DARTCSIM: AN ENHANCED USER-FRIENDLY CLOUD SIMULATION SYSTEM BASED ON CLOUDSIM WITH BETTER PERFORMANCE

Xiang Li, Xiaohong Jiang, Peng Huang, Kejiang Ye

College of Computer Science, Zhejiang University, Hangzhou 310027, China
{lixiang2011, jiangxh, phuang, yekejiang}@zju.edu.cn

Abstract: The research interest of cloud computing is increasing tremendously in recent years both in academic and industry, however, it's difficult to experiment or evaluate the performance in a real cloud environment due to its large scale and complex deployment. Some simulation tools such as CloudSim, OptorSim, and GreenCloud etc. are often used to evaluate the performance of cloud centers. CloudSim is one of the most powerful and open-sourced platforms in cloud computing simulation. But CloudSim is not easy to use from the users' perspective because a user needs to know the source code well and write hundreds lines of code to complete even a simplest simulation. We developed an enhanced cloud simulating system DartCSim based on CloudSim with a more user-friendly interface, which allows a user to configure all the data of the simulation environment with a visual interface, including the configurations of network cloudlets, network topology, and the algorithms for managing the cloud center.

Keywords: Cloud computing; Simulation; CloudSim; Performance modeling

1 Introduction

Nowadays, cloud computing is almost everywhere arranging from private cloud to public cloud. Most companies provide cloud service with large-scale data centers supported. According to Emerson Network Power, there are more than 500,000 data centers all over the world, with 285.8 million square feet of space in 2011 [1]. It is important to evaluate the efficiency of the hardware deployments of a cloud center and the performance of proposed algorithms for cloud management. But the problem is that it's difficult to do the evaluations in a real cloud environment because the costs of the experiments in a real cloud center are unsustainable. What's more, due to the complicity of the cloud environment, the experimental results may not be repeatable under the influence of many uncertain factors. Two solutions can be used to solve this problem. One is to use cloud specific simulators currently available like CloudSim [2], OptorSim [3] and GreenCloud [4]; the other way is to use a test-bed, like Open Cirrus [5] or Google/IBM cluster. A test-bed which consists of a large number of hosts and lots of networking resources is very powerful in computing and storage. The experimental results from test-beds could be more convincing than that from simulation. However, conducting experiments on such large-scale real cloud test-beds is unrealistic for most of the researchers. So simulator is still a very essential tool in the research of cloud computing.

CloudSim is an event based simulator, all functions such as creation of virtual machines (VM), allocation of cloudlets (the basic unit of task in cloud computing), etc. are processed as events. Compared with other cloud simulators, CloudSim has the following main advantages which have established its reputation in simulation of cloud computing: (1) it models nearly all the resources in cloud center, including host, cloudlet, broker, etc.; (2) it provides the power-aware model [6] and the network model [7] for simulations; (3) it is easy for the users to implement a certain algorithm; (4) it takes CloudSim only seconds to run a common simulation.

However, we found a fetal weakness when using CloudSim: bad usability, which affected its popularity a lot. To overcome this weakness, we developed an enhanced simulator DartCSim to allow users to configure all the data in a visual interface, and record their experimental configurations and experimental results. The enhancements for the CloudSim 3.0 are summarized as the following.

1) DartCSim provides a user-friendly interface, which greatly improve the usability of CloudSim.

2) DartCSim maintains all the data of configurations so that the users can repeat their experiments at any time via importing a previous project, or restart a new experiment via importing a previous project and modifying the configurations.

3) DartCSim supports rewriting all the algorithms provided by CloudSim by filling a corresponding template.

4) DartCSim provides an intuitive interface for the users to preview the construction of a data center, especially the topology of the networks. It also provides a clear view of the simulation results.

5) We reconstructed the code of CloudSim in order to eliminate the problem of hard coding. Thus to improve its modularization and to provide the access interfaces for the front-end module.

978-1-4673-1857-0/12/$31.00 ©2012 IEEE

This paper is organized as follows. In Section 2, we outline the basic information about CloudSim, including the general steps of processing a simulation. Section 3 introduces DartCSim and presents the architecture of DartCSim. Section 4 describes the implementation of DartCSim in detail. Section 5 makes a conclusion of our work and gives the future work in DartCSim.

2 Introduction of CloudSim

2.1 CloudSim architecture

CloudSim is one of the most popular simulators for cloud computing because of its sophisticated modeling, including the modeling of the hardware, cloudlets and algorithms.

