CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Conference Paper

Exploiting a Throttle Mechanism for QEMU

Renato Oliveira
Cláudio Maia
Luis Miguel Pinho

CISTER-TR-180704

2018/09/03

Exploiting a Throttle Mechanism for QEMU

Renato Oliveira, Cláudio Maia, Luis Miguel Pinho

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: prmol@isep.ipp.pt, clrrm@isep.ipp.pt, lmp@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

This paper proposes the implementation of a throttle mechanism for QEMU in order to allow the testing of Cyber-Physical Systemsvia emulation.

Exploiting a Throttle Mechanism for QEMU

Renato Oliveira, Cláudio Maia, and Luís Miguel Pinho

CISTER Research Centre, Porto, Portugal

Abstract. This paper proposes the implementation of a throttle mechanism for QEMU in order to allow the testing of Cyber-Physical Systems via emulation.

Keywords: QEMU · Cyber-Physical Systems · Emulation · Throttle.

1 Introduction

Testing Cyber-physical Systems (CPS) is challenging, especially in those scenarios where the testing environment should replicate the exact characteristics of the real environment for which they are designed, while it is not possible to have such testing facility at hand. An example of such scenario occurs in the development of aerospace systems, as for instance a satellite. In such scenarios, testing may require high amounts of money, either because destructive tests may have to be made or because replicating the hardware requires doubling the amount of money involved in the project.

In order to tackle these issues it is possible to recur to simulation. More precisely, a particular case of simulation which is known as emulation. With the use of emulation, it is possible to swap some of the real hardware components with their emulated counterparts. Several tools exist in the market but only a few are able to fulfill the necessary conditions that allows one to test Cyber-Physical Systems. An example of such tool is Quick Emulator (QEMU) [1]. QEMU is a generic and open source machine emulator and virtualizer.

Although this emulator is the most adequate for the purpose of testing CPS in an emulated environment, it presents the problem of executing emulated applications in a best effort manner, that is, it uses all the processing capacity that the host system (the underlying machine upon which QEMU is executing) allows it to use. In order to circumvent this behavior, in this research we address the problem of controlling execution time of applications under QEMU.

Controlling application execution within QEMU is of paramount importance when testing CPSs. In particular, as several CPSs have low computing frequencies (for energy management purposes), in several testing scenarios, one wants emulation to be faithful to the real system. Consequently, the expected behaviour of the emulated system must match that of the real system. In the current version of QEMU, no mechanism to reduce the execution speed of the virtual machine is provided. In order to solve this issue a throttle mechanism is needed for QEMU.

This paper describes the requirements for this mechanism, the experiments performed in order to evaluate it and their results.

This paper is structured as follows: Section 2 describes the requirements of the proposed throttle mechanism; Section 3 presents the method used to test the proposed mechanism; finally, results are presented in section 4.

2 Requirements

In the interest of providing a *close-to-reality* simulation of a system with low computing capabilities, it would be interesting to implement and test a feature that enables the control of the execution speed of applications running within QEMU. The existence of this feature means that systems with low computing capabilities can be tested within an emulation context, making emulators relevant candidates for this purpose.

As such, this mechanism, upon its implementation, must: (1) control the execution speed of a virtual machine; (2) be deterministic and exhibit a behaviour that is similar to the real environment when executing the same computational tasks; (3) and usable at runtime and on-demand. Due to space limitations the implementation details of the throttle mechanism are not provided.

Regarding the first requirement, through the use of the throttling mechanism, QEMU should be able to increase or decrease the execution speed of a virtual machine.

As for the second requirement, it states that the virtual machine should behave similarly when executing instances of the same computational tasks, when submitted to the same throttle values. This behaviour is measured in the completion time of the tasks, and for correctness purposes, we allow a threshold in the execution time of applications. That is, different instances executed under the same conditions are allowed to produce results that vary within a threshold.

The third requirement infers that this mechanism should be usable whilst the virtual machine is active. This is critical since the throttle mechanism may behave differently in different emulated systems and because of that it should be possible to adapt to these situations during runtime. Therefore, it should be possible to calibrate these systems with relative ease by allowing the update of the parameters that this mechanism uses in order to operate.

3 Testing

In order to test the throttling mechanism we execute two types of tasks and measure the time that they take to execute. In the first task, the computational load task, we measure the time that it takes to sort an array using a

$$n * log_2(n)$$

complexity algorithm, and in the second one, the time constraint task, we measure the time that it takes to execute a number of iterations with no computational load until a time constraint is met.

