DOI 10.12694/scpe.v16i3.1103
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 16, Number 3, pp. 303–319. http://www.scpe.org

EARLY PREDICTION OF THE COST OF CLOUD USAGE FOR HPC APPLICATIONS*

MASSIMILIANO RAK†, MAURO TURTUR, AND UMBERTO VILLANO‡

Abstract. After a decade of diffusion, cloud computing has received wide acceptance, but it is not yet attractive for the HPC community. Clouds could be a cost-effective alternative to clusters and supercomputers, providing economy of scale, elasticity, flexibility, and easy customization. Unfortunately, most clouds are optimized for running business applications, not for HPC. However, they can be profitably used to run small-scale parallelism codes.

This paper presents a framework built on the top of a cloud-aware programming platform (mOSAIC) for the development of bag-of-tasks scientific applications. The framework integrates a cloud-based simulation environment able to predict the behavior of the developed applications. Simulations enable the developer to predict at an early development stage performance and cloud resource usage, and so the infrastructure lease cost on a public cloud.

The paper sketches the framework organization and presents the approach followed for the performance simulation of applications, focusing on a software development methodology that hinges on early performance prediction. After showing the results of some validation tests of simulation accuracy, an example of early performance prediction is presented.

Key words: cloud computing, HPC, performance prediction, simulation

AMS subject classifications. 68M14, 68W15

1. Introduction.

After about a decade of fast and widespread diffusion, cloud computing has received wide acceptance, standing as the new information technology platform. However, it is a fact that cloud computing is not yet attractive for the HPC (High Performance Computing) community, leaving the great potential of this paradigm partly underutilized.

At least in theory, clouds could be profitably used to bring the power of economic and scalable parallel computing to the masses, as a cost-effective alternative to clusters and supercomputers. IaaS (Infrastructure as a Service) clouds easily enable users to lease a set of nodes and set up a virtual cluster. Cloud advantages include economy of scale, elasticity, flexibility, and easy customization by exploiting the virtualization of cloud resources. In particular, HPC on clouds appears to be particularly appealing for the users who cannot afford to buy dedicated HPC infrastructures, or have only sporadic parallel computing demands.

However, most clouds (whether public or private) are optimized for running business applications, not for HPC. The biggest obstacles for efficient execution of HPC applications in clouds are the performance losses due to the systematic use of virtualization and, above all, to the use of networks designed mainly for scalability, and not for performance [5]. Moreover, multitenancy, the presence of loads hidden from the user view and control and HPC-agnostic cloud schedulers induce substantial variability of performance from one run to another [20], precluding the possibility to perform fine-grained software optimization. Studies and performance measurements made in the last years [26, 31, 40, 50] have diffused the pessimistic opinion that present-day clouds are simply unfit for the majority of HPC applications.

Simply stated, the problem is that today HPC and clouds are agnostic about each other [23]. There are essentially two options: or i) to re-design both clouds and HPC applications, so as to enable efficient execution of HPC codes in the cloud [24] , or ii) to deploy on the cloud only applications that are relatively unaffected by all the above-mentioned performance inhibitors. These applications include most small-scale parallelism codes, which can be run effectively on tiny virtual clusters, as well as embarrassingly parallel and tree-structured computations, which can exploit a high number of virtual processors without significant performance inefficiency.

This paper moves a step in the second direction. There is a wide range of applications widely used in science, engineering and for commercial purposes that have highly variable response times, are moderately CPU

*This paper is an extended version of the paper "Early Prediction of the Cost of HPC Application Execution in the Cloud", presented at the MICAS 2014 Workshop, 22-23 September 2014, West University of Timisoara, Romania.

†Department of Industrial and Information Engineering, Second University of Naples, Aversa, Italia, (massiliano.rak@unina2.it)

‡Department of Engineering, University of Sannio, Benevento, Italia, (mauro.turtur@unisannio.it, villano@unisannio.it).

intensive, are not fit (unless heavily redesigned) for GPU computing and are made up of loosely coupled tasks, so that computation easily dwarfs communication times. We think that this class of "para-scientific" applications is an almost ideal candidate for execution on the cloud. The major advantage is economic: the cost for leasing a small set of virtual cores can be very low, especially if there are relaxed time constraints for obtaining the results. A wise choice among provider offerings often allows to acquire the computing resources needed at very low cost (see for example the EC2 Spot Instances offer [2]). This enables any organization to run parallel code whenever needed, at a low cost, without investing the capital in rapidly obsolescing parallel hardware. The second important issue is cloud elasticity, which allows to scale in/out the number of virtual cores *on-the-fly* (i.e., while the application is running), based on the particular job requirements, paying just for the resources actually used. In other words, cloud computing is *also* a great opportunity for everyone to experiment and to exploit parallel computing at low cost, using a comfortable pay-as-you-go model.

We think that the final step to make clouds fully advantageous for sporadic scientific users is providing simple tools to predict the performance behavior of their application, allowing them to make a tradeoff between performance and leasing costs. In a previous paper [15] we proposed the use of a cloud-enabled programming platform. This platform makes it possible to develop cloud applications on the top of a cloud-aware programming framework (mOSAIC [42, 43]) by exploiting the bag-of-tasks programming paradigm. The bag-of-tasks (BOT) paradigm, also known as master-worker, processor farm, etc., is widely understood, and ubiquitous in small and medium-scale scientific computing.

Moreover, in the past we worked on the performance prediction of cloud applications developed on the top of the mOSAIC framework [14]. Already-available tools enable us to predict the performance of a cloud application without running it on (payed) cloud resources. In this paper, we discuss the enrichment of the bag-of-task framework with performance prediction capabilities, allowing the automatic generation of the application simulation models.

The remainder of this paper is structured as follows. In the next section we will examine related work. Section 3 illustrates the rationale and the architecture of the framework we have implemented for the development of bag-of-tasks applications in the cloud. Section 4 presents our approach to early performance prediction and Section 6 shows some of our simulation validation tests. Then an example of the use of early prediction to estimate response times and associated cloud leasing costs for a huge-sized problem is presented in Section 7. The paper closes with our conclusions and plans for future research.

2. Related Work.

2.1. HPC in the Cloud. The potential of clouds for scientific computing linked to economicity and to on-demand provision is discussed in [33]. As mentioned in the introduction, in the last few years there have been many works studying the suitability of clouds for HPC computations [11, 18, 27, 28, 29, 30, 36, 37, 50], almost all pointing out the performance losses of at least one of order of magnitude as compared to HPC clusters and the extreme variability of response times. The use of clouds only for particular classes of HPC loads is suggested in [26, 31]. Reference [23] and companion papers [24, 25, 26] suggest the design of HPC-aware clouds. The seminal paper dealing with cloud benchmarking is [49]. More recent results, showing the advances in performance of commercial clouds in the last few years, can be found in [20, 40]. Paper [47] tackles the interesting problem of the optimal use of the low-cost AWS *spot instances*.

In [35], the applicability of cloud platforms, and in particular of Microsoft Azure, to scientific computing is studied by implementing a well-known bioinformatics algorithm (BLAST). An implementation of BOT similar to the one presented in this paper, although with a few significant differences, is presented in [4]. A Java framework for the development of fault-tolerant applications is proposed in [38].

