Chapter

Intelligent Workload Scheduling in Distributed Computing Environment for Balance between Energy Efficiency and Performance

Larysa Globa, Oleksandr Stryzhak, Nataliia Gvozdetska and Volodymyr Prokopets

Abstract

Global digital transformation requires more productive large-scale distributed systems. Such systems should meet lots of requirements, such as high availability, low latency and reliability. However, new challenges become more and more important nowadays. One of them is energy efficiency of large-scale computing systems. Many service providers prefer to use cheap commodity servers in their distributed infrastructure, which makes the problem of energy efficiency even harder because of hardware inhomogeneity. In this chapter an approach to finding balance between performance and energy efficiency requirements within inhomogeneous distributed computing environment is proposed. The main idea of the proposed approach is to use each node's individual energy consumption models in order to generate distributed system scaling patterns based on the statistical daily workload and then adjust these patterns to match the current workload while using energy-aware Power Consumption and Performance Balance (PCPB) scheduling algorithm. An approach is tested using Matlab modeling. As a result of applying the proposed approach, large-scale distributed computing systems save energy while maintaining a fairly high level of performance and meeting the requirements of the service-level agreement (SLA).

Keywords: energy efficiency, performance, SLA, distributed computing system, scheduling, horizontal scaling

1. Introduction

Nowadays information technologies penetrate all spheres of human life. According to the Gartner Top 10 Strategic Technology Trends for 2019 [1], the new world-driving trends are going to include augmented analytics, immersive technologies, edge computing and blockchain in the nearest future. These technologies require extremely highly efficient distributed computing systems that could process large amounts of data, consuming as little resources as possible.

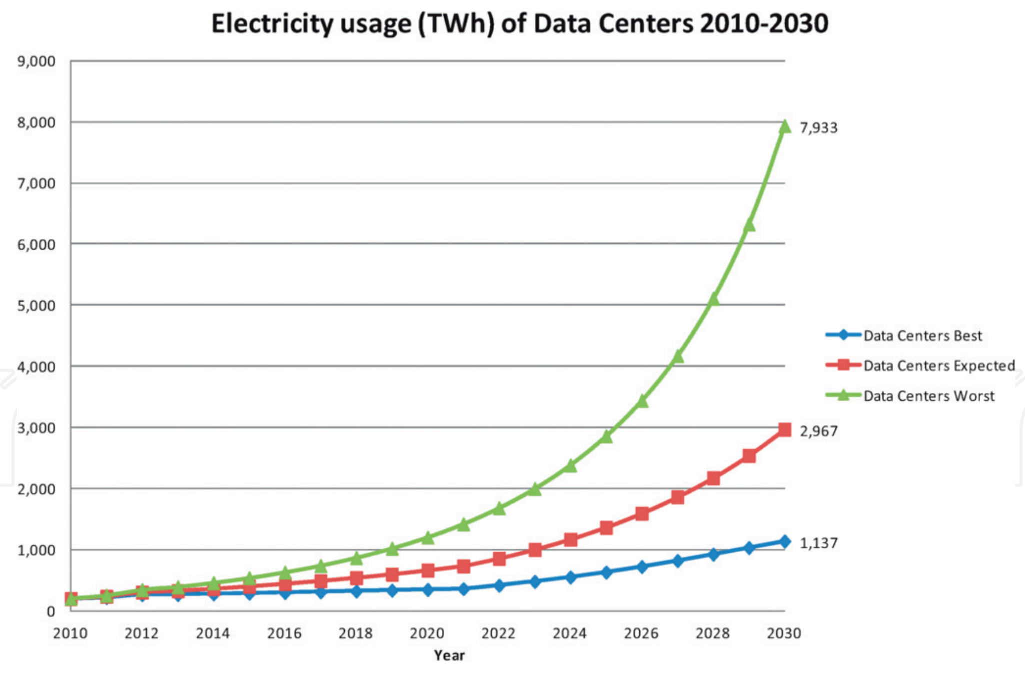


Figure 1.
Electricity usage of data centres 2010–2030 estimation [2].

Electrical power is one of the most demanded resources for the large-scale distributed computing systems. Both science and industry have made tremendous efforts to reduce the power consumption of large-scale computing systems over the past 10 years. According to the Huawei Technologies Sweden estimation, presented in [2], power consumption of data centres all over the world is still going to grow in the nearest future. However, due to the newly developed techniques, we can expect much lower growth up to 2967 TWh. On the contrary, in the worst case, data centres could consume up to 7933 TWh of energy without the use of any energy-saving approaches (**Figure 1**).

