Machine learning applied to accelerate energy consumption models in computing simulators

Gabriel G. Castañé
gabriel.gonzalezcastane@insight-centre.org

Insight-centre for Data Analytics
Computer Science Department
University College Cork
Cork, Ireland

Alejandro Calderón Mateos
acaldero@inf.uc3m.es

Computer Science and Engineering Department
Universidad Carlos III de Madrid
Leganés, Spain

Abstract

The ever-increasing growth of data centres and fog resources makes difficult for current simulation frameworks to model large computing infrastructures. Therefore, a major trade-off for simulators is the balance between abstraction level of the models, the scalability, and the performance of the executions. In order to balance better these, early forays can be found in the literature in which AI techniques are applied, but either lack of generality or are tailored to specific simulation frameworks.

This paper describes the methodology to integrate memoization as a technique of supervised learning into any computing simulators framework. In this process, a bespoke kernel was constructed for the analysis of the energy models used in most well known computing simulators -cloud and fog-, but also to avoid simulation overhead. Finally, a detailed evaluation of energy models and its performance is presented showing the impact of applying supervised learning to computing simulator, showing performance improvements when models are more accurate and computations are dense.

Keywords: Simulation, Computer Simulation, Machine learning, Memoization

1. Introduction

Cloud market has generated billions of revenues to companies, becoming a pillar for consumers and providers in recent years [1, 2]. This trend is expected to continue with the advent of edge devices and IoT, pushing companies to enlarge their infrastructures to multiple data centers and to co-operate with Internet Service Providers - ISPs -, therefore expanding in size and complexity. This infrastructural growth exposes a negative impact on the energy consumed by data centres [3]. U.S have 1,520 data centres consuming 91 billion kilowatt-hours of electricity per year. This means $10 billion expenses in electricity corresponding to 2.5% of the annual electricity revenues, equivalent to the energy consumed by all the households of New York city during two years. As more users migrate applications to the Cloud, and as we experience a growth in services, smart homes, and IoT in general, as more power is required by the ever-growing computing infrastructures. It is foreseen that the energy needed to support these infrastructures will reach 140 billion kilowatt-hour by 2020 [4].

The need to investigate new techniques to alleviate above-mentioned made academia and industry to work on this topics for more than a decade. However, the procedures for investigating optimisations on energy consumption are either supported by expensive-invasive procedures or require complex environments of edge and cloud devices. Then, most of the research in this area is conducted by the use of computing simulators. This technique has been extensively used for the last decades also to prevent costly customisations - sandboxing - in which it is possible to efficiently control experiments by making virtual representations of systems - models. These models, mimic the functionalities elements of computers allowing the end-user: to focus on the relevant aspects under study, to ease the observation of the system, and to save costs. However, simulating current computing trends is becoming challenging and the mechanisms used by traditional simulators are becoming impractical.

The performance of simulation frameworks is mainly bounded to the level of abstraction of the models [5], e.g. Hardware devices simulators require high computing power and memory availability to run experiments as they are not designed to scale up to thousands of millions of simulated resources. Foremost among these are: Disksim [6], Graphite [7] and GPGPUsim [8].

Another issue with current simulators is the balance between performance and accuracy. The simulations outputs are generated in run-time as result of multiple models performing concurrent operations, and subsequently stored in memory or persistent storage. Although to increase the frequency of these operations - sampling interval - produces an overhead in the processing and I/O, the results obtained provide a higher accuracy, allowing end users to have a better understanding of the results when changes are made in the simulation models. Thus, finding the adequate balance between the sampling interval and the performance is hard and time consuming as in this process it is common to either overburden simulation processes with unnecessary repeated operations, or to abstract models to a point where events are (frequently) hidden, thus, impeding users to dictate a verdict from the results produced by simulation experiments.

The need for higher performance and accuracy in simulation systems lead to reconsider the context of data. For instance, Monte Carlo experiments must be fed with input data, store intermediate results, and filter and merge output data in adjusted and reliably robust manner. Thus, and according to authors in [9], simulation approaches are particularly affected by the Big Data phenomenon since they need to use large data sets to enhance resolution and scale and distribute and analyse data in the different phases of the simulation-analysis pipeline. Furthermore, simulation models, in contrast to other methods such as data-driven, are widely used for production of forecast data. This data, that falls apart of historical bounds, is used for reinforced learning approaches [10, 11]. In these cases, some major challenges are to improve the performance and scalability of the solutions.

Machine learning [12] can alleviate the above described challenges in simulation environments. Although procedures to create models are different between simulation and machine learning - the first requires a deep understanding of the processes to gather relevant data and to produce realistic results while machine learning models are data-driven - it has been proven the strength of blending both methods in different areas [13, 14, 15]. This means that stochastic systems can get benefit from machine learning decisions under conditions of constrained response time, big amounts of data, or changing environments. Then, instead of using predetermined conditional constraints, simulation systems can learn the most appropriate conditions according to a set of desired objectives. This learning can be combined in different approaches depending on the placement of the machine learning component: (i) before the simulation execution to process input data and make it usable, developing heuristics that simulation agents can apply [13]; (ii) within the simulation to either train the machine learning algorithm as part of the warm-up, or to train the models as the simulation is executed as in reinforced learning approaches, or to reuse previously trained models [14]; (iii) after the simulation executions, using simulation outputs as data source for machine learning training data [15].

This work focuses on combining memoization as supervised learning mechanism within computing simulators, aligned to (ii), learning from the inputs and outputs of selected - performance wise - pieces of code, and making the results persistent and reusable between executions. It aims to reduce progressively the calculations overhead.

Memoization technique has been proven useful in the past to applications decreasing the cost of executing expensive compute functions in environments where a set of dense and exact operations were repeated during the life-time of an execution [16]. Although parallels between data caching and memoization exists - both aim to improve the performance by strategically storing operations -, the two differ in aspects such as subsystem target, data size, or re-usability of data, among others. On one hand, data caching focuses on improving the time to access data on storage subsystems, keeping blocks of data retrieved for future operations. And as it is unfeasible to store a full set of data required for an application in cache due to the trade-off between size of the storage and speed, as described by authors at [17] - section 4; and strategies of replacement of these blocks are applied when required [18] [19]. On the other hand, memoization focuses on the computing subsystem, storing the results of most frequent/dense computations on memory in order to avoid the operations. Furthermore, the main challenge of data caching [20] is to forecast the blocks that might be required in a future, while memoization challenge is on identifying - *Intercept* - those functions that trigger frequent/dense computations. This means, from a procedural point of view that the main focus is to intercept, tag, and store the results of costly operations after the first computation for later accesses.

These accesses - calls - are handled by a table lookup mechanism rather than by (re)computing them in cases where the search time is lower than the time required to obtain the result of the expensive computation. To effectively apply memoization, initial application profiling experiments are required to identify the potential functions that might be candidates to be *memoized* by analysing their accumulated execution time. To select these, a high-execution-time-to-compute criteria is applied, and the ones that are identified to have a higher frequency are chosen.

The major contributions of this work are listed as follows:

- To provide the methodology and analysis used to apply and incorporate memoization technique into multiple computing simulation frameworks.
- To provide an analysis of the state of the art energy models used in current computing simulation frameworks.
- To implement and provide a kernel for study and compare the performance of energy consumption models in computing simulators.
- To improve the performance of computing simulators applying memoization techniques and, therefore, a further reduction on the power required to run experiments.

The rest of the paper is organized as follows. In Section 2 the related work of computing simulators and memoization applied to simulation are described. In Section 3, the memoization model is introduced to unify the nomenclature used into this paper. Section 4 describes the study of the alternatives to apply memoization as machine learning technique in the case of computing simulation frameworks and the energy models used currently in these. Next, a methodology is presented in Section 5 with a detailed model on applying memoization to computing simulators. Section 6 presents a detailed analysis of the state of the art energy models used in computer simulators implemented in the proposed kernel. Finally, conclusions and future works are drawn in Section 7.

2. Related Work

Simulation techniques are being deployed and used in an ever-increasing rate, becoming a widely used tool set by the research community [21]. These tools vary from specific component simulation to the simulation of very large systems, like Computer Clusters, Cloud Computing or Fog Computing systems. For instance, different network simulator libraries and frameworks include NS-3 [22], DaSSF [23], and OMNET++ [24], and OPNET [25]. These tools focus on different network aspects, such as network protocols, path optimization, latencies, or IP fragmentation, but lack the details to simulate heterogeneous resources and detailed applications. Some examples of frameworks for computer-system architecture research are Simics [26], Gem5 [27], M5 [28] and SimFlex [29]. Simics is a full-system simulator developed by the Swedish Institute of Computer Science (SICS) capable of simulating multiple architectures for CPU processors and used as a virtual platform for prototyping embedded hardware and new processors. M5 provides a highly configurable simulation framework, multiple ISAs, and wide variety of CPU models. Gem5 provides a flexible, modular simulation system that offers a diverse set of system execution modes, memory system models, and CPU architectures. Finally, SimFlex is a component-based simulation framework for creating timing models of uniprocessor and multiprocessor server systems.

Different High Performance Computing and Grid simulation frameworks also exist, such as SIMCAN [30], GridSim [31], OptorSim [32], and GangSim [33]. These tools can simulate brokerage of resources and the execution of different types of applications on different types of computing resources. But they lack the details to simulate cloud environments and energy consumption of the resources as they have not been maintained for a long period of time. Most of these tools evolved into Cloud or Edge / Fog Computing simulators, abstracting (or removing) HPC models in favor of the performance of the new models. Specifically, reviews can be seen in the area of cloud computing [34, 35, 36] where a list of features, architecture diagrams and the current status of the simulation platforms is presented to cover most of the current cloud computing open source simulation tools. These simulators can be grouped into two classes: layered architectures representing visualized data center components (CloudSim [37], iCanCloud [38], FogNetSim++ [39], MDCsim [40], EMUSIM [41], DCSim [42], SPECI [43], Simulizar [44]) and network-based components layout (GreenCloud [45], SimGrid [46]).

In more detail, MDCSim offers an infrastructure simulation for multi-tier data centers, while iCanCloud provides a cloud infrastructure simulator that encompasses detailed hardware models, provider-specific VM instances, application repository models (including HPC/HTC), a Hypervisor model that allows studying scheduling policies and costs at Virtual Machine Manager level. However, the high degree of detail in the models hamper the scalability of this simulator to perform simulations of large scale infrastructures.