A few papers discuss how to exploit the intrinsic elasticity of clouds, i.e., the ability to increase or decrease the amount of computing resources used for application execution. In [19], the Authors present Cloudine, a platform for the development of generic scientific applications able to exploit at best cloud elasticity. The paper [45] tackles the problem of adding elasticity to existing MPI codes. This is obtained by terminating the execution and restarting the program on a different amount of resources, scaling up/down the number of computing nodes used. The execution of MPI codes over a cloud-aware communication library is discussed in [21], where CMPI, a novel MPI library based on the cloud-oriented communication optimizations proposed in [22], is presented.

2.2. Simulation and Performance Prediction. The core of our proposal is the use of a simulation-based approach for application performance prediction. The use of simulation in the HPC context is widely discussed in a number of papers, as [7, 17]. Recent efforts in this field are documented in [3] and [10].

For Component-Based Software Systems (CBSS), most approaches available in the literature focus on integrating performance prediction and evaluation techniques at design time, i.e., when the system implementation is not available. An exception, which has some similarities with our approach, is the COMPAS system [39]. The paper [32] presents a complete survey of performance strategies for CBSS.

CloudCMP [34] is one of few examples of simulation-based performance predictions of cloud applications, where the goal is to compare different cloud providers and to select a suitable one. Another notable work in this area is CloudSim [9], which targets the simulation of the entire stack of software/hardware components in a cloud infrastructure.

3. The Framework for Science Applications. The solution we proposed in [15] for BOT cloud application development requires several assumptions that span the various steps of the scientific application life-cycle:

- **Development**: the developer of the application provides only the basic sequential code blocks implementing the chosen algorithm. He should not care about communication/synchronization details, but only take into account how data are organized and elaborated.
- **Deployment**: the developer/user should be able to start the application over the cloud, choosing the amount of resources to be used and possibly scaling dynamically them up/down at run time. Fault tolerance is guaranteed by the development framework, and is completely hidden at code level.
- **Execution**: the developer/user submits multiple job to the application, which performs always the same actions over different data.

Our BOT development framework implements the simple and common *split-work-merge* solution pattern. A problem to be solved is split in sub-problems (*tasks*), and handed out to task solvers (*workers*), whose partial results are finally merged (see Figure 3.1). The resulting workflow could be applied also in the context of the map-reduce programming model [16], which involves a similar split-work-merge sequence. The difference is in the timing, as the workers in a bag-of-tasks are not constrained to proceed in lock-step, and can work on sub-jobs asynchronously among them. Bag-of-tasks applications can be developed by extending the above described components with problem specific-algorithms. The details of the development framework are presented in [15].

The BOT development framework provides all the needed components (*splitter*, *merger*, *worker* and *orchestrator*) and an API that can be used to integrate the user-supplied application code. In fact, all the supplied components are mOSAIC components. mOSAIC is a cloudware that builds up a Platform-as-a-Service on the top of computing resources leased in Infrastructure-as-a-Service mode from a single or even multiple cloud providers [42]. The mOSAIC platform offers an easy way to package and to deploy automatically in the cloud software components. Through the platform interface it is possible to deploy multiple instances of the same component and to restart them, in the case of a failure.

mOSAIC offers a set of already-developed basic components, which can be easily extended by the developer (as we have done in the case of the BOT framework). Among the standard components, `Queues` and `KVstores` play the most important roles.

The mOSAIC `Queue` component is a customized version of the RabbitMQ queue server [48]. It is a software component that offers an API to create messages queues, which can used by the applications to communicate each other. After that a queue has been created, all connected applications can send a message through it. All applications registered as consumers will be able to receive the message.

The `KVstore` component is based on Riak [1]. It offers a persistent NoSQL storage service to cloud applications. Computation and communication components can store data in the shared KVstore components, and retrieve them by a key. For example, the HTTPgw can use the KVstore to store HTTP messages that have a large body; "computing" mOSAIC components (*cloudlets*) can store in a KVstore the results of their elaboration, in order to make them be accessible to the external interface.

4. Performance prediction of bag-of-tasks applications. In the previous section we have described our bag-of-tasks framework for the development of scientific applications in the cloud. A key point is that such applications are fully defined by the number of instances of the mOSAIC components mentioned above and by

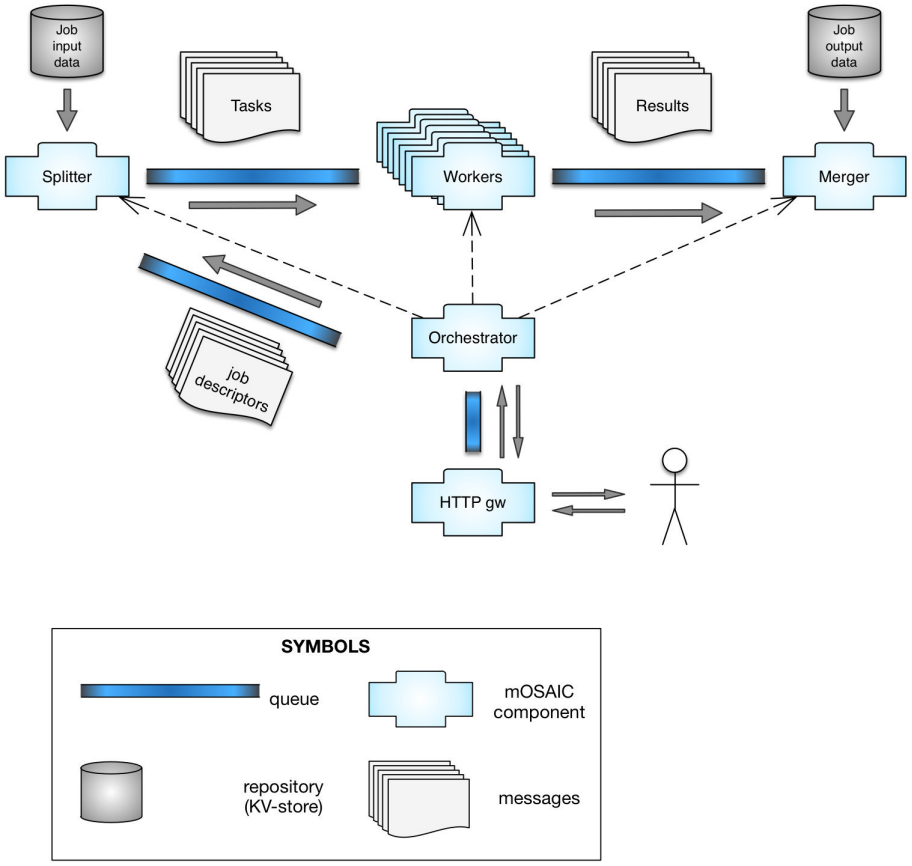

FIG. 3.1. *Architecture of the BOT framework*

their interconnections. Moving from these premises, we have devised a performance model of the application that can be used to predict its behavior and to tune its performance.

Our performance model is process-based [12, 41, 46], in that it is described through a set of discrete-event simulation components whose temporal behavior is described as a process. Event management and discrete-event actions/reactions are modeled in terms of process synchronization primitives. The simulated components have been developed by exploiting the JADES simulation library [12], which allows the description of process-oriented simulations in Java.