Although the fruitful cooperation between science and industry has already brought very good results in terms of reducing data centre power consumption, there is still a very high demand for approaches that would allow to cope with new challenges associated with the rapid development of distributed computing.

Within this study we propose an intelligent workload scheduling approach, which is aimed at improving energy efficiency of distributed computing systems through the application of energy-aware scheduling combined with scaling, taking into account data processing performance as well. An approach considers first of all inhomogeneous distributed systems designed to use cheap commodity hardware as much as possible.

The chapter is structured as follows: Section 2 contains state-of-the-art analysis of distributed computing system energy efficiency problem. Section 3 explains the problem to be solved by proposed approach. Section 4 introduces proposed intelligent workload scheduling approach efficiently combined with dynamic scaling approaches. Section 4.1 presents a model of proposed approach implementation, and Section 5 concludes the work with a summary and outlook on future work.

2. State of the art and background

Many approaches to increasing energy efficiency of the large-scale distributed computing systems (power management approaches) already exist. According to the paper [3], these approaches can be divided into static and dynamic in terms of decision-making process. Both static and dynamic approaches can be applied either at the hardware level or at the software level. At the same time, these approaches are also classified according to the scope (cloud environment, single server, etc.). The overall structure of approach classification proposed in [3] is depicted in **Figure 2**.

Since the load on computing systems usually changes dynamically (in particular, in the cloud environment), it is worth to pay attention to the dynamic approaches to power management, respectively. Thus, within this study we mainly focus on dynamic energy-efficient approaches.

Considering hardware-based approaches, one of the most common approaches is dynamic voltage and frequency scaling (DVFS). DVFS is a power management technique that is effective in reducing power dissipation by lowering the supply voltage [4]. DVFS is widely used to manage the energy and power consumption in modern processors; however, for DVFS to be effective, there is a need to accurately predict the performance impact of scaling a processor's voltage and frequency [5, 6]. Moreover, the implementation of hardware-based approaches may be challenging, especially when it comes to inhomogeneous systems consisting of commodity hardware. Such approaches usually require additional expenses in order to adapt the system to their use. However, hardware-based approaches are very efficient. And they might bring even more energy efficiency when used in conjunction with software-based techniques.

Among the software approaches, the most commonly used are scheduling and consolidation [7]. Scheduling approaches are aimed at distributing the workload among the servers in a way that no servers are underutilized, jobs' processing performance is high enough and, in the case of energy-aware scheduling, the power consumption of the whole computing system is minimized. Consolidation approaches concern virtualized environments and are designed to balance virtual machines (VMs) so that they can run on as few servers as possible. Idling servers are then shut down or switched to a standby mode. It means that the consolidation is tightly coupled with horizontal scaling approaches.

The authors of [8] proposed the performance and energy-based cost prediction framework that dynamically supports VMs auto-scaling decision and demonstrates the trade-off between cost, power consumption and performance. This framework allows to estimate auto-scaling total cost, which is essential when using consolidation and horizontal auto-scaling approaches. Dynamic auto-scaling can be a big gain in energy efficiency, but estimating the cost of automatic scaling should not lead to excessive overhead. Thus, in some cases, it may be worthwhile to avoid constant dynamic auto-scaling and basically rely on statistics instead, while adjusting the scale of the system when it is really needed.

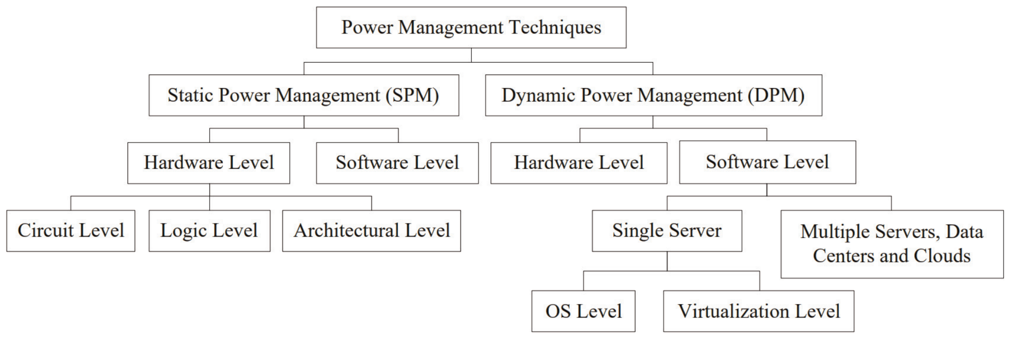


Figure 2.
Classification of power management approaches to increasing energy efficiency of computing systems [3].

