An energy saving approach

Hernán H Álvarez Valera, Marc Dalmau, Philippe Roose, Jorge Larracoechea, Christina Herzog

An energy saving approach: Understanding microservices as multidimensional entities in p2p networks

Hernan Humberto Alvarez Valera
E2S UPPA, University of Pau
Anglet, France
hhavalera@univ-pau.fr

Marc Dalmau
E2S UPPA, University of Pau
Anglet, France
Marc.Dalmau@iutbayonne.univ-pau.fr

Philippe Roose
E2S UPPA, University of Pau
Anglet, France
Philippe.Roose@iutbayonne.univ-pau.fr

Jorge Larracoechea
E2S UPPA, University of Pau
Anglet, France
jorge-andres.larracoechea@etud.univ-pau.fr

Christina Herzog
EFFICIT SAS
Mauzac, France
herzog@efficit.com

ABSTRACT

With the use of microservices, many software solutions have been improved in terms of scalability, response efficiency, ease of load balancing among others. However, it is still a challenge to dynamically deploy them according to devices' heterogeneity and energy consumption concerns, while maintaining a defined QoS. Centralized and decentralized approaches that manage microservices deployment have the traditional pros and cons long discussed over time. While the former offer greater control over distributed application components, the latter offers frugal network negotiations, no system-wide crashes, privacy, among others.

This work focuses on identifying "ideal" host candidates for microservices' execution in a decentralized network, applying runtime scheduling operations (migration or duplication) to reduce energy consumption. To do this, we created a scheduling algorithm using MAAN (a P2P approach) to interpret a decentralized network as a multidimensional resource (capacity-demand) space, which supports range queries in a logarithmic quantity of hops. In this way, a node that runs a set of microservices is able to 1) map them in terms of their execution requirements (i.e. CPU frequency, RAM capacity, Network rate and disk speed) 2) Select an ideal microservice to be moved or duplicated, 3) find ideal node(s) that meet all those requirements in an optimal computational complexity and 4) negotiate the movement or duplication of the selected microservice, by analyzing energy consumption and QoS criteria.

CCS CONCEPTS

• **Computing methodologies** → **Distributed algorithms**; • **Computer systems organization** → **Peer-to-peer architectures**; **Cloud computing**;

KEYWORDS

distributed algorithms , power consumption, energy consumption, P2P, microservices, cloud computing

ACM Reference Format:
Hernan Humberto Alvarez Valera, Marc Dalmau, Philippe Roose, Jorge Larracoechea, and Christina Herzog. 2021. An energy saving approach: Understanding microservices as, multidimensional entities in p2p networks. In *The 36th ACM/SIGAPP Symposium on Applied Computing (SAC '21), March 22–26, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3412841.3441888

SAC '21, March 22–26, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8104-8/21/03…$15.00
https://doi.org/10.1145/3412841.3441888

1 INTRODUCTION

Nowadays, the use of microservices has become widespread in the conception of distributed applications. Microservices are "small" services running in its own process and communicating with lightweight mechanisms[15] which permit to deal with problems of applications' load balancing, high availability, scalability, among others. This is why many companies such Amazon[47] and Netflix[38], among others, use microservices to deploy their applications and at the same time, offer the use of their microservices technologies through APIs in a dynamic and efficient way.

Microservices are difficult to manage despite their advantages. Their orchestration, which consists in finding the best way of deployment, scaling, load balancing and scheduling[8] remains a challenge. Criterias such as the amount of energy consumed, control of devices with heterogeneous capabilities or ensuring optimal QoS, must be evaluated by intelligent distributed scheduling algorithms to achieve desired results.

Such algorithms can be oriented to either centralized or decentralized philosophies. Each of them has advantages and disadvantages, depending on the type of application, deployment politic or deployment level (i.e. Cloud, grid or host). For example, while Netflix prefers centralized microservices orchestration (by it's well known Conductor) to choreography mainly for scalability issues[38], de langue et. al[12] took advantage of a decentralized environment to develop a non-formal learning system in CoPs (Communities of Practice) based on a microservices architecture. Similarly, there are technologies such as Kalimucho[11], which offer a middleware platform to intelligently manage microservices in a p2p network of

heterogeneous devices, enabling microservices movement, duplication and routing operations.

On the other hand, we are aware that excessive energy consumption has economic and sustainability repercussions. For this reason several works aim at optimizing the use of resources for energy saving purposes. They use the two approaches mentioned in the previous paragraph in order to manage energy consumption according to their applications and/or their architecture. Thereby, some works use centralized approaches to decrease energy consumption. For example, they studied CPU's features and capacities to be able to model mathematically the energy consumption over a period of time[42][46], in order to take centralized decisions like migrating virtual machines[43] in data centers or cloud environments, allowing the reduction of CPU frequency, use PCPG[28], or even put selected servers in sleep mode[1][48]. Facing the same problem, decentralized approaches save energy by storing information close to nodes[39] in order to reduce hops when recovering it, or establishing negotiation processes between nodes so that both information and data are relocated to more energy-efficient processing units[32], taking into account the characteristics of the software modules and those of the hardware[30].

Since the two approaches always try to optimize the use of each device hardware component to reduce energy consumption, either by (1) taking full advantage of their particular characteristics (as CPU-DVFS), (2) reducing the processing time of a task, (3) keeping only the necessary devices on, or (4)improving process scheduling; in this work, we improve the idea implemented into Kaligreen. In both of its versions[32][30], it is shown that moving, duplicating, stopping and restarting microservices taking into account the heterogeneity of hardware components in a p2p network can reduce energy consumption without establishing expensive negotiations with a single central device, which can become saturated or even fail, causing the entire network to crash.

To make this improvement, we use the concepts of overlay p2p networks and multidimensional data structures. For this, we deployed a 4-dimensional space on a MAAN[4] network overlay. This technique organizes the nodes of a non-centralized network in a circular list whose search method is based on a table of addresses called "finger table". With it, the search for information or a node through another is carried out in a logarithmic number of hops.

We use MAAN to contrast the microservices' needs for its execution and the hardware availability that each device has. Our approach is based on the fact that in a certain period of time or under certain circumstances (i.e. battery problems, overload, etc.), one device can find a peer in order to negotiate with it and move or duplicate a selected microservice M to save energy. To perform this "find" operation, we consider the processing requirements of M in terms of CPU frequency, RAM, network transfer rate, disk speed and dependencies management (applications using M and QoS repercussions), as well as the specific characteristics of the involved devices in the same terms. Finally, to carry out the negotiation process, we implemented a distributed scheduling algorithm based on Kaligreen, in which, circumstantially, each node tries to save energy by moving its heaviest/lightest microservice to another peer that can run it in a more frugal manner.

To demonstrate the efficiency of our approach, we deployed 2 types of experiments: One based on scalability in which the number of devices and microservices grows over time and the other based on stress, in which devices are saturated by a big quantity of microservices. The first one shows that our approach is capable of saving 27% of energy against a deployment without an algorithm and 11% against the one proposed by Kaligreen with an optimal QoS level. The second experiment ensures a higher quality of service by a margin of 5% against the aforementioned approaches besides displaying a higher energy consumption. To our knowledge, this is the first work that tries to focus on energy consumption using multidimensional resources (CPU, RAM, network and HDD) approach, which allows linking "ideal" microservices with "ideal devices" in a logarithmic number of hops.