The components of CloudSim are organized in three levels. The bottommost layer provides the event based simulation foundation, including the timer and queues for each entity. The second layer consists of five sub-layers. (1) User Interface layer provides APIs for the users to build simulation code; (2) Virtual Machine Services layer is responsible for cloudlets management in the virtual machines; (3) Cloud Services layer provides scheduling policies and management for cloud resources; (4) Cloud Resources layer refers to the hardware in the data center and the supports for the mechanism of event which will be discussed latter; (5) Network Modeling layer which is introduced in CloudSim Version 3.0. The layer above all is the code written by users, defines the scenarios and configurations of the cloud environment with certain algorithms.

2.2 Processing of CloudSim simulation

CloudSim is an event based simulator, i.e. the simulation is driven by events transmitted between the entities like the hosts and the switches, etc. Figure 1 demonstrates the detail process of a simplest simulation in CloudSim, focusing on the interactivities between the cloud broker and the data center, and omitting some irrelevant details like registering process of the entities. In this case, there is only one data center and only one cloudlet in the virtual machine. Note that the future (event) queue and the deferred (event) queue in CloudSim are expressed as “future” and “deferred” in Figure 1.

The simulation is processed as follows. After registering the entities (Step 1), the broker will query the detail characteristics of the data center and wait for a response from the data center (Step 2). Then after receiving the response in Step 3, the broker tries to create the virtual machines by sending VM_CREATE_ACK signal in Step 4. If all the virtual machines have been created in the target data center (Step 5), the broker then submits the cloudlets (or application in network CloudSim) to the data center to execute (Step 6), if not, the broker would keep trying to request for the creation of the rest virtual machines. After then, the data center could begin processing the cloudlets or applications (Step 7). CLOUDLET_RETURN signal will be sent back to the broker (Step 8) with the execution result if any cloudlet has been finished.

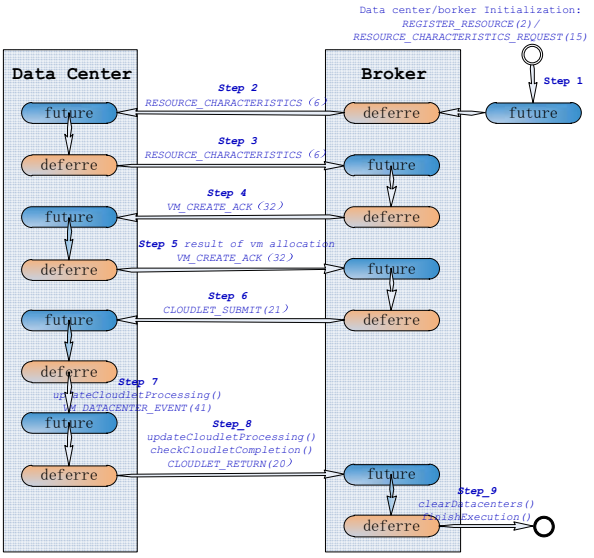

Figure 1 Steps of processing in a CloudSim simulation

The future queue and the deferred queue are of vital importance in CloudSim. CloudSim can complete the simulation only when finishes processing these two queues iteratively.

3 Simulate cloud computing with DartCSim

3.1 Introduction to DartCSim

DartCSim is a user-friendly simulation platform for cloud computing based on CloudSim. Users can use CloudSim in a visual way that all the simulation experiments or tests can be done via visual configuration without coding at all. Thus they can focus on their experiments and algorithm designs other than spending a lot of time on understanding the source code.

DartCSim hides the simulation details and provides users a very friendly interface to build the project. In general, there are four steps needed to be done by the users to complete the configuration: (1) provide basic information of the simulation, such as project name, the store path, etc.; (2) provide information of the hardware, including the hosts and the networks; (3) provide information of the virtual machines including the capacity of CPU, memory, etc.; (4) provide information of the cloudlets or the applications. DartCSim helps the users to create various kinds of cloud resources and entities in the simulation. Moreover, DartCSim provides a very intuitional way to view the topology of the networks the user configured and the results of the simulation. Besides, all kinds of algorithms supported in CloudSim can be overridden in a template by users through DartCSim. The interface of DartCSim provides three kinds of configuration guide to meet different requirements and construct a simulation quickly.

