KOALA-C: A Task Allocator for Integrated Multicluster and Multicloud Environments

Lipu Fei[†], Bogdan Ghiț[†], Alexandru Iosup[†], Dick Epema[†§]
[†]Delft University of Technology
[§]Eindhoven University of Technology
the Netherlands
{l.fei,b.i.ghit,a.iosup,d.h.j.epema}@tudelft.nl

***Abstract*—Companies, scientific communities, and individual scientists with varying requirements for their compute-intensive applications may want to use public Infrastructure-as-a-Service clouds to increase the capacity of the resources they have access to. To enable such access, resource managers that currently act as gateways to clusters may also do so for clouds, but for this they require new architectures and scheduling frameworks. In this paper, we present the design and implementation of KOALA-C, which is an extension of the KOALA multicluster scheduler to multicloud environments. KOALA-C enables uniform management across multicluster and multicloud environments by provisioning resources from both infrastructures and grouping them into clusters of resources called sites. KOALA-C incorporates a comprehensive list of policies for scheduling jobs across multiple (sets of) sites, including both traditional policies and two new policies inspired by the well-known TAGS task assignment policy in distributed-server systems. Finally, we evaluate KOALA-C through realistic simulations and real-world experiments, and show that the new architecture and in particular its new policies show promise in achieving good job slowdown with high resource utilization.**

I. Introduction

Companies and scientific communities with large compute-intensive workloads that are used to build and maintain their own hardware infrastructure, can now acquire seemingly unlimited computing power from public IaaS cloud providers, for example to compensate for transient shortages of computing resources in their own infrastructure. Such a resource usage pattern creates a new environment with heterogeneous resources including multiple clusters and multiple IaaS clouds (e.g., Amazon EC2 and private clouds), or an integrated multicluster and multicloud environment, to which traditional job scheduling and resource management has to be extended. In this paper we propose the KOALA-C resource manager along with several scheduling policies, and their analysis, for managing compute-intensive workloads in integrated multicluster and multicloud environments.

Integrating multicluster and multicluster resources raises several non-trivial challenges, which we address in this work. First, the resource provisioning interfaces and capabilities of clusters and commercial IaaS clouds may be very different, yet the integrated environment should for simplicity provide a uniform interface to its user. Multicluster resources do not need to limit the maximal allocation of resources, because the size of the physical infrastructure is limited anyway (otherwise, the use of cloud resources would not be needed at all), but cloud resources need to be provisioned under strict limits (otherwise, the user may unknowingly exceed the available budget). Among IaaS clouds, some may charge for the usage of resources (e,g., commercial public clouds such as Amazon EC2) whereas others may not (e.g., private clouds that use non-payment related accounting). For managing IaaS cloud resources, the many existing approaches [1]–[4] propose managing resources individually, creating virtual clusters by grouping and managing multiple resources for a longer period of time, etc. In this work, we employ a virtual cluster architecture that uniformly supports cluster and cloud resources.

Second, managing the integrated multicluster and multicluster resources requires a compliant scheduling policy framework. Various approaches to the design and evaluation of policies for multicluster systems (see [5], [6] and references within) and for IaaS clouds ([3], [4], [7]) have been investigated. In particular, two major types of policies, namely resource provisioning policies for making resources accessible only for as long as needed and job allocation policies for assigning jobs to accessible resources, must be supported by the scheduling policy framework. Although many scheduling policies have already been proposed, we show in this article that there is still much room for *innovation in scheduling architectures*.

Third, in using integrated multicluster and multicloud environments, we investigate the research question *how to manage the different resources efficiently?*, that is, how to provision resources and how to schedule jobs to them with good performance-cost tradeoffs? This essential scheduling problem is particularly challenging for the integrated multicluster and multicloud environment, where truly diverse jobs—short and long jobs, single-processor and highly parallel jobs, etc.—may share the same set of resources. Traditional approaches that allow jobs to share resources irrespective of the range of job runtimes have often been disadvantageous for short jobs, who may be slowed down [8] disproportionately while waiting for long jobs to complete. Slow down fairness can be achieved in offline scheduling but is expensive to compute even for single-cluster environments [9], and online fair scheduling remains challenging. Good results in online scheduling have been achieved [9]–[11] when the the jobs are split by the user into disjoint sets of short tasks, but this job model does not support the rigid parallel jobs present in scientific workloads and is thus unsuitable for our work. Isolating jobs of very different sizes (runtime, number of processors, etc.) on disjoint sets of resources has often been employed in large-scale systems, and in particular in supercomputing systems [12], with good results in alleviating the job slowdown problem, but also possibly leading to low resource utilization. In this work, we take this latter approach, and focus on a trade-off between partitioning

978-1-4799-5548-0/14/$31.00 ©2014 IEEE

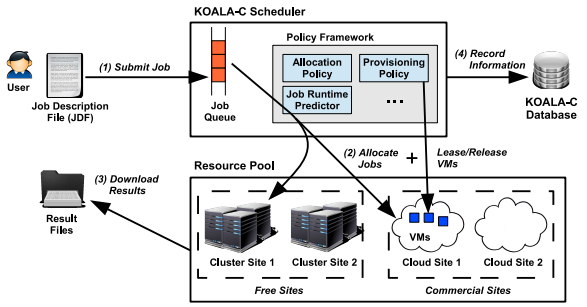


Fig. 1: The system architecture of KOALA-C.

and sharing of resources, with the aim to support diverse jobs.

Our main contribution in this work is threefold:

1) The design of KOALA-C, a resource management architecture for integrated multicluster and multicloud environments that uses the notion of *clusters of resources* or *sites* to achieve efficient system operation (Section III). Managing sites instead of single resources allows us to achieve a trade-off between quality of service for short jobs and utilization of resources, and enables a wide variety of scheduling policies.
2) To efficiently use complex sets of resources provisioned from both clusters and clouds, we design two new scheduling policies that configure sites to service jobs in specific ranges (Section IV). In addition, we adapt four traditional non-preemptive policies from single resources to sites.
3) A comprehensive experimental evaluation of our architecture and policies, through both trace-based simulations and real-world evaluation in the integrated environment consisting of our DAS-4 system [13] configured as a multicluster system or (on a subset of resources) as an OpenNebula private cloud, and Amazon EC2 (Section V). We use as baselines for comparison the four traditional policies we have adapted to sites.