GreenCloud, a packet-level simulator of energy-aware cloud computing data centers and an extension of the NS3 network simulator, focusing on energy consumption in data centers and providing detailed models for servers, and network components. However, cloud-related entities like resource abstractions or virtual machine management/containers are not modeled in detail.

CloudSim is a framework for modeling and simulation of cloud computing infrastructures and services. It supports system and behavior modeling of Cloud system components, such as data centers, virtual machines (VMs) and resource provisioning policies. Moreover, a huge set of add-ons or different simulation platforms are built on top of CloudSim for supporting the model of detail disks, fog (iFogSim [47], and EdgeCloudSim including mobility [48]), or container (ContainerCloudSim [49]). Additionally, the scalability of the DES core of CloudSim is challenging, as it was shown in CloudSimEx[50], an initiative to add extra features to CloudSim, such as the feature of parallelization of the core of CloudSim to enable running experiments in a distributed fashion. However, this feature neither alleviate the number operations per model, nor the overhead in the processes when accurate traces are required.

New techniques to improve calculations performance, and hence simulation processes execution are required to follow the trend of current and future computing systems. This can be seen in Discrete Time Simulation engines used for improving the scalability of large scale cloud models. In the case of [51], authors provide a new simulation platform able to run lightweight parallel processes for models of thousands of millions of servers. However, in spite of being a promising alternative to be explored, it provides high abstraction models, such as networks or infrastructure, but raising same issues when detailed results are needed.

Alternative mechanisms have been explored in order to improve the performance of simulations in different domains. AI techniques are used to reduce the overhead of dense operations and improve performance. In [52], authors apply machine learning techniques - Support Vector Regression, Artificial Neural Networks and Random Forest - to the simulation of rainfall-runoff relationships, using the based flow calculation method coupled with different machine learning algorithms to improve the performance of the simulations efficiently.

In the scope of molecular dynamics [53], authors apply machine learning techniques to solve both the accuracy and time-to-prediction.

In the case of memoization [54] applied as a machine learning technique, authors in [55] describe the theoretical foundations for these principles, however the work focuses the approach from the general analysis of Submodular Optimization Algorithms. We focus on the utilisation of memoization technique from the analysis and applicability to energy consumption models and its integration in the a concrete domain.

Since the first occurrence of the memoization technique (*memo functions*) in 1967 [56], the original model has been adapted to satisfy the requirements of newer software techniques, programming models and, in most of the cases was customised to incorporate this technique into concrete applications, or even applied to hardware. In the latter field it has been used as technique top boost the executions by reusing partial results on instruction blocks [57], or at CPU instruction level for arithmetic operations [58].

In the software arena, memoization was widely used in logic programming as a mechanism to keep intermediate results as predicates in Prolog language, avoiding computations [59]. It can also be seen in some other and more specific software-related areas. Foremost among these are: concurrency and communication [60], optimization of functions with stochastic inputs [61], thread scheduling [62], out-of-order processor simulation [63], as file caching of results for geoscientists in satellite data [64], and finally financial applications [65].

The stochasticity and randomness of simulations make this area complex to memoization to be applied into. Thence, there are very few previous works related to the application of this technique into simulation. Most remarkable works is: memoization on a railway power consumption simulator [66] in which authors introduce a memoization framework and a systematic process intended for general usage. Then, the proposed general memoization process is locked-in to the technology of the Railway Power consumption simulator only, and for the most demanding pure functions in the simulation (as a post-design optimization process).

Other works related to simulation and memoization can be seen into [67, 68, 69] where authors describe a mechanism to automatically identify memoization units in simulation. Initial work identifies promising code blocks manually to subsequent works that present the transformation of the identified code blocks into a memoized variant, and the automation of the holistic process. This work is more related to refactoring techniques (in order to work with code blocks and transform into pure functions to be memoizated).

In contrast to previous works in simulation, the research presented in this paper focuses the model of memoization as a technique and its applicability into any computing simulator, more specifically the energy models, and independently to the underlying platform. Another interesting contribution is to introduce memoization as a type of learning technique (related with AI) that can be used within simulations. Works such as [70] uses memoization in Reasoning Systems, but not in simulations as learning technique alternative.

3. Memoization Overview

This section describes the general model of memoization used in this work. Figure 1 shows the memoization mechanism applied to an application (A). As internal software components that encompass A, a set of *Caller Functions* F: $\{f_1, f_2, .., f_n\}$ invokes a set of *Pure Functions* P:$\{p_1, p_2, .., p_i\}$ to perform operations and calculations that satisfy:

$$\forall f \in F, \forall p \in P \mid f, p \subset A \tag{1}$$

The P functions perform dense computations and return the same results (ρ) over the time, i.e. j, k instants of time and $j \neq k$; when input parameters (${}^{x}/_{x} = 1..y$) to $p \in \{p_1, p_2, .., p_i\}$ are the same:

$$p_j(x) \;=\; p_k(x) \;=\; \rho \tag{2}$$

Another essential feature of P is the scope where they execute. A p_i function do not cause any side effect by using resources allocated out of the application scope.

The memoization layer is composed by a set of Intercept functions, $I : \{g_1, g_2, .., g_m\}$ that depict calls captured from f where a p is involved and memoization can be applied into. The requirements to effectively capture a caller function f and apply any g_i must satisfy one of following conditions:

- An injective relation between F and P as $g_i \in I{:}F \rightarrowtail P$, if $f_1 = f_2 \in F$, then $p_1 = p_2 \in P$;
- A surjective relation between F and P as $g_i \in I{:}F \twoheadrightarrow P \;/\; \forall f \in F, \exists p \in P \,.\, g_i(f) = p$

I functions require to be identified and partially modified so they can avoid the dense computations by recovering previously stored data for the operation. This process can be performed manually by the developers or in an automated/semi-automated fashion. The most efficient consists in the manual identification of the f functions that invokes p by profiling several execution traces, followed by an ordering and selection process of I. However, the major drawback lays

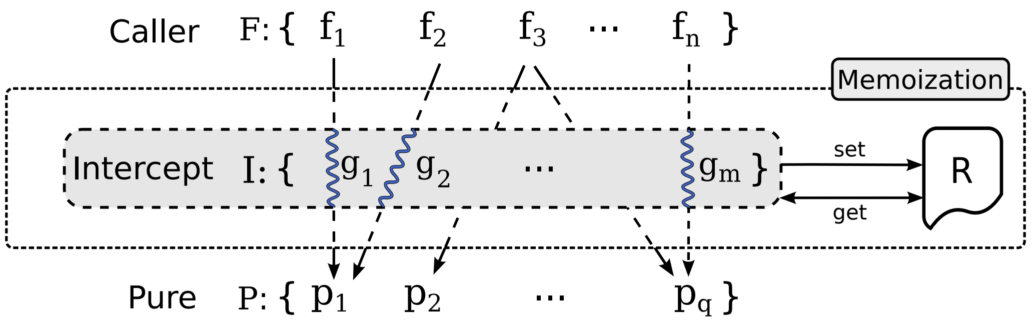

Figure 1: Memoization architecture overview for an application *A*.

in the labor-intense and error-prone effort carried out by the model developer. This approach can be widely found in the literature[66, 71, 72], and it is the one applied in this work. Furthermore, in order to alleviate the time-consuming manual procedure, automating it within compilers - Haskell, Lisp, and C++ [73, 74, 75] - has been a target since the past decades, but also, in a more generic manner, a minor subset of generic functions under temporal patterns [76]. Another alternative can be found as a semi-automated memoization process in which programmers must annotate under their criteria the code blocks that correspond to I functions [77, 78]. However, for the semi-automated and the automated approaches, although the degree of transparency to the end user is higher, the efficiency of the approaches is lower as most of the studies show not all pure functions are correctly identified as it can be done through the manual profiling.

R is a store where previous dense or very frequent operations (P) are stored. To store without loosing performance, an in-memory key-value store is typically used for (R) as current disk drives do not provide same I/O performance as memory. Additionally, to avoid the recalculation between multiple executions, data stored in R can be saved to persistent storage. Hence, when an execution is newly triggered, data previously computed can be used and therefore save energy and compute time.

When a $g_i \in I$ is invoked, it performs a seek into R of a prior execution of a $p_i \in P$ function with same input parameters. In case no entry satisfies the requirements for a set of parameters given, p_i is invoked with those parameters and the result of the operation is subsequently stored as a new entry into R to avoid the re-compute in future calls. Figure 1 shows these operations as *get* and *set* arrows going from the Intercept square to the storage R both within the *memoization* scope.

4. Targets for memoization in computing simulation frameworks

Computing simulators are composed of independent models that, appropriately linked and cooperating together, build virtual representations able to describe behavioral patterns of real computing ecosystems. Physical resources, resource abstractions, resource managers and users' applications are represented in most current computing simulation frameworks with different granularity levels. Hence, models of physical resources can be seen in a fine grain perspective as complex compositions of another intra-connected models (processing units, memories, disks, network interfaces) each one of these composed by another internal model/s, or simplified and abstracted as one unique model with a mathematical model parametrized by a set of attributes, describing these behaviors. Analogously, resource abstractions models into which virtual machines and containers are encompassed can be seen as sets of interconnected operative system, file systems, and virtual processor models, or gross grain abstracted as single application models.

This paper focuses on cloud computing simulators as these models have been widely validated and used over the last decade. They are a superset of previous grid and cluster technology models with an added virtualization and user models, and a subset of the holistic view of Fog and Edge computing environments in which is a matter of scale rather than the lack of additional models to simulate these scenarios. In this section, the models of cloud computing simulators are classified as potential targets for applying memoization techniques, showing that energy models are the most suitable models for this aim. Additionally, it is shown an analysis of the energy models implemented in current (most) well-known cloud computing simulators.

4.1. Model alternatives for memoization

Simulators dynamically generate model replicas as objects in memory during runtime, parametrized by their attributes and operations. Although multiple objects can be replicas of the same model, the computation overhead of an object in a precise instant of simulated time depends on the operation that is executed and the input parameters associated with it.

To identify where to apply memoization based in profiling, it is required a trace with data for every model for several executions. This is composed by objects created in runtime, their associated operations, and values of the parameters where to seek dense and frequent computations. This approach can be unrealistic when hundreds of models, and hence hundreds of thousands of operations are sought. Therefore, a simplification can be made by grouping the models by commonalities. This strategy encompasses into each group the set of models with a *shared model goal*. Thence, operations and attributes between these are close in a multidimensional search space.

