Cloud-based Concurrent Simulation at Work: Fast Performance Prediction of Parallel Programs

Massimiliano Rak  
Seconda Università di Napoli, Dipartimento di Ingegneria dell’Informazione, Aversa, Italy  
massimiliano.rak@unina2.it

Antonio Cuomo, Umberto Villano  
Università del Sannio, Dipartimento di Ingegneria, Benevento, Italy  
{antonio.cuomo, villano}@unisannio.it

Abstract—The performance prediction of parallel programs is a time-consuming process due to the need to execute a large number of simulations for different values of parameters linked to the software decomposition and to the target hardware environment. This paper shows how mJADES, a novel environment for running concurrent simulations in the cloud, can provide an effective support to a simulation-based performance prediction framework. The accuracy of predictions and the practical validity of the approach followed are evaluated through experimentation on a simple but realistic case study.

Index Terms—Cloud Computing, Discrete-event Simulation, Performance Prediction

I. INTRODUCTION

Over the last few years, cloud computing [1] is steadily diffusing as a means to provide computing resources, platforms and software in the form of simple and economically convenient pay-per-use services. We have recently presented a discrete-event simulation environment [2] running in the cloud in Software-as-a-Service (SaaS) mode. mJADES is a cloud application that allows its users to execute multiple simulations leasing resources from the cloud. These simulations are executed concurrently, in that the inherent parallelism of the cloud is exploited to execute multiple sequential simulation tasks at the same time. This is useful whenever multiple simulations are required, for example to sweep one or several simulation parameters, or to answer to “what if” system configuration questions. This is not the canonical approach, as almost always in the past parallel hardware has been used to speed up a single simulation, leading to the development of parallel and distributed simulation techniques [3], [4], [5].

It is well known that the discrete-event simulation of complex systems is a time expensive and resource consuming task. mJADES aims at running sequential, monolithic discrete-event simulations, with the possibility to run multiple simulation instances at the same time, leasing automatically a suitable number of computing resources from a cloud. As discussed in the companion paper introducing mJADES [2], the rationale of this design is clear. Whenever a simulation must be executed many times, there is an exploitable source of parallelism in running together multiple instances of the same basic simulation task. This happens when simulations are stochastic and should be repeated to perform statistical validation, or for parameter-sweep simulations, where one or multiple input parameters vary over a range of values. To make the whole process clearer, in this paper we will present a practical example of application of the concurrent simulation modality provided by mJADES. As will be shown later, this example is not an academic case study, but has a substantial practical utility for parallel programmers.

Parallel program performance prediction is a research field that aims at gaining insight on a parallel code performance behavior without resorting to extensive performance testing. The idea is to know with reasonable precision how the code will perform without executing many times the code on the real hardware and, possibly, even in the absence of fully-developed code. Our research group has shown that this is possible through the simulation of the program behavior [6], [7]. Most of our past results have been obtained by running a program simulator fed with actual or synthetic traces [8], [9], [10], [11]. Though decidedly faster than actual program executions, simulation of a complex code running on a high number of processors is a lengthy process. The construction of performance diagrams of a given code (most of the times, the evaluation of speed-up for progressively higher number of processors until the knee of the curve is reached) requires many simulations. This is exactly the point where concurrent simulation comes into play. We will show that, given a scalable trace (i.e., a trace valid for different number of processors), the required simulations can be straightforward executed in a single step through the mJADES system. This is a genuine advance in the performance prediction tools usability. A performance measurement methodology proven to be valid, but relatively impractical, becomes fast and efficient thanks to the cloud.

The mJADES simulation system exploited in this paper to run concurrent simulation founds on a Java-based simulation library, JADES [12], which provides the basic sequential simulation capabilities. On the other hand, the cloud interfacing relies on the mOSAIC platform [13], [14], a cloud middleware that provides a cloud-enabled, component-based programming interface. These two enabling technologies have been described elsewhere and will be only sketched here.

This research is partially supported by FP7-ICT-2009-5-256910 (mOSAIC) and by MIUR-PRIN 2008 project “Cloud@Home: a New Enhanced Computing Paradigm”
where the focus will be on the practical use of mJADES to obtain straight performance predictions of parallel code. The remainder of this paper is structured as follows: the next section presents related work, pointing out the original features of our approach. Section III gives an overview of mJADES and of the technologies on which it is based. The use of mJADES for parallel software performance prediction is dealt with in section IV. Section V draws the conclusions and sketches our future research directions.