In this chapter we use this idea, and we are mainly focused on the efficient combination of horizontal scaling and energy-aware scheduling approaches. In the field of energy-aware scheduling, there are many approaches developed so far. Some of them are also aimed to combine different fundamental ideas of energy-efficient computing.

The authors of the paper [9] proposed adaptive energy-efficient scheduling (AES) approach which combines scheduling with the dynamic voltage scaling (DVS) technique. This approach is proposed to be used in homogeneous computing systems. It consists of two phases—in the first phase, an adaptive threshold-based task duplication strategy is used, which can adjust the optimal threshold according to the specified schedule length, network power, processor power and application; in the second phase, DVS-enabled processors that can scale down their voltages are used for processing. A proposed approach gives a gain of 31.7% in terms of energy efficiency without performance loss that is quite a good result. Within this study we also use an idea of combining scheduling approaches with other methodologies; however, the authors of [9] use DVS and are limited by using DVS-enabled hardware. Their approach is designed for homogeneous computer environments, while we are focused on inhomogeneous environments, since it makes sense for service providers to use cheap commodity hardware with different physical parameters in order to reduce capital expenses.

Another example of energy-aware scheduling approach is Min_c [10]. This strategy takes into account the variety of tasks that comes to the computing system and the fact that the resources required by these tasks are different. The main drawback of this approach is that the model of energy consumption is the same for each node. It has nonlinear character that is close to reality, but the characteristics of different machines can differ. Therefore, we propose defining $P = f(CPU)$ dependencies for each machine individually.

Within our previous research [11–13], energy-aware scheduling approach called power consumption and performance balance (PCPB) was proposed and experimentally tested. In [13] two possible modifications to the PCPB were described—one of them uses tasks classification; another one applies scaling in addition to energy-aware scheduling in order to further increase energy efficiency. It was determined that developed energy-aware scheduling approach PCPB gives the best results in terms of energy efficiency (energy savings up to 33.59%) while being used in conjunction with horizontal scaling (scale-in and scale-out). Thus, in this chapter we elaborate on this basic idea and propose to enhance PCPB with scaling techniques and smartly power off and on idling servers while distributing the workload between them using PCPB.

3. Problem definition

Consider a distributed computing system consisting of N nodes. Each node N_j is described by the following:

- V_j—the amount of available RAM
- $flops_j$—the productivity of the node N_j, which has k_cores_j of the computing cores
- $P_j = f(CPU_j)$—the power consumption function of the node, which is experimentally determined for each N_j

Job is the unit of computing. The component of job is called a task. Physically, one job is represented as a single computing process of a computer. Subprocesses or child processes in turn appear to be tasks.

We will use the term "job" to determine the atomic computational problem that can be located and executed on one of the computing nodes of the system (i.e. "job" is the unit of load distribution). The income jobs form the queue.

Let us introduce the assumptions:

1. The queue consists of Q positions.

2. Jobs come to the queue at random moments of time τ. If the length of the queue is $0 < l < Q$, the job is placed on a free space in the queue. Otherwise, the job is rejected.

3. All jobs in the queue are independent to each other.

4. The job may have one of the five possible states: "preparation for execution", "readiness for execution", "in process of execution", "execution successfully completed", and "execution interrupted".

5. Each i^{th} job is characterized by the following parameters:

   - The volume of job—the number of floating-point operations that must be performed by the machine within this job (the number of floating-point operations is not natively used to measure the jobs' volume; however, we use this artificial unit to express the amount of work that is required by job to be performed):

$V_i = const\ [operations]$

   - The maximum execution time (or timeout):

$\Delta t_{max\,i} = const\ [\text{sec}]$

is the time period from the moment the job arrives at the processing, after which it must be completed. If the job has not been transferred to the state of "execution successfully completed" after the time $\Delta t_{max\,i}$, it is transferred to the state of "execution interrupted" and is output from the system.