The following alternatives are considered as potential memoization targets:

- *Alternative 1, applications and resource abstraction models*. These models cover a large number of objects created in simulation runtime. Most cloud simulators are provided with detailed models to calculate the performance degradation of several virtual machines, applications, and containers running in the same host.
- *Alternative 2, resource provisioning techniques, orchestration, and allocation models*. There is a huge variety of scheduling policies applied to cloud computing orchestration of resources. These models perform dense computations as they need to evaluate a large number of inputs to find a host for placing each application.
- *Alternative 3, infrastructure and hardware subsystem models*. processing, memory, storage, and networking The infrastructure models are static, being created during the initial phase of the simulation process, and staying until the process finishes. This alternative encompasses a huge set of components and, depending on the level of abstraction used to model each server, it might be a set of objects performing operations. However, these operations do not perform dense computations and also are application-dependent.
- *Alternative 4, energy consumption models associated with the physical infrastructural models*. When an object belonging to the infrastructure models change its state or load, the power state and consumption is updated in the simulation process and, subsequently the energy consumption. The energy consumption needs to be calculated periodically to log the behavior of one or more targeted resources during a simulation runtime.

In case of *Alternative 1*, performance degradation, noisy neighbours, and other techniques to model the overhead between applications and resource abstractions have a negative impact on the applicability of memoization into this Alternative. In these models, a valid operation can be represented by a $g_a \in I$ as: $g_a : F \rightarrow P / f_1 = f_2 \in F \wedge p_1 \neq p_2 \in P$

These operations can produce different outputs for identical sets of input parameters, contradicting the surjective condition described in section 3. Thus, there is no certainty on the correctness of memoized results stored into R to previous dense or frequent computations if memoization is applied to these models.

For *Alternative 2*, stochastic models and Random Number Generators (RNG) used to mimic the uncertainty in reality, influence negatively to the results of applying memoization to orchestration model operations. Orchestration techniques might create new objects as load balancers, databases instances, web server replicas in response to certain workload or energy consumption conditions. By applying memoization to these models can produce the inhibition of stochastic events intentionally added by programmers. Additionally, the large set of inputs that needs to be controlled due to the randomness between several executions would increases the size of R until potentially turning into inefficient the management of stored data to apply memoization.

Similarly to *Alternatives 1* and *2*, the applicability of memoization to the models encompassed into *Alternative 3* has a direct dependency on the application models and their requested operations performed by the infrastructure models and their subsystems. However, these operations are lack of dense computations since they are directly dependent on the application, virtual machines, containers models. In addition, these models can inherit some g_i operations, being not feasible to apply imprecise memoization to these models.

Our approach targets the energy models, *Alternative 4*, as computing simulators perform periodic dense operations to calculate the energy consumed by the infrastructure models.

Next, the energy models are explained in detail for a better understanding of the model of memoization applied to this Alternative and described in Section5

4.2. Energy Simulation models in Cloud Computing Simulators

Although a simulation framework can be customized to enable the study of energy consumption by using any energy models, as it is described by authors in [79] each simulator is restricted to use one or several models currently available in the literature to perform these calculations.

Most simulators provide *linear interpolation* as a method to calculate energy consumption. The programmatic software complexity is low, reduced to a few lines of code, and requiring only idle and maximum power consumption values of a server as input parameters to the model. A *Sqrt*, *Square*, and *Cubic* distributions can be applied to the linear interpolation model as variants providing better accuracy on the energy calculated but requiring higher performance. The algorithm complexity to calculate the linear interpolation, and any of the before described variants is given by $O(N_s(n))$.

A particular case of linear interpolation (*piecewise-defined linear interpolation*) is the case of the SPEC [80] energy models. In this models, the energy consumption of some commercial workstations was measured in under certain conditions allowing to divide the consumption space in equally distributed intervals, splitting by intervals of 10% from idle, the initial interval or 0%, to full performance or 100%. Intermediate values are estimated using piecewise linear interpolation, having a complexity of $O(N_s(log(n) + m))$ where N_s is the number of servers and m the number of break data points (10 for SPEC values).

The *per component model* is a discretized model in which the energy consumption of a server calculated as result of the aggregation of the energy consumed by each main subsystems. This model, that do not use an analytical distribution to describe the behavior of the system or subsystems, uses a matrix of power data to identify each component consumption as the time spent in each state and the power consumption in that state. Operations such as multiplications and aggregations on the data of this matrix are periodically performed to calculate the energy consumption of each server. Although high accuracy can be obtained from the per component model, it requires detailed models for each subsystem and increases the complexity to $O(N_s(mn))$.

Energy models and their associated complexities are captured in Table 1.

		Complexity	DCSim	CloudSim	GreenCloud	iCanCloud
Models	Linear	$O(N_s(n))$	yes	yes	no	yes
	Piecewise	$O(N_s(log(n)+m))$	no	yes	no	no
	Sqrt	$O(N_s(n))$	no	yes	no	no
	Square	$O(N_s(n))$	no	yes	no	no
	Cubic	$O(N_s(n))$	no	yes	yes	no
	Component	$O(N_s(mn))$	no	no	yes	yes

Table 1: Summary of simulators models for cloud computing systems.

The linear interpolation approach is provided by iCanCloud, DCSim, and CloudSim in which is also provided the Sqrt, Square variants.

GreenCloud provides models in which a cubic relationship between operating frequency and power consumption described in [81], thus forming a model using these metrics $O(N_s(n))$, and also included into CloudSim.

The piecewise-defined linear interpolation alternative is implemented in CloudSim.

The per component model is implemented in GreenCloud and iCanCloud [82]. GreenCloud uses the network interface card and an abstracted model of CPU to calculate the energy consumption while iCanCloud uses network, memory, storage and a fine-grained CPU model in which energy is calculated at p-states and frequency levels.

5. Memoization model applied to energy models in computing simulators

Figure 2 depicts an example of possible representation of a topology segment associated with a data center model. This segment has two server clusters connected by a router in the center of the Figure, labelled as e_1^1. The bottom-right part of the Figure is a group of five homogeneous servers: e_3^1, e_3^2, e_3^3, e_3^4, and e_3^5; and a group of three homogeneous servers in the bottom-left part: e_2^1, e_2^2, and e_2^3.

Each group of elements, more specifically e_2^1, e_1^1, and e_3^1 from left to right respectively, has a dotted-line rectangle showing an exploited view of the internal components that potentially have energy operations associated with it. Models described in Section 4.2 can be independently applied to each hardware device model: CPU, memory, network card, and disk drive in a per-component model, or can be generalised into an operation for the entire server in which each hardware device model is parametrised as an attribute, thus, the element-component relation is identified as 1-to-1.

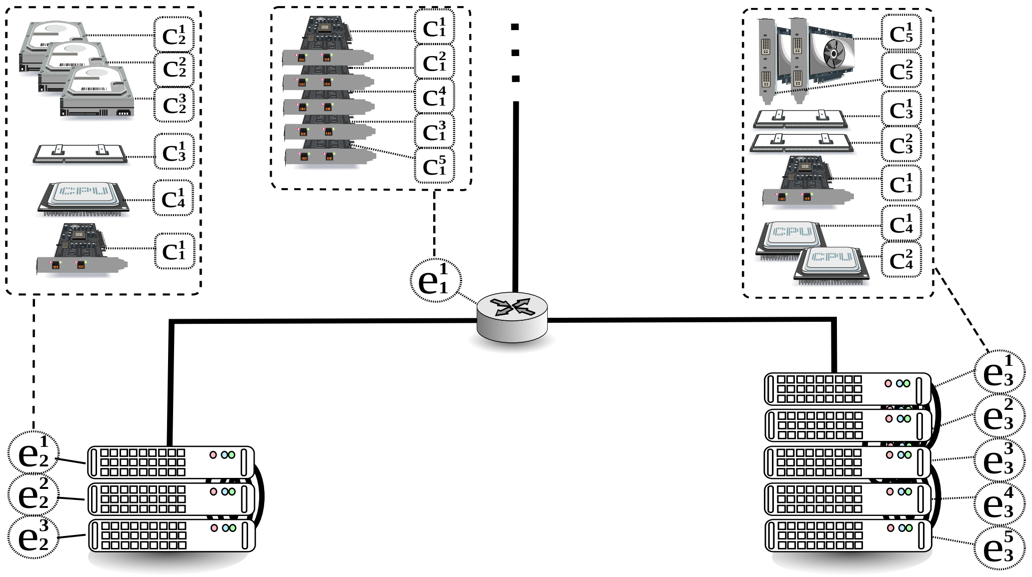

Figure 2: Example of a data center model consisting of two groups of servers heterogeneous between them and a router

In the Figure, the router, labeled as e_1^1, is composed by seven network cards (c_b^a where $a \in \{1..3\}, b = 1$). All these are identical replicas of the same network card model. In the right hand, a GPU cluster is shown in which each server has two CPU cards, two CPU processors, high memory capacity, and no disk drive. Moreover, the left-hand cluster models a group of storage servers consisting of three disk drives and low processor and memory features.

Next sections describe the most representative variables, notation and constraints used for modelling energy consumption in computing simulators, and the memoization model applied to them.

5.1. Computing simulators, energy model

Simulation model notations

Let e_i^j be an *element* where i is a unique identifier associated with a specific model in a simulated scenario and j depicts the number of replica of that specific model.

Let c_s^q be a *component* where s define a concrete component model and q identifies the number of replica into the model super-set e_i^j.

Let C be the set of components encompassed into an e_i^j where $|C|$ is the number of components.

Let $\mathcal{H}$ be a set of element replicas (homogeneous group of elements) and let $|\mathcal{H}|$ be the number of $\mathcal{H}$ groups. Two elements e_i^j and $e_{i'}^{j'}$ are considered homogeneous, or belonging to the same $\mathcal{H}_i$ group, if $i = i'$ and it is satisfied:

$$\forall c_s^q \in e_i^j : \exists c'^{q'}_{s'} \in e_{i'}^{j'} / c_s^q = c'^{q'}_{s'} \wedge |C_i| = |C_{i'}| \tag{3}$$

Let $\mathcal{M}$ be the set, union of the elements contained into all $\mathcal{H}$ groups as $\mathcal{M} = \{\mathcal{H}_1 \cup .. \cup \mathcal{H}_n\}$. Let $|\mathcal{M}|$ the sum of all $|\mathcal{H}_i|, \forall i \in \mathcal{M}$.

Let e_a be the notation used to refer a unique element within the $\mathcal{M}$ space. This is $a \in \{1..|\mathcal{M}|\}$.