A noteworthy feature of the solution devised is that the simulation models expressed through the JADES library can be easily evaluated through the mJADES platform [13]. mJADES is a recently-developed system that supports the distribution of multiple JADES simulations on cloud resources. The mJADES simulation system is based on a Java-based modular architecture. The mJADES *simulation manager* produces simulation tasks from simulation jobs, and schedules them to be executed concurrently on multiple instances of the *simulation core*. This is a process-oriented discrete-event simulation engine based on the JADES simulation library. The outputs from the runs are handed on to a *simulation analyzer*, whose task is to compute aggregates and to generate reports for the final user. mJADES has been developed as a collection of mOSAIC components, and so the evaluation of the models can be performed on any mOSAIC platform. This could be the one where the application under study is deployed and executed, or even a different one.

4.1. Bag-of-tasks Simulation Model. To model a bag-of tasks application, we developed a simulation component for each of the core components making up the bag-of-task framework (`Splitter`, `Worker` and `Merger`), and for each component devoted to manage the cloud application execution (`HTTPgw`, `Orchestrator`),

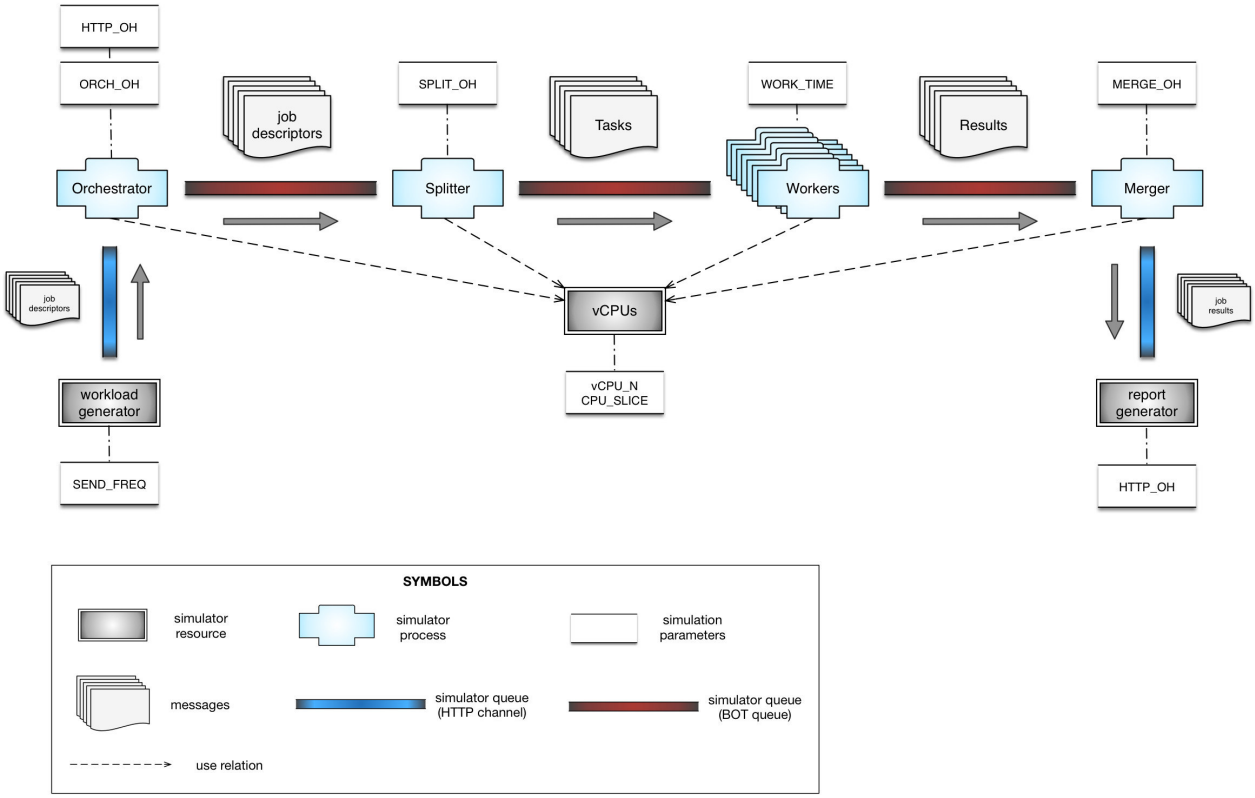

FIG. 4.1. *Bag-Of-Tasks simulation model*

communication and storage (`Queue` and `KVstore`) (Figure 4.1).

The management components of the bag-of-task framework (`HTTPgw` and `Orchestrator`) have the role of managing the cloud application execution, offering an interface to end users (`HTTPgw`) and orchestrating the execution, forwarding the messages to the right splitter when multiple bag-of-tasks are executed on the same resources and starting/stopping the triple Splitter-Worker-Merger.

Our simulation model is driven by a `workload generator`, which is in charge to generate the sequence of requests the users issue in time. At the start of simulation, it begins sending out the messages to the `Orchestrator` according to the chosen workload, described in a configuration file. Two simple workload models are currently available: a set of requests with fixed inter-arrival time, and a Poisson arrival model that generates messages with random exponential inter-arrival time. Moreover, it is also possible to start a multiple number of concurrent workloads miming the load generated by multiple users.

The `Orchestrator` model is fairly simple: it continuously receives `Jobs` from a queue (that mimics an HTTP channel) and forwards them to the `Splitter`. At the state of the art, our model is very simple, since it is based on the assumption of a fixed application configuration, which cannot be dynamically altered (i.e., we cannot start a new set of Split-Work-Merge components during the simulation). So the `Orchestrator` has just the role of routing the message to the `Splitter` instances. We aim at improving the `Orchestrator` model in the future.

The `Splitter`, `Merger` and `Worker` models behave in a similar way, receiving and forwarding the messages from/to the internal queues accordingly to the described bag-of-tasks pattern. The `Merger` process, after collecting all the intermediate job `Result` messages, sends a `job Result` message to a special simulator component, the `report generator`, which gathers the results and produces the final simulation reports.

The computational resources consumed by our core components (i.e., `Orchestrator`, `Splitter`, `Merger`

LISTING 1. *The Queue Simulation Process*

```
public class Queue extends it.unisannio.ing.perflab.jades.core.Process {
    private Mailbox inputMailbox;
    private Mailbox outputMailbox;

    public Queue(String name, double beta0, double beta1, double beta2) {
        super(name);
        inputMailbox = new Mailbox(name + "inputMailbox");
        outputMailbox = new Mailbox(name + "outputMailbox");
    }

    public void send(Object m) {
        inputMailbox.send(m);
    }

    public void run() {
        while (true) {
            int msgSize = (Integer) inputMailbox.receive();
            int msgInQueue = inputMailbox.msg_in_queue()+1;
            hold(beta2 * msgSize + beta1 * msgInQueue + beta0);
            outputMailbox.send(msgSize);
        }
    }

    public Object receive() {
        return outputMailbox.receive();
    }

}
```

and `Worker` processes) are taken into account by means of a component that simulates CPU resource sharing. At simulation start-up it is necessary to provide the actual number of available virtual CPUs (vCPUs) and the allocation to vCPUs of the framework components involved in a run.

The response time of the queues (i.e., the time needed to notify a message to a process, once it has been published on the queue) is modeled as a function of the number of queued messages and of the dimension of the messages (according to the iLDS model [44]). Listing 1 shows the code of the process simulating the behavior of the communication queues.