II. System Model

In this section, we first describe our job model, and then, we propose a scheduling structure for executing sequential and parallel compute-intensive jobs on multiple clusters and clouds.

A. Job Model

In this paper, we only consider compute-intensive workloads consisting of sequential or parallel high-performance applications. The jobs running these applications are assumed to be rigid in that they do not change their sizes (in terms of the numbers of processors they use) during their runtimes. In the simulations they are represented simply by their size and runtime, in the real experiments they run actual MPI applications. We assume that a parallel job can use any subset of nodes of a single cluster or of a public or private cloud. Support for other types of compute-intensive applications such as Bags-of-Tasks and Workflows does exist in the original multicluster version of KOALA, but so far we have not extended it to clouds, which would be much of an engineering exercise. In some of our policies jobs may need to be preempted, but then they will be restarted rather than resumed from where they were stopped, so we don't require special support for preemption.

B. Resource Management Model

We assume that the integrated multicluster and multicloud system to be considered consists of sites representing separate clusters, private, or public clouds that each can be accessed separately. We treat these sites as being homogeneous and we don't distinguish resources based on their network location. Each cluster site is assumed to have its own Local Resource Manager (LRM) responsible for managing and allocating resources through which jobs can be submitted, either direcly by the user or by some higher-level resource manager. Similarly, clouds, whether private or public, can be accessed through the cloud API offered by the specific cloud manager installed.

The aim of this paper is to design and implement a Global Resource Manager (GRM) that interfaces with the LRMs of the cluster sites and with the cloud API of the cloud sites. The GRM we present in this paper is KOALA-C, which is an extension of KOALA [5]. One of the components we added to KOALA, which already did have an interface to SGE as the LRM of clusters, were interfaces to the OpenNebula and Amazon EC2 APIs for submitting jobs. All jobs are submitted by users to the GRM, which selects appriopriate sites for them to be executed. Every job is assumed to be able to be executed on all sites. The two main differences between clusters and cloud for the purposes of this paper are that the resources of clusters are "always present and accessible," while cloud resources first have to be leased before they can be allocated by KOALA-C, and that with clouds, a cost is associated, which we use as an additional metric in our experiments.

III. KOALA-C: A Multicluster, Multicloud Resource Management Architecture

In this section, we present the KOALA-C resource management system for integrated multicluster and multicloud environments. First, we explain the overall system architecture of KOALA-C, and then we present the scheduling policy framework created by its architecture.

A. System Architecture

The architecture of the KOALA-C scheduler, which is shown in Figure 1, is a combination of the common architectures used for multicluster and for (multi-)cloud scheduling. Steps 1–4 in Figure 1 are commonly employed in multicluster systems (e.g., in our KOALA [5] multicluster scheduler). Users submit jobs (step 1 in Figure 1) that are enqueued by the scheduler. Jobs are allocated and later executed on available resources through a *resource manager* and its scheduling framework, configured by policies (step 2). The results of job execution are returned to the user (step 3). Scheduler and system operations are logged for later accounting processes (step 4, concurrently with the other steps). From existing cloud schedulers, such as our earlier SkyMark [4], KOALA-C borrows

the common architecture of provisioning resources via leasing and releasing operations (step 2).

In addition to features borrowed from existing multicluster and multicloud architectures, KOALA-C has two distinctive features: uniform resource management across multicluster *and* multicloud resources leading to the single notion of *sites*, and a mechanism for configuring and managing *sets of sites*.

Uniform resource management across multicluster and multicloud resources: The resource manager of KOALA-C dynamically provisions heterogeneous resources from multicluster systems and from multiple cloud environments, and groups them into clusters of resources. To disambiguate them from physical clusters, from hereon they are referred to as *sites*.

We design sites to use a dynamic *virtual cluster structure* (VCS) architecture, which consist of two types of nodes: one head node and a number of worker nodes with local storage. The head node controls all the nodes in the VCS, and runs a daemon that can launch jobs for execution on the worker nodes. The head node can configure the VMs leased from clouds, so that user jobs can be executed on clouds without modification. This architecture allows sites to support uniformly resources from the two types of infrastructure, clusters and clouds, despite their different resource management. When a site is composed only from the resources of a cluster or of a cloud, we refer to it as a cluster site and a cloud site, respectively. The VCS is a simplified version of the generalized virtual cluster [14]–[16].

Management of sites: Given the complete set of sites a KOALA-C deployment has to work with, the KOALA-C resource manager can configure this set into multiple subsets of sites (*subsystems*) for use by scheduling policies. For instance, two disjoint sets of sites may service the jobs of different user communities; or one set may include all the (free) cluster sites in one set and another all the (pay-per-use) cloud sites. In the simplest configuration, there is only a single set containing all sites. We introduce for this case, in Section IV-A, four traditional policies that can be used in our scheduling framework. For multiple subsystems, we define in Section IV two new scheduling policies, and we conduct a comprehensive evaluation of these policies in Section V.

Architecturally, a key design choice for KOALA-C is that sites are grouped into subsystems that may *share* resources. Traditional, mutually exclusive partitioning [12] only allows sites to belong to a single subsystem. However, mutually exclusive partitioning can lead to low resource utilization and to high job slowdown, when large and small jobs coexist in the system [17]. In our design, we allow sites to belong to one or *several* subsystems, which enables resource multiplexing and promises to enable a trade-off between improved resource utilization and reduced job slowdown. We explore in Section IV-C the use of subsystems that share sites.