Pure Functions associated with Energy operations

Let consider $\mathcal{E}$ as the pure functions p associated with dense or frequent energy consumption operations in the complete set of pure functions that a simulator have P as $p \in \mathcal{E} \subseteq P$.

Let p_{ci} be the operations required to calculate the energy consumption of a component c that is identical for all its replicas. Let consider the set of operations encompassed by this calculation as an atomic – non-divisible – operation.

Let p_{ei} be the set of operations required to calculate the energy consumption of the element e and, thus, all elements of a homogeneous group $\mathcal{H}_i$.

Let define the following properties for $p \in \mathcal{E}$ associated with the scope of computer simulators:

1. Element-component model property: $\forall e_i \in \mathcal{H}_i : |p_{ei}| = 1 \rightarrow p_{ei} = p_{ci}$;
2. Per-component model property: in an element consisting of $|C|$ components, $|C| > 1$, a group of operations are required to calculate the energy consumption where $i \in 2, |C|$;
3. Pure function-component bijection property: $\forall c_b^a \;/\; \forall a : (b = k_1) \rightarrow (p_c = k_2)$, where k_1, k_2 are constants. This means that the every component has associated a unique pure function to calculate the energy, independently on the replica.
4. Homogeneity property: given $e_i^j, e'^{j'}_{i'} / (p_{e1} = p_{e2}) \rightarrow \{j \wedge j'\} \in \mathcal{H}_a$, where $a \in 1, |\mathcal{H}|$

Energy and power consumption operations

Computer simulators have a diverse number of energy models implementations to calculate energy consumption in the same component model, as described in Section 4.2. However, similarities exist in all methods. Therefore, these have been identified, generalised, and described in this subsection.

Let u be a possible status or utilisation score of a component c, and $\mathcal{U}$ be the set of all possible statuses c can have. And let $|\mathcal{U}|$ be the number of possible statuses or utilizations in which a component can switch, and identical between $c_i \in \alpha_i, \forall i \in 1, |\mathcal{M}|$.

Let Pw be a function that calculates the power consumption of a c_s^q, and let φ be the result of the Pw operation as:

$$\varphi = Pw_i(u, t_j),\ i \in [1, |C|] \tag{4}$$

according to the energy models, that are described in Section 6, function of utilisation in an instant of time t_j.

Let $\overline{E}$ be a function that calculates the energy consumed of C in an interval of time $t_{a,b}$, that is $\forall t_i \in [a, b]$, and let ϕ be the result to this operation as:

$$\phi = \overline{E}(\varphi_i, t_{(a,b)}) = t_{(a,b)} \times (\sum_{i=1}^{|C|} Pw_i(u, t_j)) \quad , \quad i \in [1, |C|] \tag{5}$$

where in case number of components $c_i > 1$ then, the energy consumption of an element e_i is calculated as the aggregation of φ partial results associated with C. Next, main properties associated with Pw and $\overline{E}$ are listed:

1. as φ is the result of a function depending of u, then $|\varphi| = |\mathcal{U}|$, where $|\varphi|$ is all possible values that Pw can have associated to each of the components of an element, and $|\mathcal{U}|$ the composition of components and the combination of their statuses.
2. ϕ depends on the sum of φ associated to each component, therefore a composition of φ will produce an identical $|\phi|$ over executions.

3. the number of different ϕ is bounded to $|\varphi|$
4. a composition of φ produce a unique ϕ_i associated to an element of the simulation model $\forall i \in |\mathcal{H}_i|$.
5. the space of possibilities, upper bound, for $|\varphi|$ is defined by $\binom{|C| \times |\varphi|}{|C|}$.

Energy and power consumption operations time during a simulation

Let t_{sim} be the instant of simulated time when a simulation process end, and let $t \in [0, t_{sim}]$ be a variable that denotes an instant of simulated time during the simulation run-time process.

Let $t_{comp} \in [0, t_{sim}]$ be a continuous variable that denotes the time required to calculate a ϕ_i. Then, and according to Section 4.2, the time t_{el} to provide the energy of an e_b^a with C components is depicted in Equation 6 as the aggregation of the partial consumption of each c_i, with $i = [1, C] \in e_b^a$.

$$t_{el}(t) = \sum_{i=1}^{|C|} t_{comp}(t, i) \tag{6}$$

Moreover, other frequent operation in which energy consumption operations are used and, therefore, p functions, is associated with the calculating the total consumption of M (or a subset) in an instant of time t. This operation, described by Equation 7, requires t_M time to sum the energy consumed by $\mathcal{E} = \{e_{\mathcal{H}1} \cup .. \cup e_{\mathcal{H}i}\} = |\mathcal{M}|$ elements, and for each $e_{\mathcal{H}}$ the sum of the energy consumed by $|C|$ components as described by Equation 6.

$$t_M(t) \;=\; \sum_{z=1}^{\mathcal{E}} \sum_{i=1}^{|C|} t_{comp}(t, i) = \sum_{z=1}^{\beta} t_{el}(t) \tag{7}$$

Then, T_{total} time is needed to calculate the energy consumed by the model $\mathcal{M}$ using a number of η samples (periodic energy checkpoints), during t_{sim} (usually user predefined) time. This is given by:

$$T_{total} \;=\; \sum_{t=\frac{0}{\eta}}^{\frac{t_{sim}}{\eta}} t_{\mathcal{M}}(t) \tag{8}$$

Where the sample space during a simulation experiment (N_{sample}) is defined as the normalization of the simulated time over the sample space, $N_{sample} = \frac{t_{sim}}{\eta}$. This is a constant set of operations that are required by any simulator that perform energy calculations in run-time.

5.2. Memoization model

Figure3 depicts a generalisation of a simulation model (S) grouping e and $\mathcal{H}$ in a two dimensional table. The Y axis shows for each e, replicas and components encompassed from 1 to $|C|$ not necessarily equal between them. The X axis groups into i heterogeneous groups $\mathcal{H}$ the elements and components that build the simulation model.

The *Storage* row at bottom shows an in-memory table R grouped per heterogeneous $\mathcal{H}_i$ element column. This model facilitates a size reduction of the storage as a lower number of memoized entries is stored and, thus, the complexity for insertions in R_i decrease accordingly. Additionally, keeping a partitioned storage model enables the partial serialisation of R_i elements to a secondary storage in run-time in case the system where simulations run has limited resources.

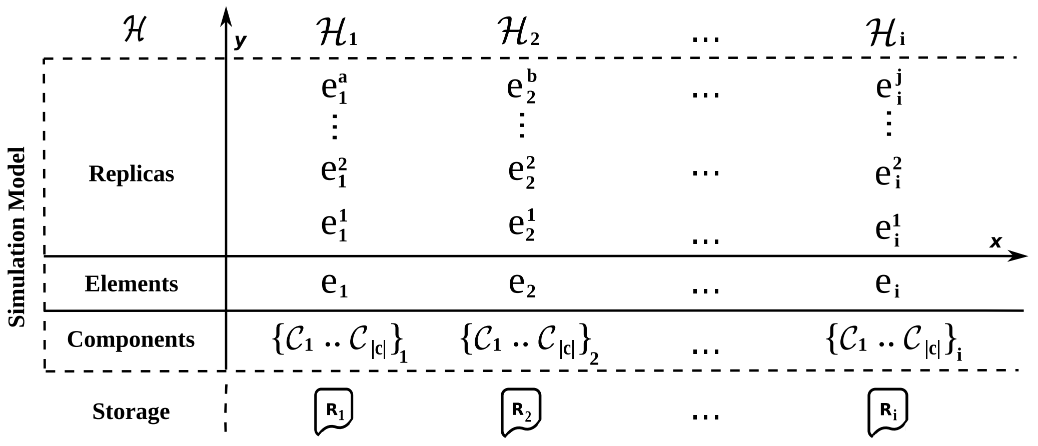

Figure 3: Generalisation of elements and components for heterogeneous computer simulation models

Moreover, more strategies to improve the performance based on the partitioning of the memoization storage can be investigated to increase the performance but are beyond the scope of this paper.

To effectively apply memoization, the first time a pure function (p_i) requiring an energy calculation associated with a component is executed, φ is calculated and subsequently stored into the in-memory storage, look-up table, R, requiring a time to execute of t_{el} + t_{ins}, where the latter term represents the time to insert into the storage. Afterwards, subsequent caller functions (f calls) invoke p_i, an intercept function will capture the call, extracting previous stored result from the look-up table and returning the stored value to f, hence reducing the time from t_{el} to the time to search in a look up table, t_{search}, and the complexity from what it is shown in next section 4.6 to O(1).

The overhead during a non-deterministic simulation can not be modeled with exactitude as the probability is a fundamental part during the executions. However, it is possible to define upper and lower performance bounds for simulations executions. The lower bound for performance, T_{lbmemo} will occur in case all the executions are insertions in the in-memory table:

$$T_{lbmemo} = \eta \times T_{ins} \times t_{el} \tag{9}$$

Analogously to Eq. 9, for the upper bound operations results are previously calculated and stored into the corresponding R_i and therefore, the time is reduced to T_{ubmemo} to perform all pure operations associated with energy calculations:

$$T_{ubmemo} = \eta \times T_{search} \tag{10}$$

This means that a combination of both Eq. 9 and 10 describes the time bounded by the number of operations that are invoked during a simulation experiment and where $\alpha + \beta = \eta$. This can be described as:

$$T_{memo} = T_{search} \times \alpha + T_{ins} \times t_{el} \times \beta \tag{11}$$

Note that, in the process to improve the performance and acknowledging to a limited number of entries in the in-memory storage per component - combinations of components and energy statuses per element; by loading previously calculated memoization storages (R_i) the performance between simulation experiments will increase, as it can be considered that memoization is *previously trained* for that specific element. Note this aspect in Figure 3 where each R_i has been represented to keep independence between components, but shared between replicas. Therefore, consecutive experiments using same element already executed with memoization will avoid dense computations by obtaining results previously calculated from the corresponding in-memory storage R.

By reusing the *trained* memoized R_i across multiple executions, the execution time will decrease towards converge in $T_{memotrained}$, that consists of time to perform searches, and time to restore ($t_{restore}$) all the elements that construct the simulation model of the current experiment. This is:

$$T_{memotrained} \;=\; \eta(f) \times t_{search} + t_{restore} \times i \tag{12}$$

Where i is the number of independent storage R_i associated with the different elements of the simulation model executed.