II. RELATED WORK

mJADES is designed in such a way that parallel or distributed hardware is exploited by running multiple concurrent instances of a traditional sequential simulator. This makes it substantially different from the majority of the systems developed in main body of research on parallel/distributed/web-based simulation. However, the idea to speed-up simulations by launching multiple concurrent runs (“replications”) is not new, and was explored in a seminal paper by Heidelberger in the eighties [15], comparing the obtained performance to the one attained by a more conventional parallel simulator. The possibility to offer simulation software as-a-service has also been considered before. The paper [16] presents a Java-based architecture for replicated simulations on a network of processors, called “alliance”. However, our work relies on the more recent cloud technologies and exploits an open-source simulator engine.

Papers on the use of cloud resources for computer simulation are just beginning to appear in the literature. Experimentation of simulation as-a-service in a cloud are presented in [17] and [18]. References [19] and [20] deal with the porting of traditional parallel/distributed simulation schemes to a master/worker pattern that is more suitable for deployment in a public or private cloud. The Artis/GAIA+ project [21] aims at balancing adaptively the simulation by model decomposition, dynamic partitioning and bottleneck removal, to better suit the variable and dynamic performance levels observed in clouds.

The development of applications targeted to cloud environments is a hot research topic, and only few complete solutions are currently available. An analysis of what a cloud-enabled API should look like is out of the scope of this paper. Some of the open issues are sketched, along with the solution adopted for the mOSAIC middleware, in [13], [14].

Application of simulation to performance prediction of parallel programs dates back to the early nineties. In addition to the work by some of the authors of this paper ([16], [10]), early efforts include the execution-driven MPI-Sim [22] and the trace-driven Dimemas network simulator [23]. Among the more recent contributions in literature, Phantom [24] is a hybrid prediction framework that uses direct execution of computational blocks together with communication traces. Optimized replay techniques are used to obtain the duration of computational blocks on a small sample of the target machine, but trace collection requires the execution of the application on a host with the same number of processes and same problem size. MPI-PERF-SIM [25] is a performance simulator of hierarchical clusters and uses an hybrid benchmark/model based approach with polynomial regression to predict performance of computational blocks, and linear regression to predict communication time. BSIM [26] is targeted at large scale systems. It works by skeletonizing the source code (substituting intensive computational code with delays) and replacing MPI communication calls with simulated calls. BigSim [27] is an online simulator which models sequential code on target platform by user-supplied estimates or by applying a scaling factor to measurements on a host machine.

III. mJADES OVERVIEW

mJADES is a system able to produce simulation tasks from simulation jobs and to run them in parallel by means of multiple instances of the simulation engine, which are executed as mOSAIC cloudlets. The output variables or performance indicators are successively aggregated and presented to the final user. As pointed out by its name, mJADES comes from a fusion of two enabling technologies: the mOSAIC cloud middleware and the JADES simulation library.

A. mOSAIC and JADES

As mentioned in the introduction, mJADES is a cloud application built on the top of the mOSAIC middleware [13], [14]. mOSAIC aims at providing a simple way to develop cloud applications. In mOSAIC, a cloud application is structured as a set of components running on resources leased from a cloud provider and able to communicate with each other. Cloud applications are often provided in the form of Software-as-a-Service, and can also be accessed/used by users other than the mOSAIC developer (i.e., by final users).

A mOSAIC application is built up as a collection of interconnected mOSAIC components. Components may be (i) core components, i.e., predefined helper tools offered by the mOSAIC platform for performing common tasks, (ii) COTS (commercial off-the-shelf) solutions embedded in a mOSAIC component, or (iii) cloudlets developed using the mOSAIC API and running in a Cloudlet Container. mOSAIC cloudlets are stateless, and developed following an event-driven asynchronous approach [13], [14].

The mOSAIC platform offers ready-to-use components such as queueing systems (RabbitMQ and zeroMQ), which are used for component communications, and an HTTP gateway, which accepts HTTP requests and forwards them to application queues, NO-SQL storage systems (as KV store and columnar databases). mOSAIC components run on a dedicated virtual machine, named mOS (mOSAIC Operating System), which is based on a minimal Linux distribution. The mOS is enriched with a special mOSAIC component, the Platform Manager, which makes it possible to manage set of virtual machines hosting the mOS as a virtual cluster.

