general enough to be reused for similar cloud extensions in other science gateways.

The VM control algorithm presented in Section IV is able to reach specific points of the performance-cost trade-off through its parameter λ. Basic estimations of queuing, booting, and execution time proved usable, but they may not be sufficient when these parameters vary rapidly over time. Considering

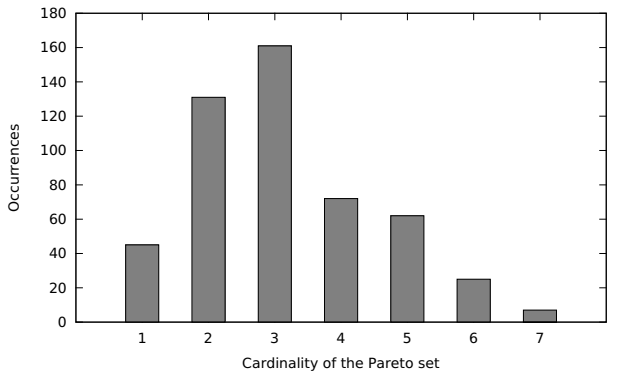


Fig. 6. Distribution of the cardinality of the Pareto set (all workloads)

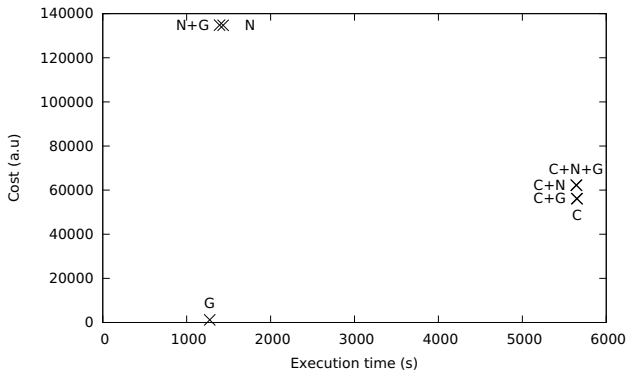


Fig. 7. Representation of the action set at the first VM factory iteration for 55 images/task, 1 burst, 25 tasks (Figure 4), when round-robin was faster than $\lambda = 1$ and $\lambda = 2/3$. C stands for `Creatis`, G for `Guillimin` and N for `Nimbus`.

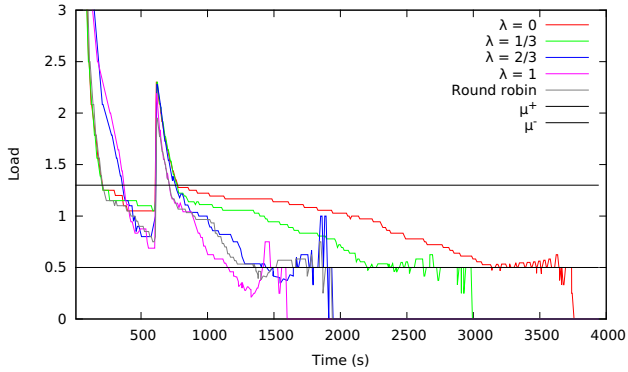


Fig. 8. Load measured by VM factories (25 tasks/burst, 2 bursts, 1 image/task)

active tasks in these estimations could help coping with brutal parameter increases.

The estimation of the execution time of an action, represented by variable r in Algorithm 3, also deserves deeper investigation. In particular, estimating the task execution time as $\phi_k \bar{e}$ ignores the risk to replicate VMs on a site with poorly performing computing nodes. It could be worth studying a probabilistic model where $\phi_k \bar{e}$ would be replaced by the expectation of a distribution modeling the fact that VMs are more likely to execute on sites with short queuing times, but may still end up on the other ones.

Finally, the remarkable performance of round-robin for makespan optimization makes it an interesting reference to evaluate VM submission strategies for this criteria, although it may not be so well performing when more sites are used.