6. Memoization analysis on energy calculation techniques

6.1. Energy models

In "A comparative study of CPU power consumption models for cloud simulation frameworks" [83] authors reviewed the main four power models used in cloud computing simulation frameworks such as CloudSim and DCSIM: linear, square, cubic and square root. Authors also include two novel models [51, 84] in this work based on a third degree polynomial form and some variant that are named as Proposed-1 and Proposed-2. Proposed-1 requires less computing power but it provides less accuracy than Proposed-2, that is a performs a dense computation and provides quite accurate results when compared with real data as described by authors. These are:

- Linear: $P(u) = P_{min} + (P_{max} - P_{min}) * u$
- Square: $P(u) = P_{min} + (P_{max} - P_{min}) * u^2$
- Cubic: $P(u) = P_{min} + (P_{max} - P_{min}) * u^3$
- Square root: $P(u) = P_{min} + (P_{max} - P_{min}) * \sqrt{u}$
- Proposed-1: $P(u) = P_{min} + \frac{P_{max}-P_{min}}{100} * u + \frac{P_{min}}{2} * sin(\frac{2*\pi}{100}) * u$
- Proposed-2: $P(u) = (\frac{11}{27} * P_{max} - \frac{7}{6} * P_{min}) * (2 * u - 1)^3 + (2 * P_{min} - \frac{2}{9} * P_{max}) * u^2 + (\frac{11}{27} * P_{max} - \frac{2}{3} * P_{min}) * u + (\frac{11}{27} * P_{max} - \frac{1}{6} * P_{min})$

These models are also valid for all the computing simulators built using CloudSim as core and with target edge and container models. These are: iFogSim, EdgeCloudSim, MyiFogSim, ContainerCloudSim.

6.2. Kernel and evaluation setup

Furthermore, and as a main contribution of the work proposed in this paper, the evaluation has been conducted by designing and developing a kernel aimed to help simulation developers to test current and future energy model, and its impact on performance. This kernel is publicly available at: `https://github.com/acaldero/energy-kernel`

The kernel implements the Subsection 6.1 power models and provides to the researchers a mechanism to obtain the time - in microseconds - and energy consumption of a compute node - cloud or edge - for a period of time. With this tool, users and developers can analyse existing and new energy models, the impact of memoization or other techniques on these computations, and the accumulated error in case they have real measurements collected from a real node. Furthermore, for those simulators that calculate the energy consumption by components granularity, as it is the case of GreenSim, iCanCloud and FogNetSim++, the evaluation can be seen as a composition of components and the aggregation of their consumption over time.

The kernel was executed on a system with a Intel(R) Core(TM) i7 920 @ 2.67GHz CPU (8 MiB of cache size), 32 GiB of RAM, Ubuntu 17.10 64 bits distribution, and GCC version 7.2.0.

For the evaluation, the experiments are designed for a group of elements - computer nodes/edge devices or internal components - from 1 to 15. Moreover, due to the models are simple common arithmetic operations, the evaluation is extended to encompass multiple optimisation levels in order to analyse the dependency between the assembly code generated by the compiler and the performance. Foremost among these, the options analysed are: -O0, -O1, -O2, -O3, and -Ofast.

An additional experiment to the comparison of the six energy models - *LINEAR*, *SQUARE*, *SQUARE-ROOT*, *CUBIC*, *PM1*, AND *PM2* - is added to represent the use of memoization - *FIND*. For the later, the model used is *CUBIC* to be the one that both: requires more computing power and is implemented in a higher number of simulators. In spite that *PM1* and *PM2* provide a higher accuracy in the results, they are quite novel and not resident in the source code of any of the simulators described in Section 2. From the executions of this model, we used the kernel to memoize all those results. Then, the results were added to the comparison.

6.3. Results analysis

Tables A.2 and A.3 shows both: (a) the time to compute the energy estimation for the kernel with six optimization levels (-O0, -O1, -O2, -O3, -Ofast, -g) with six energy models plus memoization (*CUBIC*, *LINEAR*, *SQUARE*, *SQUARE-ROOT*, *PM1*, *PM2*, and *FIND*) and up to 15 components. (b) the estimation of energy value itself for the six optimization levels, six energy models and up to 15 components. The kernel is executed ten times, the average results are shown in the table for both. The standard deviation is less than 2% so are near the same each time the kernel is executed.

On table A.2 the time in microseconds for the combination of: (a) different optimization levels (-O0, -O1, etc.), (b) different energy models (*CUBIC*, (*LINEAR*, etc.), and (c) different number of elements (0, 1, ..., 15). Figure 4 represents the table results in a compact view. Figure 5 shows the accumulated time for 1 up to 15 elements.

With the highest level of optimization (-Ofast) the time is reduced a lot because "Enable all optimizations of -O3 plus optimizations that are not valid for standard-compliant programs, such as re-ordering operations without regard to parentheses" [85]. The (-g) is not an optimization level but the flag/option to add debugging information into the executable in order to developer be able to debug is code (and be able to see the original code while debugging). In this case, it is added to the default optimization level (-O0) it adds some minor overload.

The order from slower one up to faster one estimation model in general is: *PM2*, *CUBIC*, *PM1*, *FIND*, *SQUARE-ROOT*, *SQUARE*, *LINEAR*. There some optimizations that has minor influence in some consecutive pairs of previous models.

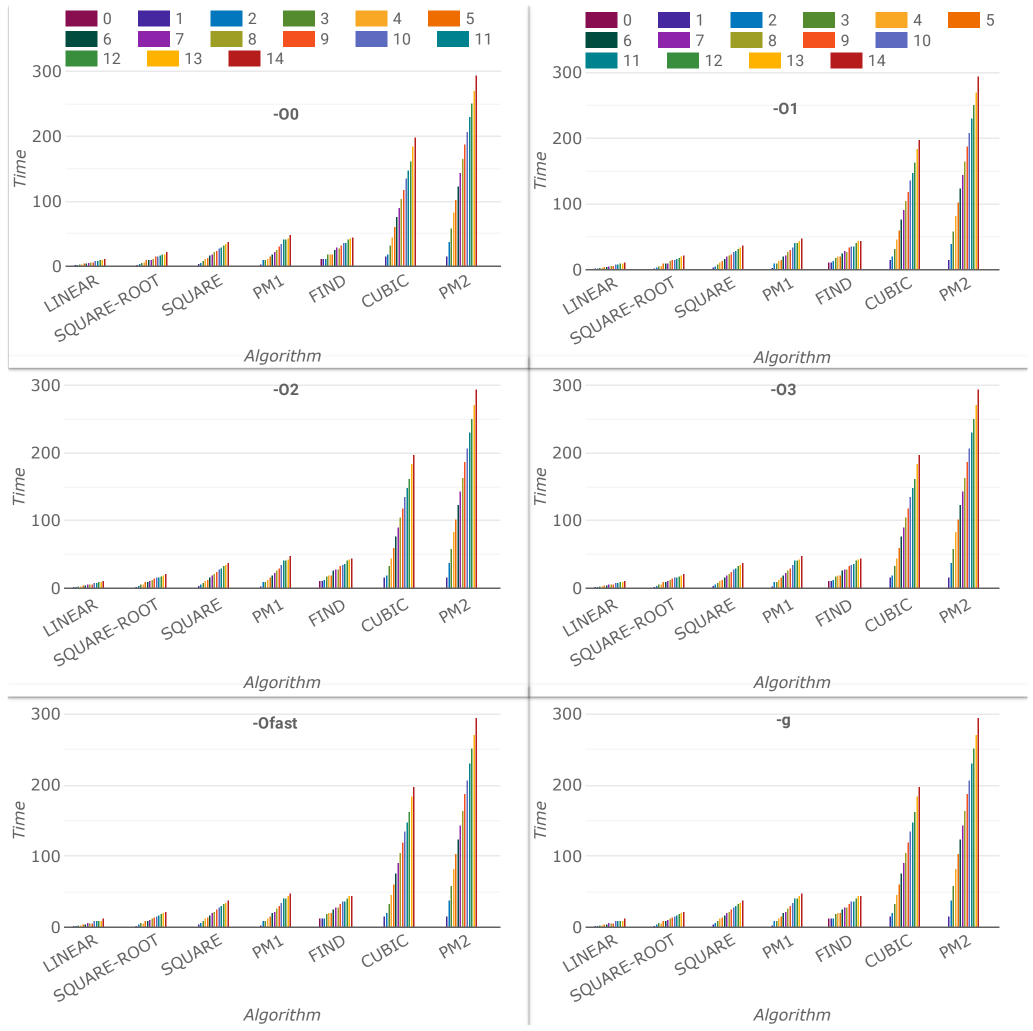

Figure 4: Time needed on Intel i7 920 @ 2.67GHz (for several elements, algorithms and compiler optimisation options).

For some energy models the number of elements has a linear influence because the O(n) order of complexity in the computation performed. Examples of these models are the *LINEAR* or the *SQUARE-ROOT*. On the other hand, there are more complex models and with more components per node to be estimated (up to 15) the time became near ten times larger compared with one element. Examples of these are *PM2* or *CUBIC*. Figure 4 shows the influence of the number of elements too. The execution time for *FIND* grows near linear with the number of elements following a growth defined by $f(x) = 0,7397x+2,775$ with $R^2 = 0{,}9642$; whereas *CUBIC* grows much more quickly following a second degree polynomial: $f(x) = 0,223x^2 + 9,8936x - 9,575$ with $R^2 = 0,9966$.

Table A.3 shows the computed values, and an important aspect here to notice is the differences among them. The energy model ordered from lower value up to higher value are: *PM1*, *PM2*, *CUBIC*, *SQUARE*, *LINEAR*, *SQUARE-ROOT*. The difference in some cases is larger than 50% of the value. Memoization option (*FIND*) has the same value as *SQUARE-ROOT* because it was the chosen one as reference, but could be used any other. We want highlight that *PM2*, *PM1* and *CUBIC* are the energy models with high computational time but lower value. On the opposite side, *SQUARE* and *LINEAR* have a higher value but produce a minor computational time.

Given the general order from slower to faster (*PM2*, *CUBIC*, *PM1*, *FIND*, etc.), the memoization option is the faster of the four that require heavier computations and produce more accurate energy models. Due to it is using the value of other precise model during the first execution - the *CUBIC* in this case - it is as accurate as the selected one. Figure 5 depicts the accumulated time for 1 up to 15 elements. Using Memoization - *FIND* - the *CUBIC* model speeds-up up to four times.