B. Scheduling Policy Framework

KOALA-C provides a *scheduling policy framework* (shown in Figure 1) , which consists of a module that configures and manages subsystems, a resource provisioning module for IaaS clouds, and, among the auxiliary modules, a predictor module. Using this framework, KOALA-C may dynamically change the amount and shares of resources given to each subsystem through specific resource provisioning policies, while at the same time reconfiguring the policies of each subsystem. The current version of KOALA-C provides a library of scheduling policies, which we describe in Section IV.

The key module of the KOALA-C policy framework is the module for the configuration and management of subsystems, which allows the system administrator to select, from an existing library, the policy that creates a *subsystem organization* (chain, hierarchy, etc.), and that allocates jobs dynamically to each and *between* the subsystems. This also enables the design and implementation of a wide variety of scheduling policies that dynamically decide on which resources each job will run, and in particular how to organize subsystems of multicuster and multicloud environments.

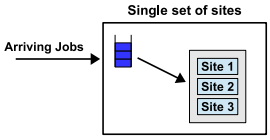
We further describe the resource provisioning and the predictor modules, in turn.

Resources from IaaS clouds have to be specifically provisioned, subject to cost and other restrictions addressed by a provisioning policy. Although KOALA-C can conceptually employ any of the many provisioning policies commonly used in clouds (see, for example, our recent exploration of 8 provisioning policies [4]), we use in this paper a single, traditional resource provisioning policy, On-Demand (OD, explored for example in [4]). The OD policy leases more VMs and assigns them to a cloud site to meet job demand, and releases idle VMs during off-peak hours. This policy treats differently clouds that provide resources for free (for example, private clouds) and clouds that charge for resources (for example, public commercial clouds such as Amazon EC2, which charges resource usage in hourly increments). On a cloud with free resource use, OD releases VMs after they have been idle for a certain amount of time. On a non-free cloud, OD releases idle VMs to match the charging model; for Amazon, resources are released when they are close to reaching another hour of their lifetime.

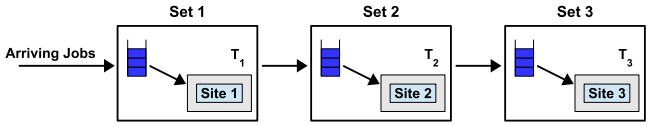
The auxiliary predictor module also reuses previous work in job runtime and wait time estimation. Two of the allocation policies we define in Section IV-A (SJF and HSDF) require job runtime estimation. Instead of relying on inaccurate estimations provided by users, we estimate the runtime of a newly arrived job as the mean of the runtimes of the last two finished jobs submitted by the same user [18]. Many other runtime estimation techniques have been explored in the past, but this simple method has been shown to give very good, and often even the best results [18], [19]. Similarly, we also use traditional wait-time estimation [19].

IV. Job Scheduling in Koala-C

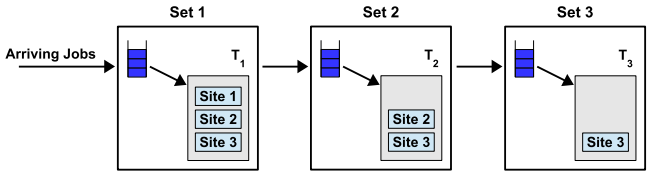
In this section, we propose a scheduling model for placing jobs across multiple sites managed by KOALA-C. Towards this end, we adapt four common scheduling policies for single sets of sites which service jobs irrespective their runtimes. Furthermore, we design two new policies for multiple sets of sites each of which is configured to service only jobs whose runtimes are in a specific range. The latter policies are non-trivial adaptations to our system model of the well-known TAGS task assignment policy in distributed server systems [20].



(a) The structure of the traditional scheduling policies with one set of sites.



(b) The structure of the TAGS-*chain* policy with a chain of three sites ($T_3 = \infty$).



(c) The structure of the TAGS-*sets* policy with three decreasing sets of sites with three, two, and one site, respectively ($T_3 = \infty$).

Fig. 2: The structure of the KOALA-C scheduling policies.

A. Traditional Scheduling Policies

We consider four common scheduling policies which we use as baselines in our experiments. All these policies assume a system consisting of a single set of sites that are ordered in some arbitrary way (see Figure 2a). Arriving jobs are entered into a global queue, and periodically (every 30 seconds in our experiments) the selected scheduling policy is invoked. On each site, jobs run to completion without preemption of any form. The four policies are:

1) *First-Fit* (FF): The FF policy uses a FCFS queue to dispatch every job to the first site in which it fits, i.e., that has sufficient resources to execute the job.
2) *Round-Robin* (RR): Similarly to FF, the RR policy services jobs in FCFS order and assigns them in a cyclical fashion to sites with enough resources.
3) *Shortest-Job-First* (SJF): The SJF policy uses job runtime predictions to prioritize the shortest job on the first site that has sufficient resources to run it. We also use a variation of this policy, SJF-*ideal*, which assumes perfect prior knowledge of the job runtimes.
4) *Highest-SlowDown-First* (HSDF): The HSDF policy is similar to SJF except that it also uses job wait-time prediction in order to allocate the job with the highest estimated slowdown first.

B. TAGS-*chain: Scheduling across a Chain of Sites*

The TAGS-*chain* policy considers a system to consist of a *chain* of sites each of which is associated with a *job runtime limit* that indicates the maximum amount of time jobs are allowed to run on the resources of the site (see Figure 2b). More generally, one might for this policy assume a system to consist of a chain of *sets* of sites, with runtime limits for complete sets, as we will have in the TAGS-*sets* policy (see Section IV-C). However, in this paper for TAGS-*chain* we restrict such sets to single sites. The runtime limits are increasing across the chain, and have to be set by the system administrator. Jobs submitted to the system are first appended to the queue of site 1, which has the lowest job runtime limit. Within each site, jobs are served in FCFS fashion, and run until completion or until exceeding the site's runtime limit. When a job reaches the runtime limit of site 1, it is killed and appended to the queue of site 2. When it is then scheduled, it will start from scratch again. This process continues until a job either completes, or until it reaches the last site, where it will be allowed to run to completion (that is, the job runtime limit of the last site is always set to ∞).