   - Minimal amount of resources required to complete the job:

     - Minimal amount of RAM needed:

$RAM\_\text{min}\_i\ [Gb]$

     - Minimal required number of processing cores:

$k\_cores\_\text{min}\_i\ [cores]$

An optimal parallelization for the task is not considered within this study. Let us consider that the tasks are parallelized beforehand and just declared how many cores they require to be processed, as an input data for processing system.

     - Minimal required volume of persistent storage:

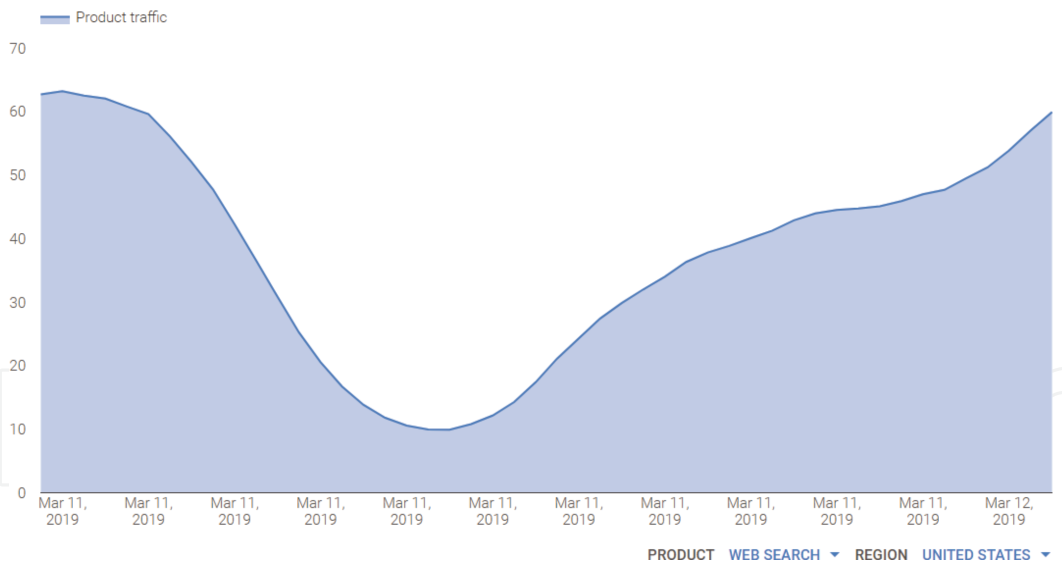


Figure 3.
Workload statistic for the Google web search service (11 March 2019) [14].

$storage\_\min\_i\ [Gb]$

- Job's priority is an optional numeric parameter that can be set for a job and

  determine the priority of its execution in comparison with other jobs. Let the priority $priority_i$ be determined by an integer from 1 to 10. In this case, priority processing requires tasks with priority 1. Let the default value of the priority of the job be 10 units.

Consider the daily workload curve to be a part of an input data for the problem. Every service provider is able to gather the daily statistics of the workload in his computing system. As a result, the pattern of the day workload can be generated. For example, Google provides such statistics of using its web search service in the Transparency Report [14] (**Figure 3**).

Under the certain circumstances (holidays, festivals, special events, etc.), this pattern can be changed. It means that it is possible to create certain system configuration patterns for the stable workload, respectively, but there is a need to adapt the system in the case of changes being foreseen.

Given an input data, the problem is formulated as follows: to reduce total power consumption of considered computing system while increasing the performance of data processing and meeting service-level agreement (SLA) requirements.

4. Proposed approach

Within the previous studies [11–13], it was determined that the use of energy-aware scheduling is most effective when used in conjunction with consolidation approaches and further scaling the whole system in and out. In [13] it was shown that powering off idle servers could save up to 33.59% of the energy in the considered computing cluster.

Within the current study, the system model and main idea of proposed approach were modified and can be described as follows:

1. To define experimentally individual dependencies $P_j = f(CPU_j)$ for all computing nodes of the system as power consumption functions. Having power consumption functions and performance value of each node, to range all nodes according to their integrated performance and power consumption criteria from the best one to the worst one.

