International Journal of Advances in Computer Science and Its Applications– IJCSIA
Volume 4: Issue 2 *[ISSN: 2250-3765]*
Publication Date : 25 June 2014

On the Scalability of Simulating Cloud Systems

[1]Alberto Núñez, [1]Manuel Núñez, [1]Mercedes G. Merayo and [2]Sergio Núñez

***Abstract*—Simulation of cloud computing systems remains to be a challenge due to the high number of issues that hamper this task. Basically, the main issue for achieving simulations of cloud systems is two-fold. First, the enormous amount of time required to execute those simulations. Second, the large amounts of memory required for simulating the high number of elements that constitute the model. In most cases, those systems contain thousands of computing nodes, tens of storage nodes, communication networks, and communication switches, whereof the algorithms required for modelling and simulating all those elements require huge amounts of CPU power. In this paper we present a complementation for INET, a framework used for modelling and simulating networks in cloud systems models.**

***Keywords*—Simulation, Modelling of Cloud Systems**

I. Introduction

Nowadays, cloud computing systems are increasing their role due to the fast evolution on computer networks and communication technologies. Those systems grow more complex with each generation, becoming a difficult and time-consuming task analysing and predicting their behaviour. It makes simulators a very important choice for designing and analysing large and complex architectures. Network behaviour is one of the main assets in this kind of simulations. On the one hand, network performance has a critical impact on the overall system performance. On the other hand, the strategy used for modelling and simulating the network system is the key for obtaining a simulation with the level of detail, performance and scalability required.

Simulation of cloud systems is currently a fashion topic. In most cases, the simulations require a great computer power to be executed. Basically, the main issue for achieving simulations of cloud systems is two-fold. First, the enormous amount of time required to execute the simulation. Second, the large amount of memory required for simulating the vast number of elements that constitute the model. In most cases, systems contain thousands of computing nodes, tens of storage nodes, communication networks, and communication switches, whereof the algorithms required for modelling and simulating all those elements require huge amounts of CPU power.

[1] University Complutense of Madrid, Spain

[2] University Carlos III of Madrid, Spain

The state of the art shows several simulation frameworks for modelling these kind of systems. All of them ease many of the tasks required to develop simulations. One of these simulation frameworks that is gaining momentum is the INET simulation framework [1]. INET has been developed as a module on top of the more generic OMNeT++ simulation framework [2]. The INET framework allows an easy development of computing network simulations. It also allows a high level of detail and it can be scaled, with little effort, up to the existing computer power. Furthermore, INET can easily work with any other simulated subsystem built on top of OMNeT++ (processor, memory, I/O, power management, etc).

OMNeT++ is a C++ discrete event simulator designed for modelling computing networks and parallel and distributed systems. OMNeT++ is adequate for large-scale simulations because it uses hierarchical models and reusable components. The INET framework is an open-source network simulation package built upon OMNeT++ that uses the same concept: modules communicating by message passing. This framework contains IPv4, IPv6, TCP, UDP protocol implementations. Supported link-layer models are PPP, Ethernet and 802.11.

The main problem with the INET framework is its lack of scalability, because it is not adapted to develop parallel simulations. Thus, it is limited to the capacity of a single computer, even though the underlying OMNeT++ offers facilities to develop parallel simulations. For this reason we decided to parallelize the INET framework by removing the existing scalability restrictions and providing the capacity for developing parallel simulations.

The limitations found in INET when we tried to simulate large-scale systems are in the genesis of this work. In order to accomplish our research studies, we demonstrate the usefulness of our proposal by using the iCanCloud simulator [3] to simulate, in parallel, cloud systems following the methodology described in this work.

The rest of the paper is structured as follows. Section II presents some related works. Section III describes the motivation for increasing the performance for simulating highly distributed systems. Section IV describes our proposal to increase the scalability for simulating large scenarios. Section V shows several performance experiments. Finally, Section VI presents some conclusions and future work.

II. Related Work

At present we have found in literature some works to increase both scalability and performance in simulations.

The authors of [4] describe an ad-hoc simulation approach, as well as an optimistic synchronization algorithm. An ad-hoc distributed simulation is a collection of autonomous on-line simulations brought together to model an operational system. They offer the potential of increased accuracy, responsiveness, and robustness compared to centralized approaches. This work differs from conventional distributed simulations in that it has been created bottom-up rather than top-down. Other works based on online simulation are DDDAS [5] and the LEAD project [6].