In order to explain the abstract part of our approach, this article introduces multidimensional spaces in section 2, describes our multidimensional resources space in section 3; and explains the relation between hardware component's load and energy consumption in section 4. Then, to explain the implementation of our approach, it describes multidimensional P2P approaches in section 5, the implementation of our multidimensional space in section 6, our distributed scheduling algorithm in section 7; and the experiments and results in section 8. Finally, the article explains our conclusions and future works in section 9.

2 MULTIDIMENSIONAL SPACES

Data management is a great concern in computer science. For this reason, there are structures that allow modeling data sets as spaces with as many dimensions as data sets have characteristics. They allow data to be inserted, identified and retrieved at efficient operation costs. As one dimensional examples, all binary-based trees like AVL, RedBlack and B* perform data management operations in O(Log(N)) in the average case; while hash based structures do the same in O(1) in the best case. However, these operations become more complicated with the increment of search dimensions (data characteristics). To deal with this problem, several solutions based on metrics (defined by distance functions) and/or n-dimensional spaces have been created. These are able to manage multidimensional data sets from the viewpoint of: 1) The distances among its elements[5] (e.g. BKT, BT or LAESA) or 2) the abstraction of the universe in which they exist[16] (e.g. K-D-Tree, R*tree or Z-ordering).

These data management approaches allow not only to process queries based on a single characteristic/dimension to obtain a single element, but also to retrieve all data that meets various criteria in a certain range or region, either from the viewpoint of another element or from the viewpoint of the entire universe.

Edgar Chavez et al.[5] describe the possible queries in metric multidimensional spaces, also applicable in spatial access methods:

- Range query $(q, r)_d$. Retrieve all elements which are within distance r to q. This is, $\{u \in U \mid d(q, u) \leq r\}$.
- Nearest neighbor query $NN(q)$. Retrieve the closest element to q in U. This is, $\{u \in U \mid \forall v \in U, d(q, u) \leq d(q, v)\}$.
- K-Nearest neighbor query $NN_k(q)$. Retrieve the k closest elements to q in U. This is, retrieve a set $A \subset U$ such that $|A| = k$ and $\forall u \in A, v \in U - A, d(q, u) \leq d(q, v)$.

In this paper, we propose that querying a multidimensional space of hardware resources is very useful to understand how to save energy in a microservices environment, since energy consumption is strongly related to the way of using hardware resources. In other words, at a certain moment, for a microservice M that is executed in a device D generating a load of L and an energy consumption E, it is possible to find a device D' that: (1) has at least availability of resources C, in such a way that $C >= L$ and (2) the energy consumption E' generated by M in D' is less than E (because of current load or particular characteristics of involved hardware components). In the next section, we will explain our hardware resource space and its features.

3 MULTIDIMENSIONAL RESOURCES SPACE

In a decentralized network environment, managing microservices is challenging. Concepts such as load balancing, microservices discovery or energy saving are difficult to analyze in the absence of a central entity that contains all the information necessary to apply scheduling heuristics. However, we believe that by organizing the devices of a P2P network in an intelligent way, it is possible to replace centralized scheduling by optimal negotiations between its members. Thereby, (1) a device with overload or energy problems (as low battery situation or fans energy spending) can move a microservice to another peer to decrease its load and save energy or (2) an underloaded device can offer help to other peers to process microservices in a more energy-efficient way[32]. This is why multidimensional spaces are the cornerstone of our work. The basis of our approach is to abstract and efficiently query hardware resources as a 4-dimensional space. It contains devices' available capabilities in terms of CPU frequency, RAM capacity, network rate and hard drive speed.

Definition 3.1. Given a set of devices connected to a common network, we define a space U made up of 4 dimensions/axis: (1) CPU frequency in GHz(2) RAM capacity in MB (3) network transfer rate and (4) Hard disk speed. Devices are elements of U in terms of its hardware resources current availability.

Queries in this space will allow a device to know which peers have sufficient resources to run a certain microservice. For this, in order to decide whether a microservice is a candidate to be moved or duplicated in a subsequent negotiation process with another peer, a device should analyze the microservice's processing requirements, type and restrictions. The authors of Kaligreen V2[30] made an interesting analysis of the characteristics that should be analyzed before operating a microservice. For example, they considered that:

- There are microservices which remain running until a target is achieved, while others are kept running permanently pending requests.
- There are microservices that can be moved or not based on their objectives. For example, while a microservice which performs high-cost CPU operations could be moved or duplicated to balance load; a microservice that represents an element of a graphical interface[37] should not always be moved since it works usually with a single screen.

Then, in order to perform microservices movement or duplication operations, Kaligreen authors contrasted this information with: 1) its CPU load, 2) its size in RAM memory, 3) the amount of data it handles on disk and 4) the amount of data it transfers to other microservices and/or other external entities. In our case, we will use the same analytical criteria; however: (1) instead of taking into account the amount of data that a microservice manages on the disk, we study the latency it produces since we are also interested in the energy consumed by this device in terms of its load, leaving microservices-driven data analysis to future work and (2) we do not take into account the functional aspects of each microservice as axes of our space U. Instead, we study them at the time of prior negotiation of a scheduling operation.

As it is possible to see, in this work a device can self-analyze in terms of its load, its energy situation and the microservices it executes. Moreover, it has the ability to decide which microservice to move or duplicate based on energy saving and load balancing scheduling heuristics. Finally, it will be able to efficiently search for candidate peers to execute, after a negotiation process, the selected microservice(s) querying the resources U space.

In the next section, we will explain the relation between the load of each hardware component and energy consumption.

4 ANALYSIS OF HARDWARE COMPONENTS (AXES OF SPACE U)

Whenever energy saving is sought in any network of devices, it is necessary to study the characteristics of devices' hardware components and how they are used by running processes. In our case, as our approach is based on microservices execution taking into account their load in terms of CPU frequency, RAM, network rate and disk speed (axis of our 4-dimensional space), we will explain in the following paragraphs some formulas that allow understanding for each component: (1) How much energy is consumed in terms of its load, (2) how much energy is consumed by a running microservice and (3) How much energy would a microservice consume if migrated or duplicated on another device.

4.1 CPU

Several works study power consumption by analyzing running software features and the physical properties that a CPU has. Some of these approaches consist of:

- Finding the relation between a standardized workload and energy consumption[46][3][19].
- Finding the relation between the type of task and the clock cycles it generates[22][24][18].
- Finding the consumption difference between real or virtualized environments[6].
- Using capabilities like DVFS or PGPC[30][28].
- Finding the relation between CPU energy consumption and its frequency[12].