2. To use energy-aware PCPB scheduling approach proposed in [11] to distribute the workload in the system.

3. On the basis of the daily workload statistics, to determine a set of scaling patterns describing the state of the computer system (in particular the number of active nodes of the system) such that:

    - The probability of job loss is minimized and is less than what is required by SLA.
    - Workload is processed with sufficient performance.
    - Total energy consumption of the system is minimized. Individual functions of power consumption are used to define the total energy consumption of the system.

Scale the system horizontally with the respect of defined patterns. Patterns should be formed only once on the basis of long-term workload statistics.

4. To detect the deviation of the current workload from the statistical one and adjust scaling patterns dynamically for the fulfillment of the conditions listed above.

4.1 Individual $P_j = f(CPU_j)$ dependencies definition

As the first step of the approach, it is proposed to define individual power consumption functions $P_j = f(CPU_j)$ for each node of the system. This process is described in details in [12]. In a nutshell, it is done as follows:

- An appropriate stress test is chosen in order to load each node's CPU from 0 to 100%.
- A power consumption of each node is measured for a set of CPU utilization levels (this can be done using special hardware (multimeter/wattmeter) or software that also evaluates power consumption according to computer hardware models. It is recommended to use hardware measurement tools as they provide higher measurement accuracy).
- An analytical function is obtained from the experimental data using interpolation [15].

As a result, $P_j = f(CPU_j)$ functions are obtained in a form of polynomials (Eq. (1)).

$$P_j(CPU_j) = a * CPU_j^4 + b * CPU_j^3 + c * CPU_j^2 + d * CPU_j^1 + e * CPU_j^0 \tag{1}$$

where a, b, c, d and e are fourth degree polynomial coefficients, defined within interpolation process. An example of the $P_j = f(CPU_j)$ curves obtained for five computing nodes is presented in **Figure 4**.

Having $P_j = f(CPU_j)$ functions and performance values $flops_j$ (in FLOPS) for all nodes in the system, we can range them from the worst one to the best one as follows:

1. For each node calculate the area under power consumption curve using Eq. (2).

$$S_j = \int_0^{100} P_j(CPU_j)dCPU_j, \tag{2}$$

In Eq. (2) CPU load is expressed as a percentage from 0 to 100%.

2. Sort all nodes by their S_j value in descending order.

3. Sort all nodes by their $flops_j$ value (performance metric) in ascending order.

4. Grant each node with a mark that equals the sum of the node positions in two sorted arrays. The node with the highest mark is considered to be the best one.

Thus, using this simple approach, we are able to form sorted list of nodes in order to make a decision, in which the node is the most inefficient one and should be switched off first during scale-in process (**Figure 4**). Obviously, this only makes sense for inhomogeneous systems where nodes are different from each other and can be sorted.

4.2 Basic scheduling scheme

As a basic scheduler in a system, it is proposed to use PCPB energy-aware scheduler that was proposed and further developed in [11–13]. The main idea of this

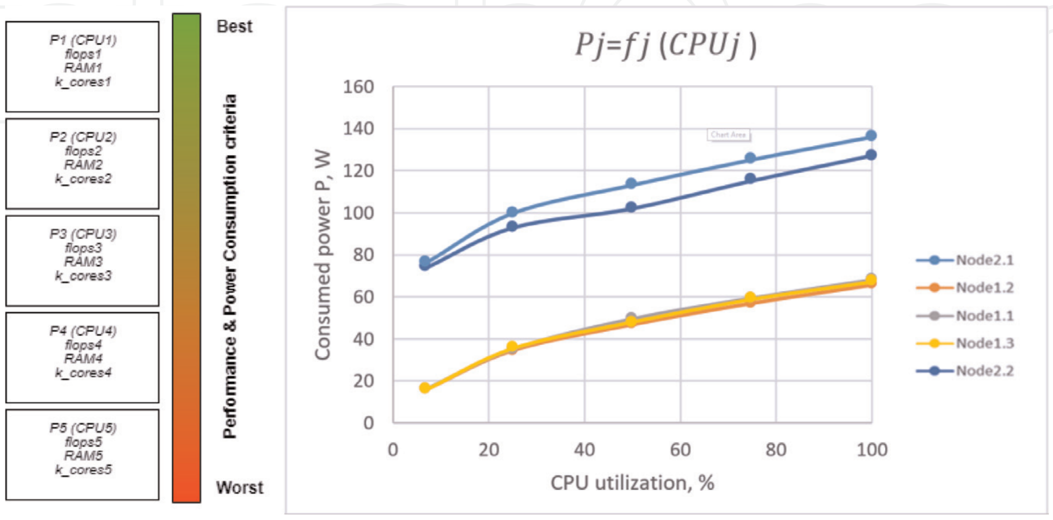


Figure 4.
Sorted list of five nodes according to their integrated performance and power consumption criteria and their $P_j = f(CPU_j)$ curves obtained experimentally.