The work described in [7] presents a parallel SystemC simulation kernel, which is implemented using parallel programming techniques and leverages the parallel execution capabilities of multi-core machines to speed up hardware simulation. In this work [8], the authors propose a machine-learning algorithm as an aid in making decisions about the simulation execution. The algorithm is based on the well-known K-Nearest Neighbour algorithm. After an extensive training regime, it was shown to make a correct prediction 99% of the time on whether to use a parallel or sequential simulator. The author of [9] describes a parallel processing method for simulations of large-scale networks with a hybrid traffic representation combining both a time-stepped fluid model and a discrete-event packet-oriented model. This paper also shows the benefit of the parallel hybrid model through a series of simulation experiments of a large-scale network consisting of over 170,000 hosts and 1.6 million traffic flows on a small parallel cluster.

III. Motivation and main goals

The worst-case scenario for simulating cloud systems arises when those simulations require a great level of detail and a high level of scalability. In these cases a great computer power is necessary and performing those kinds of simulations in a single computer, using sequential simulation techniques, is not feasible due to the lack of scalability. This limitation lies basically in the computer power and memory provided by the single machine in which simulation is executed. In this case, large-scale models can take weeks, and even months of simulation time, therefore, much more CPU power is required to obtain results in a reasonable time-frame.

Nowadays, the best way to obtain this kind of computing power is to use parallel computing systems, like clusters. Those systems require the developing of parallel applications that differs from traditional applications. Parallel applications consist of several computing processes that work coordinated by interchanging information (usually using message passing methods). Then, a feasible solution is to use parallel simulation techniques for simulating large-scale environments that require both great amounts of processing power and memory. This is the reason why simulators of distributed systems must provide scalability. In this context, scalability means whether the corresponding simulator is able to simulate with enough performance large-scale systems, by increasing the number of machines, which make up the corresponding architecture to be simulated. Likewise, performance determines the speed which a simulator executes a corresponding simulation. In general, the larger the size of the architecture to be simulated, the greater the time needed to execute the simulation.

In this paper we propose an approach to perform parallel simulations of large-scale models using the INET framework. As we said in section 1, SIMCAN has been built upon the OMNeT++ framework. The reason to choose OMNeT++ as base framework is that it is popular in academia for its extensibility since it is also open sourced and there are plentiful online documentations. Moreover, several open source simulation models have been published in the field of network simulations such as IP, IPv6, MPLS, mobility and ad-hoc simulations. Similarly, the INET framework is a well-known framework for modeling and simulating the network system. Moreover, a lot of research works use it, as ReaSE [10,11], MiXiM [12], xMIPv6 [13] and overSIM [14], just to name few. Therefore, our proposal is not only focused on a specific simulation platform. Instead, any simulator or tool that uses the INET framework for modeling and simulating the network system can take advantage of this approach for performing parallel simulations.

IV. Increasing scalability in large simulations

OMNeT++ simulation programs have to assign manually which modules will share the same computing node. This is done by classifying the modules into logical partitions (LPs). Each logical partition will be executed on a different computing node. This classification has to be performed before the execution of the simulation, because there is no way to perform a dynamic classification during execution time.

In order to locate and reference the remote modules, OMNeT++ uses "placeholders". An OMNeT++ placeholder is just a proxy of a remote module placed on the partition instead of the original one. Its mission is to forward all the messages to the remote module itself. All the partitions of the computing nodes have a placeholder for each one of their remote high-level modules. A compound module can have their sub-modules spread across some nodes. In this case, the compound module is duplicated on all the corresponding nodes, and the remote sub-modules are replaced by placeholders.

The INET framework is not able to perform parallel simulations. These simulations cannot be executed on a single computer because of the lack of memory or the computing time required. OMNeT++ applies the message-passing paradigm to implement parallel simulation (MPI [15]). The main issue of performing parallel simulations lies in the method of assigning both network and MAC addresses to the nodes and switches of the modelled environment.

International Journal of Advances in Computer Science and Its Applications– IJCSIA
Volume 4: Issue 2 [ISSN: 2250-3765]
Publication Date : 25 June 2014

It is important to remark that, in parallel simulations, the model is split in different sub-domains, called partitions, in which a part of the model is executed in a single computer. Therefore, using different machines, each executing a part of the complete model, the simulation can share resources of these nodes, like CPU and memory. However, the performance of using parallel simulation must consider the overhead caused by synchronizing these partitions.

Since each partition is in charge of managing a sub-set of modules from the model, the local view of each partition is not able to see the rest of the modules. Hence, a local partition does not know how many other nodes or switches must be configured in foreign partitions. Therefore, the main difficulty lies in how to assign IP and MAC addresses without repeating the same address in different partitions.

