Planning Mobile Cloud Infrastructures Using Stochastic Petri Nets and Graphic Processing Units

Francisco Airton Silva  
Matheus Rodrigues and Paulo Maciel  
Informatics Center  
Federal University of Pernambuco, Brazil  
Email: {faps, mdr, prmm}@cin.ufpe.br

Sokol Kosta and Alessandro Mei  
Computer Science Department  
Sapienza University of Rome, Italy  
Email: {kosta, mei}@di.uniroma1.it

Abstract—Mobile Cloud Computing (MCC) combines mobile computing and cloud computing aiming to aid performance of mobile devices. The idea is simple: thin devices offload heavy methods to resource-rich servers in the clouds. We believe that in the near future MCC will adopt more advanced offloading techniques. In particular, in this paper we envision a scenario where offloading frameworks will have to deal with GPU code offloading. Amazon already offers instances with Graphics Processing Units (GPU), which can be used for this purpose. We propose and implement MCC-Adviser, a simulation tool that can predict the performance of GPUs with different number of cores using Stochastic Petri Nets. We tested MCC-Adviser in a case study with one of the expensive Amazon GPU instances. The simulations showed that it is possible to minimize costs, while satisfying user’s quality of service requirements, by utilizing less powerful instances.

Index Terms—Stochastic Petri Nets, Mobile Cloud Computing, Graphic Processing Unit.

I. INTRODUCTION

Mobile Cloud Computing (MCC) is a successful solution for prolonging battery life of mobile devices and reducing execution time of mobile applications [6], [7]. Whenever a heavy task is encountered on the smartphone, it is offloaded to remote powerful servers for execution. Once the execution on the remote server finishes, the result is returned back to the device and execution continues normally. Usually, the remote side is a virtual machine on the cloud running the same operating system as the aided mobile device.

Until now, MCC is limited to CPU code offloading. Inspired by the recent support for GPU computation on the cloud [1], and the initial tentatives of using these GPU-capable virtual machines for data-intensive processing [12], [10], we envision a future where MCC will embrace the enormous possibilities offered by GPU computation offloading. General purpose computing on GPU (GPGPU) enables the possibility of optimizing the execution time of many parallel applications thanks to their large number of cores compared to the CPU. Imagine a normal smartphone being able to run the latest GPU-powered photo editor or to perform GPU-accelerated virus scanning [10]; all thanks to the cloud. We believe researchers will extend on previous works and integrate GPU code offloading into their offloading frameworks.

Unfortunately, aside from Amazon [1], no other public cloud provider accommodates virtual machines with GPU support. Not only that, the only choices offered by Amazon are the g2.2xlarge and g2.8xlarge instances, both of them very powerful and expensive: $0.65/h and $2.6/h, respectively. The first instance type has 1 GPU, while the second one has four. The model is the same for both: High-performance NVIDIA, with 1,536 CUDA cores and 4GB of video memory, which is one the best graphic cards on the market. Final users would be enforced by the limited choice to pay for these very high-performance instances, even if their requirements were not so high.

We believe that in the near future this will not be the case anymore. The other providers are going to catch up Amazon and provide GPU VM instances as well. Not only that, we also believe that all providers will offer a broader spectrum of instance types so that users with different needs can choose accordingly and minimize their costs.

In this paper, we tackle the problem of choosing the optimal GPU instance in order to satisfy user’s Quality of Service (QoS) requirements, while reducing its costs. Aiming to better understand the future use of GPUs in MCC and analyze the available public cloud service, the contributions of this paper are the following: (i) we propose a Stochastic Petri Net (SPN) pattern to model GPU parallel processes (Section II-B1); (ii) we propose a tool called MCC-Adviser which can generate SPNs representing GPU architectures while running evaluations that predict the application behavior (Section II-B2); and (iii) by using MCC-Adviser we performed an experiment that evidenced a need for smaller GPUs in terms of CUDA cores depending on the user requirements (Section III).

II. PLANNING MOBILE CLOUD INFRASTRUCTURE BASED ON SPNs

A. Illustration

To make the presentation clear, we now consider a trivial example where the offloading framework should choose among three different GPU virtual instances. This example, presented in Figure 1, shows the execution time of a hypothetical application on the three GPUs. We assume GPU_1 is the worst performing, while GPU_2 and GPU_3 have almost the same performance. Since GPU_3 performs slightly better than GPU_2 it can have a higher cost. From the user’s point of view...
however, this slight performance can be insignificant, so she can prefer saving money and use GPU_2.

Figure 1: Execution time of a hypothetical application on three different types of GPU.