The simulation model sketched in Figure 4.1 can be coded as illustrated in Listing 2. The proposed listing shows only the code fragment in charge of building the simulation application. For brevity's sake we have replaced the actual parameters used for object creation with the placeholder <`parameters`>. It is worth pointing out how closely the simulation code resembles the structure of the bag-of-tasks application.

As previously pointed out, even if the actual framework behavior strictly depends on the specific algorithm to be implemented, in any case the core bag-of-tasks components receive messages from queues and forward them to other queues, consuming a suitable amount of CPU time. Starting from this consideration, we can fully describe a bag-of-tasks instance by means of two sets of parameters.

The *application instance* parameters represent the values under the control of the framework user. They describe the application and the BOT framework configuration to be simulated, as follows:

- **virtual CPUs** (vCPU_N): vCPUs available to the mOSAIC platform;
- **vCPU time slice** (vCPU_SLICE): vCPU scheduler preemption time;
- **workers** (WORKERS): number of Worker instances;
- **splitters** (SPLITTERS): number of Splitter instances;
- **jobs** (JOBS): number of jobs generated and submitted;

Listing 2. *The Application Model Simulation Process*

```
public void run() {
        FilterCloudlet orchestrator, splitter, merger, workers[];
        double setupTime = 0.0;
        hold (setupTime);

        Queue gwQueue=new Queue("gwQueue", <parameters> );
        Queue fromOrchestratorToSplitter=new Queue("orchToSplit",<parameters>);
        Queue fromSplitterToWorker=new Queue("SplitToWork",<parameters>);
        Queue fromWorkerToMerger=new Queue("workToMerge", <parameters>);

        double orchestratorHoldTime =
                Double.valueOf(myProp.getProperty("receive_from_http_overhead"));
        orchestrator = new FilterCloudlet("orchestrator_cloudlet",
                orchestratorHoldTime, gwQueue, fromOrchestratorToSplitter);
        double splitterHoldTime =
                Double.valueOf(myProp.getProperty("single_split_overhead"));
        splitter = new Splitter("splitter_cloudlet",
                splitterHoldTime, fromOrchestratorToSplitter, fromSplitterToWorker);

        add(gwQueue);
        add(fromOrchestratorToSplitter);
        add(fromSplitterToWorker);
        add(fromWorkerToMerger);
        add(outQUeue);
        add(orchestrator);
        add(splitter);

        int workers_num = Integer.valueOf(myProp.getProperty("workers"));
        workers = new FilterCloudlet[workers_num] ;
        double workerThinkTime =
                Double.valueOf(myProp.getProperty("worker_thinktime"));

        for (int i =0 ; i < workers_num ; i++ ){
                workers[i] = new  FilterCloudlet("worker_"+i,
                        workerThinkTime , fromSplitterToWorker, fromWorkerToMerger);
                add(workers[i]);
        }

        double mergerHoldTime =
                Double.valueOf(myProp.getProperty("single_merge_overhead"));
        merger = new Merger("merger_cloudlet",
                mergerHoldTime, fromWorkerToMerger, outQUeue);
        add(merger);

        //create jobs
        int jobs = Integer.valueOf(myProp.getProperty("jobs"));
        double jobs_frequency =
                Double.parseDouble(myProp.getProperty("send_frequence"));
        int tasks = Integer.valueOf(myProp.getProperty("tasks"));

        for (int i =0 ; i < jobs ; i++ ){
        gwQueue.send(new Message(String.valueOf(i)+"_"+tasks));
                hold(jobs_frequency);
        }
}
```

- **send job frequence** (SEND_FREQ): job send rate;
- **tasks** (TASKS): number of tasks generated by the `Splitter`;
- **worker overhead** (WORK_TIME): estimate of the vCPU time required by a `Worker` to process a `Task`;
- **allocation map**: allocation matrix describing the allocation of the framework processes to the available vCPUs.

The *framework tuning parameters* are used to model a specific framework instance, taking into account the overhead introduced by the framework itself, by the platform and by any underlying software layer. After they have been estimated for a framework instance, they will not vary across different simulation runs. In the following subsection we will describe a methodology for the evaluation of such values. The parameters are:

- **HTTP overhead** (HTTP_OH): the communication delay introduced by an HTTP communication channel;
- **Orchestrator overhead** (ORCH_OH): the orchestrator overhead (vCPU time) introduced to process each submitted job and to forward the descriptor to the `Splitter`;
- **Splitter overhead** (SPLIT_OH): overhead introduced (vCPU time) to execute a single *split* operation, to create and to forward a `Task`;
- **Merger overhead** (MERGE_OH): overhead (vCPU time) introduced to execute a single *merge* operation;
- **Job Descriptor message** (JOB_MSG_SIZE): size of the `Job Descriptor` messages;
- **Result message** (RESULT_MSG_SIZE): size of the `Result` messages;
- **Task message** (TASK_MSG_SIZE): size of the `Task` messages;
- **HTTP channel**: *beta0*, *beta1*, *beta2* parameters (see Listing 1) to model the HTTP channel;
- **bag-of-tasks queues**: *beta0*, *beta1*, *beta2* parameters to model the framework internal queues.

4.2. Tuning of Algorithm-dependent Components. The above presented simulation model is completely independent of the algorithm implemented in the bag-of-tasks application. The specific characterics of the application affect the model only as far as the number and dimension of messages the `Splitter` generates, the `Worker(s)` elaborates and the `Merger` joins together are concerned. Moreover, they affect the amount of CPU time required by each of the above components to perform its own actions.

The first set of information (number and dimension of messages generated) is usually dependent only on the problem data dimension, and so is known beforehand, when the simulation takes place. On the other hand, the CPU time needed for each elaboration needs to be estimated. Moreover, the times spent in the other bag-of-tasks framework components (queues, storages, orchestrator) need to be evaluated taking into account the type and number of resources leased from the cloud that will be used to execute the application.

In order to estimate the time behavior of the above mentioned components we use an approach based on the execution of *ad-hoc* benchmarks, which run using the benchmarking framework integrated in mOSAIC API [8]. The methodology used is thoroughly dealt with in paper [14]. Following this approach, we develop dedicated benchmark application for each specific component of the application; the mOSAIC framework automates the process of their execution. Benchmarks for the basic components, namely queue Servers and KV stores, are offered by the mOSAIC framework out-of-the-box. The corresponding benchmarking micro-applications are organized as shown in Figure 4.2. We have developed similar benchmarking applications for the core components of our bag-of-tasks framework (`Splitter`, `Worker` and `Merger`, `Orchestrator`). The execution of such applications, automated by the framework, makes it possible to collect response times in a single csv file, from which, using regression models, it is possible to find the value of the timing parameters to be used for simulation.