Currently, INET uses a module called Flat Network Configurator (FNC). This module assigns IP and MAC addresses automatically in execution time. This solution works fine for sequential simulation because the FNC module has access to all modules involved in the simulation. Thus, both IP and MAC addresses can be assigned. However, this is not a feasible solution for simulating cloud systems, where thousands of physical machines have to be simulated. The problem arises in parallel simulations, where each FNC is executed in a different partition. Each FNC only has access to those modules executed in the same partition, but not to the rest of the modules simulated in other partition. Hence, a mechanism of synchronization is required among all the partitions to cooperate and assign the corresponding IP and MAC addresses without collisions.

Figure 1 shows a network model made up of 8 computing nodes and 2 switches. In this example, the simulation is executed sequentially. The FNC has access to all modules that compose the model and it can assign IP and MAC addresses to the modules that need one.

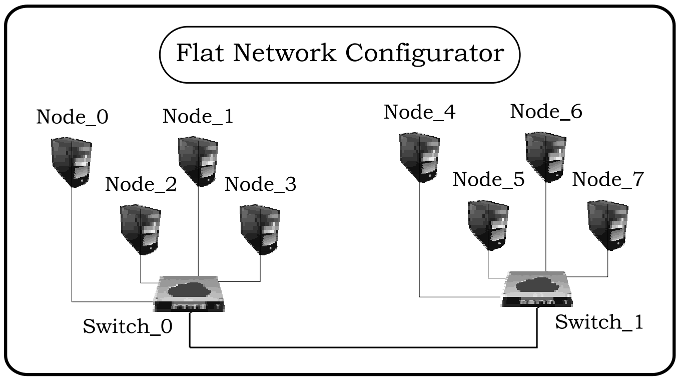

Fig 1 Sequential simulation example

Figure 2 shows an example of a parallel simulation corresponding to the model shown in Figure 1. In this example the complete model is simulated using two LPs, where LP_0 is in charge of simulating switch_0 and computing nodes 0, 1, 2 and 3. LP_1 is in charge of simulating switch_1 and computing nodes 4, 5, 6 and 7.

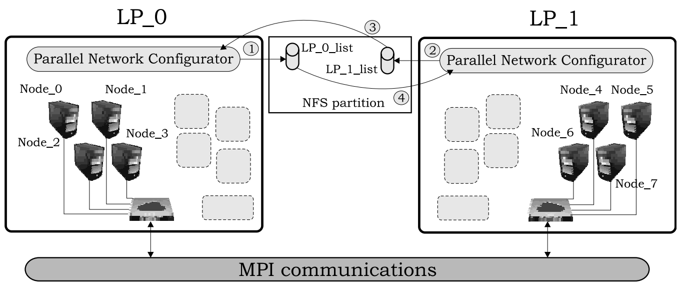

Fig 2 Parallel simulation schema

From the LP_0 point of view there are only 5 modules: Switch_0 and node 0, 1, 2 and 3. LP_0 knows that other 5 modules exist, but it does not know the type of the modules, or if those modules need an IP address or a MAC address. These kind of modules are placeholders, represented in Figure 2 by grey squares. The FNC of each partition has no access to the complete list of modules that need an IP or a MAC address. Consequently, it cannot assign the addresses correctly, and it cannot avoid repeated addresses.

We propose a method to assign appropriately both IPs and MAC addresses to all the nodes and switches in a distributed cloud model. The solution basically consists on using a new module called Parallel Network Configurator (PNC). The main purpose of this module is to build a complete list of those modules that require an IP address or a MAC address. Each LP must contain one PNC module (see Figure 2).

At initialization stage, each PNC builds a list that contains the local modules that require an address. Then, this module writes the list to a file, which is stored in a file system shared by all partitions (step 1 and 2).

When all those files are created, each PNC has to read them (step 3 and 4). Those files are merged with the purpose to create a unique list for each kind of addresses. Therefore, each PNC contains a complete list of those modules that need IP or MAC addresses. In this way, each PNC can assign both IP and MAC addresses correctly by avoiding collisions.

Finally, this process is used when a pair of modules, placed in different LPs, requests to create a network connection. In this case, the PNC module offers the facility to translate the name of the destination module to its corresponding IP address. This is done by seeking on the list those modules that use IP address created by the PNC.

V. Performance experiments

In this section, we report the results of a set of simulation experiments that have been perrformed. The main goal of those experiments is to check both the scalability and performance obtained using the proposal described in this work.