VII. Conclusion and Future Work

The presented evolution of the CBRAIN science gateway to deploy virtual machines on clusters and clouds and schedule application tasks on these VMs results in a unique system exploiting heterogeneous computing resources with a very homogeneous software environment. We plan to progressively integrate these new features in the production CBRAIN environment, to enable its exploitation by users, and to collect more feedback about its use at a large scale. We also envisage to exploit it to exchange applications with other science gateways used in medical imaging.

We expect that data transfers and storage will be a limitation in the exploitation phase, when the number of disk images becomes important. Some practical policy may be enforced to limit this issue, but the VM factory could also be revised to control data storage. Storage utilization could for instance be a third objective optimized by the VM factory, tending to group VMs of a given disk image on the same site to avoid replicating the disk image.

Task scheduling on the available VMs could also be improved by taking into account data localization and storage utilization, by grouping tasks in the same VM to reduce costs, and by considering multi-core tasks. Finally, task pre-emption could also be envisaged, for instance based on VM snapshoting and migration.

VIII. Acknowledgment

We thank Compute-Canada[6] and Calcul Québec[7] for providing the infrastructure to perform the experiments presented in this paper: Alain Veilleux, Michel Barette, Michael Jeanson and Jean-François Landry at the University of Sherbrooke[8], Bryan Caron and his team at the HPC center of the McGill University[9]. We also thank Natacha Beck and Tarek Sherif for their developments and support in CBRAIN. We thank Fabrice Bellet for administrating the Creatis cluster, in particular the KVM installation. The research leading to this publication received funding from the EC FP7 Programme under grant

[6]https://computecanada.ca
[7]http://www.calculquebec.ca
[8]http://www.usherbrooke.ca/
[9]http://www.hpc.mcgill.ca

agreement 312579 ER-flow = Building an European Research Community through Interoperable Workflows and Data.

REFERENCES

[1] T. Kiss, "Science Gateways for the Broader Take-up of Distributed Computing Infrastructures," *Journal of Grid Computing*, vol. 10, no. 4, pp. 599–600, Nov. 2012.

[2] E. H. B. M. Gronenschild, P. Habets, H. I. L. Jacobs, R. Mengelers, N. Rozendaal, J. van Os, and M. Marcelis, "The effects of FreeSurfer version, workstation type, and Macintosh operating system version on anatomical volume and cortical thickness measurements." *PloS one*, vol. 7, no. 6, p. e38234, Jan. 2012.

[3] T. Sherif, P. Rioux, M.-E. Rousseau, N. Kassis, N. Beck, T. Glatard, R. Adalat, S. Das, and A. Evans, "CBRAIN: A web-based, distributed computing platform for collaborative neuroimaging research," *Frontiers of Neuroscience*, 2014, submitted, in revision.

[4] G. B. Frisoni, A. Redolfi, D. Manset, M.-E. Rousseau, A. Toga, and A. C. Evans, "Virtual imaging laboratories for marker discovery in neurodegenerative diseases." *Nature reviews. Neurology*, vol. 7, no. 8, pp. 429–38, Aug. 2011.

[5] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the Use of Cloud Computing for Scientific Workflows," in *2008 IEEE Fourth International Conference on eScience*. IEEE, Dec. 2008, pp. 640–645.

[6] C. Vecchiola, S. Pandey, and R. Buyya, "High-Performance Cloud Computing: A View of Scientific Applications," in *2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*. IEEE, 2009, pp. 4–16.

[7] P. Kacsuk, Z. Farkas, M. Kozlovszky, G. Hermann, A. Balasko, K. Karoczkai, and I. Marton, "WS-PGRADE/gUSE Generic DCI Gateway Framework for a Large Variety of User Communities," *Journal of Grid Computing*, vol. 10, no. 4, pp. 601–630, Nov. 2012.