3.2 Architecture of DartCSim

DartCSim is a multi-tiered, complete system which integrates the front-end (user interface) and the back-end (simulation core). As it is illustrated in Figure 2, DartCSim can be divided into four levels: user interface, JNI, CloudSim and persistence. In addition, we use DartCSim management to represent the functionality of project management, data management, etc. Note that the components in brown are the original CloudSim (Version 3.0) modules, and all the other modules are newly added components in DartCSim.

The above layer is the user interface which provides a visual window to allow users to input simulation configuration data and view the simulation results in an easier way. When the user has input all the configurations and starts the simulation, the parameters will be passed to the back-end (CloudSim). CloudSim takes over the task, run the simulation, and send back the results after completing the simulation.

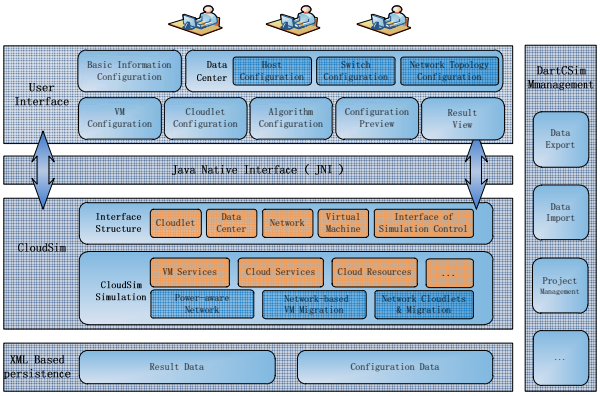

Figure 2 Architecture of DartCSim

The kernel level is CloudSim which performs simulation in the back-end. We add three new modules in this layer. (1) Power-aware network module provides power consuming model in a network simulation. (2) Network-based VM migration module simulates more real virtual machine migration through networks considering the network utilization and overheads. (3) Network Cloudlets & Migration module supports both the network cloudlets (applications) and network based migration at the same time. We will introduce the implementation details of the three newly added modules in another paper due to space limitation of this paper.

We use Java native interface (JNI) [8] to let the front-end and the back-end communicate with each other because CloudSim is coded in java, while the user interface is coded in Qt, a C++ graphic framework. The bottommost layer is responsible for data persistence, i.e. store all the data including configurations of the simulation projects and the corresponding results.

The right part of Figure 2 means the DartCSim management, through which we can import the certain data to the simulation project, export and store the configurations and result data in persistence layer. We provide a navigation pane to manage all the available simulation projects. The users could view all the information for the configurations and the result of each project. Users can also modify the configuration data and run a previous simulation again.

4 Implementation of DartCSim

This section introduces the functionalities and the implementation of DartCSim in detail in top down order according to the architecture we introduced in the last section.

4.1 User interface

User interface is implemented on the top level of the Architecture. A complete and integrated interface is used to configure the data of a simulation project and to view the result returned back from simulation back-end. The graphic interface is written in Qt which is a cross platform application and C++ based UI framework [9]. The main reasons that we choose Qt are that (1) Qt can bring us excellent user experiences; (2) Qt is a cross platform framework which allows us to target multiple platforms with coding only once.

1) Functionalities and implementation—The user interface provides the following functions.

a) Configure the simulation. The configuration includes basic information, data center information (networks included), virtual machines information, cloudlet or application information and algorithm information. The basic information of a simulation consists of the name of the project, the path specified by the user to save the project and the type of the simulation project (we will discuss about this later). In order to configure a data center, the user should also configure the characteristics of the hosts and the networks. And the characteristics of a VM include its operating system, capacity of CPU and Memory, etc. In the basic type simulation, we use cloudlet but not application or network cloudlets.

b) Configure and preview the cloud topology. Through DartCSim, the user can configure the topology of the cloud center by input the relationship and the dependences between the nodes (hosts, VMs, or switches). After doing that, DartCSim provides a preview of the topology of the cloud center. Moreover, once we click on a certain node, we can view the detail information of this node. DartCSim generates the preview of topology by Qt Graphics View Framework, which provides an item based approach to model-view programming and a surface for managing and interacting with a large number of custom-made 2D graphical items. All the switch nodes in a data center and the links between them are drawn by QGraphicsItem class provided by Qt API. The main features in the function of topology preview are as follows. (1) It has a clear and concise structure based on three-level topology. (2) The scale of the network could be extremely flexible. DartCSim can manage all the

components in the network and it automatically choose which component to be shown and which to be hidden.