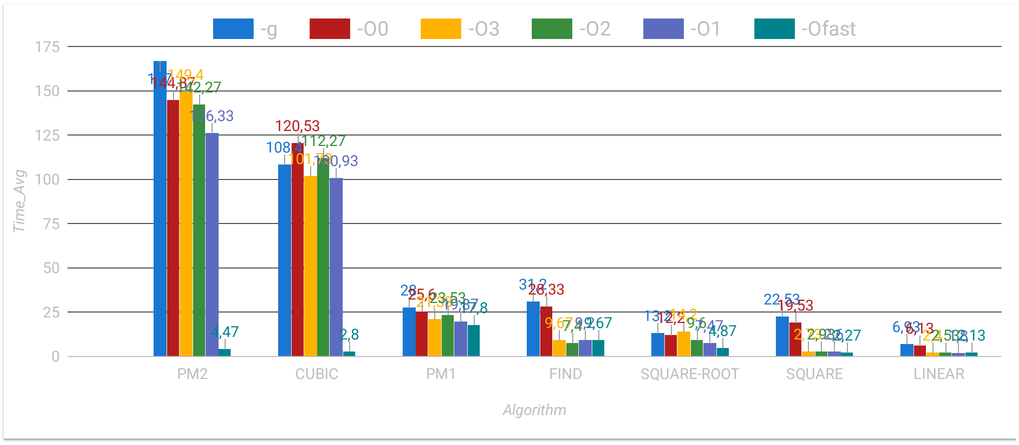

Figure 5: Accumulated Time needed on Intel i7 for 1 up to 15 elements (for several algorithms and compiler optimisation options).

7. Conclusion and Future work

Modelling and simulation have been used to study Cloud Computing, for example working on energy modelling [86], to improve performance [87], to support internal organisation and management [88], etc.

Big Data systems have been based on Cloud Computing [89]. Nowadays IoT is an important source of big data [90]. The growth rate of IoT system is strong [91], so more data is generated by them. In order to better scale, Edge Computing move part of the computation near of the IoT system [92]. Again, modelling, analysis, and simulation should be used to study how to design, test, and operate Systems based on Cloud/Edge/Fog [93] too.

Unsurprisingly, energy efficiency is a major focus of cloud, fog and edge computing research including the optimisation of resource allocations under energy, performance, and QoS constraints [94]. The main goal of this work is to improve how energy consumption models are simulated. Related to the main goal, main contributions of this work includes: (i) to study supervised learning opportunities in simulation using memoization as a mechanism for this aim; (ii) to introduce an advanced model for memoization applied to energy models in computing simulators; (iii) and a complete analysis on techniques to simulate energy in computer environments.

As summary of the memoization analysis on the energy calculation, Figure 5 shows that *FIND* provides four times faster the same aggregated results of *CUBIC*. And Figure 4 shows that execution time for *FIND* grows near linear with the number of elements whereas *CUBIC* grows much more quickly. Both Figures demonstrate that memoization can be used for learning from past energy consumption simulations, and improve how consumption models are simulated. This work opens new possibilities required on hybrid environment with multiple Clouds and Edge resource simulations. To have a better understanding on these latter environments, the re-usability of this work within these environments might be analogous or requiring a low adaptation. Finally, as was shown in our work, supervised learning through memoization applied to computing simulators reduces the execution time required to perform simulations and therefore, reduce energy consumed by these as well.

As Future Works we plan to study the portability to IoT platforms based on ARM, so learned results in a high-performance node CPU will be used in mid-range SoC element, such as edge devices. Furthermore, next steps on supervised learning for computing simulation are on studying the impact of imprecise computation in the precision of the results under different tolerated ranges of error.

8. Acknowledgements

This material is based upon works supported by the Science Foundation Ireland under Grant No. 12/RC/2289 P2 which is co-funded under the European Regional Development Fund.

This work was partially supported by the Spanish Ministry of Economy, Industry and Competitiveness under the Grant No. TIN2016-79637-P (Towards Unification of HPC and Big Data Paradigms).

9. Bibliography

References

[1] S. R. Group, 2016 Review Shows $148 billion Cloud Market Growing at 25% Annually, Technical Report, Synergy Research Group, Reno, NV, United States, 2017.

[2] Garner, Gartner Says Worldwide IT Spending Forecast to Grow 2.7 Percent in 2017, Technical Report, Garner, STAMFORD, Conn, United States, 2017.

[3] O. Zik, A. Shapiro, Coal computing: How companies misunderstand their dirty data centers, 2017.

[4] J. Whitney, P. Delforge, Data center efficiency assessment-scaling up energy efficiency across the data center industry: Evaluating key drivers and barriers, NRDC and Anthesis, Rep. IP (2014) 14–08.

[5] T. E. Carlson, W. Heirmant, L. Eeckhout, Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation, in: 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 1–12.

[6] J. S. Bucy, J. Schindler, S. W. Schlosser, G. R. Ganger, The disksim simulation environment version 4.0 reference manual (cmu-pdl-08-101), Parallel Data Laboratory (2008) 26.

[7] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, A. Agarwal, Graphite: A distributed parallel simulator for multicores, in: High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on, IEEE, pp. 1–12.

[8] J. Power, J. Hestness, M. S. Orr, M. D. Hill, D. A. Wood, gem5-gpu: A heterogeneous cpu-gpu simulator, IEEE Computer Architecture Letters 14 (2015) 34–36.

[9] C. Grelck, E. Niewiadomska-Szynkiewicz, M. Aldinucci, A. Bracciali, E. Larsson, Why High-Performance Modelling and Simulation for Big Data Applications Matters, Springer International Publishing, Cham, pp. 1–35.

[10] K. Kang, S. Belkhale, G. Kahn, P. Abbeel, S. Levine, Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight, CoRR abs/1902.03701 (2019).

[11] K. Lowrey, S. Kolev, J. Dao, A. Rajeswaran, E. Todorov, Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system, in: 2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR), IEEE, pp. 35–42.

[12] I. El Naqa, M. J. Murphy, What is machine learning?, in: Machine Learning in Radiation Oncology, Springer, 2015, pp. 3–11.

[13] T. Weber, J. Sossenheimer, S. Schäfer, M. Ott, J. Walther, E. Abele, Machine learning based system identification tool for data-based energy and resource modeling and simulation, Procedia CIRP 80 (2019) 683–688.

[14] D. B. Fuller, V. J. M. Ferreira Filho, E. F. de Arruda, Oil industry value chain simulation with learning agents, Computers & Chemical Engineering 111 (2018) 199–209.

[15] C. Sobie, C. Freitas, M. Nicolai, Simulation-driven machine learning: Bearing fault classification, Mechanical Systems and Signal Processing 99 (2018) 403–419.

[16] S. Mittal, A survey of techniques for approximate computing, ACM Comput. Surv. 48 (2016) 62:1–62:33.

[17] A. W. Burks, H. H. Goldstine, J. von Neumann, Preliminary discussion of the logical design of an electronic computing instrument, Technical Report, Institute for Advanced Study, Princeton, NJ, USA, 1946.

[18] J. L. Hennessy, D. A. Patterson, Computer architecture: a quantitative approach, Elsevier, 2011.

[19] J. Isaac Katuka, G. Dams, S. Danjuma, Architectures and technologies of cache memory: A survey, International Journal of Advanced Studies in Computer Science and Engineering 3 (2014) 7–13.

[20] A. J. Smith, Cache memories, ACM Comput. Surv. 14 (1982) 473–530.

[21] E. Winsberg, Science in the Age of Computer Simulation, University of Chicago Press, Chicago, IL, USA, 2010.

[22] T. R. Henderson, M. Lacage, G. F. Riley, Network simulations with the ns-3 simulator, in: In Sigcomm (Demo, pp. 527–528.

[23] J. Liu, D. Nicol, B. Premore, A. Poplawski, Performance prediction of a parallel simulator, in: Proceedings Thirteenth Workshop on Parallel and Distributed Simulation. PADS 99. (Cat. No.PR00155), pp. 156–164.

[24] A. Varga, The OMNeT++ Discrete Event Simulation System, Proceedings of the European Simulation Multiconference (ESM'2001) (2001).

[25] S. S. Rutherford, S. P. Gordon, Using opnet to simulate an ip network, in: Proceedings of the 36th Conference on Winter Simulation, WSC '04, Winter Simulation Conference, 2004, pp. 2107–2207.

[26] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, Simics: A Full System Simulation Platform, Computer 35 (2002) 50–58.

[27] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, D. A. Wood, The gem5 simulator, SIGARCH Comput. Archit. News 39 (2011) 1–7.

[28] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, S. K. Reinhardt, The m5 simulator: Modeling networked systems, IEEE Micro 26 (2006) 52–60.

[29] N. Hardavellas, S. Somogyi, T. F. Wenisch, R. E. Wunderlich, S. Chen, J. Kim, B. Falsafi, J. C. Hoe, A. G. Nowatzyk, Simflex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture, SIGMETRICS Perform. Eval. Rev. 31 (2004) 31–34.

[30] A. Núñez, J. Fernández, J. D. Garcia, L. Prada, J. Carretero, Simcan: A simulator framework for computer architectures and storage networks, in: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, Simutools '08, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 2008, pp. 73:1–73:8.

[31] R. Buyya, M. M. Murshed, Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, CoRR cs.DC/0203019 (2002).

[32] W. H. Bell, D. G. Cameron, A. P. Millar, L. Capozza, K. Stockinger, F. Zini, Optorsim: A grid simulator for studying dynamic data replication strategies, The International Journal of High Performance Computing Applications 17 (2003) 403–416.

[33] C. L. Dumitrescu, I. Foster, Gangsim: a simulator for grid scheduling studies, in: Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on, volume 2, IEEE, pp. 1151–1158.

[34] W. Zhao, Y. Peng, F. Xie, Z. Dai, Modeling and simulation of cloud computing: A review, in: Cloud Computing

Congress (APCloudCC), 2012 IEEE Asia Pacific, IEEE, pp. 20–24.

[35] A. Ahmed, A. S. Sabyasachi, Cloud computing simulators: A detailed survey and future direction, in: Advance Computing Conference (IACC), 2014 IEEE International, IEEE, pp. 866–872.

[36] T. Lynn, A. Gourinovitch, J. Byrne, P. Byrne, S. Svorobej, K. Giannoutakis, D. Kenny, J. Morrison, A preliminary systematic review of computer science literature on cloud computing research using open source simulation platforms, 2017.

[37] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and experience 41 (2011) 23–50.