5. Early Prediction Development Methodology. Program performance simulation is an interesting matter, but it becomes a fundamental technique if it can be adopted at the very early development stages, before the complete development and deployment of an actual application. The use of performance prediction as an integral part of a parallel program development methodology (*performance debugging*) has turned out to be particularly efficient if performance prediction tools are used at an early stage of the software life cycle (i.e., before a complete the implementation has been devised) [6, 7]. The idea is to exploit a partially implemented program design to obtain performance data before the complete software implementation. This allows a sort of

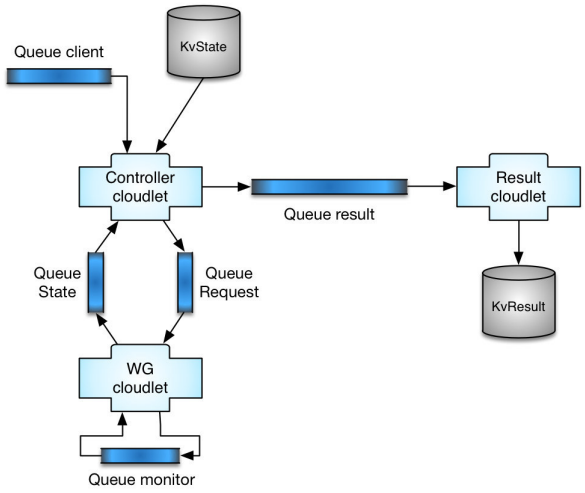

FIG. 4.2. *The Queue Benchmark mOSAIC Application*

performance-driven parallel program design, where algorithm choices can be evaluated and possibly compared very early in the development process.

The "classical" method to obtain early performance data before code finalization is to resort to *prototypes*, i.e., to skeletons of code where some of the computations interleaved between communications are not fully specified, and are represented for simulation purposes by delays equal to the expected time that will be spent in the actual code (for details, please refer to the above cited [6, 7]). In fact, most of the times the specification and the use of code prototypes for simulation purposes is not a simple and error-proof procedure. In the case of BOT code, developed within the proposed framework, instead, the adoption of a "fixed" execution model not only lends itself to a direct construction of a simulation model, as shown in the previous section, but leads to a very simple and well-defined performance prediction development methodology.

This methodology can be directly integrated in the BOT application life cycle (Figure 5.1), and it is made out of the following steps:

1. **Identify the Split/Work/Merge Algorithms**: the developer has to rethink of his/her algorithm in order to identify the role of the core bag-of-tasks components. This is not a complex task, because the BOT paradigm is usually close to the behavior of most common applications. In this phase the developer starts writing and rethinking his/her own code.
2. **Prepare the set of data to be compared**: concurrently with algorithm development, the developer identifies the way in which the application will be used in the future, i.e., how many requests will be served and how the work will be split among workers. This activity can be conducted concurrently with development, so that it is possible to adapt splitting and merging algorithms in order to reduce the execution costs in future, on the basis of the simulation predictions.
3. **Execute the benchmarks and collect the results**: the benchmark applications are launched in the target mOSAIC cloud platform, i.e., in the same environment where the final code will be executed. This makes it possible to obtain performance figures for every component *on the actual execution platform*. It should be noted that, if the application is not fully developed, the developer will only run the benchmarks for the core components. The CPU time requirements of not-fully developed code can be estimated by means of static software analysis. Of course, this is likely to affect adversely the prediction accuracy.
4. **Obtain a performance prediction by executing the simulation model**: the simulation model can be executed with multiple synthetic workloads, using the parameters estimated as described in the previous steps, obtaining performance predictions for different scenarios of interest.
5. **Cost evaluation**: the response times of the application, possibly under several different configuration

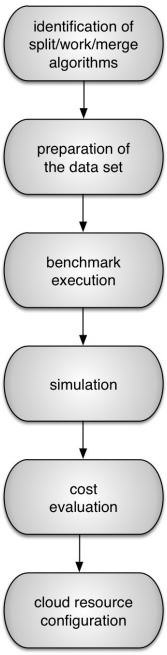

FIG. 5.1. *The Phases of the Early Prediction Methodology*

options, can be "projected" on the cloud offerings of multiple providers, considering alternative pricing plans. The objective here is to find the application execution cost under any reasonable configuration.

6. **Cloud resource configuration**: the output of the previous step makes it possible for the developer to compare providers, pricing plans and configuration options. Here the developer may choose to trade off performance for reduced cost of leased resources, or to focus on performance at the expense of a higher cost of computing resources.

Following such approach, the developer is able to predict the performance and resource usage (and related costs) from the early development stages of his/her cloud application. It is worth pointing out that the proposed procedure does not require the complete set of cloud resources corresponding to the chosen run-time configuration to find predicted response times. In other words, it is possible to build the model and to obtain by benchmarking the performance figures of every system component on a small-scale virtual machine configuration. Then, it is possible to obtain by simulation the overall application response time on a much larger machine configuration. This option will be discussed in detail in the example presented in Section 7.

6. Validation of Simulation Results. As outlined in the previous sections, the framework we propose is composed of (i) the cloud BOT development framework, which enables the developer to run applications on cloud environment, (ii) the simulation framework, which enables the developer to predict application performance even in the early development stages and (iii) the benchmark applications, which are used to evaluate the timing parameters for simulation.

The first step needed to evaluate the feasibility of the proposed approach is to validate the (small-scale) performance results obtained by simulation. A more comprehensive performance prediction and analysis example, including cost evaluation and comparisons, will be presented in the next Section.

In order to validate the approach, we run benchmark applications on the target VMs to gather the timing parameters for the simulation environment, using a minimal set of resources. Then we compare a real application with its simulation, varying both the workload and the amount of resources assigned to the application run.

6.1. Measurement of Timing Parameters. Our BOT framework makes it possible to obtain applications that are completely independent of the real environment on which it will run. The number and the

characteristics of cloud resources leased for execution only affect performance. We capture the characteristics of the actual execution environment through a set of benchmarks, whose results produce the timing parameters successively used for simulation.

For our tests, we have deployed the mOSAIC platform on Virtual Machines (VM) leased from the Amazon Web Services (AWS) infrastructure. The characteristics of the acquired VMs are shown in Table 6.1. Running our benchmark suite, we obtained the values in Table 6.2.

TABLE 6.1
AWS VM instance details

Parameter	Value
instance type	c1.medium
vCPU	2
RAM	1.7 GB
storage	350 GB
network performance	moderate

TABLE 6.2
Testbed timing parameters

Parameter Name	Value
HTTP_OH	50 ms
ORCH_OH	10 ms
SPLIT_OH	5 ms
MERGE_OH	5 ms
HTTP_CHANNEL beta0	1500
HTTP_CHANNEL beta1	0.500
HTTP_CHANNEL beta2	0.010
BOT_QUEUES beta0	65
BOT_QUEUES beta1	0.003
BOT_QUEUES beta2	0.001

6.2. Simulation Validation Varying the Workload. We have developed a skeletal BOT application, where the work method in the `Worker` class is able to process a given load, expressed in MFLOPS and obtained as a parameter from the task description. The Splitter and Merger do not actually perform splitting/merging, but just activate Workers and collect response times, respectively. We submitted the same workload both to the skeletal application running in the real execution environment and to the simulator, comparing the measured and predicted completion times. For our tests, we used the workloads briefly described in Table 6.3, which are representative of light (WORK_TIME=15 ms) and medium-heavy (WORK_TIME=200 ms) load for the workers. To vary dynamically the number of tasks, we used a pseudo-random uniform distribution.

Figure 6.1 shows on the x-axis the job number (30 jobs are submitted, according to Table 6.3) and on the y-axis its completion time. The completion time associated to job #30 is the total completion time for the whole burst of 30 jobs. The simulation results are summarized in Table 6.4, where is also reported the estimated vCPU usage.

Summarizing the results shown in Figure 6.1, for Test 1 (WORK_TIME=15 ms) we measured a completion time of 512 ms versus a predicted one of 551 ms, with a relative error of 7.61% (considering also intermediate jobs completion times, the error ranges from a minimum of 0.77% to a maximum of 28.57%). For Test 2 (WORK_TIME=200 ms) we measured a completion time of 1383 ms against a predicted of 1134 ms, with a relative error of about 18% (considering intermediate jobs, the error ranges from 4.37% to 18.62%).