The characteristics of the TAGS-*chain* policy derive from the analytical study of the original TAGS policy [20] in a system with distributed servers and sequential jobs. In our system model, each site is dedicated to the jobs with a specific range of runtimes. As shown in previous work on TAGS, short jobs will not suffer from huge values of slowdowns due to the presence of very large jobs. As a consequence, TAGS-*chain* helps in achieving relatively uniform values of the job slowdown across jobs with varying runtimes.

In order to further distribute the load in a large system, we generalize the TAGS-*chain* policy to multiple chains of sites managed by KOALA-C, each identical in structure to a single chain as in Figure 2b. Then, when a job is submitted to KOALA-C, it is immediately dispatched to one of these chains selected either uniformly random (RND), or in round-robin fashion (RR). We explore the ability of the chain selection policy to extend the properties of TAGS in a single chain in Section V. Intuitively, TAGS-*chain* may lead to low resource utilization when the amounts of resources consumed by jobs in the different runtime ranges are imbalanced.

C. TAGS-*sets: Scheduling across Decreasing Sets of Sites*

To alleviate the loss of resource utilization that may be the result of using TAGS-*chain*, we design the policy TAGS-*sets* for integrated multicluster and multicloud environments. TAGS-*sets* still uses the main motivation of TAGS-*chain*, which is to let short jobs have short wait times and to let long jobs have long wait times by running each on specific subsets of resources. Although the structure of TAGS-*sets* resembles that of TAGS-*chain*, there are two important differences: in TAGS-*sets*, the sets of sites configured by KOALA-C consist of multiple sites, with runtime limits associated with complete sets, and the sets of sites are decreasing, with set 1 consisting of the complete system, and with each next set being a proper subset of the previous set. Similarly as with TAGS-*chain*, jobs upon arrival join the queue of set 1, the runtime limit increases from one set to the next, and jobs that are preempted and killed because they reach a runtime limit are started from scratch in the next set of sites. Within each set, jobs are serviced using the FF policy defined in Section IV-A. In Figure 2c, we show the structure of TAGS-*sets* with three sets of sites for a system with in total three sites. In this example, site 1 is dedicated to short jobs (site 1), site 2 runs short and intermediate-sized jobs, and site 3 is the only site to run long jobs.

TABLE I: Overview of the experiments, with in bold the distinguishing features of each experiment.

Section	Experiment Goal	Experiment Type	Used Policies	Used Environment
V-B	**Policy configuration**	Simulation	TAGS-*chain*, TAGS-*sets*	Multicluster
V-C1	**Policy comparison**	Simulation	Traditional, TAGS-*chain*	Multicluster
V-C2	Policy comparison	Simulation	Traditional, TAGS-*sets*	Multicluster and **Multicloud**
V-D	**Real-world vs. Simulation**	**Real-world** and Simulation	Traditional, TAGS-*sets*	Multicluster and Multicloud

TABLE II: Summary of the characteristics of the five PWA traces (top) and three GWA traces (bottom). Count (%) indicates the percentages of jobs after eliminating those larger than 32 nodes from the original trace. The Job Runtime statistics are only for the remaining jobs.

Trace	Jobs (Count, %)	Job Runtime (minutes)				
		mean	Q1	median	Q3	99th
KTH-SP2	$27K$ (96%)	149	1	14	164	1,323
CTC-SP2	$74K$ (94%)	184	2	16	223	1,081
SDSC-BLUE	$175K$ (76%)	46	1	2	14	961
SDSC-SP2	$63K$ (94%)	96	1	8	42	1,068
DAS2-fs3	$65K$ (99%)	12	0	1	1	92
AuverGrid	$50K$ (100%)	218	1	17	123	1,645
LCG	$50K$ (100%)	113	2	4	27	2,901
NorduGrid	$50K$ (100%)	4,315	32	1,403	7,720	17,836

There is one subtle issue in scheduling jobs in TAGS-*sets*. One may question the benefit of killing a job in a certain set that already runs on a site that is also part of the next set (e.g., an intermediate-sized job running on site 2 in set 1), and later restarting it on that same site as part of the next set (e.g., restarting the job on site 2 in set 2). The reason is that whenever the KOALA-C scheduler is invoked, it scans the queues of the sets of sites for jobs to be scheduled in increasing order, starting at the queue of set 1. Thus, priority is given to shorter jobs over longer jobs. In the given example, after the job is killed on site 2 in set 1, first jobs in the queue of set 1 get a chance to be scheduled before the very same job will be rescheduled and may restart on site 2.

We conduct an extensive experimental analysis of TAGS-*sets* in Section V. Intuitively, TAGS-*sets* is very useful for a system that requires a differentiation between two job classes, i.e., short and long jobs. For this situation, TAGS-*sets* may improve the overall utilization when there are enough small jobs (because the small jobs can be assigned to any sites) without making small jobs wait unnecessarily for long jobs (because at least some subset of sites is exclusively used by short jobs).

V. Experimental Results

We perform both simulations and real-world experiments to analyze the architecture and the policies presented in this work. In this section, we first describe the experimental setup, then present the simulation results, and finally the real-world results. In Table I, we provide an overview of our experiments with their goals and their distinguishing features.

A. Experimental Setup

We use eight real-world traces of scientific workloads, five from the Parallel Workloads Archive (PWA[1]) and three from the Grid Workloads Archive (GWA) [21]. From the PWA traces, we use the complete trace, while from the GWA traces, we use the first 50,000 jobs. Given the system configurations we use in our experiments, we limit the size of jobs to 32 nodes and omit larger jobs from the traces. We summarize the properties of these traces in Table II. The numbers in parentheses are the percentages of jobs remaining from the original traces after removing the jobs of size exceeding 32. For each trace, we show the average job runtime along with the job runtime at different percentiles.