More concretely, consider the case where the developer has a requirement on the task execution time to be smaller than 120ms. With high probability, GPU_1 will not be able to satisfy this requirement. Choosing between GPU_2 and GPU_3 is the only alternative. Since GPU_2 has lower costs and can deliver the task result in time, the offloading framework should prefer that one instead of GPU_3.

B. Solution

1) Modeling GPU processes using Stochastic Petri Nets: Petri Nets (PN) [9] are a family of formalism well suited for modeling several types of system, since concurrency, synchronization, communication mechanisms and probabilistic delays are naturally represented. This work adopts a particular extension, namely, Stochastic Petri Nets (SPN) [8], which allows the association of stochastic delays to exponential transitions using exponential distribution. We propose to use SPN modeling pattern to represent and analyze the performance of parallel executions on the GPU in the context of mobile cloud offloading.

Figure 2 depicts the basic structure of a process by using SPN representation.

The SPN process is composed of three places—START, READY_TO_EXEC, and FINISH—and two transitions—trigger_time and proc_time. In addition, an outer place RSRC_POOL contains a reference about the number of current available resources (Resources Number, RN). The place START is the initial phase of the process, which contains tokens representing the input to be processed. The input tokens switch place based on transitions. The transition trigger_time is the delay before starting the process. Since such a time is usually depressive, an immediate transition time is adopted (delay equals to zero). Input tokens move from START to READY_TO_EXEC place throughout trigger_time transition, which causes one resource from RSRC_POOL to be consumed at a time and the number of available resources to be decreased. Next, the execution is triggered by the proc_time transition, and when it finishes the resource token come back to RSRC_POOL place. The transition proc_time follows an exponential distribution and represents the average processing time of the task being analyzed. The last place, FINISH, represents the correct completion of process execution.

Based on the basic SPN model, we extend the proposal to use parallel processes running in a set of GPU cores. Thus, a more complex SPN with parallel processes emerges. Figure 3 depicts the corresponding SPN with n parallel processes. First of all, one initial place SYSTEM_INACTIVE is included representing the state before the parallel processes are called for execution. Then, two final places and a transition are added aiming to perform a “reduce” task, which is meant for organizing partial results of each separate process.

Using the proposed parallel SPN model we can calculate important metrics about the task execution. For example, we can calculate the probability that the task will finish before a given time. Although the proposed SPN pattern has simple rules, no other modeling platform can generate SPN models for the context presented in this paper. Existing platforms, such as TimeNET [3] and Mercury [2] [13] [14], have no support for building a desired SPN with large number of parallel processes in an easy way. In the context of GPU computation, the number of processes can vary from less than 100 to more than 1500. In this paper, we propose and implement a tool called MCC-Adviser that is precise, scalable, and easy to use.

2) MCC-Adviser: MCC-Adviser is a tool based on the Mercury platform that aims to assist software engineers planning mobile cloud infrastructure by calculating probabilities of finishing the application execution before a given time. Figure 4 illustrates our approach. Given a task, for each considered GPU we only need two parameters: the number of cores CN and the Execution time per Core EtpC. While the number of cores can be easily obtained by reading the hardware specifications of each GPU, to measure the EtpC the developer should run the task on the given GPUs, measure its total execution time, and calculate the EtpC by dividing the total execution time by the number of GPU cores.

After that, MCC-Adviser
1) generates one SPN for each GPU;
2) configures the SPNs with the two aforementioned parameters;
3) runs the transient evaluation generating probabilities with time tending to one value t. In other words, it computes the probability to absorption in [0,t), thorough transient evaluation, where F(t) approaches 1. One could either adopt the SPN Generated Distribution (SPN_GD) function or apply a distribution fitting method and then adopt a theoretical distribution that fits SPN_GD;
4) plots a graph with Cumulative Distribution Functions (CDF) for each GPU model.

Regarding CDFs, non-negative random variables may have the probability distributions defined in terms of its probability
density function, where \( F(x) = \int_0^x f(\mu) \, d\mu \), CDFs allows a broader cost-effective architectural simulation of mobile cloud applications. Analyzing the hypothetical example of CDF line plot in Figure 4, two types of interpretation may be traced:

- **Probability of Finishing Execution Before Time** \( t \): Considering one specific execution time point \( t \), the graph returns the probability \( P(T < t) \) of finishing the execution before such time for each GPU. The probability of finishing the execution before 150ms is equal to 66% for GPU_1, 95% for GPU_2, and 98% for GPU_3.