[38] A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, I. M. Llorente, icancloud: A flexible and scalable cloud infrastructure simulator, Journal of Grid Computing 10 (2012) 185–209.

[39] T. Qayyum, A. W. Malik, M. A. Khan Khattak, O. Khalid, S. U. Khan, Fognetsim++: A toolkit for modeling and simulation of distributed fog environment, IEEE Access 6 (2018) 63570–63583.

[40] S.-H. Lim, B. Sharma, G. Nam, E. K. Kim, C. R. Das, Mdcsim: A multi-tier data center simulation, platform, in: Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on, IEEE, pp. 1–9.

[41] R. N. Calheiros, M. A. Netto, C. A. De Rose, R. Buyya, Emusim: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications, Software: Practice and Experience 43 (2013) 595–612.

[42] M. Tighe, G. Keller, M. Bauer, H. Lutfiyya, Dcsim: A data centre simulation tool for evaluating dynamic virtualized resource management, in: Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm), IEEE, pp. 385–392.

[43] I. Sriram, Speci, a simulation tool exploring cloud-scale data centres, in: IEEE International Conference on Cloud Computing, Springer, pp. 381–392.

[44] M. Becker, S. Becker, J. Meyer, Simulizar: Design-time modeling and performance analysis of self-adaptive systems., Software Engineering 213 (2013) 71–84.

[45] D. Kliazovich, P. Bouvry, S. U. Khan, Greencloud: a packet-level simulator of energy-aware cloud computing data centers, The Journal of Supercomputing 62 (2012) 1263–1283.

[46] H. Casanova, A. Legrand, M. Quinson, Simgrid: A generic framework for large-scale distributed experiments, in: Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on, IEEE, pp. 126–131.

[47] H. Gupta, A. Vahid Dastjerdi, S. Ghosh, R. Buyya, ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments, Software: Practice and Experience (2016).

[48] C. Sonmez, A. Ozgovde, C. Ersoy, Edgecloudsim: An environment for performance evaluation of edge computing systems, in: 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), pp. 39–44.

[49] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, R. Buyya, Containercloudsim: An environment for modeling and simulation of containers in cloud data centers, Software: Practice and Experience 47 (2017) 505–521.

[50] D. Stamenov, M. Kostoska, Virtual machine migration in cloud–techniques, challenges and cloudsim migration simulation, in: 14th International Conference on Informatics and Information Technologies, Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University in Skopje, Macedonia, 2017, pp. 1–408.

[51] C. Filelis-Papadopoulos, G. Gravvanis, P. Kyziropoulos, A framework for simulating large scale cloud infrastructures, Future Generation Computer Systems 79 (2017).

[52] Z. Yang, Y. C. Yabansu, R. Al-Bahrani, W. keng Liao, A. N. Choudhary, S. R. Kalidindi, A. Agrawal, Deep learning approaches for mining structure-property linkages in high contrast composites from simulation datasets, Computational Materials Science 151 (2018) 278 – 287.

[53] A. Pérez, G. Martínez-Rosell, G. D. Fabritiis, Simulations meet machine learning in structural biology, Current Opinion in Structural Biology 49 (2018) 139 – 144. Theory and simulation • Macromolecular assemblies.

[54] J. L. Bentley, Writing efficient programs, Prentice Hall, Inc., Upper Saddle River, NJ, USA, pp. 1–170.

[55] R. K. Iyer, J. A. Bilmes, A memoization framework for scaling submodular optimization to large scale problems, CoRR abs/1902.10176 (2019).

[56] D. Michie, U. of Edinburgh. Department of Machine Intelligence & Perception, Memo Functions: A Language Feature with "rote-learning" Properties, MIP-R-, Edinburgh University, Department of Machine Intelligence and Perception, 1967.

[57] K. Kamimura, R. Oda, T. Yamada, T. Tsumura, H. Matsuo, Y. Nakashima, A speed-up technique for an auto-memoization processor by reusing partial results of instruction regions, in: Networking and Computing (ICNC), 2012 Third International Conference on, IEEE, pp. 49–57.

[58] S. E. Richardson, Caching Function Results: Faster Arithmetic by Avoiding Unnecessary Computation, Technical Report, Sun Microsystems, Inc., Mountain View, CA, USA, 1992.

[59] I. Bratko, Prolog programming for artificial intelligence, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[60] L. Ziarek, K. Sivaramakrishnan, S. Jagannathan, Partial memoization of concurrency and communication, SIGPLAN Not. 44 (2009) 161–172.

[61] E. H. Mulalic, M. S. Stankovic, R. S. Stankovic, Memoization technique for optimizing functions with stochastic input, CoRR abs/1211.5173 (2012).

[62] H. Cui, J. Wu, C.-C. Tsai, J. Yang, Stable deterministic multithreading through schedule memoization, in: Proceedings of the 9th USENIX conference on Operating Systems design and implementation, USENIX Association, pp. 1–13.

[63] E. Schnarr, J. R. Larus, Fast out-of-order processor simulation using memoization, SIGOPS Oper. Syst. Rev. 32 (1998) 283–294.

[64] D. Yamada, T. Sonobe, H. Tezuka, M. Inaba, Grid spider: a framework for data intensive research with data process memoization cache, in: INTENSIVE 2012, The Fourth International Conference on Resource Intensive Applications and Services, pp. 5–8.

[65] G. Agosta, M. Bessi, E. Capra, C. Francalanci, Automatic memoization for energy efficiency in financial applications, Sustainable Computing: Informatics and Systems 2 (2012) 105 – 115.

[66] A. Calderón, A. García, F. García-Carballeira, J. Carretero, J. Fernández, Improving performance using computational compression through memoization: A case study using a railway power consumption simulator, The International Journal of High Performance Computing Applications 30 (2016) 469–485.

[67] M. Stoffers, R. Bettermann, K. Wehrle, Automated memoization: Automatically identifying memoization units in simulation parameter studies, in: 21st IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2017, Rome, Italy, October 18-20, 2017, pp. 33–42.

[68] M. Stoffers, R. Bettermann, K. Wehrle, Automated memoization: Automatically identifying memoization units in simulation parameter studies, in: 2017 IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications (DS-RT), IEEE, pp. 1–10.

[69] M. Stoffers, D. Schemmel, O. Soria Dustmann, K. Wehrle, Automated memoization for parameter studies implemented in impure languages, in: Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, SIGSIM-PADS '16, ACM, New York, NY, USA, 2016, pp. 221–232.

[70] M. J. Maher, A. Rock, G. Antoniou, D. Billington, T. Miller, Efficient defeasible reasoning systems, International Journal on Artificial Intelligence Tools 10 (2001) 483–501.

[71] K. Boos, D. Chu, E. Cuervo, Flashback: Immersive virtual reality on mobile devices via rendering memoization, in: Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys '16, ACM, New York, NY, USA, 2016, pp. 291–304.

[72] W. Lee, E. Slaughter, M. Bauer, S. Treichler, T. Warszawski, M. Garland, A. Aiken, Dynamic tracing: memoization of task graphs for dynamic task-based runtimes, in: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, pp. 441–453.

[73] P. Norvig, Techniques for automatic memoization with applications to context-free parsing, Comput. Linguist. 17 (1991) 91–98.

[74] R. Hinze, Memo functions, polytypically!, in: Proceedings of the 2nd Workshop on Generic Programming, Ponte de, pp. 17–32.

[75] P. McNamee, M. Hall, Developing a tool for memoizing functions in c++, SIGPLAN Not. 33 (1998) 17–22.

[76] G. Tziantzioulis, N. Hardavellas, S. Campanoni, Temporal approximate function memoization, IEEE Micro 38 (2018) 60–70.

[77] M. Stoffers, D. Schemmel, O. Soria Dustmann, K. Wehrle, Automated memoization for parameter studies implemented in impure languages, in: Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, ACM, pp. 221–232.

[78] M. Stoffers, D. Schemmel, O. S. Dustmann, K. Wehrle, On automated memoization in the field of simulation parameter studies, ACM Transactions on Modeling and Computer Simulation (TOMACS) 28 (2018) 26.

[79] K. M. Giannoutakis, A. T. Makaratzis, D. Tzovaras, C. K. Filelis-Papadopoulos, G. A. Gravvanis, On the power consumption modeling for the simulation of heterogeneous HPC clouds, in: Proceedings of the 1st International Workshop on Next Generation of Cloud Architectures, CloudNG:17, ACM, New York, NY, USA, 2017, pp. 1:1–1:6.

[80] SPEC - Standard Performance Evaluation Corporation, `http://www.spec.org`, 2019. [Accessed on 2019-May-15].

[81] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, N. Gautam, Managing server energy and operational costs in hosting centers, in: Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '05, ACM, New York, NY, USA, 2005, pp. 303–314.

[82] G. G. Castañé, A. Nuñez, P. Llopis, J. Carretero, E-mc2: A formal framework for energy modelling in cloud computing, Simulation Modelling Practice and Theory 39 (2013) 56–75.

[83] A. T. Makaratzis, C. K. Filelis-Papadopoulos, K. M. Giannoutakis, G. A. Gravvanis, D. Tzovaras, A comparative study of CPU power consumption models for cloud simulation frameworks, in: Proceedings of the 21st Pan-

Hellenic Conference on Informatics, PCI 2017, ACM, New York, NY, USA, 2017, pp. 10:1–10:6.

[84] C. Filelis-Papadopoulos, G. Gravvanis, P. Kyziropoulos, K. Giannoutakis, G. Sikotidis, C. Kouzinopoulos, D7.1.1 large scale modelling and simulation framework, `https://www.ec.europa.eu/research/participants/documents/downloadPublic\?documentId\s=080166e5b475e772\&appId\=PPGMS`, 2017. Accessed: 2019-Aug-08.

[85] GNU Compiler Collection Developers, GNU Compiler Collection Flags, 2019. Accessed: 2019-May-15.

[86] G. G. Castañé, A. Núñez, P. Llopis, J. Carretero, E-mc2: A formal framework for energy modelling in cloud computing, Simulation Modelling Practice and Theory 39 (2013) 56 – 75. S.I.Energy efficiency in grids and clouds.

[87] G. G. Castañè, A. Núneñ, R. Filgueira, J. Carretero, Dimensioning scientific computing systems to improve performance of map-reduce based applications, Procedia Computer Science 9 (2012) 226 – 235. Proceedings of the International Conference on Computational Science, ICCS 2012.