scheduler is to use individual power consumption functions $P_j = f(CPU_j)$ within the scheduling process in order to determine dynamically, on which node the task is going to consume less power. This approach considers performance criteria as well. The main idea of approach is similar to that one described for the nodes' ranging; however, in PCPB the current state of computing nodes is considered. As it possible to see in **Figure 4**, the increase in energy consumption at different levels of CPU load is different (the curves grow steeper when the load changes from 0 to 25% and flatter when the load changes from 75 to 100%).

The PCPB scheduling algorithm is described by **Figure 5**.

Scheduler gathers the data regarding current load of the nodes in a system. In Step 1 it evaluates the CPU utilization of each node at the time moment τ_{k-1}, before scheduling the next job in a queue.

Having the knowledge on how many resources the job_i requires, scheduler excludes those nodes that currently do not have enough RAM or cores available for the execution (Step 2).

In Step 3 scheduler calculates the theoretical total power that is going to be consumed by the whole computing system supposing that job_i is given for the processing to the node N_j in the moment τ_k (Eq. (3)):

$$P_{\Sigma_i}\big|\tau_k \& N_j = P_\Sigma\big|\tau_{k-1} + \Delta P_{ij}|\tau_k, \tag{3}$$

where $P_\Sigma|\tau_{k-1}$ is the total power consumption of the whole system at the moment τ_{k-1} and $\Delta P_{ij}|\tau_k = f(CPU_j|\tau_k)$ is the increase of power consumption, in the case of job_i being allocated to the node N_j at the moment τ_k.

After that the nodes are sorted according to the current $P_{\Sigma_i}|\tau_k \& N_j$ values and according to their performance value $flops_j$. The nodes get the mark for their position in both sorted lists. The job is allocated to the node with the maximal mark (the detailed description of PCPB approach is presented in [11]).