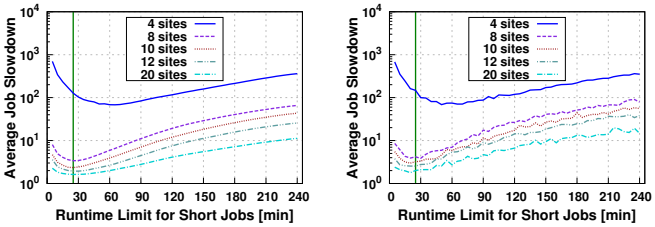
We simulate two environments, a multicluster-only environment for evaluating TAGS-*chain* and an integrated multicluster and multicloud environment for evaluating TAGS-*sets*. The two setups are summarized as follows:

1) *Multicluster Environment*: We set up different numbers (1 to 10) of identical chains of sites, with two sites of 32 nodes each, and the same runtime limit for short jobs. In this environment, we evaluate six job allocation policies: FF, RR, SJF, SJF-*ideal*, HSDF, and TAGS-*chain*. We use the RND and RR policies as dispatching policies across the different chains of sites.
2) *Multicluster and Multicloud Environment*: We configure five sites: three cluster sites, a private cloud site, and a public cloud site. Each cluster site has 32 nodes. The private cloud site has up to 64 VMs and there are always at least 8 VMs running on it. The public cloud has up to 128 VMs and no VMs are maintained to perform quick service. Cost is only charged for public cloud usage, using the same pricing policy as Amazon EC2, which is \$0.065/VM-hour. In this environment, we evaluate the same job allocation policies as in the multicluster environment, but with TAGS-*chain* replaced by TAGS-*sets*.

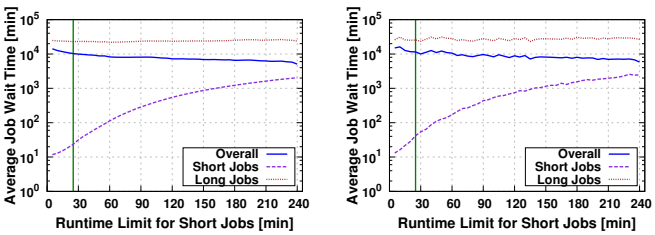
In the real-world experiment with KOALA-C, we use two cluster sites (called fs1 and fs2) of the DAS-4 system, each configured with 32 nodes, an OpenNebula-based private cloud of DAS-4 with up to 32 VMs, and Amazon EC2 as public cloud with up to 64 VMs. The workload we use here is a part of the CTC-SP2 workload of approximately 12 hours that imposes an average utilization of 70% on the system computed using the maximum cloud sizes. In this case we evaluate three policies: FF, SJF, and TAGS-*sets*. For the latter policy, set 1 of sites for short jobs consists of all four sites, and set 2 for short and long jobs consists of three sites, omitting fs1. The runtime limit of short jobs is set to 10 minutes.

The metrics we use are the average job slowdown, the average job wait time, and total cost of using the public cloud. In the real-world experiment, we also measure the makespan and the utilization.

[1] http://www.cs.huji.ac.il/labs/parallel/workload/logs.html

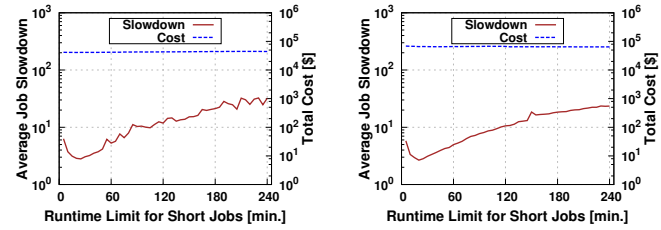


(a) Slowdown vs. Runtime (RR) (b) Slowdown vs. Runtime (RND)

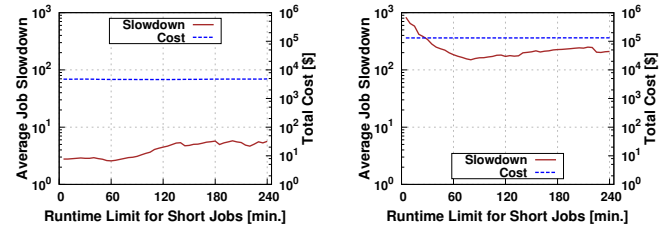


(c) Wait Time vs. Runtime (RR) (d) Wait Time vs. Runtime (RND)

Fig. 3: TAGS-*chain*: The average job slowdown and the average job wait time vs. the runtime limit for short jobs for different numbers of chains of sites (a,b) and for four chains of sites (c,d), each having two sites, for the RR and RND policies for dispatching jobs across chains of sites. Logarithmic scale on vertical axes.



(a) CTC-SP2 (b) SDSC-BLUE



(c) AuverGrid (d) NorduGrid

Fig. 4: TAGS-*sets*: The average job slowdown and the total cost vs. the runtime limit for short jobs for CTC-SP2, SDSC-BLUE, AuverGrid, and NorduGrid. Logarithmic scale on vertical axes.

B. *Configuration of* TAGS-*chain and* TAGS-*sets*

We first investigate how to configure the TAGS-*chain* policy when there are only two classes of jobs, short and long, i.e., to what value to set the runtime limit of short jobs in order to achieve the lowest average job slowdown. To this end, we simulate the scenarios of the multicluster environment detailed above, with the runtime limit for short jobs enforced on one site and the other site having no such limit.

In Figure 3, we show the results of the PWA trace KTH-SP2. From Figure 3a, we can see that using RR job dispatching, the lowest average job slowdown is achieved when we set the runtime limit for short jobs to 25 minutes. RND shows a similar pattern, but with more variance. However, for all the traces we evaluated, there is no common runtime limit to achieve the lowest slowdown in all cases, and in Table III, we list the setup for each trace. We also show the average wait times of short jobs, of long jobs, and of all jobs with four chains of sites in Figures 3c and 3d, and we find that TAGS-*chain* indeed gives short jobs shorter wait times in order to reduce the average job slowdown.