The second enabling technology exploited by mJADES implementation is the JADES simulation engine. JADES is a system for the development and evaluation of discrete-event simulation models. It supports the process-oriented view,
allowing the modeler to describe his model in terms of interacting processes. It is not a complete stand-alone simulation language, but a Java library that exploits continuations to implement the simulated processes. The aim of JADES is to provide modelers with an implementation of the process-oriented paradigm which is both effective (as it offers a wide range of building blocks to compose models) and efficient (allowing the creation of large, complex models and their fast evaluation). A JADES simulation is composed of processes, which are active components able to act upon (and interact through) passive objects or resources. A model is a logical organization of simulation processes, together with a description of the simulation details, report generation and tracing.

JADES is based on the use of continuations, which enable it to run all the simulation processes in a single Java thread. This has relevant effects in terms of performance [12], due to the absence of thread switch overhead, and also helps the integration with mOSAIC cloudlets, which act on a completely asynchronous base and are not thread-safe.

B. mJADES architecture

mJADES is a system able to produce simulation tasks from simulation jobs and to run them in parallel by means of multiple instances of the simulation engine, which are executed as mOSAIC cloudlets. The output variables or performance indicators are successively aggregated as needed, and presented to the final user. The mJADES architecture is represented in Figure 1, which follows the mOSAIC symbol and color conventions: mOSAIC components are represented by green boxes, cloudlets are yellow diamonds and other cloud resources (queues, storage, databases) are in azure. The application is composed of three cloudlets: mJADES Manager, mJADES Simulation Core and mJADES Simulation Analyzer. To support the ax-a-service approach, final users can interact with the application through the HTTP interface made available by the mOSAIC HTTP gateway component. The next subsections will describe in detail the cloudlets introduced above and their interactions with the predefined mOSAIC components.

C. The mJADES Manager cloudlet

The mJADES Manager cloudlet coordinates the distributed simulation execution. It receives simulation job requests from its users in the form of messages, which may come from the HTTP interface or from other components. Requests are translated into jobs, which receive a global identification key to keep track of the job metadata. Metadata include job status (maintained in the Simulation Audits Key-Value Store) and its outputs (that will be grouped in the Simulation Results). Then the simulation jobs are decomposed into the set of simulations tasks that must be performed. These are dispatched to multiple instances of the Simulation Core (described next), which perform the actual model evaluation required. The key function of the manager is the decomposition of the job into simulation tasks. Currently, the manager supports two configuration modes to carry out this decomposition:

- the job can specify a number of iterations of the same simulation task to be performed, until some statistical property is met. Iterations are transformed into replications, in that multiple instances of the task are launched in parallel;
- the job can require a parameter sweep over one or several simulation parameters. A simulation task is generated for each distinct parameter tuple.

In the current implementation, the simulation tasks are managed statically, in that all the tasks launched are completed independently of the simulation results (which may show, for example, that further simulations are not necessary because the process has converged). Smarter task management will be added in the future versions of mJADES. We are also studying algorithms to speed up the parameter sweep, by cutting the solution space to be explored through the use of meta-heuristics (tabu search, simulated annealing, ...).

As tasks are generated, they become part of the job metadata and receive an identification key which, together with the job identifier, allows all the cloudlets to maintain and to update the global state through the dedicate Key-Value Store. The task status is fed to the Simulation Audits, while the task output goes into the Simulation Results store.

When all the tasks have been completed, the Manager launches the Simulation Analyzer cloudlet to operate on the raw simulation data and to compute the final output to be presented to users. The simulation analyzer accepts as input a message containing the job key, so that it can retrieve the list of all the associated tasks.

D. The mJADES Simulation Core cloudlet

The mJADES Simulation Core cloudlet builds up a simulation engine around the JADES (Java Discrete Event Simulator) library described in subsection III-A. The Core executes a single simulation task and generates a simulation report, which is sent out on a dedicated output queue. It should be noted that, thanks to the mOSAIC approach, the process of launching multiple simulation tasks is made totally transparent for the user. The cloudlet container adapts automatically the
number of cloudlet instances to the number of simulation requests, in order to attain high performance, possibly acquiring new resources from the cloud. As shown in the bottom of Figure 1, the Simulation Core cloudlet has just two connectors (mOSAIC callbacks), connected to two different queues, to receive simulation requests and to generate simulation results, respectively. Following the mOSAIC approach, the cloudlet is completely stateless: each message contains all the information needed to perform a single simulation task. Moreover, each simulation task is completely independent of the others: it is up to the simulation manager and to the simulation analyzer to correlate information from different simulation tasks.