We will base our study on the last two points because we consider the first three more appropriate for the analysis of energy consumption in the stages of application conception, analysis and design; while the other two are useful for our analysis of the scheduling of already running distributed applications. Consequently, in order to find the amount of energy consumed in a certain time T by the CPU of a device, henceforth device D, in terms of its load, we will use the formula provided by intel[19] (Ohm's law). This

formula alone calculates the power consumption in watts of $D's$ CPU P_{CPU_D} in terms of its capacitance being switched per clock cycle (C), voltage (V), and the processor frequency F (cycles per second). To obtain the energy consumption in joules (watts/s), we add the factor T in seconds to the formula.

$$E_{CPU_D} = C_D V_D^2 F_D T_D \tag{1}$$

4.1.1 Microservice's energy consumption in terms of the CPU. In order to know the energy consumed by a microservice, henceforth M, in terms of $D's$ CPU, we interpret equation 1 proportionally to the load that M generates in this component. Thereby, If we know this processor's capacitance C, voltage V and the total frequency F in a certain moment; and if M loads it at $X\%$, the energy consumed to process M will be calculated in the same proportion as the mentioned load, as shown in Equation 2:

$$E_{CPU_D} = \frac{XC_D V_D^2 F_D T_D}{100} \tag{2}$$

4.1.2 Microservice's energy consumption in terms of an external CPU. In order to know the energy consumed by M on an external device's CPU, we will interpret Equation 2 in such a way that we deduce the number of $M's$ program instructions taking into account the number of instructions per unit of time that the processor can execute. Thereby, if (1) $D's$ CPU is capable of executing I_D program's instructions in a time $T1$ and (2) M consumes $X\%$ of this component capacity in $T1(X_{M_D})$, then M executes I_{M_D} program's instructions in a proportional way too:

$$I_{M_D} = \frac{X_{M_D} I_D}{100} \tag{3}$$

Then, in the context of another device $D2$, it is possible to deduce the percentage $X_{M_{D2}}$ of its CPU load if M would be executed in, based on the number of instructions I_{D2} in $T1$ that $D2$ can execute:

$$X_{M_{D2}} = \frac{X_{M_D} I_D}{I_{D2}} \tag{4}$$

Finally, the energy that $D2's$ CPU would consume to process M $E_{M_{D2}}$ can be calculated following the relation shown in equation 1; but taking into account the proportions that we have just explained in the last two paragraphs:

$$E_{M_{D2}} = \frac{X_{M_D} I_D C_{D2} V_{D2}^2 F_{D2} T}{100 I_{D2}} \tag{5}$$

Equation 5 allows obtaining from D an estimation of the energy that $D2's$ CPU would consume if it executes M. Thus, in a negotiation process, this consumption can be analyzed before a migration or duplication operation.

Finally, we are aware that other characteristics (DVFS, PCPF, etc) have energy consumption repercussions; however, a mathematical model considering them that takes these criteria into account will be presented in a future work.

4.2 RAM

Studying RAM's energy consumption from the viewpoint of running applications is a complex task. On one hand, RAM's load does not represent a highly relevant factor to energy consumption and on the other hand, energy consumption by a number of operations is not easily measurable[25]. For these reasons, we will interpret the energy consumption of a device's RAM E_{RAM_D} as a constant power value X_D multiplied by the analysis time T in seconds (watts/s).

$$E_{RAM_D} = X_D T \tag{6}$$

4.2.1 Microservice's energy consumption in terms of the RAM. Although RAM's load results in relatively small increases in memory power consumption, some authors[22][33] study RAM energy consumption by focusing on criteria such as the number of accesses and operations the RAM performs in the context of a program execution. These criteria beyonds the scope of this work, since they belong to the stage of application's design and development[33]. On the other hand, we have not found a way to model the energy consumption of an already running microservice in terms of the load it generates in the RAM. For this reason, for us it is important to evaluate the RAM consumption only to avoid overload situations since this could have serious consequences for application's performance (due to increased disk latency) and energy consumption.

4.3 Network and disk devices

To model the energy consumption of device's network interfaces and disk units, it is necessary to consider their heterogeneity. In the case of the network interfaces, different types of connection (i.e. ethernet, WIFI and 2, 3, 4, 5G ,etc.) should be considered. Furthermore, there are methodologies that allow finding the energy consumption of a specific network card: (1) from its operating state (i.e. sleep, idle, downlink, uplink)[7], taking into account its data sheets or using external measuring devices[7]; or (2) by simulation[40][9]. Thereby, many works perform their energy saving strategies in terms of NICs, by scheduling the amount of data transmitted[26][14][54], by making improvements on the network topology[34][2], or by improving communication paths[53][29].

Although a general energy consumption model was not introduced by the aforementioned approaches, we observe from them that there is a direct relation between the power consumption of a network card and the transmission rate it operates at.

In the same way, the power consumption produced by the hard disk is also related to its transfer rate. In fact, many hard disk brands specify the consumption of their products when they are in the active state (read/write) and in the idle state[45][44].

For that reason, we define the power consumption for any of these two components in the same way and they will be further referenced as ND in this section. We relate their power consumption at a given time T with their transfer rate, which is the average of its send/receive and idle states in T. Thus, for a certain device D, we established a W_u and W_i values which represent the power consumption in watts when its ND is in an active or idle state respectively. For a current transfer rate L ,W_u will multiply L relative to its maximum transfer capacity L_{MAX}. Then, the result will be added to the idle state consumption. For this, W_i will multiply the complement of ND's load (i.e. $L_{MAX} - L$) relative to L_{MAX}. Finally, in order to obtain energy consumption in joules (watts/s), we take into account T in seconds, as indicated in equation 7.

$$E_{ND} = (W_u \frac{L}{L_{MAX}} + W_i \frac{L_{MAX} - L}{L_{MAX}}) T_D \quad (7)$$

4.3.1 Microservice's energy consumption in terms of the NIC or Disk. In order to model the energy consumption of a microservice M in terms of its load generated in a device's (D) NIC or disk (ND), we will use the last equation; but only taking into account the load generated by M: L_M; and considering the idle state that M generates proportionally in the same way that L_M is for the current ND's load L, as it is possible to see in equation 8.

$$E_{ND_M} = (W_u \frac{L_M}{L_{MAX}} + Wi \frac{L_{MAX} - L}{L_{MAX}} * \frac{L_M}{L}) T_D \quad (8)$$

4.3.2 Microservice's energy consumption in terms of an external NIC or Disk. To model the power consumption that a microservice M would produce in terms of its load generated in another device we use, once more, the previous logic; but considering $D1's\ ND$: (1) Its maximum transfer capacity and (2) Its current load, as seen in the equation 9:

$$E_{M_{D1}} = (W_{u_{D1}} \frac{L_{M_D}}{L_{MAX_{D1}}} + Wi_{D1} \frac{L_{MAX_{D1}} - L_{D1}}{L_{MAX_{D1}}} * \frac{L_{M_D}}{L_{D1}}) T_D \quad (9)$$

So far, we have described the structure of our multidimensional space and the value-energy consumption relation that each of its axes has. In the following paragraphs, we will introduce how we deploy this space in a P2P overlay which allows us to understand a decentralized network as a multidimensional data structure. In it, a device is able to find other peers according to their available resources to migrate or duplicate microservices with the aim of saving energy.