Similar to TAGS-*chain*, we investigate the configuration for TAGS-*sets*. We let short jobs use all five sites and we let long jobs use three of them (one cluster and both clouds). In Figure 4, we show the average job slowdown and the total cost of using the public cloud versus the runtime limit of short jobs. We find that for TAGS-*sets*, changing the runtime limit for short jobs has little impact on the average slowdown, but the total cost may vary significantly. But overall, we find that in most cases, a runtime limit of 40 minutes is suitable for most traces, and we use this value to evaluate all policies in the multicluster and multicloud environment.

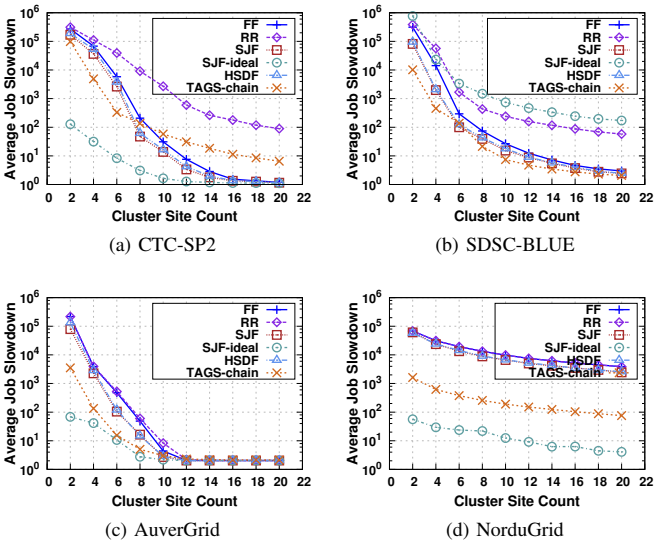
C. *Results for Trace-Based Simulations*

1) Multicluster Environment: In Figure 5, we show the average job slowdown and the average job wait time of all policies for the four traces CTC-SP2, SDSC-BLUE, AuverGrid, and NorduGrid versus the number of cluster sites. We find that RR almost always has the highest job slowdown, and that in most cases, heuristic policies such as SJF and HSDF are better than the policies with no prior knowledge such as FF and RR. TAGS-*chain* outperforms the other policies when the system is small, and in the best case in NorduGrid, it lowers the slowdown more than 25 times against the other policies when the system size is small. The reason is that in NorduGrid SJF and HSDF perform poorly because they extremely overestimate (underestimate) the runtimes of short (long) jobs, while TAGS-*chain* yields wait times that are better in proportion with job runtimes.

2) Multicluster and Multicloud Environment: In Figure 6, we show the average job slowdown and the total cost of all policies for all traces. We find that RR is by far the worst policy in terms of performance, followed by FF. Simple heuristic

TABLE III: TAGS-*chain*: The optimal runtime limit of short jobs for all traces (rounded to multiples of 5 minutes).

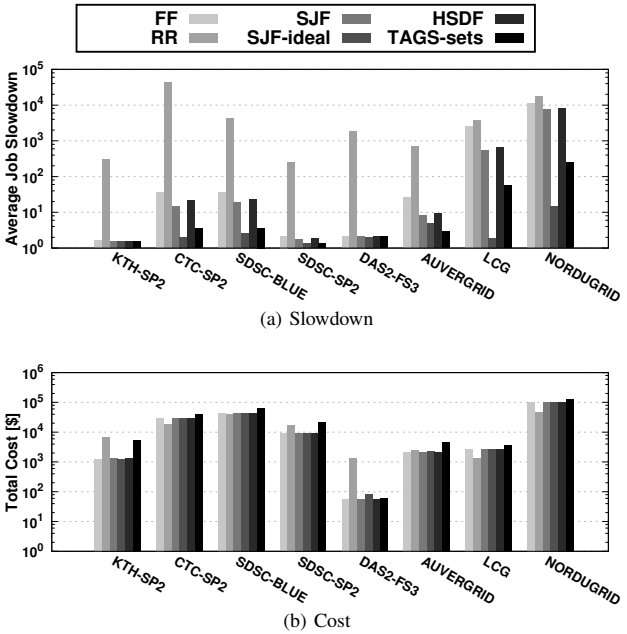
Trace	Limit [min]	Short Jobs
KTH-SP2	25	55%
CTC-SP2	125	69%
SDSC-BLUE	55	87%
SDSC-SP2	40	75%
DAS2-FS3	25	98%
AuverGrid	60	67%
LCG	20	69%
NorduGrid	120	32%



(a) CTC-SP2 (b) SDSC-BLUE

(c) AuverGrid (d) NorduGrid

Fig. 5: The average job slowdown of all policies for CTC-SP2, SDSC-BLUE, AuverGrid, and NorduGrid. Logarithmic scale on vertical axes.



(a) Slowdown

(b) Cost

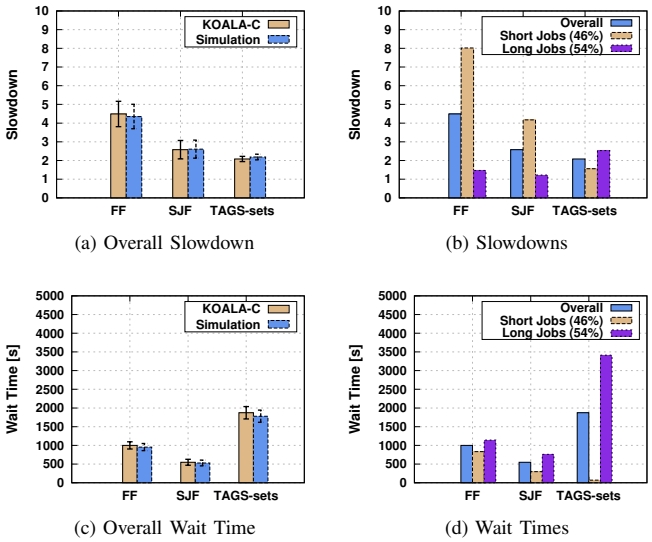
Fig. 6: The average job slowdown (a) and the total cost (b) for all policies for all traces. Logarithmic scale on vertical axes.