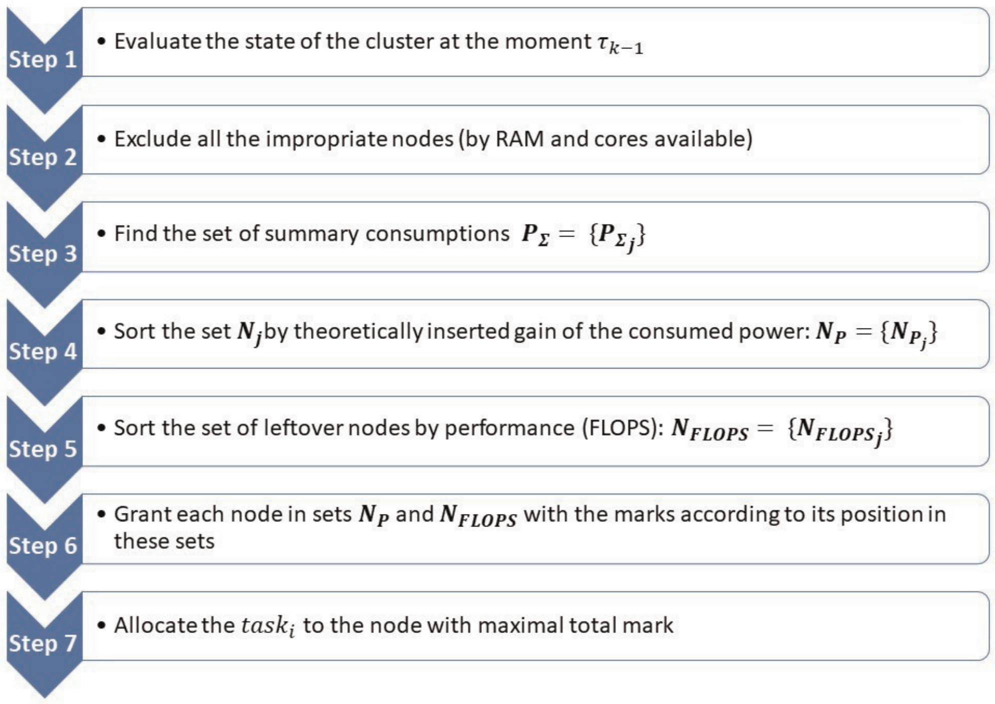

Step 1
• Evaluate the state of the cluster at the moment τ_{k-1}
Step 2
• Exclude all the impropriate nodes (by RAM and cores available)
Step 3
• Find the set of summary consumptions $P_\Sigma = \{P_{\Sigma_j}\}$
Step 4
• Sort the set N_j by theoretically inserted gain of the consumed power: $N_P = \{N_{P_j}\}$
Step 5
• Sort the set of leftover nodes by performance (FLOPS): $N_{FLOPS} = \{N_{FLOPS_j}\}$
Step 6
• Grant each node in sets N_P and N_{FLOPS} with the marks according to its position in these sets
Step 7
• Allocate the $task_i$ to the node with maximal total mark

Figure 5.
PCPB scheduling algorithm.

4.3 System state pattern determination

Since within previous research [13], it was determined that energy-aware scheduling gives the best results in terms of energy efficiency while being used in conjunction with horizontal scaling; let us define the optimal system scale patterns in a form of set $\{n, t_{start}, t_{end}\}$, where n is a number of active computing nodes in the system and $\{t_{start}, t_{end}\}$ denotes the boundaries of time interval of the day, during which these patterns remain optimal. These patterns are formed with respect to the power consumption and job loss probability metrics. In order to find an optimal pattern for each time interval, it is a need to determine these two metrics as functions of number of active nodes in the system.

The certain level of system availability for service user is guaranteed by the SLA as the probability that service will be available (i.e. the user's job will be served) when it is requested. Let the probability of the job loss required by SLA be defined as in Eq. (4):

$$p_{loss_{SLA}} = 1 - p_{SLA} \tag{4}$$

where P_{SLA} is guaranteed by SLA probability that the job will be served.

The more nodes available in the system, the less likely it is that a new incoming task will find the system in a state where all nodes and the queue are fully loaded. It means that the probability of job loss depends on the number of available nodes in computing system and can be presented as a function $p_{loss} = f(n)$, where n is the number of active nodes in the system.

Thus, the purpose of scaling patterns is to answer the question: How many active nodes does the system need to have in order to fulfill the requirements to p_{loss} and minimize total power consumption P_{Σ}?

In order to estimate the dependency between the probability of job loss p_{loss} and number of active nodes *n,* consider the queueing system with a finite queue. The basic concepts of queuing theory are requests (or customers) and servers [16]. The natural way of modeling considered system as a queueing system would be to treat jobs as requests and computing nodes as servers. However, in this case we would have to eliminate the fact that the job may be served by several cores of one server and one server may serve several jobs simultaneously as well. These facts bring the certain complexity to the queueing system modeling. To be more precise, consider one computing core to be a server in terms of queueing system. In order to model input workload as requests of queueing system, we introduce a correction factor $k = cores_{\min avg}$ that denotes how many cores in average are requested by an input job. Using this correction factor, we can represent an input workload as a number of queueing system requests per time unit [Eq. (5)]:

$$\lambda(t) = k * \lambda_{stat}(t), \tag{5}$$

where $\lambda_{stat}(t)$ is the statistical workload represented as a number of computational jobs.

Input data for queuing system:

- Number of active servers in the system:
- n computing nodes or $n' = \sum_{j=1}^{n} k_{core_j}$ servers in terms of queueing system, where k_{core_j} is the number of cores of j^{th} node.

- Queue length: Q jobs or $k * Q$ queueing system requests

- Service discipline:

Serving discipline is basically defined by scheduler that is being used. In considered case, using PCPB, the jobs are selected from the queue according to the first-in-first-out (FIFO) discipline.

- *The mean arrival rate:*

The mean arrival rate can be defined on the basis of the workload statistics as it is done in Eq. (5).

However, in order to build a model, it is worthy to operate with the value λ_{Tmax} that is the maximal value of load during the period T (T defines the time of one pattern validity).

- *The mean service rate:*

In general case, in order to evaluate the service rate for a particular server of the system, it is necessary to determine the average number of requests that leave the server after a successful processing per time unit. This is a random value; within this study we will consider service rate to be determined experimentally (on the basis of statistic of load processing for a particular system) and defined by its mean value μ_{avg}. In this case, the model is actually simplified, but still allows us to estimate the general relationship between the probability of loss and the number of nodes in the system.