[8] S. Marru, R. Gardler, A. Slominski, A. Douma, S. Perera, S. Weerawarana, L. Gunathilake, C. Herath, P. Tangchaisin, M. Pierce, C. Mattmann, R. Singh, T. Gunarathne, and E. Chinthaka, "Apache airavata," in *Proceedings of the 2011 ACM workshop on Gateway computing environments - GCE '11*. New York, New York, USA: ACM Press, Nov. 2011, p. 21.

[9] T. A. Wassenaar, M. Dijk, N. Loureiro-Ferreira, G. Schot, S. J. Vries, C. Schmitz, J. Zwan, R. Boelens, A. Giachetti, L. Ferella, A. Rosato, I. Bertini, T. Herrmann, H. R. A. Jonker, A. Bagaria, V. Jaravine, P. Güntert, H. Schwalbe, W. F. Vranken, J. F. Doreleijers, G. Vriend, G. W. Vuister, D. Franke, A. Kikhney, D. I. Svergun, R. H. Fogh, J. Ionides, E. D. Laue, C. Spronk, S. Jurkša, M. Verlato, S. Badoer, S. Dal Pra, M. Mazzucato, E. Frizziero, and A. M. J. J. Bonvin, "WeNMR: Structural Biology on the Grid," *Journal of Grid Computing*, vol. 10, no. 4, pp. 743–767, Nov. 2012.

[10] E. Afgan, D. Baker, N. Coraor, B. Chapman, A. Nekrutenko, and J. Taylor, "Galaxy CloudMan: delivering cloud compute clusters." *BMC bioinformatics*, vol. 11 Suppl 1, no. Suppl 12, p. S4, Jan. 2010.

[11] I. Dinov, K. Lozev, P. Petrosyan, Z. Liu, P. Eggert, J. Pierce, A. Zamanyan, S. Chakrapani, J. Van Horn, D. S. Parker, R. Magsipoc, K. Leung, B. Gutman, R. Woods, and A. Toga, "Neuroimaging study designs, computational analyses and data provenance using the LONI pipeline." *PloS one*, vol. 5, no. 9, p. e13070, Jan. 2010.

[12] G. Antoniu, A. Costan, B. D. Mota, B. Thirion, and R. Tudoran, "A-Brain: Using the Cloud to Understand the Impact of Genetic Variability on the Brain," *ERCIM News*, vol. 2012, no. 89, 2012.

[13] R. Figueiredo, P. Dinda, and J. Fortes, "A case for grid computing on virtual machines," in *23rd International Conference on Distributed Computing Systems, 2003. Proceedings.* IEEE, 2003, pp. 550–559.

[14] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo, "VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing," in *Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference*, 2004, p. 7.

[15] A. M. Matsunaga, M. O. Tsugawa, S. Adabala, R. J. Figueiredo, H. Lam, and J. A. B. Fortes, "Science gateways made easy: the In-VIGO approach," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 6, pp. 905–919, Apr. 2007.

[16] M. A. Murphy, B. Kagey, M. Fenn, and S. Goasguen, "Dynamic Provisioning of Virtual Organization Clusters," in *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE, May 2009, pp. 364–371.

[17] C. V. Blanco, E. Huedo, R. S. Montero, and I. M. Llorente, "Dynamic Provision of Computing Resources from Grid Infrastructures and Cloud Providers," in *2009 Workshops at the Grid and Pervasive Computing Conference*. IEEE, May 2009, pp. 113–120.

[18] F. Harald Barreiro Megino, D. Benjamin, K. De, I. Gable, V. Hendrix, S. Panitkin, M. Paterson, A. De Silva, D. van der Ster, R. Taylor, R. A. Vitillo, and R. Walker, "Exploiting Virtualization and Cloud Computing in ATLAS," *Journal of Physics: Conference Series*, vol. 396, no. 3, p. 032011, Dec. 2012.

[19] C. Vázquez, E. Huedo, R. S. Montero, and I. M. Llorente, "On the use of clouds for grid resource provisioning," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 600–605, 2011.