In both cases the simulation offers good predictive capacities. Moreover, even at this stage (no actual application developed) it offers interesting information about resource usage. The vCPU usage in Test 1 (the one with lower WORK_TIME) is very low, as most of execution time is spent in communications. This is a

TABLE 6.3
Workload for Test 1 and 2

parameter	value
vCPU	2
JOBS	30
SEND_FREQ	1 s
TASKS	uniform(100,400)
WORK_TIME	15 ms; 200 ms
WORKERS	4
SPLITTERS	2
TASK_MSG_SIZE	400 B
JOB_MSG_SIZE	20 B
RESULT_MSG_SIZE	20 B

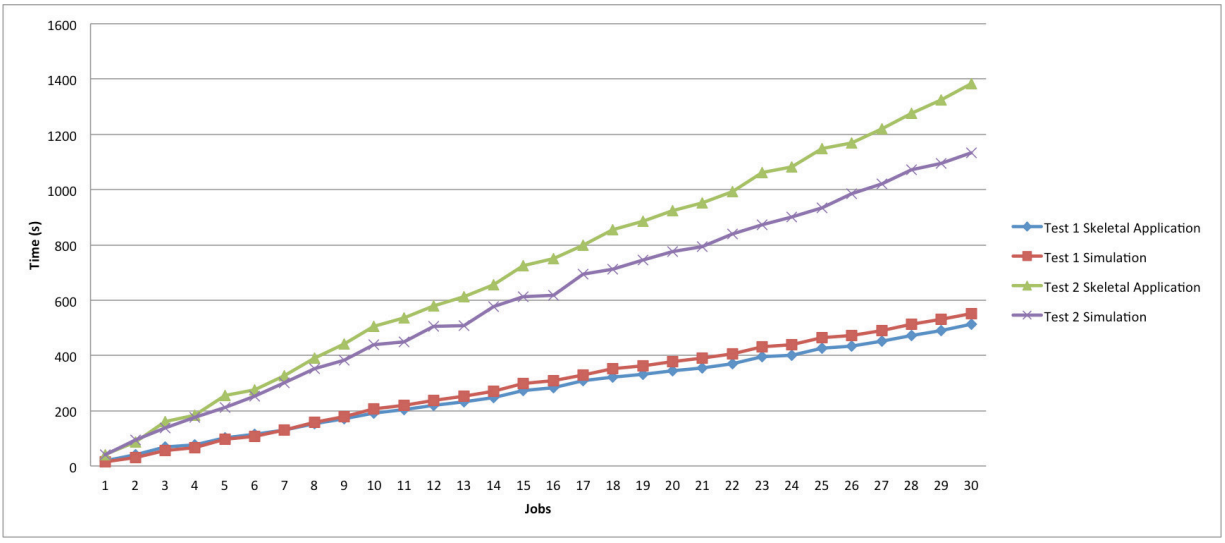

FIG. 6.1. *Comparison between skeletal application and simulated completion times*

relevant hint for the developer of real code, who can tune its implementation so as to obtain a coarser task granularity and hence a more efficient balance of computing and communication.

6.3. Simulation Validation Varying the Resources Used. In the following we compare the performance of the skeletal application executed on real resources to the one obtained by simulation, using a variable number of workers for the real and the simulated run. It should be noted that a higher number of vCPUs is clearly necessary for the execution of these tests on real resources. On the other hand, if we assume to lease additional vCPUs on AWS of the same type used in the previous test, we can use once again the timing parameters in Table 6.2. So the simulation of a larger-scale run does not involve additional leasing costs. Only the allocation matrix in the simulator configuration file has to be updated.

We submitted the workload described in Table 6.5 to the synthetic application, varying the number of workers and comparing it to the real measurements on the skeletal application. The test outcome matches the results proposed in [15], where we evaluated the framework overhead.

Table 6.6 shows the simulation results compared to the actual figures obtained through the real execution of the synthetic application. It should be noted that the simulation output offers the same information on scalability as tests on real hardware (presented previously in [15]). This is a proof of the simulation ability of to catch and to reproduce the behavior of the real system. This ability can be fruitfully exploited to avoid expensive executions on real resources.

TABLE 6.4
Test simulation results

Metric	Test 1	Test 2
Completion Time	551	1134
Tasks sent	7354	7354
vCPU1 usage	0.29%	0.99%
vCPU2 usage	0.04%	0.99%

TABLE 6.5
Workload for Worker Scalability Test

parameter	value
vCPU	4
JOBS	10
SEND_FREQ	1 s
TASKS	30
WORK_TIME	100 ms
WORKERS	1 to 5
SPLITTERS	1
TASK_MSG_SIZE	20 B
JOB_MSG_SIZE	20 B
RESULT_MSG_SIZE	20 B

7. Cost/Performance Analysis by Early Prediction. The goal of this paper is not to provide ultimate performance measurements for real-world applications. Our objective is just to evaluate the feasibility of the proposed approach, i.e., the development of scientific code on the top of a cloud-aware programming framework, exploiting early performance prediction techniques for making cost/performance trade-offs.

As mentioned before, the main advantage of the adoption of simulation is that it enables us to make predictions of performance (and related costs) even when complete code and/or resources are not available. Assuming that the simulation results are sufficiently accurate, as in the example of the previous section, it is possible to test different cloud resource configurations/options at no additional cloud leasing cost. The evaluations thus obtained, as early as at application development time, can help developers to make the choice most fit for their needs.

7.1. Application Configuration Optimization. The simulator helps to drive application development and deployment, predicting the effect of different configurations and of the amount of computing resources used. It should be noted that resorting to real measurements to support development and configuration choices is inefficient and expensive. Furthermore, real measurements require the availability of (at least) a code skeleton, whose construction is likely to be cumbersome. A possible option to perform predictions and comparisons would be the use of an analytic model of the application. However, the construction of an analytic model requires an in-depth knowledge of the performance behavior of system and application. All things considered, the most simple solution is the use of the general-purpose simulation model presented in Section 4, which makes it is possible to obtain reasonably-accurate evaluations in a simple and straightforward way.

The objective of the following case study is to use performance and cost prediction to support the choice of an optimal number optimal number of vCPU for the workload shown in Table 7.1. This is very different from the workloads of Tables 6.3 and 6.5, since there just one huge job (and not a stream of small successive jobs) made out of many tasks (100,000) with long work time (3000 ms). The investigation is conducted for a number of vCPUs ranging from 1 to 20, assuming to map a worker on each vCPU.

The use of the same AWS instances as in the previous section (see Table 6.1) and hence of the same timing parameters (Table 6.2) made it possible to run all the required simulations on a PC in few minutes, without leasing resources from the cloud. The obtained completion times are shown in Table 7.2. Taking into account

TABLE 6.6
Scalability: Real and Simulated Response times

Number of Workers	Real App Response Time	Simulated Time (s)
1	84	94
2	44	49
3	31	35
4	30	33
5	30	33

TABLE 7.1
Case study workload

parameter	value
vCPU	1,2,4,8,16, 20
JOBS	1
SEND_FREQ	N/A
TASKS	100000
WORK_TIME	3000 ms
WORKERS	2,4,8
SPLITTERS	1
TASK_MSG_SIZE	20 B
JOB_MSG_SIZE	20 B
RESULT_MSG_SIZE	20 B

that the C1 medium instances are priced at 0.130/hour[1], it is possible to plot response time and running cost as shown in Figure 7.1. Without entering for brevity's sake into the detail of the comparison between different providers, type of instances and pricing plans, the proposed figure makes it possible to observe that for more of 8 vCPUs the cost rises without any significant performance increase. In these conditions, the most reasonable choice is probably the use of 4 or 8 vCPUs for workers.