5 MULTIDIMENSIONAL P2P APPROACHES

As previously mentioned, this work seeks to take advantage of some features offered by decentralized network environments in order to manage microservices based deployments. Thereby, for a device to find other peers, it is necessary that all the devices of the network are organized in an intelligent way. For this reason, there are several network overlay approaches that allow devices searching for information or peers in a logarithmic number of hops by abstracting the network as a data structure deployment such as: (1) an ordered circular list iterated with hash techniques, whose nodes and information are indexed through the same universe of IDs[49][27], being Chord one of the best known techniques; (2) a binary tree that can be explored through bit space and logical operations like XOR[36]; or (3) a linked list of nodes that can search for elements by making smart hops in the structure[35].

However, there are circumstances in which a network needs to be queried by more than one criterion (dimension). In our case, for example, we query a p2p device network to obtain the best candidate in terms of CPU, RAM, network and disk availability to process a microservice. Some pre-existing approaches deploy multidimensional data structures such as Rtree[13], or KD Tree[17] in a distributed network, keeping the parent-child relation of the tree nodes in the network devices. These last approaches allow range multidimensional querying in an efficient way; but the balancing or restructuring operations when inserting or deleting nodes could involve the participation of many devices in the network to perform expensive and complex data movements.

On the other hand, other approaches seek to partition multidimensional spaces into more independent structures. For example MAAN[4] generalizes the Chord approach to deploy a multidimensional space in as many circular device lists as the space has dimensions. In the following paragraphs, we will explain in detail how this technique works since, despite the positive characteristics that other approaches may have, we have arbitrarily (since it is not the objective of this work to demonstrate the advantages and disadvantages of P2P overlays) chosen MAAN to deploy our multidimensional space U described in section 3.

Before explaining the structure of MAAN, it is necessary to understand the Chord structure. Chord[50] deploys an ordered circular list of devices where they are organized in a clockwise fashion according to their ID. Each one of them, as well as the elements to look for (i.e. data, processes, etc.), are identified with the same universe of identifier numbers.

In addition, each device (node) keeps a table of addresses called finger table. This table has as many elements as the length of the bit string that represents the highest ID of all elements. Then, each index in the table will be defined by the id of each node added to 2^i, where i goes from 0 to the length of the aforementioned bit string. Every time an element E is searched through a node N, N looks for the $E's$ index successor in its finger table, that is, the highest possible index that does not exceed $E's$ id. Thus, the data structure used to search for elements is a consistent hashing table.

On the other hand, because Chord manages an ever changing network of devices, nodes can join and leave at any time. Due to this, it is important to understand the way each join/leave operation achieve said dynamism:

The join operation: There are several operations a new node nN and other nodes in the network must perform in order for nN to join the Chord network. First, the entering node nN gets assigned a unique ID by using a SHA-1 function to hash it's IP address. Then, an external mechanism guarantees a connection between nN and a Chord network pre-existent node which is found by executing a query inside of the network to find a node representative of an immediate successor of nN: $nN1$.

The knowledge of $nN1$, consisting of a immediate circle predecessor node reference and a finger table, is then used by nN to assign its own circle predecessor reference to the one of $nN1$ and fill its own finger table by copying the one in $nN1$. Finally, the predecessor reference of $nN1$ is re-set to nN at this moment.

Once nN has been initialized, an update must be issued walking counterclockwise along the circle to update previous nodes' finger tables, acknowledging $nN's$ existence and preserving finger tables' consistency. This is done by assigning $nN's$ ID to the appropriate finger table references inside of the circle predecessor node P finger table if both of the following conditions are met: (1) P precedes nN by at least 2^{i-1}, where i is delimited from 1 to m where m is the length of finger tables and (2) the last finger entry of node P succeeds nN. This update operation is sustained until a previous node P whose i^{th} finger precedes nN is reached. Finally, the responsibility of corresponding keys gets delegated from $nN1$ to nN.

The leave operation: To perform the leave operation, first a leaving node LN informs its predecessor node pLN that $LN's$ immediate successor node in the circle, $LN1$, is now its immediate successor in the circle. Then, the circle predecessor reference of $LN1$ is set to pLN. An update operation similar to the one described for the join operation is performed with the ID of the successor of LN. The goal of this update operation is to keep the finger tables of nodes located counterclock along the circle updated. Finally, the responsibility for $LN's$ keys is delegated to $LN1$.

MAAN:. As mentioned above, MAAN[4] generalizes the Chord approach to support n-dimensional queries as well as range queries. To do this, It uses a SHA-1 hashing function in order to assign a m-length bits identifier for each node and a locality preserving hashing function (a function designed to output consecutive values from related inputs) that, in contrast with the original Chord approach, is used to assign each attribute an identifier in the m-bit space according to its value (the same space shared by the m-bits node identifiers specified before). The latter is specifically important because the SHA-1 function would destroy the locality of keys since MAAN seeks to pair numerical attribute values to them instead of objects' names. Thereby, this allows the execution of 2 different multi-attribute query resolution approaches that would not be possible within Chord's original system. These two approaches are:

- The iterative approach, which consists of a resource search originated by a node N and composed by a M number of following sub-queries according to the n number of attribute dimensions available. This approach returns a series of candidate lists that are later intersected in origin node N with the purpose of finding the fittest candidate.
- The single attribute dominated query resolution aims to find appropriate candidates by performing a search fulfilling the needs of a single dominating attribute, returning a candidate set X. Then, sub-queries for other attributes are applied on X to obtain a single set of candidates whose members meet the needs of other resources. Finally, origin node N proceeds with the selection of the fittest candidate by correlation. This second approach may result inconvenient due to the amount of candidates that can exist in set X and do not fulfill the needs of any other sub-queries.

In MAAN, join and leave operations are kept the same as the original Chord system as it remains the core of MAAN.

6 IMPLEMENTING THE SPACE OF RESOURCES U IN MAAN

As defined previously in section 3, device resources exist inside of space U based on the availability of their processing capabilities. With these, we aim to efficiently find/index resources to perform migration and duplication operations on microservices to achieve energy savings, therefore, an appropriate system to efficiently store and retrieve candidate host devices in U is needed. Furthermore, due to space U consisting of 4 dimensions/axis of resources, this system must support multidimensional indexing.

6.1 Resources to index

According to our previous definition of space U, 4 different dimensions/axes are related to 4 different hardware resources: (1)CPU frequency in GHz (2)RAM capacity in MB (3)network transfer rate in Mbps and (4)Hard disk speed in MBps. Thereby, MAAN will be used to index each device resource availability value on the corresponding U axis.