c) View the result of a simulation. After all the required configurations, DartCSim could run the project if no errors exists. The result of the simulation will be shown in DartCSim in the corresponding page, including the results of the cloudlets, data centers, and some statistics like the overall debts, success rate of cloudlets, etc. DartCSim shows the results in the auto-generated forms.

d) Override the algorithms provided by CloudSim. As we know, CloudSim provides the users with a variance of algorithms, including the algorithm for allocating the virtual machines in a data center and the algorithm of cloudlet scheduling, etc. However, the algorithms provided by CloudSim are naive and simple that cannot meet the user' requirements of using intelligent or optimized algorithms. So CloudSim need to be extended with interfaces to integrate user defined algorithms. We achieved this by providing a configuration guide. Firstly, we give the user a list of the methods in the parent class and let the user choose which to override. And then we render a code editor to the user. DartCSim generates as much code as possible before the user' editing, including the copyright information, declaration of the methods choose by the user (and the methods which must be overridden), etc. After that, we compile the code of the algorithm written by the user with the existed code of CloudSim by Java Development Kit (JDK). If any errors found by JDK, DartCSim would pop out a box to inform the user, if not, the algorithm will be compiled, saved and become an optional item, through which the user can choose his own algorithm to run the simulation.

2) Three simulation types—DartCSim provides three types of configuration guides: (1) basic type, (2) network type without power-aware model, and (3) power-aware network type. Table I shows the supported modules in each type.

Table I Architecture of DartCSim

Module	Basic Simulation	Application	Network Topology	Migration	Power-aware
Type A	Yes	No	No	No	No
Type B	Yes	Yes	Yes	No	No
Type C	Yes	Yes	Yes	Yes	Yes

The first type of the configuration guide in DartCSim provides the basic functionality. The user can conduct his basic simulation through this guide. The second type of configuration guide is called Network Simulation. This type uses the application instead of the cloudlet which is represented only by the instruction counts in the basic type. A typical application consists of lots of network cloudlets. There are several stages in a single network cloudlet. The so called stage in a network cloudlet is either computing, waiting for receiving a message, or waiting for sending a message. The messages are passed to a different host through the network layer as mentioned in Section 2. The last type we provided is the Power-aware Network simulation. This kind of simulation is based on the improved CloudSim, which has integrated the power-aware module and the network module together.

4.2 Java native interface

The middle layer, CloudSim, is actually the back-end of DartCSim that provides the simulation logic. As we know, CloudSim is written in Java. To solve the problem of interaction of the two layers which coded in C++ and Java respectively, we employ JNI as the bridge to communicate with each other. It means that C++ could involve the Java methods and vice versa. Sun's Java Native Interface is the first API for building Java-enabled native applications that are binary compatible across multiple Java Virtual Machines [10]. Figure 3 shows the general steps of using Java in C++ code.

First, A Java file is created with the declarations and the implementations of the methods and then it is compiled to a class file. Users should get the signature of a method through javap before a Java method is invoked. Once the class file is built and the signatures of the methods are obtained, a Java Virtual Machine (JVM) can be created after including the header file, jni.h, which is provided by JDK in C++ code. Through the JVM and the signature of the function, the Java methods can be invoked freely.

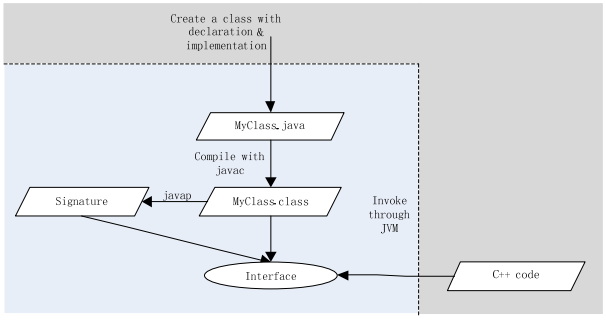

Figure 3 Steps of invoking java in C++

In our case, we first compile the CloudSim source files to class files. Then we create the Java Virtual Machine in C++ code and then invoke the methods in CloudSim. That is to say, we can invoke all the APIs provided by CloudSim in the C++ code which builds the user interface. For conciseness, we implemented a serious of classes in C++ to encapsulate the corresponding class in CloudSim. In each class, we provide methods with the same name in CloudSim. However, in each method, we only load the corresponding methods through JNI and JVM but not implement them respectively. The names of the classes and the methods we use in C++ are nearly

the same as those in CloudSim. So the methods can be used just like using the APIs in CloudSim. This is important because we separate the interface from implementation. Thus the follow-up developers of the user interface can use C++ classes created by us just like using APIs provided by CloudSim.