- **Probability Interval**: The software engineer may obtain the probability of finishing the execution between a time interval \((t_1, t_2)\), which is calculated as \( P(t_1 < T < t_2) = P(T < t_2) - P(T \leq t_1) \). Thus, we can calculate the probability of finishing the execution between 20ms and 80ms resulting in 9.2% for GPU_1, 47% for GPU_2, and 65% for GPU_3. As expected, the better GPU has higher probability of satisfying the constraint.

If we relax the timing constraints and increase the time interval between 100ms and 200ms, the probability of finishing the execution with less resources becomes higher. Specifically, for each device we obtain the following values: 68% for GPU_1, 29% for GPU_2, and 16% for GPU_3.

### III. Case Study

Using the MCC-Adviser tool, mobile cloud offloading frameworks can automatically decide which instance type to use so that user’s quality of service requirements are satisfied, while minimizing the costs. We run a benchmark application from NVIDIA’s samples on a **g2.2xlarge** Amazon instance and measured the execution time, which was 5.48ms on average over 100 runs. We then divided the execution time by the number of CUDA cores in order to obtain the execution time per core (EtpC) and feed it to the MCC-Adviser tool. In absence of Amazon GPU instances with less CUDA cores, we defined four other types with 98, 256, 512, and 1024 cores, assuming they use the same GPU model as the real one. Then, we used the previous measured execution time per core to estimate the probabilities for each of the defined instances.

In Figure 5 we present the results of the MCC-Adviser tool for the real GPU instance with 1536 cores and for the other instances defined by us. If the user requires her task to finish before 4.5ms, with probability higher than 95% even the less powerful instance would satisfy her needs. If the desired execution time was less or equal than 2.5ms, the probability of the less powerful instance drops to 76% and maybe another instance is better in this case.

We are aware that the example reported has very small delay, however, considering real-time applications such result is totally possible. Shi et al. [11], for example, proposes a real-time video encoding method for mobile cloud gaming in which...
some procedures take around 4ms in average. The purpose of this real case study is to simulate, for the first time, the choice of a GPU powered virtual machine in the cloud considering user’s quality of service requirements. Our tool is extremely flexible. Producing the same results for other applications is just a matter of measuring the execution time per core and feeding the value to the SPN simulator.

Aiming to better present the results of this case study, a dynamic public web page is delivered through the URL: http://cin.ufpe.br/~faps/mcc-adv-gpu/. The web page presents the CDF chart generated for this case study in conjunction with a way to access the specific probabilities. Thus, it is possible to visualize both, the probability of finishing execution before a give time \( P(T < t) \), and the probability interval \( P(t_1 < T < t_2) \).

### IV. RELATED WORK

Mobile Cloud Computing is seen as a solution for enhancing the performance of mobile devices and improve their battery life. ThinkAir [6], is an offloading framework for Android, which performs offloading of heavy methods from mobile device to remote virtual machines on the cloud. ThinkAir, however, doesn’t support GPU computation. To the best of our knowledge, [12] is the only paper that deals with GPU code offloading. In this paper, the authors propose an OpenCL-extend framework to federate the computation resources of mobile devices with cloud service so as to share its workload and shorten application response times. It allows a computation request to be conducted either on CPU on mobile device or GPU on connected servers. The authors of [10] use the power of GPU for fast malware signature scanning and show that the run-time is reduced if compared to traditional scanning.

Regarding the use of CDFs, such a strategy is very widespread in the research literature. In the field of computer science, CDFs have been used mainly to model the behavior of packet traffic within TCP/IP networks [5] [4].

### V. CONCLUSION

In 2013 Amazon launched the first virtual machines with GPU support. Two years later, we are still stuck with the same instances. No other public cloud company provides these type of machines yet, due to technological difficulties and high costs, in our opinion. Mobile devices have been broadly using cloud computing to increase applications performance. Following the successful trend of mobile application offloading towards powerful servers, we believe that in the very near future mobile GPU offloading will be a reality. This work presented an approach to represent GPU parallel processes with Stochastic Petri Nets. Using such a representation we implemented a tool, called MCC-Adviser, that can simulate GPU executions and plot cumulative distribution functions in a highly flexible way. Using the MCC-Adviser, we plot the probabilities of satisfying user requirements when using GPUs with different number of cores. We found that user can reduce her costs by opting for a less powerful GPU virtual machine, while still satisfying application’s requirements in terms of execution time.

### ACKNOWLEDGMENT

This work has been partially supported by the European Commission under the Horizon 2020 Program through the RAPID project (H2020-ICT-644312).

### REFERENCES