6.2 Node join operation

When a device intends to join the MAAN network, 4 different logical nodes are created in 4 different dimensions in accordance with the axis of space U. These nodes follow exactly the same operations as the ones described above in Chord's node join operations with some technical differences. In our approach, we lack an IP address identifier for each device. Therefore we created a locality preserving formula that, for any dimension, transforms a numerical value representing the availability of a resource into a corresponding ID coherent to the ID space available. Formula 10 states a superior limit for numerical entry values and a maximum number of nodes supported. Thus, a numerical input value V is multiplied by the maximum number of supported (at any time) nodes N and the product divided by the maximum device's resource limit value SL, where SL and N values are relative to each dimension constraints.

$$ID = \left\lceil \frac{V * N}{SL} \right\rceil \tag{10}$$

Thereby, in our approach, finger tables contain records in pairs of m-bit space identifiers and node references. These elements grant us the execution of n-dimensional queries by applying an iterative approach like the one used in MAAN's original proposal.

On the other hand, it's important to say that nodes with a repetitive value for any attribute can join the network resulting in an identical ID assignment as a pre-existing node in the network. This would result in a collision causing any entering node to place itself between a successor circle node and (if existent) a node identical to itself, further referenced as $twinnode$. Due to this, we decided to implement a technique inside our approach that manages nodes involved in this phenomenon. When a twin node T tries to join the network and finds a predecessor node rN (root node which has assigned the same ID as T) counter clock along the circle, T enters a twin list of rN where each member has a twin list predecessor node and twin list successor node references. This list keeps the twin nodes of rN stored to execute node leave related operations further explained below in section 6.4.

6.3 Querying for resources

In order to perform queries, we follow MAAN's iterative approach with a couple of new techniques applied on any candidate list X. First, we introduce a "candidate lock" meant to prevent resource selection collisions. Resource selection collisions occur when the same host candidate is selected as the fittest candidate by several different query executions and simultaneous microservices migration/duplication operations occur towards this node, causing saturation of the new host candidate due to newly received microservices competing for resources and consequently inducing a higher power consumption in the new host device. The candidate lock is applied

on each member $x \in X$ which means that devices can be candidates to exclusively one query at a time.

Even though this candidate lock strategy is enough to prevent selection collisions, another selection strategy had to be applied inside of our querying approach to prevent a "circular saturation" phenomenon. A circular saturation phenomenon occurs when a saturated device finds a fit host device and migrates its microservices but causes a final higher energy consumption by either: (1) causing a high enough demand that triggers external device mechanisms to mitigate thermal energy elevation such as a fan or (2) causing the new host to initiate a migration strategy resulting in further network strain. We prevent circular saturation by performing a calculation inside the querying node that, for every $x \in X$, predicts the future resources consumption. This is done by projecting the future percentage of each resource consumption should the candidate microservice be hosted there, discarding hosts whose projection would result equal or greater to a specific desired percentage threshold.

6.4 Node leave operation

The node leave operation is kept mostly the same as the one in the original Chord's proposal having the handling of twin nodes as an added feature. For any twin list inside of any pre-existent parent root node rN, if rN leaves the network, the first twin node TN of the twin list inside rN gets assigned as the new rN by copying $rN's$ elements, such as the counter clock circle predecessor reference and its finger table. In the event that a twin node inside of $rN's$ twin list leaves, all left to do is remove it from $rN's$ twin list and reset the references (previously mentioned above) of its twin list predecessor and twin list successor to fill the gap between them.

6.5 Data frames

We decided to implement the concept of "data frames" inside of our approach to recollect the change per resource availability in each device allowing us to control the frequency with which nodes get dynamically reindexed into MAAN. Data frames are, in essence, a limited (to a deliberate amount) collection of resource availability measurement values related to a single resource. This allows us to perform the re-indexing process through a supervisory service that makes the pertinent node leave and then join again if there is a substantial change in its ID. The calculation deciding whether a node must be re-indexed or not works the following way: (1) find the average value in the data frame to (2) get an ID related to it and, (3) if the ID is different to the currently indexed one, perform a re-index operation on that node.

7 THE SCHEDULING ALGORITHM

Once a device is inserted into our MAAN's network configuration containing our space U deployed, it starts the continuous execution of a supervisor service[32]. The objectives of this service are (1) to evaluate every certain time elapsed the load sustained by each of the device's hardware components, (2) to manage the set of microservices that the device runs (i.e. start, stop, migrate or duplicate); and (3) provide information about its device such as hardware components load, hardware components capabilities and working time; and information about the microservices it runs like processing requirements, execution time and generated load. Furthermore, it analyzes its device data frames to keep the resources nodes updated. We use this service in such a way that, through collaborative negotiations, the devices can reach an efficient load-energy state. In order to achieve this, we propose an algorithm based on the following premises:

- A device can find another peer to process a microservice fulfilling its processing requirements with a lower energy consumption.
- A device sustaining overload conditions in its CPU and/or disk components can activate heat dissipating mechanisms (such as a fan), which result in a higher energy consumption.
- There are situations in which it is possible to migrate microservices to turn off a device, saving energy.
- A device may migrate high/low consumption microservices.

These premises allow us to reach an initial pivot point in the algorithm which involves two main scenarios:

(1) The device is saturated and migration operations need to be performed.
(2) The device is not saturated but energy consumption can be mitigated/reduced by reducing hardware load with the migration of microservices.

For the first scenario, we decided that a device is saturated based on previous behavioural observations of the algorithm when one of the hardware resources available is being consumed at 85%. In addition, a deliberate tax of an additional 20% power consumption is applied on a saturated device in an effort to emphasize such a situation. Once a saturation condition is met, the algorithm proceeds by selecting the heaviest microservice and then executes a search for candidates that meet the requirements to host it. If there are no available candidates, it proceeds to search again with the requirements of the smallest microservice to finally migrate the microservice in either of cases to a random fit candidate. For the second scenario, a fan off and sleep strategies were included. The fan off strategy consists of reducing the load of a device to a deliberate 80% or less, mitigating the deliberate extra 5% energy consumption induced by the fan. The sleep strategy sends the device to sleep reducing the energy cost of the device to a deliberate 5%.

8 EXPERIMENTS AND RESULTS

In order to test scenarios and perform our experiments, we used the DRACeo[31] simulator to deploy devices specifying their resources capabilities, microservices with their resources demands, and network overlays through an API. In addition, DRACeo allows the definition of the mathematical model that explains the hardware component load-power consumption relation also through an API.

8.1 Methodology of testing

To perform our experiments, we considered a collection of metrics to define devices' capabilities and hardware consumption variables gathered from several sources like data sheets and public hardware benchmarks. The objective of this is to approximate results to real scenarios, facilitating their interpretation.

8.1.1 Defining devices capabilities. As previously mentioned, we consider 4 different device's hardware capabilities to study the relation between load, QoS and energy consumption: CPU frequency (Ghz), RAM capacity (MB), disk data transfer rate (MB/s) and finally NIC data transfer rate (Mb/s). For this work and further explained test scenarios, a random delimited value for each resource shown in table 1, establishing heterogeneity among devices.

These values are mostly considered by averaging data from publicly available sources. For instance, the CPU and RAM possible values were considered by analyzing publicly available hardware benchmarks performed independently by millions of users[52][41], providing an insight into hardware resources available in the market, while NIC and disk values were based on hardware and software specialized websites that perform independent test[51][23].