[88] C. Filelis-Papadopoulos, H. Xiong, A. Spătaru, G. G. Castañé, D. Dong, G. A. Gravvanis, J. P. Morrison, A generic framework supporting self-organisation and self-management in hierarchical systems, in: 2017 16th International Symposium on Parallel and Distributed Computing (ISPDC), pp. 149–156.

[89] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, S. U. Khan, The rise of "big data" on cloud computing: Review and open research issues, Information Systems 47 (2015) 98 – 115.

[90] M. Chen, S. Mao, Y. Liu, Big data: A survey, Mobile Networks and Applications 19 (2014) 171–209.

[91] P. Suresh, J. V. Daniel, V. Parthasarathy, R. H. Aswathy, A state of the art review on the internet of things (IoT) history, technology and fields of deployment, in: 2014 International Conference on Science Engineering and Management Research (ICSEMR), pp. 1–8.

[92] M. Satyanarayanan, The emergence of edge computing, Computer 50 (2017) 30–39.

[93] G. Kecskemeti, G. Casale, D. N. Jha, J. Lyon, R. Ranjan, Modelling and simulation challenges in Internet of Things, IEEE Cloud Computing 4 (2017) 62–69.

[94] S. Svorobej, P. Takako Endo, M. Bendechache, C. Filelis-Papadopoulos, K. M. Giannoutakis, G. A. Gravvanis, D. Tzovaras, J. Byrne, T. Lynn, Simulating fog and edge computing scenarios: An overview and research challenges, Future Internet 11 (2019).

Appendix A. Evaluation Results

Opt	Algorithm	Number elements / Time elapsed (us)															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-O0	CUBIC	1	20	22	35	51	65	83	99	112	130	144	161	176	190	324	195
	LINEAR	1	1	2	2	4	4	4	6	6	6	8	8	9	9	10	12
	SQUARE	0	4	5	8	11	13	14	17	19	21	25	26	29	31	33	37
	SQUARE-ROOT	1	2	4	5	7	8	10	10	12	13	15	17	17	19	20	23
	PM1	1	3	9	9	12	16	18	20	23	27	31	36	40	42	46	51
	PM2	1	14	33	52	72	89	106	123	142	161	180	199	220	235	251	295
	FIND	11	12	13	18	20	20	26	27	28	30	31	32	37	37	39	44
-O1	CUBIC	0	16	21	31	44	57	72	137	102	114	120	131	143	156	171	199
	LINEAR	1	1	0	0	1	1	2	1	3	2	2	3	2	2	3	3
	SQUARE	1	1	1	1	1	2	2	2	3	2	3	3	4	4	4	5
	SQUARE-ROOT	1	1	1	3	3	5	5	6	8	8	9	10	12	12	13	15
	PM1	1	2	5	7	9	11	14	16	18	21	24	27	31	35	36	41
	PM2	0	11	30	44	64	80	96	111	128	147	160	175	188	199	211	251
	FIND	3	4	4	6	6	6	8	8	8	11	11	10	12	13	13	15
-O2	CUBIC	1	17	23	34	60	63	86	95	107	124	140	153	168	181	199	233
	LINEAR	0	1	1	1	2	2	2	2	3	3	3	3	3	4	4	4
	SQUARE	1	1	1	1	2	2	2	3	3	3	3	4	4	5	4	5
	SQUARE-ROOT	1	1	2	3	4	6	7	8	10	10	12	14	14	16	17	19
	PM1	1	3	6	8	11	14	16	18	21	26	28	32	38	41	42	48
	PM2	1	13	35	54	75	95	112	132	150	171	189	202	221	300	186	198
	FIND	3	3	3	5	4	5	7	7	6	9	9	8	10	10	10	12
-O3	CUBIC	1	15	18	31	44	57	72	87	98	114	130	141	154	167	183	214
	LINEAR	0	1	1	1	2	1	2	2	3	2	3	3	3	4	4	4
	SQUARE	0	1	1	1	2	2	2	2	3	3	3	4	4	4	4	5
	SQUARE-ROOT	0	1	3	5	7	8	11	12	14	16	17	20	22	23	25	29
	PM1	1	2	9	7	10	12	15	17	20	25	27	28	31	38	36	42
	PM2	0	12	32	48	69	86	102	121	137	158	378	192	201	215	229	261
	FIND	4	4	4	6	7	6	9	9	9	11	11	11	13	13	13	15
-Ofast	CUBIC	1	1	2	2	2	2	3	3	2	3	3	4	3	3	4	4
	LINEAR	1	1	1	2	1	2	2	2	1	2	3	2	3	3	3	3
	SQUARE	0	1	1	1	1	2	2	3	2	3	3	3	3	3	3	3
	SQUARE-ROOT	0	2	3	5	7	8	5	5	3	4	5	6	4	5	6	5
	PM1	0	6	10	7	11	12	14	17	17	19	23	26	23	24	28	30
	PM2	0	2	2	3	4	3	4	5	4	5	5	6	5	6	6	7
	FIND	3	3	4	6	7	6	9	9	8	12	11	11	13	13	14	16
-g	CUBIC	1	16	20	33	46	61	77	91	105	119	136	149	163	185	198	226
	LINEAR	1	2	2	3	3	5	5	6	7	7	9	9	10	10	12	13
	SQUARE	1	4	6	9	12	14	17	20	22	25	28	30	33	35	38	44
	SQUARE-ROOT	1	2	4	6	7	10	10	11	13	15	16	17	19	20	22	25
	PM1	1	3	10	10	13	16	20	23	27	31	35	42	42	44	48	55
	PM2	1	16	39	59	83	103	124	144	165	188	208	231	252	271	295	326
	FIND	12	12	13	19	20	20	26	29	28	34	36	37	42	44	45	51

Table A.2: Energy simulation models analysis for cloud computing systems: time spent for computing values.

Opt	Algorithm	Number elements / Values computed															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-O0	CUBIC	0	100	200	300	401	502	605	709	816	926	1041	1162	1289	1424	1569	1893
	LINEAR	0	100	206	318	435	559	688	824	965	1112	1265	1424	1588	1759	1935	2306
	SQUARE	0	100	200	302	405	510	619	731	848	971	1099	1233	1375	1525	1683	2029
	SQUARE-ROOT	0	100	224	359	501	649	803	963	1127	1295	1468	1645	1825	2009	2197	2582
	PM1	0	100	200	301	401	502	604	705	807	909	1011	1113	1216	1319	1422	1629
	PM2	0	100	190	275	357	439	524	614	712	818	934	1061	1200	1350	1512	1868
	FIND	0	100	224	359	501	649	803	963	1127	1295	1468	1645	1825	2009	2197	2582
-O1	CUBIC	0	100	200	300	401	502	605	709	816	926	1041	1162	1289	1424	1569	1893
	LINEAR	0	100	206	318	435	559	688	824	965	1112	1265	1424	1588	1759	1935	2306
	SQUARE	0	100	200	302	405	510	619	731	848	971	1099	1233	1375	1525	1683	2029
	SQUARE-ROOT	0	100	224	359	501	649	803	963	1127	1295	1468	1645	1825	2009	2197	2582
	PM1	0	100	200	301	401	502	604	705	807	909	1011	1113	1216	1319	1422	1629
	PM2	0	100	190	275	357	439	524	614	712	818	934	1061	1200	1350	1512	1868
	FIND	0	100	224	359	501	649	803	963	1127	1295	1468	1645	1825	2009	2197	2582
-O2	CUBIC	0	100	200	300	401	502	605	709	816	926	1041	1162	1289	1424	1569	1893
	LINEAR	0	100	206	318	435	559	688	824	965	1112	1265	1424	1588	1759	1935	2306
	SQUARE	0	100	200	302	405	510	619	731	848	971	1099	1233	1375	1525	1683	2029
	SQUARE-ROOT	0	100	224	359	501	649	803	963	1127	1295	1468	1645	1825	2009	2197	2582
	PM1	0	100	200	301	401	502	604	705	807	909	1011	1113	1216	1319	1422	1629
	PM2	0	100	190	275	357	439	524	614	712	818	934	1061	1200	1350	1512	1868
	FIND	0	100	224	359	501	649	803	963	1127	1295	1468	1645	1825	2009	2197	2582
-O3	CUBIC	0	100	200	300	401	502	605	709	816	926	1041	1162	1289	1424	1569	1893
	LINEAR	0	100	206	318	435	559	688	824	965	1112	1265	1424	1588	1759	1935	2306
	SQUARE	0	100	200	302	405	510	619	731	848	971	1099	1233	1375	1525	1683	2029
	SQUARE-ROOT	0	100	224	359	501	649	803	963	1127	1295	1468	1645	1825	2009	2197	2582
	PM1	0	100	200	301	401	502	604	705	807	909	1011	1113	1216	1319	1422	1629
	PM2	0	100	190	275	357	439	524	614	712	818	934	1061	1200	1350	1512	1868
	FIND	0	100	224	359	501	649	803	963	1127	1295	1468	1645	1825	2009	2197	2582
-Ofast	CUBIC	0	100	200	300	401	502	605	709	816	926	1041	1162	1289	1424	1569	1893
	LINEAR	0	100	206	318	435	559	688	824	965	1112	1265	1424	1588	1759	1935	2306
	SQUARE	0	100	200	302	405	510	619	731	848	971	1099	1233	1375	1525	1683	2029
	SQUARE-ROOT	0	100	224	359	501	649	803	963	1127	1295	1468	1645	1825	2009	2197	2582
	PM1	0	100	200	301	401	502	604	705	807	909	1011	1113	1216	1319	1422	1629
	PM2	0	100	190	275	357	439	524	614	712	818	934	1061	1200	1350	1512	1868
	FIND	0	100	224	359	501	649	803	963	1127	1295	1468	1645	1825	2009	2197	2582
-g	CUBIC	0	100	200	300	401	502	605	709	816	926	1041	1162	1289	1424	1569	1893
	LINEAR	0	100	206	318	435	559	688	824	965	1112	1265	1424	1588	1759	1935	2306
	SQUARE	0	100	200	302	405	510	619	731	848	971	1099	1233	1375	1525	1683	2029
	SQUARE-ROOT	0	100	224	359	501	649	803	963	1127	1295	1468	1645	1825	2009	2197	2582
	PM1	0	100	200	301	401	502	604	705	807	909	1011	1113	1216	1319	1422	1629
	PM2	0	100	190	275	357	439	524	614	712	818	934	1061	1200	1350	1512	1868
	FIND	0	100	224	359	501	649	803	963	1127	1295	1468	1645	1825	2009	2197	2582

Table A.3: Energy simulation models analysis for cloud computing systems: computed values