4.3 CloudSim

We implement the core simulation of DartCSim based on CloudSim 3.0. We extended the simulation functions in three aspects. First, we created new classes to represent the power-aware network entities so that the network models could support power model all the same. Second, we extended the migration simulation of the virtual machines considering the influence of network in the model of network-based migration. Third, we extended CloudSim to support both network cloudlets and migration implemented in the module of network cloudlet & migration. In addition, we modified some parts of the code in CloudSim to eliminate hard coding problem and improve the modularity to provide the front-end layer of DartCSim with concise interfaces.

4.4 XML based persistence

The bottommost layer of DartCSim architecture handles the task of persistence of the simulation project data. All the configurations and result data are stored in a user's specific folder in Extensible Markup Language (XML) format. We use the XML read and write interface provided by Qt. The Qt XML module is very convenient in parsing and generating a XML file. It provides a stream reader and writer for XML documents, and C++ implementations of SAX (Simple API for XML) and DOM (Document Object Model). We adopted the latter one because its convenience in operating XML files and it has better performance when the scale of data is not large.

4.5 DartCSim management

DartCSim management module provides the following functionalities: (1) export the data of configuration to the specified XML file; (2) import data of configuration from the XML file stored before so that the user can reuse the previous data easily; (3) manage all the simulation projects by providing a navigation through which the user can review, edit, delete and restart the project. Note that the configuration data could be imported or exported in any level. For example, the user can import the data of a single CPU or alternately import the data of a whole data center.

5 Conclusions and future work

We developed an enhanced cloud simulation platform based on CloudSim 3.0, extending the functionalities with user-friendly interface and new modules. There are still future work to do to improve the functionality and performance of DartCSim. We here figure out several future directions in the improvement of DartCSim.

1) DartCSim should be improved to provide more user customized options for choosing the usage of the functions provided by CloudSim. We now provide three types of simulations in DartCSim, which may not cover all the requirements of the users.

2) The network architecture of CloudSim is based on three-level topology. More flexible network topology should be supported by CloudSim. On the other hand, DartCSim should provide a way which is more easy-to-use to configure the topology of the network, such as through dragging, etc.

3) Power-aware CloudSim provides several algorithms to select a host to be the migration target [6]. However, all these algorithms do not take the network into consideration. We should also implement some network based selection algorithms [11] in the future.

Acknowledgements

This work is supported by National High Technology Research 863 Major Program of China (No.2011AA01A207) and National Science Foundation of China (No.61272128).

References

[1] Emerson Network Power, http://www.emersonnetwork power.com

[2] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environment and Evaluation of Resource Provisioning Algorithms. Software: Practice and Experience (SPE), 41(1), pp, 23-50, 2011.

[3] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar,K. Stockinger, and F. Zini. OptorSim: A Grid Simulator for Studying Dynamic Data Replication Strategies. International Journal of High Performance Computing Applications, 17(4), pp, 403-416, 2003.

[4] D. Kliazovich, P. Bouvry, and S. Khan, Greencloud: A Packet-level Simulator of Energy-aware Cloud Computing Data Centers. Global Telecommunications Conference, 2010.

[5] Ng kwang Ming, Luke J-Y and Han. Namgoong. Open Cirrus: A Global Cloud Computing Testbed. Computer, 43(4), pp, 35–43, 2010.

[6] Anton Beloglazov, Rajkumar Buyya. Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers. Concurrency and Computation: Practice and Experience, 00: 1-24, 2011.

[7] Saurabh Kumar Garg and Rajkumar Buyya, NetworkCloudSim: Modelling paraller Application inCloud Simulations. Proceesings of the 4th IEEE/ACM International Conference on Utility and Cloud Computing, 2011.

[8] JNI, http://java.sun.com/docs/books/jni/

[9] Qt, http://qt.nokia.com/products/

[10] Robert Gordon. Essential Jni: Java Native Interface. Prentice Hall PTR, March 1998.

[11] Kejiang Ye, Zhaohui Wu, Xiaohong Jiang, Qinming He. Power Management of Virtualized Cloud Computing Platform. Chinese Journal of Computers, 35(6), 1262-1285, 2012.