Table 1: Devices capabilities

CPU (Ghz)	RAM (MB)	NIC (Mb/s)	Disk (MB/s)
1.2-4.8	2000-32000	101.0-1000	SSD: 101.0-800 HDD: 80-160

8.1.2 Defining power consumption variables. Having defined above the capacity for each resource, we defined variables to model power consumption by interpreting information gathered into possible magnitudes of values shown in table 2. For instance, we selected CPU consumption values following CPU data sheets[21] and CPU overclocking techniques[20] involving voltage among others[10]. RAM and disk values were simple to define as hardware manufacturers have previously justified a delimited estimated consumption[45][44]. On the other hand, we deliberated NIC consumption variables by averaging a recollection of several NIC hardware specifications[7].

Table 2: Components power variables

CPU	RAM	NIC	Disk (SSD/HDD)
Capacitance:10pF Voltage:1.2v	3-5W	Idle:0.4944W Active: 1.1349W	Idle:0.05/5.4W Active:2.2/8.0W

8.2 Tests definition

8.2.1 Scalability test. The objective of this test was to challenge the scheduling algorithm and energy saving strategies with increasing numbers of microservices and devices through time. The test begins with 20 heterogeneous devices and 40 heterogeneous microservices (MS.) varying among 3 different types with diverse resource demands shown in table 3, where the microservice size is the data size to be duplicated/ migrated during scheduling.

The duration of the experiment consists of 100 seconds and a deployment of 10 random microservices and 10 random devices each 10 seconds for the first 80 seconds, leaving 20 final seconds to the scheduling algorithm for stabilizing. This experiment was performed 100 times in order to ensure repeatability of testing and consistency of the results.

Table 3: Microservices Requirements

Resources	GUI MS.	Control MS.	DB. MS.
CPU (Ghz)	1.2	1.8	0.8
RAM (MB)	200	100	150
Network (MB)	0	0	0
Disk (MB)	0.1	50	50
Microservice size (MB)	50	50	50

8.2.2 Stress test. The objective of this experiment was to prove resilience of the scheduling algorithm when faced against a constant change in microservices execution. This test consisted of 250 random microservices of the same types as the ones in the previous experiment randomly deployed among 50 heterogeneous devices. Furthermore, we introduced a time constraint feature inside of microservices which gave them a specific "time to finish" ranging from 5 to 30 seconds for a whole text duration of 100 seconds. This meant that each microservice had a temporal operational impact before being removed from the network and redeploying another random one. In this experiment, the sheer amount of microservices is guaranteed to stress the whole network inducing constant scheduling and, while introducing a "time to finish" would relief it, we decided to keep a varying (due to the randomness of the deployment and type of microservice) constant stress by deploying a microservice each time one microservice finished. This experiment was also performed 100 times in order to ensure repeatability of testing and consistency of the results.

8.3 Tests metrics definition

In order to evaluate the performance of our approach, we considered to analyze 5 variables during the 2 experiments performed:

- The energy consumed by the entire network (Joules).
- The QoS achieved on average by all microservices that are executed in the network.
- The average number of migrations performed.
- The data transferred by microservices movements (MB).
- Energy cost of microservices movements (Joules).

8.4 Definition of success

In order to properly interpret the data collected as a success, we compared the performance results of our algorithm in the tests previously mentioned against the performance results of Kaligreen algorithm and no algorithm in the same tests under the same conditions and number of experiment repetitions. Kaligreen algorithm gets executed when a supervisor microservice in a device detects a resource consumption level surpassing a deliberate threshold. Once this saturation is detected, the supervisor microservice selects the heaviest microservice currently hosted in the device and migrates it to the freest available neighbor peer. Finally, the performance of our algorithm should be catalogued as a success if, after analyzing the performance results, energy consumption is lowered or kept the same and there is an increase on QoS, or, QoS is bettered at the expense of slightly more energy consumption.

8.5 Scalability test results

As shown in table 4, our scheduling algorithm proved to consume a total average of 27% less energy (692020.73 joules) against the experiments with no algorithm (95452.72 joules) , and 11% less energy against the naive algorithm (78570.94 joules) experiments. Furthermore, the average QoS sustained by our algorithm was greater than the one kept by both of the alternative experiments, where our algorithm scored an average of 99.24% of QoS against 94.04% of the no-algorithm experiments and 99.21% of the naive algorithm. Even though an observable difference in number of movements performed by our algorithm against the naive algorithm, having 55.70 and 50.35 respectively, and an observable higher average data transfer is present, having 850.29 MB against 775.83 MB respectively, the amount of total energy saved is well worth it even after considering the energetic cost of movements done, having 0.002 joules against 0.001 joules consumed by the naive algorithm. After the evaluation of these results, we can consider our experiment a success, stating that our algorithm is scalable and stable.

Table 4: Scalability test results

Average of:	Algorithm	No algorithm	Dummy
Total energy	69202.7337	95452.72	78570.94
QoS	99.2414925	94.04	99.21
Movements	55.7014925	0	50.35
Data transfer	850.298507	0	775.83
Movements' Energy consumption	0.00210982	0	0.00176535

8.6 Stress test results

This experiment revealed an interesting behaviour against the other alternatives. As it is possible to see in table 5, our algorithm consumed a total average of 69% more energy (578315.4 joules) against the no-algorithm experiments (340372.9 joules) and 4% less energy than the naive algorithm (598737.0 joules). In addition, the most revealing information shown by this test was the average QoS sustained by each scheduling algorithm or lack of. Our scheduling algorithm scored an average of 92.71% of QoS against the average 87.61% sustained by the naive algorithm and 87.05% sustained when no algorithm was applied. This demonstrates that the movements performed by our algorithm did not only save energy compared against the naive algorithm, they were also wiser, keeping the QoS level as high as possible under heavy load. On the other hand, the amount of these movements is higher when compared with the naive algorithm, having an average of 912.6 movements against 798.5 respectively. Even though this results in a higher average data transfered of 12610.29 MB and 13243.3 MB respectively the difference is intranscendental considering they consumed 0.030 joules against 0.031 joules consumed by the naive algorithm. This is easily explained by taking into consideration that, under heavy network resources load, mostly small microservices are moved. Overall, our algorithm proved to be resilient under heavy loads, keeping energy as low as possible and, in an inverse way, QoS as high as possible.

Table 5: Stress test results

Average of:	Algorithm	No algorithm	Dummy
Total energy	578315.444	340372.995	598737.071
QoS	92.7182353	87.05	87.6183333
Movements	912.647059	0	798.583333
Data transfer	12610.2941	0	13243.3333
Movements' Energy consumption	0.03020178	0	0.03184407

9 CONCLUSIONS AND FUTURE WORKS

In this article, we have shown that it is possible to save energy by running microservices on "ideal" devices. To do this, we have evaluated the device load and microservices execution requirements using a P2P overlay called MAAN which is based on a multidimensional data approach and offers stability, efficiency and independence of operations. This structure allows a device to find in a logarithmic number of hops another peer with sufficient and optimal capacities in order to negotiate and then execute the migration or duplication of a microservice. The negotiation criterion handled by each device is managed by its scheduling algorithm.