A data-center of a cloud system has been modelled by using both INET and iCanCloud. This model represents a data-center that consists of 6 elements:

- **Rack**: It is used in order to group computing nodes in large-scale systems. Due to managing large amounts of nodes hampers the task of configuring the entire system, racks contain a set of board nodes in order to ease deploying tasks.
- **Board node**: It is used with the same purpose as racks, for grouping and managing nodes. This element contains a set of computing nodes and one switch. The switch is used for interconnecting the set of nodes inside a node board with the rest of the architecture.
- **Node**: It is used to perform processing. Usually the nodes contain a set of CPU cores for executing several processes in parallel.
- **Storage node:** These elements are used for managing data.
- **Switches**: They are used for interconnecting the elements of the distributed architecture.
- **Communication network**: It defines the speed of each communication link in the architecture.

The modelled data-center contains 1024 nodes, grouped in 8 racks, where each rack contains 8 board-nodes. Each board node contains 16 nodes. The number of storage nodes used is 16. Finally, the network used is an Ethernet 10 Gbps.

The behaviour of a map-reduce based application has been modelled to achieve the simulation experiments. Basically, the application model used in this work is a simplified version of the map-reduce model proposed by Google [16]. This model uses an initial data-set as the size of the problem. By size of the problem we mean the amount of data that has to be processed in order to accomplish the execution of the application completely. The size of the data-set used in those experiments is 128 GB.

In the experiments, the complete model is distributed among a set of LPs. Each LP is executed on a single machine, and MPI is used to communicate and synchronizing the set of LPs. In order to split the model in a set of LPs, every element of the model has to be allocated to a specific LP. We have considered the next three modules as indivisible blocks to perform this process: racks, switches and storage nodes. That is, all the elements contained in the same block will be executed in the same machine. For instance, all nodes included in the same rack are located in the same LP, therefore all of them will be executed in the same machine.

The simulation experiments have been executed using two different methods for splitting the model. Each method uses a different strategy for allocating each block of elements to a LP.

The first method consists on allocating each block of elements to a corresponding partition using a random distribution. On the contrary, the second method uses domains to achieve the allocation of each block. In this case, we define domain as a group of elements which probability to share a communication link to interchange data is high. For example, two racks that are connected to the same switch have a high probability to interchange data between them. The main goal of this method is to reduce the communication overhead among communicating elements executed in different LPs.

The simulation has been executed in a 16-node cluster. Each node contains an Intel Xeon CPU 2.00Ghz and 4 GB of RAM memory.

Figure 3 shows the execution time of the simulation experiments using the two previously described allocation methods. First, we can see that using parallel simulation, the performance obtained is clearly improved. That is due to the fact that several machines work in parallel for simulating the complete model. This chart also shows that the improvement of the performance obtained when the number of LPs increases does not follow a linear tendency. This is caused because of the communication overhead, necessary to synchronize all LPs in the model. The greater the number of LPs, the greater the overhead to synchronize the complete model. However, this overhead can be alleviated by using the technique of grouping elements into domains, instead of allocating blocks to LPs randomly.

When 8 and 16 LPs are used, performance obtained is practically identical. The cause of this lies in the architecture of the model to simulate, which contains 8 racks. Therefore, the level of parallelism cannot be increased by distributing the racks in 16 LPs. Furthermore, the level of parallelism obtained for simulating the storage nodes in parallel is lost by the overhead caused to synchronize the LPs that contain them.

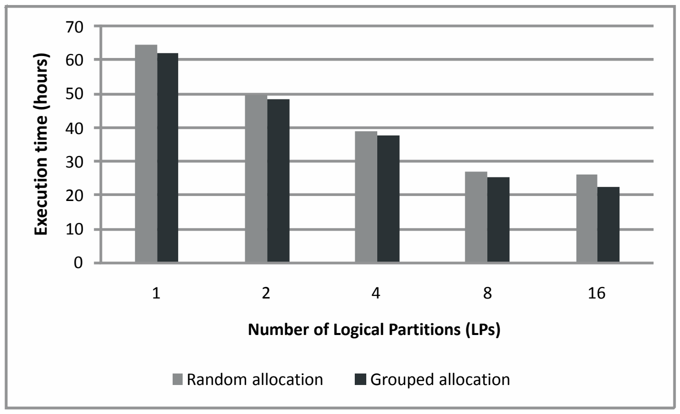

Fig 3 Time required to simulate the cloud model

Figure 4 shows the amount of memory required for executing each experiment. Those results show that the memory requirements for both allocation methods are practically the same. Using a grouped allocation consumes slightly less memory than random allocation, but it is almost insignificant. It is worth to note that parallel simulations require almost the same memory that sequential simulations. However, using parallel simulation provides more scalability because the model is not limited to the memory provided by a single machine. Instead, the sum of the memories where the simulation is executed can be used.