The presented trends can be obtained at early application development stage, since the process requires, in essence, only to predict the computational load to process each task. On the contrary, the other parameters represent design choices.

8. Conclusions and Future Work. The aim of the work described in this paper is to propose an approach that integrates the development of scientific application on top of a cloud platform and its performance prediction through a dedicated simulation environment.

To obtain this integration, on one hand we have built a development framework, currently specialized for the bag-of-task paradigm, which exploits the API and the components provided by the mOSAIC platform. On the other, we have developed a set of simulation components for JADES. These simulation components correspond one-to-one to the mOSAIC components for the BOT framework. Besides presenting the approach, we have discussed the outcome of our preliminary performance tests used to evaluate the simulation accuracy.

In our tests, we offered some examples of usage under different workloads and using different amount of resources, in order to show how it will be possible for a developer to predict the behavior of is application, its completion time and the associated cloud resource leasing cost without running it on (paid) cloud resources, but simply using the associated simulation environment.

Our future research work will focus on the extensive testing of the framework and simulation components, by collecting measurements on real-world scientific codes running in private and commercial cloud environments. We also plan to implement alternative frameworks for additional programming paradigms.

[1]Price checked on 10/6/2015: *on-demand* pricing plan, US East (North Virginia) data center.

TABLE 7.2
Completion times vs. number of workers

# Workers	Completion Time (h)
1	83472,32278
2	41793,99694
4	20885,73444
8	10461,88806
12	7396,314722
16	6503,216667
20	6139,974722

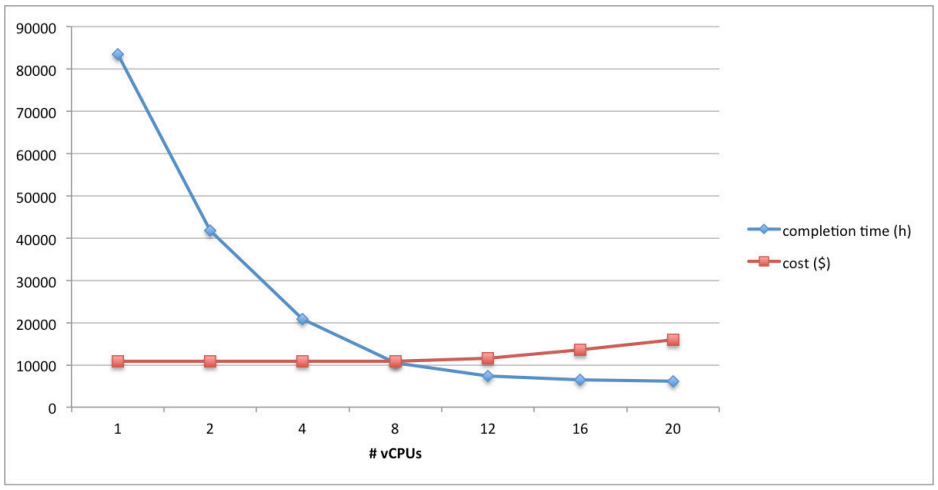

FIG. 7.1. *Completion time and cost vs. number of vCPUs*

REFERENCES

[1] *Basho products.* "http://basho.com/riak/".
[2] *ec2 spot instances.* "https://aws.amazon.com/ec2/purchasing-options/spot instances/".
[3] S. ACHOUR, M. AMMAR, B. KHMILI, AND W. NASRI, *MPI-PERF-SIM: Towards an automatic performance prediction tool of MPI programs on hierarchical clusters*, in Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on, IEEE, 2011, pp. 207–211.
[4] D. AGARWAL AND S. PRASAD, *Azurebot: A framework for bag-of-tasks applications on the azure cloud platform*, in Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International, May 2013, pp. 2139–2146.
[5] M. AL-FARES, A. LOUKISSAS, AND A. VAHDAT, *A scalable, commodity data center network architecture*, SIGCOMM Comput. Commun. Rev., 38 (2008), pp. 63–74.
[6] R. AVERSA, A. MAZZEO, N. MAZZOCCA, AND U. VILLANO, *Developing applications for heterogeneous computing environments using simulation: A case study*, Parallel Computing, 24 (1998), pp. 741 – 761.
[7] R. AVERSA, A. MAZZEO, N. MAZZOCCA, AND U. VILLANO, *Heterogeneous system performance prediction and analysis using ps*, Concurrency, IEEE, 6 (1998), pp. 20–29.
[8] G. AVERSANO, M. RAK, AND U. VILLANO, *The mosaic benchmarking framework: Development and execution of custom cloud benchmarks.*, Scalable Computing: Practice and Experience, 14 (2013).
[9] R. CALHEIROS, R. RANJAN, A. BELOGLAZOV, C. DE ROSE, AND R. BUYYA, *Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms*, Software: Practice and Experience, 41 (2011), pp. 23–50.
[10] P. CLAUSS, M. STILLWELL, S. GENAUD, F. SUTER, H. CASANOVA, AND M. QUINSON, *Single node on-line simulation of MPI applications with SMPI*, in 25th IEEE International Parallel and Distributed Processing Symposium, 2011. IPDPS'11, IEEE, 2011, pp. 664–675.
[11] E. F. COUTINHO, G. PAILLARD, AND J. N. DE SOUZA, *Performance analysis on scientific computing and cloud computing environments*, in Proceedings of the 7th Euro American Conference on Telematics and Information Systems, EATIS '14, New York, NY, USA, 2014, ACM, pp. 5:1–5:6.