[20] S. Ostermann, R. Prodan, and T. Fahringer, "Dynamic Cloud provisioning for scientific Grid workflows," in *2010 11th IEEE/ACM International Conference on Grid Computing*. IEEE, Oct. 2010, pp. 97–104.

[21] V. Pareto, "Manuale di economica politica, societa editrice libraria," *Manual of political economy*, vol. 1971, 1906.

[22] H. M. Fard, R. Prodan, J. J. D. Barrionuevo, and T. Fahringer, "A Multi-objective Approach for Workflow Scheduling in Heterogeneous Environments," in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012)*. IEEE, May 2012, pp. 300–309.

[23] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[24] T. Lorido-Botrán, J. Miguel-Alonso, and J. A. Lozano, "Auto-scaling Techniques for Elastic Applications in Cloud Environments," Department of Computer Architecture and Technology University of the Basque Country, Tech. Rep. EHU-KAT-IK-09-12, Sept. 2012.

[25] S. Chaisiri and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in *2009 IEEE Asia-Pacific Services Computing Conference (APSCC)*. IEEE, Dec. 2009, pp. 103–110.

[26] H. Casanova, "On the Harmfulness of Redundant Batch Requests," in *International Symposium on High-Performance Distributed Computing*, Los Alamitos, CA, USA, 2006, pp. 255–266.

[27] W. Cirne, F. Brasileiro, D. Paranhos, L. F. W. Goes, and W. Voorsluys, "On the Efficacy, Efficiency and Emergent Behavior of Task Replication in Large Distributed Systems," *Parallel Computing*, vol. 33, pp. 213–234, 2007.

[28] R. da Silva, G. Juve, E. Deelman, T. Glatard, F. Desprez, D. Thain, B. Tovar, and M. Livny, "Toward Fine-Grained Online Task Characteristics Estimation in Scientific Workflows," in *8th Workshop On Workflows in Support of Large-Scale Science (WORKS)*, Denver, United States, 2013, p. to appear.

[29] S. Verboven, P. Hellinckx, F. Arickx, and J. Broeckhove, "Runtime Prediction Based Grid Scheduling of Parameter Sweep Jobs," in *2008 IEEE Asia-Pacific Services Computing Conference*. IEEE, Dec. 2008, pp. 33–38.

[30] J. Mościcki, M. Lamanna, M. Bubak, and P. Sloot, "Processing moldable tasks on the grid: Late job binding with lightweight user-level overlay," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 725–736, 2011.

[31] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, July 2002.

[32] N. Brook, A. Bogdanchikov, A. Buckley, J. Closier, U. Egede, M. Frank, D. Galli, M. Gandelman, V. Garonne, C. Gaspar, R. G. Diaz, K. Harrison, E. van Herwijnen, A. Khan, S. Klous, I. Korolko, G. Kuznetsov, F. Loverre, U. Marconi, J. P. Palacios, G. N. Patrick, A. Pickford, S. Ponce, V. Romanovski, J. J. Saborido, M. Schmelling, A. Soroko, A. Tsaregorodtsev, V. Vagnoni, and A. Washbrook, "DIRAC - Distributed Infrastructure with Remote Agent Control," June 2003.

[33] T. Maeno, "PanDA: distributed production and distributed analysis system for ATLAS," *Journal of Physics: Conference Series*, vol. 119, no. 6, p. 062036, July 2008.

[34] R. Viennet, C. Fonteix, and I. Marc, "Multicriteria optimization using a genetic algorithm for determining a Pareto set," *International Journal of Systems Science*, vol. 27, no. 2, pp. 255–260, Feb. 1996.

[35] D. Lingrand, J. Montagnat, and T. Glatard, "Modeling user submission strategies on production grids," in *International Symposium on High Performance Distributed Computing*, Munchen, Germany, June 2009.