As regards the Simulation Core, the callback handlers from the connectors (which implement the asynchronous behaviour typical of mOSAIC) enable the cloudlet to be awakened on message arrivals. Messages contain the simulation description in JavaScript Object Notation (JSON)\(^1\). The Core cloudlet decodes the message content and starts up the requested simulation model. The simulation output is returned as a JSON object, which will be sent on the output message queue to be handled by the Manager, stored in the Simulation Results store and processed by the Analyzer.

E. The JADES Simulation Analyzer cloudlet

The third cloudlet, Simulation Analyzer, is dedicated to the analysis of the simulation logs. It receives analysis requests from the Manager containing the ID of the job to be analyzed. The Analyzer retrieves the simulation task results for the job from the Simulation Results Key-Store and performs the required analysis (e.g., statistical measurements, sensitivity analysis, summary statistics) on them, writing back the aggregated results in the repository. The simulation analyzer is able to send out messages on the HTTP gateway queue to provide the analysis results to the final users. The analyzer is also capable of generating complex reports, charts and graphs.

IV. mJADES for Performance Prediction of Parallel Programs

This section will show the practical application of mJADES to performance prediction. First, the steps needed to derive a performance model for simulation are described using as case study the parallel Jacobi iteration. Then, the simulation results obtained through the performance model are shown, discussing the practical validity of the distribution of the concurrent simulation workload to computing resources hosted in a cloud.

A. Performance Modeling of the Jacobi Iteration

The Jacobi method is an iterative solver used to compute a solution of the two-dimensional Laplace equation by the finite differences method [28]. Given a 2D grid representing the region of interest, a stencil code can be used to update the interior points of the grid to the average of neighboring points, until convergence is reached. A parallel implementation of the algorithm can be easily constructed by assigning rows of the grid to cooperating processes. To perform the stencil computation, each process needs two boundary rows not belonging to its subdomain (the last row of the preceding process and the first row of the following process). So, each process allocates space for its rows and a couple of “ghost” rows, which mirror the content of the boundary rows residing on neighbour processes. The updated values of these rows must be received at every iteration. It is worth noting that even if the parallel Jacobi iteration is not the best algorithm for the numerical solution of systems of linear equations, its structure (row-wise decomposition plus ghost rows exchange) is very similar to the one used in more efficient methods, as Conjugate Gradient or Multigrid. The structure of the algorithm is thus made of the following operations, repeated in a loop:

- Communicate ghost rows: The processes exchange boundary rows to be able to compute their internal rows. This is done through send and receive primitives.
- Local computation: Processes update the internal points; a helper grid is used to store the new values. At the end of the iteration, the helper grid is used to update the main grid and the local error is computed.
- Global error computation: All local errors are compared to find the maximum error. If the value of the global error is under a given threshold the processes terminate, otherwise a new iteration is started. To compute the global error, an allreduce collective operation using the maximum function is inserted.

Starting from this specification, the Jacobi simulation model can be derived by modeling the three steps of the algorithm. To model the computation cost, a regression model has been constructed from measurements of the execution time of sample iterations for various number of processes. To model communication, simulated communication calls must be derived that correspond to ghost rows exchange and global error computation through reduction. A framework for parallel programs modeling has been developed on the top of the basic mJADES platform. This allows mJADES simulation processes to model local computations (CPU/memory operations) and communication (message-passing calls) by simulated delays. The framework can be used to construct model instances that mimic the behavior of a given application.

B. Experimental results

The performance study carried out on the mJADES platform has focused on two main issues:

- assessing the effect of the number of processes into which the application is decomposed. Each process is supposed to be allocated on a dedicated CPU;
- assessing the effect of the interconnect, switching between a Fast Ethernet (100 Mbit/s) and a Myrinet [29].

The two interconnects have been simulated for every number of processes in the range of interest on the experimental platform described in Figure 2. Simulation requests for the Jacobi performance model are submitted through the HTTP gateway component. In all the experiments, the problem grid size was fixed to 2000x2000 points. For the first analysis,
a sweep was configured over the $nP$ parameter, which represents the number of simulated processes. For the second analysis, a sweep was performed over a compound parameter representing latency and bandwidth of the interconnection network. The values of these parameters for Fast Ethernet and Myrinet networks were obtained on our PoweRcost cluster through benchmarking.