To test our approach, we used DRACeo as a tool for simulating our scheduling algorithm in MAAN-based P2P environments. Our two experiments were aimed at evaluating energy consumption in situations of scalability and stress, obtaining good results in both cases: saving up to 27% of energy in the first experiment and ensuring an optimal level of 93% of QoS in the second.

On the other hand, we are aware that it is necessary to evaluate this technique in more realistic architectures where important criteria typical of the software production stage are taken into account, such as microservices discovery, replication restrictions, load balancing, etc. We will develop these analysis in future work.

REFERENCES

[1] Nour M. Azmy, Islam A.M. El-Maddah, and Hoda K. Mohamed. 2016. Adaptive Power Panel of Cloud Computing Controlling Cloud Power Consumption. In *Proceedings of the 2Nd Africa and Middle East Conference on Software Engineering (AMECSE '16)*. ACM, New York, NY, USA, 9–14. https://doi.org/10.1145/2944165.2944167

[2] Kaibin Bao, Ingo Mauser, Sebastian Kochanneck, Huiwen Xu, and Hartmut Schmeck. 2016. A Microservice Architecture for the Intranet of Things and Energy in Smart Buildings: Research Paper. In *Proceedings of the 1st International Workshop on Mashups of Things and APIs (MOTA '16)*. ACM, New York, NY, USA, Article 3, 6 pages. https://doi.org/10.1145/3007203.3007215

[3] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. 2018. SPEC CPU2017: Next-Generation Compute Benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18)*. ACM, New York, NY, USA, 41–42. https://doi.org/10.1145/3185768.3185771

[4] M. Cai, M. Frank, J. Chen, and P. Szekely. 2003. MAAN: a multi-attribute addressable network for grid information services. In *Proceedings. First Latin American Web Congress.* 184–191.

[5] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. 2001. Searching in Metric Spaces. *ACM Comput. Surv.* 33, 3 (Sept. 2001), 273–321. https://doi.org/10.1145/502807.502808

[6] Qiao Chen and Jian Li. 2013. The Balance Mechanism of Power and Performance in the Virtualization. In *Proceedings of the Second International Conference on Innovative Computing and Cloud Computing (ICCC '13)*. ACM, New York, NY, USA, Article 189, 4 pages. https://doi.org/10.1145/2556871.2556912

[7] Salvatore Chiaravalloti, Filip Idzikowski, and Łukasz Budzisz. 2011. Power consumption of WLAN network elements. https://doi.org/10.13140/2.1.4424.8005

[8] IBM Cloud. 2019. Orquestacion de contenedores explicada. https://www.youtube.com/watch?v=kBF6Bvth0zw&t=425s.

[9] B. F. Cornea, A. Orgerie, and L. Lefèvre. 2014. Studying the energy consumption of data transfers in Clouds: the Ecofen approach. In *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*. 143–148.

[10] Cypress. 2017. Understanding Load Capacitance and Access Time. https://www.cypress.com/file/202411/download.

[11] Keling Da, Marc Dalmau, and Philippe Roose. 2014. Kalimucho: Middleware for Mobile Applications. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC '14)*.

[12] Peter de Lange, Bernhard Göschlberger, Tracie Farrell, and Ralf Klamma. 2018. *A Microservice Infrastructure for Distributed Communities of Practice: 13th European Conference on Technology Enhanced Learning, EC-TEL 2018, Leeds, UK, September 3-5, 2018, Proceedings*. 172–186. https://doi.org/10.1007/978-3-319-98572-5_14

[13] C. du Mouza, W. Litwin, and P. Rigaux. 2007. SD-Rtree: A Scalable Distributed Rtree. In *2007 IEEE 23rd International Conference on Data Engineering*. 296–305.

[14] Niloufar Piroozi Esfahani and Alberto E. Cerpa. 2015. Poster: Energy Optimization Framework in Wireless Sensor Network. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems (SenSys '15)*. ACM, New York, NY, USA, 441–442. https://doi.org/10.1145/2809695.2817904

[15] Martin Fowler. 2014. Microservices - a definition of this new architectural term. https://martinfowler.com/articles/microservices.html.

[16] Volker Gaede and Oliver Günther. 1998. Multidimensional Access Methods. *ACM Comput. Surv.* 30, 2 (June 1998), 170–231. https://doi.org/10.1145/280277.280279

[17] Prasanna Ganesan, Beverly Yang, and Hector Garcia-Molina. 2004. One Torus to Rule Them All: Multi-Dimensional Queries in P2P Systems. In *Proceedings of the 7th International Workshop on the Web and Databases: Colocated with ACM SIGMOD/PODS 2004 (WebDB '04)*. Association for Computing Machinery, New York, NY, USA, 19–24. https://doi.org/10.1145/1017074.1017081

[18] Shuai Hao, Ding Li, William G. J. Halfond, and Ramesh Govindan. 2012. Estimating Android Applications' CPU Energy Usage via Bytecode Profiling. In *Proceedings of the First International Workshop on Green and Sustainable Software (GREENS '12)*. IEEE Press, Piscataway, NJ, USA, 1–7. http://dl.acm.org/citation.cfm?id=2663779.2663780

[19] Intel. 2004. Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor. (2004).

[20] Intel. 2020. How to Overclock Your Unlocked Intel Core™ Processor. https://www.intel.com/content/www/us/en/gaming/resources/how-to-overclock.html.

[21] Intel. 2020. Intel® Core i5-8400 Processor. https://ark.intel.com/content/www/us/en/ark/products/126687/intel-core-i5-8400-processor-9m-cache-up-to-4-00-ghz.html.

[22] Abhishek Jaiantilal, Yifei Jiang, and Shivakant Mishra. 2010. Modeling CPU Energy Consumption for Energy Efficient Scheduling. In *Proceedings of the 1st Workshop on Green Computing (GCM '10)*. ACM, New York, NY, USA, 10–15. https://doi.org/10.1145/1925013.1925015

[23] jpy software. 2020. Calculating network data transfer speeds. https://news.jpy.com/kbase/support-articles/2015/9/7/calculating-network-data-transfer-speeds.

[24] SeungGu Kang, Hong Jun Choi, Cheol Hong Kim, Sung Woo Chung, DongSeop Kwon, and Joong Chae Na. 2011. Exploration of CPU/GPU Co-execution: From the Perspective of Performance, Energy, and Temperature. In *Proceedings of the 2011 ACM Symposium on Research in Applied Computation (RACS '11)*. ACM, New York, NY, USA, 38–43. https://doi.org/10.1145/2103380.2103388

[25] Alexey Karyakin and Kenneth Salem. 2017. An Analysis of Memory Power Consumption in Database Systems. In *Proceedings of the 13th International Workshop on Data Management on New Hardware (DAMON '17)*. Association for Computing Machinery, New York, NY, USA, Article 2, 9 pages. https://doi.org/10.1145/3076113.3076117

[26] Simon Kiertscher and Bettina Schnor. 2015. Scalability Evaluation of an Energy-aware Resource Management System for Clusters of Web Servers. In *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (Spects '15)*. Society for Computer Simulation International, San Diego, CA, USA, 1–8. http://dl.acm.org/citation.cfm?id=2874988.2875004

[27] Jan Krajewski, José Lozano, Julian Driver, Emmanuel Escandon, Sumith Kumar, Silvia Malatini, and Jose Lozano Hinojosa. 2006. *PASTRY: the third generation of peer -to -peer networks*. Ph.D. Dissertation.