policies such as SJF and HSDF never perform worse than FF and RR. However, in most cases, TAGS-*sets* has the best performance except for SJF-*ideal*, and noticeably, in LCG and NorduGrid, TAGS-*sets* improves the slowdown against SJF over 10 times and more than 25 times, respectively. From Figure 6b, we find that TAGS-*sets* costs more than most of the other policies, by a factor ranging from 1.3 to 4.2 times, but if we take into account the significant performance improvement TAGS-*sets* achieves, the cost increment is acceptable. Therefore, we conclude that TAGS-*sets* can achieve a good performance-cost trade-off in an integrated multicluster and multicloud environment.

D. Results for Real-World Experiments

In this section we present the experiments performed with KOALA-C in the integrated multicluster and multicloud environment as explained in Section V-A. In Figure 7, we show the average overall job slowdown and job wait time of the three policies FF, SJF, and TAGS-*sets* with both real-world experiments using KOALA-C and simulations, and the same metrics for short and long jobs separately (and again all jobs together) for the real experiments only. The error bars are calculated using the standard error of the mean defined as $\sigma/\sqrt{n}$, where σ is the standard deviation and n is the sample size. In all of our experiments, KOALA-C completes all jobs correctly, thus, we conclude that KOALA-C operates correctly in a multicluster and multicloud environment.

The results of the real experiments and the simulations in Figure 7 are remarkably close. We find that TAGS-*sets* has the lowest job slowdown compared to FF and SJF, but the highest average job wait time. Figures 7b and 7d show the reason for this phenomenon: TAGS-*sets* succeeds in giving short jobs short wait times and long jobs long wait times, exactly what it was designed for!

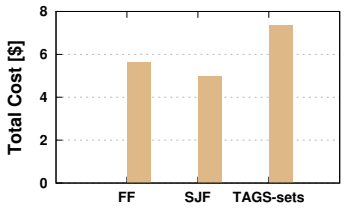


(a) Overall Slowdown (b) Slowdowns

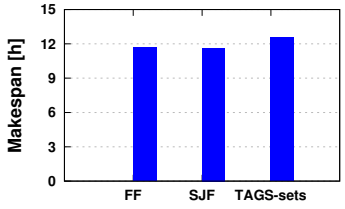
(c) Overall Wait Time (d) Wait Times

Fig. 7: The average job slowdown for all jobs (a) and per job type (b) in the real experiments with KOALA-C (a,b) and in the simulations (a), and the average wait time for all jobs (c) and per job type (d) in the real experiments with KOALA-C (c,d) and in the simulations (d). The fractions of short and long jobs are shown in parentheses.

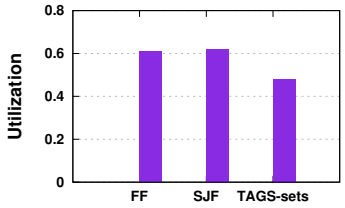
In Figure 8 we show, the cost, the makespan, and the utilization of the three policies. We find that similar to the results in the simulations, TAGS-*sets* has a higher cost. However, this is because we let long jobs use the public



(a) Cost



(b) Makespan



(c) Utilization

Fig. 8: Real experiments with KOALA-C: The cost, the makespan, and the utilization of FF, SJF, and TAGS-*sets*.

cloud, which can eventually incur higher cost. Thus, with a different configuration, the result can be different. We also find that the makespan of TAGS-*sets* is higher than FF and SJF, and its overall utilization is lower. This is also caused by its preemptive nature, and moreover, long jobs have to race for limited resources (three sites) while privileged short jobs can use all resources.

In Figure 9, we show the fractions of jobs completed on each of the four sites, which for FF and SJF are ordered as fs1, fs2, ONE, and Amazon EC2, used in the real experiments. FF and SJF have similars pattern in site usage, and they both use the cluster site fs1 the most. However, TAGS-*sets* balances the load much better across the cluster sites fs1 and fs2. SJF is basically FF with a preference for jobs with shorter runtimes (not necessarily below the runtime limit of TAGS-*sets*), and this is reflected in a somewhat higher completion rate on fs1 and a much higher rate on fs2. TAGS-*sets* schedules short jobs (runtime 10 minutes) across all four sites, but as on fs1 there are no competing long jobs, the majority of short jobs (41%, out of a total of 46%) are completed on fs1; the decreasing number of completions on fs1, fs2, ONE, and EC2, reflects well TAGS-*sets*'s preference in using first local, then free private cloud, then paid public cloud resources, respectively.

VI. Related Work

In this section, we summarize the related work from two aspects: the systems and approaches for cluster or/and cloud

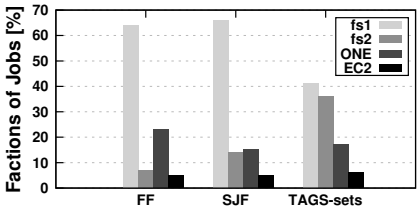


Fig. 9: Real experiments with KOALA-C: The fractions of jobs completed on each site with FF, SJF, and TAGS-*sets* (fs1 and fs2 are two DAS-4 cluster sites, ONE is the OpenNebula cloud of DAS-4, and EC2 is Amazon EC2).

environments and the policies of job allocation and resources provisioning. Our work differs fundamentally from all related work, through our novel architecture and our new subsystem-aware policies. Moreover, we conduct both real-world and simulated (long-term) experiments, which only a few of the related studies do.