[12] A. Cuomo, M. Rak, and U. Villano, *Process-oriented discrete-event simulation in Java with continuations:quantitative performance evaluation*, in Proc. of the International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), SciTePress, 2012, pp. 87–96.
[13] A. Cuomo, M. Rak, and U. Villano, *Concurrent simulation in the cloud with the mjades framework*, International Journal of Simulation and Process Modelling, (2013), pp. 212–226.
[14] A. Cuomo, M. Rak, and U. Villano, *Performance prediction of cloud applications through benchmarking and simulation*, International Journal of Computational Science and Engineering, in the press (2014).
[15] A. De Benedictis, M. Rak, M. Turtur, and U. Villano, *Cloud-aware development of scientific applications*, in 23rd IEEE International WETICE Conference WETICE-2014, June 2014, pp. 149–154.
[16] J. Dean and S. Ghemawat, *Mapreduce: Simplified data processing on large clusters*, Commun. ACM, 51 (2008), pp. 107–113.
[17] B. Di Martino, E. Mancini, M. Rak, R. Torella, and U. Villano, *Cluster systems and simulation: from benchmarking to off-line performance prediction*, Concurrency and Computation: Practice and Experience, 19 (2007), pp. 1549–1562.
[18] C. Evangelinos and C. N. Hill, *Cloud computing for parallel scientific HPC applications: Feasibility of running coupled atmosphere-ocean climate models on Amazon's EC2*, in 1st Workshop on Cloud Computing and its Applications (CCA), 2008.
[19] G. Galante and L. Bona, *Constructing elastic scientific applications using elasticity primitives*, in Computational Science and Its Applications ICCSA 2013, B. Murgante, S. Misra, M. Carlini, C. Torre, H.-Q. Nguyen, D. Taniar, B. Apduhan, and O. Gervasi, eds., vol. 7975 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 281–294.
[20] L. Gillam, B. Li, J. O'Loughlin, and A. P. S. Tomar, *Fair benchmarking for cloud computing systems*, Journal of Cloud Computing: Advances, Systems and Applications, 2 (2013).
[21] Y. Gong, B. He, and J. Zhong, *An overview of CMPI: Network performance aware MPI in the cloud*, in Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '12, New York, NY, USA, 2012, ACM, pp. 297–298.
[22] Y. Gong, B. He, and J. Zhong, *Network performance aware MPI collective communication operations in the cloud*, Parallel and Distributed Systems, IEEE Transactions on, (2013).
[23] A. Gupta, *Efficient high performance computing in the cloud: Keynote talk*, in Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing, VTDC '15, New York, NY, USA, 2015, ACM, pp. 1–1.
[24] A. Gupta and L. Kale, *Towards efficient mapping, scheduling, and execution of HPC applications on platforms in cloud*, in Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International, May 2013, pp. 2294–2297.
[25] A. Gupta, L. Kale, D. Milojicic, P. Faraboschi, and S. Balle, *HPC-aware VM placement in infrastructure clouds*, in Cloud Engineering (IC2E), 2013 IEEE International Conference on, March 2013, pp. 11–20.
[26] A. Gupta and D. Milojicic, *Evaluation of HPC applications on cloud*, in Open Cirrus Summit (OCS), 2011 Sixth, Oct 2011, pp. 22–26.
[27] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, *Case study for running HPC applications in public clouds*, in Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, New York, NY, USA, 2010, ACM, pp. 395–401.
[28] Z. Hill and M. Humphrey, *A quantitative analysis of high performance computing with Amazon's EC2 infrastructure: The death of the local cluster?*, in Grid Computing, 2009 10th IEEE/ACM International Conference on, Oct 2009, pp. 26–33.
[29] A. Iosup, S. Ostermann, M. Yigitbasi, R. Prodan, T. Fahringer, and D. H. J. Epema, *Performance analysis of cloud computing services for many-tasks scientific computing*, Parallel and Distributed Systems, IEEE Transactions on, 22 (2011), pp. 931–945.
[30] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright, *Performance analysis of high performance computing applications on the Amazon web services cloud*, in Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, IEEE, 2010, pp. 159–168.
[31] G. Juve, E. Deelman, G. B. Berriman, B. P. Berman, and P. Maechling, *An evaluation of the cost and performance of scientific workflows on Amazon EC2*, Journal of Grid Computing, 10 (2012), pp. 5–21.
[32] H. Koziolek, *Performance evaluation of component-based software systems: A survey*, Performance Evaluation, 67 (2010), pp. 634–658.
[33] C. A. Lee, *A perspective on scientific cloud computing*, in Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, New York, NY, USA, 2010, ACM, pp. 451–459.
[34] A. Li, X. Yang, S. Kandula, and M. Zhang, *Cloudcmp: comparing public cloud providers*, in Proceedings of the 10th annual conference on Internet measurement, ACM, 2010, pp. 1–14.
[35] W. Lu, J. Jackson, and R. Barga, *Azureblast: A case study of developing science applications on the cloud*, in Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, New York, NY, USA, 2010, ACM, pp. 413–420.
[36] A. Marathe, R. Harris, D. K. Lowenthal, B. R. de Supinski, B. Rountree, M. Schulz, and X. Yuan, *A comparative study of high-performance computing on the cloud*, in Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing, HPDC '13, New York, NY, USA, 2013, ACM, pp. 239–250.
[37] P. Mehrotra, J. Djomehri, S. Heistand, R. Hood, H. Jin, A. Lazanoff, S. Saini, and R. Biswas, *Performance evaluation of Amazon EC2 for NASA HPC applications*, in Proceedings of the 3rd Workshop on Scientific Cloud Computing Date, ScienceCloud '12, New York, NY, USA, 2012, ACM, pp. 41–50.
[38] E. Mocanu, V. Galtier, and N. Tapus, *Generic and fault-tolerant bag-of-tasks framework based on javaspace technology*, in 2012 IEEE International Systems Conference, March 2012, pp. 1–6.
[39] A. Mos and J. Murphy, *A framework for performance monitoring, modelling and prediction of component oriented dis-*

tributed systems, in Proceedings of the 3rd international workshop on Software and performance, ACM, 2002, pp. 235–236.

[40] J. Napper and P. Bientinesi, *Can cloud computing reach the top500?*, in Proceedings of the Combined Workshops on UnConventional High Performance Computing Workshop Plus Memory Access Workshop, UCHPC-MAW '09, New York, NY, USA, 2009, ACM, pp. 17–20.

[41] K. Perumalla and R. Fujimoto, *Efficient large-scale process-oriented parallel simulations*, in Proc. of the 30th Winter Simulation Conference, 1998, pp. 459–466.

[42] D. Petcu, C. Craciun, M. Neagul, S. Panica, B. D. Martino, S. Venticinque, M. Rak, and R. Aversa, *Architecturing a sky computing platform*, in ServiceWave Workshops, M. Cezon and Y. Wolfsthal, eds., vol. 6569 of Lecture Notes in Computer Science, Springer, 2010, pp. 1–13.

[43] D. Petcu and M. Rak, *Open-source cloudware support for the portability of applications using cloud infrastructure services*, in Cloud Computing, Z. Mahmood, ed., Computer Communications and Networks, Springer London, 2013, pp. 323–341.

[44] M. Rak, R. Aversa, B. D. Martino, and A. Sgueglia, *Web services resilience evaluation using lds load dependent server models*, JCM, 5 (2010), pp. 39–49.

[45] A. Raveendran, T. Bicer, and G. Agrawal, *A framework for elastic execution of existing MPI programs*, in Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on, May 2011, pp. 940–947.

[46] H. Schwetman, *CSIM19: a powerful tool for building system models*, in Proc. of the 33nd Winter Simulation Conference, IEEE, 2001, pp. 250–255.

[47] M. Taifi, *Banking on decoupling: Budget-driven sustainability for HPC applications on auction-based clouds*, SIGOPS Oper. Syst. Rev., 47 (2013), pp. 41–50.

[48] A. Videla and J. J. W. Williams, *RabbitMQ in action: distributed messaging for everyone. Rabbit MQ in action*, Manning, Shelter Island NY, 2012.

[49] E. Walker, *Benchmarking Amazon EC2 for high-performance scientific computing*, ; login:: the magazine of USENIX & SAGE, 33 (2008), pp. 18–23.

[50] Y. Zhai, M. Liu, J. Zhai, X. Ma, and W. Chen, *Cloud versus in-house cluster: Evaluating Amazon cluster compute instances for running MPI applications*, in State of the Practice Reports, SC '11, New York, NY, USA, 2011, ACM, pp. 11:1–11:10.

Edited by: Dana Petcu
Received: June 16, 2015
Accepted: Aug 21, 2015