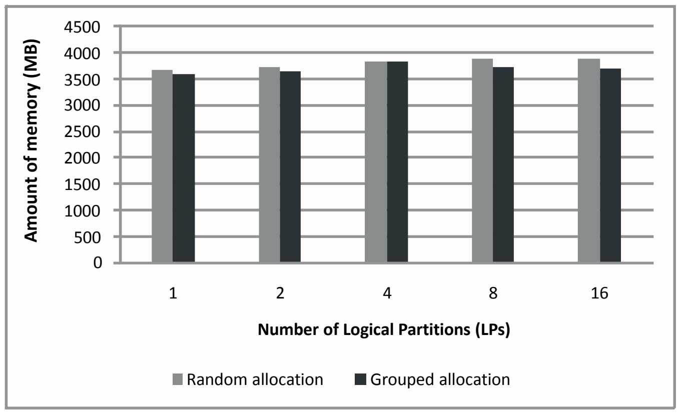

Fig 4 Amount of memory required to simulate the cloud model

VI. Conclusions and Future work

In this work we have presented the parallelization of the INET framework. This method consists on generating both IP and MAC addresses appropriately among the different partitions that execute the simulation. In order to demonstrate the usefulness of our proposal, a cloud computing data-center has been modelled and simulated.

The solution developed relies on a Parallel Network Configurator that alleviates the problem of configuring the network features in a large environment. The results of the simulation show that our proposal increases both the scalability and performance for simulating cloud systems.

Future works will include increasing the size of the model and the number of nodes for performing parallel simulations.

Acknowledgment

This research was partially supported by the Spanish MEC projects TESIS (TIN2009-14312- C02-01) and ESTuDIo (TIN2012-36812-C02-01) and the Spanish Ministry of Science and Innovation under the grant TIN2010-16497. We would also like to thank Jesús Carretero for his assistance this work.

References

[1] (2013) Andras Varga. INET Framework. "http://inet.omnetpp.org".

[2] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 1–10.

[3] (2013) The iCanCloud simulator. http://www.arcos.inf.uc3m.es/ /~icancloud/Home.html

[4] R. Fujimoto, M. Hunter, J. Sirichoke, M. Palekar, H. Kim, and W. Suh, "Adhoc distributed simulations," in PADS '07: Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation. Washington, DC, USA: IEEE Computer Society, 2007, pp. 15–24.

[5] D. Brogan, P. Reynolds, R. Bartholet, J. Carnahan, and Y. Loitire, "Semi-automated simulation transformation for DDDAS," in *In Computational Science - ICCS 2005: 5th International Conference*. Springer-Verlag, 2005, pp. 721–728.

[6] Realization of Dynamically Adaptive Weather Analysis and Forecasting in LEAD: Four Years Down the Road, vol. 4487/2007. Beijing, China: Springer, 05/2007 2007.

[7] E. P, P. Chandran, J. Chandra, B. P. Simon, and D. Ravi, "Parallelizing SystemC Kernel for Fast Hardware Simulation on SMP Machines," in *PADS '09: Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 80–87.

[8] Q. Xu and C. Tropper, "On determining how many computers to use in parallel VLSI simulation," in *PADS'09: Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 122–128.

[9] J. Liu, "Parallel simulation of hybrid network traffic models," in *PADS'07: Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 141–151.

[10] T. Gamer and M. Scharf, "Realistic simulation environments for IP-based networks," in Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 83:1–83:7.

[11] T. Gamer and C. P. Mayer, "Large-scale evaluation of distributed attack detection," in *Simutools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 68:1–68:8.

[12] K. Wessel, M. Swigulski, A. Kpke, and D. Willkomm, "MiXiM The Physical Layer An Architecture Overview," in *2nd International ICST Workshop on OMNeT++*. ICST, May 2009.

[13] F. Z. Yousaf, C. Bauer, and C. Wietfeld, "An accurate and extensible mobile IPv6 (xMIPV6) simulation model for OMNeT++," in *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 88:1–88:8.

[14] I.Baumgart, B.Heep and S. Krause, "OverSim: A Scalable and Flexible Overlay Framework for Simulation and Real Network Applications," in *IEEE P2P'09: Proceedings of the 9th International Conference on Peer-to-Peer Computing*, H. Schulzrinne, K. Aberer, and A. Datta, Eds. IEEE, 2009, pp. 87–88.

[15] W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. N. amd W. Saphir, and M. Snir, *MPI: The Complete Reference*. MTI-Press, 1998, vol. 2 – The MPI-2 Extensions.

[16] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, January 2008.