Much related work in this field reveals different approaches for cluster or/and cloud environments. An early work of Marshall et al. [1] presents a way of using IaaS clouds as elastic sites to expand the capacity of existing clusters. They use a similar virtual cluster approach on Amazon EC2 with Torque as the resource manager. Mesos [22] provides a platform which shares clusters between various application frameworks, such as MPI and Hadoop. Its scheduling mechanism is based on resource offers and fairly assigns resources to each framework. The framework can either accept or reject the resources based on their own policies. The resource isolation of Mesos gives high resource utilization and efficiency. Wrangler [2] is an automated deployment system for IaaS clouds. The system aims to solve the problem that the VMs on IaaS clouds do not provide any resource management functionality once that are provisioned. Wrangler presents a way to automatically provisioning virtual clusters with Condor as the underlying resource manager for users. Kim et al. [23] present a system that explores the efficiency of scientific workflow on hybrid environments with grid and cloud.

Several previous studies [3], [4], [7] show various approaches in policy design and performance evaluation on IaaS clouds. Chard et al. [24] present several economic resource allocation strategies for grid and cloud systems. Deng et al. [25] introduce a portfolio scheduler for IaaS clouds and evaluate policies through simulation.

VII. Conclusion

Extending existing (multi-)cluster resource managers with the ability to manage resources provisioned from IaaS clouds would be beneficial for various companies, research labs, and even individuals with high computational demands. Towards this goal, in this work we have designed, implemented, deployed, and evaluated KOALA-C, a task scheduler for integrated multicluster and multicloud environments.

KOALA-C is designed to support in a uniform way resources from clusters and clouds, which it can integrate into virtual clusters. The KOALA-C architecture includes a component for managing sets of sites and a scheduling policy

framework. We have equipped KOALA-C with a library of traditional scheduling policies for single-set system structures, and with two novel scheduling policies for multi-set structures with runtime limits aiming at uniform job slowdowns. Last, we have shown through realistic simulations and real-world experiments that the two new policies can achieve significant improvements over the other policies we have evaluated with respect to job slowdown and resource utilization.

For the future, we plan to extend the set of policies provided with KOALA-C with more job dispatching methods, and to adapt the concept of Portfolio Scheduling [25] to multicluster-and-multicloud evironments.

REFERENCES

[1] P. Marshall, K. Keahey, and T. Freeman, "Elastic Site: Using Clouds to Elastically Extend Site Resources," 2010.

[2] G. Juve and E. Deelman, "Automating Application Deployment in Infrastructure Clouds," *IEEE CloudCom*, 2011.

[3] M. D. De Assunção, A. Di Costanzo, and R. Buyya, "Evaluating the Cost-Benefit of Using Cloud Computing to Extend the Capacity of Clusters," *HPDC*, 2009.

[4] D. Villegas, A. Antoniou, S. M. Sadjadi, and A. Iosup, "An Analysis of Provisioning and Allocation Policies for Infrastructure-as-a-Service Clouds," *CCGrid*, 2012.

[5] H. Mohamed and D. Epema, "Koala: A Co-allocating Grid Scheduler," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 16, 2008.

[6] A. Iosup, T. Tannenbaum, M. Farrellee, D. Epema, and M. Livny, "Inter-Operating Grids through Delegated Matchmaking," *Scientific Programming*, vol. 16, no. 2, 2008.

[7] S. Genaud and J. Gossa, "Cost-Wait Trade-Offs in Client-Side Resource Provisioning with Elastic Clouds," *IEEE Cloud*, 2011.

[8] D. G. Feitelson and L. Rudolph, "Metrics and benchmarking for parallel job scheduling," in *JSSPP*, 1998, pp. 1–24.

[9] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Choosy: Max-min fair sharing for datacenter jobs with constraints," in *EuroSys*, 2013, pp. 365–378.

[10] A. Iosup, O. O. Sonmez, S. Anoep, and D. H. J. Epema, "The performance of bags-of-tasks in large-scale distributed systems," in *HPDC*, 2008, pp. 97–108.

[11] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *EuroSys*, 2010, pp. 265–278.

[12] W. Cirne and F. Berman, "Adaptive Selection of Partition Size for Supercomputer Requests," *JSSPP*, 2000.

[13] "The Distributed ASCI Supercomputer 4," http://www.cs.vu.nl/das4.

[14] K. Govil, D. Teodosiu, Y. Huang, and M. Rosenblum, "Cellular Disco: Resource Management Using Virtual Clusters on Shared-Memory Multiprocessors," *OSR*, vol. 33, no. 5, 1999.

[15] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. E. Sprenkle, "Dynamic Virtual Clusters in a Grid Site Manager," *HPDC*, 2003.

[16] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, and X. Zhang, "Virtual Custers for Grid Communities," *CCGrid*, 2006.

[17] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *JSSPP*, 1997, pp. 1–34.

[18] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling using System-Generated Predictions rather than User Runtime Estimates," *IEEE TPDS*, vol. 18, no. 6, 2007.

[19] O. Sonmez, N. Yigitbasi, A. Iosup, and D. Epema, "Trace-based Evaluation of Job Runtime and Queue Wait Time Predictions in Grids," *HPDC*, 2009.

[20] M. Harchol-Balter, "Task Assignment with Unknown Duration," *ICDCS*, 2000.

[21] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. Epema, "The Grid Workloads Archive," *FGCS*, vol. 24, no. 7, 2008.

[22] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A Platform for Fine-grained Resource Sharing in the Data Center," *NSDI*, 2011.

[23] H. Kim, Y. el Khamra, S. Jha, and M. Parashar, "Exploring Application and Infrastructure Adaptation on Hybrid Grid-Cloud Infrastructure," *HPDC*, 2010.

[24] K. Chard, K. Bubendorfer, and P. Komisarczuk, "High Occupancy Resource Allocation for Grid and Cloud Systems, a Study with DRIVE," *HPDC*, 2010.

[25] K. Deng, R. Verboon, and A. Iosup, "A Periodic Portfolio Scheduler for Scientific Computing in the Data Center," *JSSPP*, 2013.