Thus, for testing purposes, a *C++* client/server application was developed, where the server side of the application is running in the QEMU *guest* system (i.e., emulated system), and the client side part is executing on the *host* machine (i.e., the system executing QEMU). When the client connects to the server, it starts measuring the execution time. Then, the server starts one of the two tasks, i.e., either the computational load task or the time constraint task. In the end of execution, the server sends a message to the client so that it can stop measuring the time.

Four experiments were performed. The first three experiments involved the execution of the first task, with array sizes of 10000000, 5000000 and 2500000, respectively. The fourth experiment involved the execution of the second task. In each of the experiments, the test application was executed 200 times where the throttling value was varying in steps of 10%, from 0% up to 90% of throttle, and each step executed the testing application 20 times.

4 Experimental results

This section presents the results of the experiments reported in the previous section (section 3).

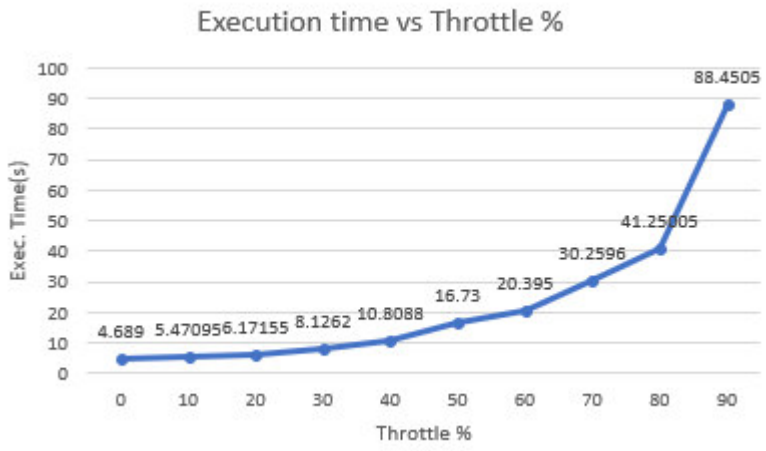

Fig. 1. Data obtained from ordering vector with 10000000 positions

In the first figure, the results obtained while executing the computational load task with an input array of 10000000 positions is depicted. From the figure, it is possible to assert that the data curve is quadratic. The same behaviour was observed for the same test with different inputs.

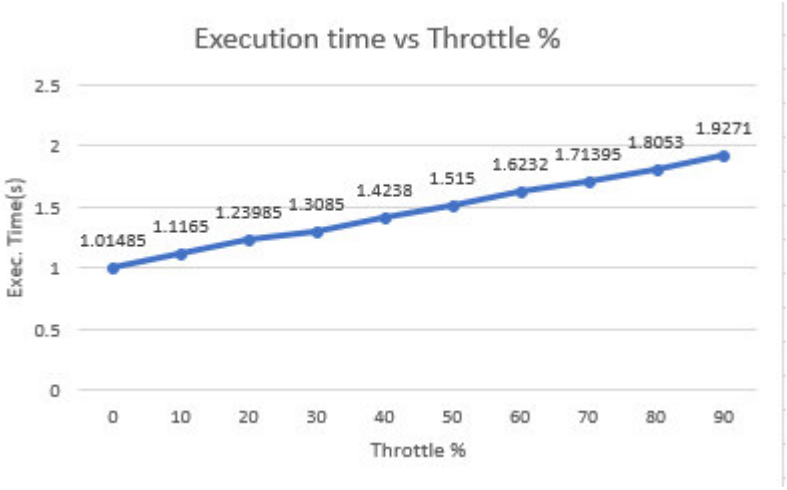

Fig. 2. Data obtained from running iterations without computational load for 1 second

In the second figure, the results obtained while executing the time constraint task with an input time of one second are depicted. From the figure, it is possible to infer that the obtained data is linear.

Because of the difference in the obtained data curves, we are able to infer that the throttle mechanism behaves differently when the system is under computational load.

5 Conclusion and Future Work

In this paper we proposed a mechanism that allows us to change the execution time of applications while using QEMU. After testing the proposed mechanism, we concluded that it behaves differently if the active task has inherent computational load versus having no computational load.

For future work, different task types can be created, one of which should be a mix between the two different tasks, in order to obtain data that could cover more execution scenarios.

Acknowledgements

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (CEC/04234).

References

1. QEMU Documentation, https://qemu.weilnetz.de/doc/qemu-doc.html. Last accessed 2 July 2018