The simulation request is translated into a simulation job, which is decomposed by the mJADES manager into a sequence of simulation tasks. Each task corresponds to a unique combination of values for number of processes and latency-bandwidth. These tasks are sent to mJADES core cloudlets, where the simulations are performed. The mJADES manager is configured to use $nC$ instances of the core cloudlet. Tasks are distributed using a dynamical scheduling approach (the task at the head of the queue is handed to the first idle cloudlet). The details of the workload distribution to cloudlets are not trivial, since the time required to execute a simulation task is not constant, but grows with the number of processes to be simulated. To help load balancing, the tasks are handed from the largest (the one for the highest value of $nP$) to the smallest. The experimental platform has been deployed on a testbed consisting of a private Eucalyptus cloud composed of four physical nodes hosting the virtual machines on which cloudlets are deployed. One node hosts the HTTP gateway and the mJADES manager cloudlet, while three nodes are dedicated to host instances of the mJADES core cloudlets.

Even if the objective of this paper is to stress the validity of concurrent simulation in a cloud, it is interesting to present the simulation results and to compare them to the actual results measured in the real target machine. We can’t refrain from pointing out once again that simulation results are practically indistinguishable from actual measurements, and so that performing long and exhausting performance testing on the target machine is often just a waste of time. Figure 3 shows the plots of the speedup of an MPI version of the code of the Jacobi iteration, as measured on the PoweRcost cluster (the machine that provided the measurements used in the regression) and as predicted by simulation. The maximum relative error throughout the range from 2 to 16 processes was 5.35%.

Simulation can be very effective but, on the bad side, launching a large number of simulation sessions for different values of the parameters can be as frustrating as running tests on the real hardware (the latter requires hardware availability, in addition). This is the point where the almost unlimited resources of clouds and mJADES come into play. Table I shows the effect of running a composite simulation task, a sweep over a number of processes ranging from 2 to 1024 in successive powers of two, on a variable number of cloudlets. Running all the 11 simulations on the same cloudlet requires about 412 seconds of simulation time, while using three cloudlets the time drops to 256 s. The overall time required for simulation is determined at large extent by the duration of the largest simulation ($nP = 1024$), which lasts 249 s when executed as a single simulation task. In practice, as the longest simulation is running in one cloudlet, the other two execute the smaller simulation tasks. The overhead introduced
by the workload distribution mechanism of mJADES is almost irrelevant. In our opinion, this is an interesting result, showing the substantial validity of the concurrent simulation approach, which spreads beyond the field of application presented here as an example.

<table>
<thead>
<tr>
<th>nP</th>
<th>Workload</th>
<th>Exec. Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>nP = 2 . . . 1024</td>
<td>411,768</td>
</tr>
<tr>
<td>2</td>
<td>nP = 2 . . . 1024</td>
<td>286,397</td>
</tr>
<tr>
<td>3</td>
<td>nP = 2 . . . 1024</td>
<td>256,283</td>
</tr>
</tbody>
</table>

**TABLE I: Execution times for nP = 2 . . . 1024**

V. CONCLUSIONS AND FUTURE WORK

In this paper we have shown a practical application of mJADES, an architecture for concurrent simulation on resources automatically leased from a cloud, mJADES relies on two existing technologies, mOSAIC and JADES, which provide cloud interfacing and discrete-event simulation capabilities, respectively. Their integration can be very useful for performing complex, repetitive simulation tasks.

The experimentation presented here shows that the concurrent simulation approach can be very useful in the context of the simulation of parallel software, making it possible to automatize and to speed up considerably the launch of a number of simulation tasks that would otherwise require long waiting times. The accuracy of the performance prediction that can be obtained by simulation (errors smaller than 5%) is not an absolute novelty, but deserves a mention. More important here is the possibility opened by mJADES to execute a huge number of simulation tasks in parallel with little or no user intervention. Our future work will try to automatize the software modeling step, and to optimize the mJADES prototype used for the experiments presented in this paper.

Acknowledgements

The Authors wish to thank Giuseppe Aversano for support in the experimentation and Institute e-Austria Timisoara (IeAT) for providing expertise and resources for the testbed.

REFERENCES