[28] Jacob Leverich, Matteo Monchiero, Vanish Talwar, Parthasarathy Ranganathan, and Christos Kozyrakis. 2009. Power Management of Datacenter Workloads Using Per-Core Power Gating. *Computer Architecture Letters* 8 (02 2009), 48–51. https://doi.org/10.1109/L-CA.2009.46

[29] Yung-Feng Lu, Jun Wu, and Chin-Fu Kuo. 2014. A Path Generation Scheme for Real-time Green Internet of Things. *SIGAPP Appl. Comput. Rev.* 14, 2 (June 2014), 45–58. https://doi.org/10.1145/2656864.2656868

[30] H. H. lvarez Valera, M. Dalmau, P. Roose, and C. Herzog. 2019. The Architecture of Kaligreen V2: A Middleware Aware of Hardware Opportunities to Save Energy. In *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*. https://doi.org/10.1109/IOTSMS48152.2019.8939237

[31] H. H. lvarez Valera, M. Dalmau, P. Roose, J. Larracoechea, and C. Herzog. 2020. DRACeo: A smart simulator to deploy energy saving methods in microservices based networks. *29th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises – WETICE 2020* (2020).

[32] Hernn H. lvarez Valera, Philippe Roose, Marc Dalmau, Christina Herzog, and Kyle Respicio. 2018. KaliGreen: A distributed Scheduler for Energy Saving. *Procedia Computer Science* (2018). https://doi.org/10.1016/j.procs.2018.10.172

[33] Dmitry Maevsky, E. Maevskaya, and E. Stetsuyk. 2017. *Evaluating the RAM Energy Consumption at the Stage of Software Development*. Vol. 74. 101–121. https://doi.org/10.1007/978-3-319-44162-7_6

[34] Tanu Malik, Ligia Nistor, and Ashish Gehani. 2010. Middleware for Managing Provenance Metadata. In *Middleware '10 Posters and Demos Track (Middleware Posters '10)*. ACM, New York, NY, USA, Article 5, 2 pages. https://doi.org/10.1145/1930028.1930033

[35] S. Mandal, S. Chakraborty, and S. Karmakar. 2012. Deterministic 1–2 skip list in distributed system. In *2012 2nd IEEE International Conference on Parallel, Distributed and Grid Computing*. 296–301.

[36] Petar Maymounkov and David Eres. 2002. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. *Kademlia: A Peer-to-peer Information System Based on the XOR Metric* 2429. https://doi.org/10.1007/3-540-45748-8_5

[37] Microsoft. 2018. Creating composite UI based on microservices. https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/microservice-based-composite-ui-shape-layout.

[38] Netflix. 2016. Netflix Conductor: A microservices orchestrator. https://netflixtechblog.com/netflix-conductor-a-microservices-orchestrator-2e8d4771bf40.

[39] Dirk Neumann, Christian Bodenstein, Omer F. Rana, and Ruby Krishnaswamy. 2011. STACEE: Enhancing Storage Clouds Using Edge Devices. In *Proceedings of the 1st ACM/IEEE Workshop on Autonomic Computing in Economics (ACE '11)*. ACM, New York, NY, USA, 19–26. https://doi.org/10.1145/1998561.1998567

[40] A. Orgerie, L. Lefèvre, I. Guérin-Lassous, and D. M. Lopez Pacheco. 2011. ECOFEN: An End-to-end energy Cost mOdel and simulator For Evaluating power consumption in large-scale Networks. In *2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*. 1–6.

[41] PassMark. 2020. Memory Benchmarks. https://www.memorybenchmark.net/amount-of-ram-installed.html.

[42] Michael Pawlish, Aparna S. Varde, Stefan A. Robila, and Anand Ranganathan. 2014. A Call for Energy Efficiency in Data Centers. *SIGMOD Rec.* 43, 1 (May 2014), 45–51. https://doi.org/10.1145/2627692.2627703

[43] Giuseppe Procaccianti, Patricia Lago, and Grace A. Lewis. 2014. Green Architectural Tactics for the Cloud. In *Proceedings of the 2014 IEEE/IFIP Conference on Software Architecture (WICSA '14)*. IEEE Computer Society, Washington, DC, USA, 41–44. https://doi.org/10.1109/WICSA.2014.30

[44] Samsung. 2017. Samsung V-NAND SSD 860 EVO. https://www.samsung.com/semiconductor/global.semi.static/ Samsung_SSD_860_EVO_Data_Sheet_Rev1.pdf.

[45] Segate. 2016. Desktop HDD Product Manual. https://www.seagate.com/www-content/product-content/barracuda-fam/desktop-hdd/barracuda-7200-14/en-us/docs/100686584v.pdf.

[46] Dongyou Seo. 2012. A Study of Workload Consolidation and Power Consumption on a Multi-core Processor. In *Proceedings of the 2012 ACM Research in Applied Computation Symposium (RACS '12)*. ACM, New York, NY, USA, 457–458. https://doi.org/10.1145/2401603.2401702

[47] Amazon Web Services. 2019. Implementing Microservices on AWS. 2019 (2019).

[48] I. Siddavatam, E. Johri, and D. Patole. 2011. Optimization of Load Balancing Algorithm for Green IT. In *Proceedings of the International Conference & Workshop on Emerging Trends in Technology (ICWET '11)*. ACM, New York, NY, USA, 1344–1346. https://doi.org/10.1145/1980022.1980321

[49] Ion Stoica, Robert Morris, David Karger, M. Kaashoek, and Hari Balakrishnan. 2001. Chord: A scalable peer-to-peer lookup service for internet applications, Vol. 149-160. 149–160. https://doi.org/10.1145/383059.383071

[50] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. 2003. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Trans. Netw.* 11, 1 (Feb. 2003), 17–32. https://doi.org/10.1109/TNET.2002.808407

[51] tomshardware. 2019. SSD vs HDD Tested: What's the Difference and Which Is Better? https://www.tomshardware.com/features/ssd-vs-hdd-hard-drive-difference/2.

[52] Tuxera. 2020. cpu-frequency. https://a1dev.com/sd-bench/stats/cpu-frequency/.

[53] Elias Yaacoub, Abdullah Kadri, and Adnan Abu-Dayya. 2012. Cooperative Wireless Sensor Networks for Green Internet of Things. In *Proceedings of the 8H ACM Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet '12)*. ACM, New York, NY, USA, 79–80. https://doi.org/10.1145/2387218.2387235

[54] Kelie Zhan, Chung-Horng Lung, and Pradeep Srivastava. 2014. A Green Analysis of Mobile Cloud Computing Applications. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC '14)*. ACM, New York, NY, USA, 357–362. https://doi.org/10.1145/2554850.2555069