Queueing system problem:

To define the probability of job loss, $p_{loss} = f(n)$ as the function of the number of active servers n (nodes of computing system), such that $2 < n < N$, where N is the total number of nodes in the system.

This system can be represented with a Markov chain (**Figure 6**), where S_0 denotes the state of the system, when 0 servers are occupied and the queue is empty, $S_1 - S_{n'}$ denotes there are 1 *to* n'occupied servers, and $S_{n'+1} - S_{n'+kQ}$ denotes all servers are occupied and some requests are placed to a queue.

The probability of loss for the queueing system with a finite queue is defined by Eq. (6) [16]:

$$p_{loss} = p_0 * \frac{\rho^{n'+kQ}}{n'!n'^{kQ}} \tag{6}$$

where

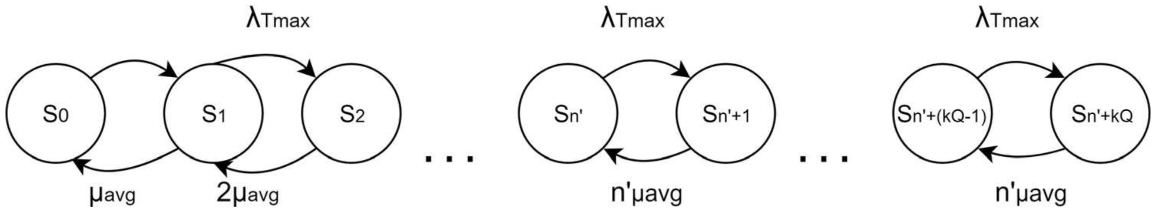


Figure 6.
System model as a Markov chain.

$$p_0 = \left(1 + \frac{\rho}{1!} + \frac{\rho^2}{2!} + \ldots + \frac{\rho^{n'}}{n'!} + \frac{\rho^{n'+1}}{n' * n'!} * \frac{\left(1 - \left(\frac{\rho}{n'}\right)^{kQ}\right)}{1 - \frac{\rho}{n'}}\right)^{-1}$$

$$\rho = \lambda_{Tmax}/\mu_{avg}$$

n'—number of servers in queueing system.

kQ—length of the queue.

Thus, the simplified model described above allows us to estimate the dependency between the number of nodes of the system and probability of the loss. This model should be precisely adjusted to each real system, but the general principle may still stay the same.

Let us estimate the power consumption of the system in order to find an optimal number of nodes so that the power consumption is minimized and probability of loss does not exceed the permissible by SLA value.

As the first step, we defined individual functions of power consumption $P_j = f(CPU_j)$ for each node in the system. These functions may be formally represented in the form of polynomials (Eq. (1)).

The total power consumption of the whole system (as a sum of nodes' individual power consumptions) depends on the load that is being processed by the nodes. We have the daily workload on the system as an input data. However, according to the system model, current load of each system node highly depends on the scheduling technique.

Initially, input workload is represented as a "number of jobs per time unit". Each job may differ computationally and can be evaluated by its maximal execution time or job volume (as a number of floating-point operations). For the sake of simplicity, let us consider the static situation in some point of time, when we have k jobs that would totally fit to the processing nodes. Assume that these jobs in total form the load to the system of $\sum_{i=1}^{k} V_i[operations]$. In this case, scheduler will schedule these jobs accordingly to its policy. In the simplest case, if round-robin fair scheduler is used [17], it would schedule tasks in a way that each node would get an average $s = \frac{\sum_{i=1}^{k} V_i}{n}$ of load. Let s be a scheduler coefficient, which should be defined for each concrete scheduler. In the case of using round-robin, the dependency of power consumption of the node from its load can be formulated as Eq. (7):

$$\begin{aligned} P_j = f(CPU) = f\left(\frac{\sum_{i=1}^{k} V_i}{n} * C\right) = a * \left(\frac{\sum_{i=1}^{k} V_i}{n} * C\right)^4 + b * \left(\frac{\sum_{i=1}^{k} V_i}{n} * C\right)^3 + c * \\ * \left(\frac{\sum_{i=1}^{k} V_i}{n} * C\right)^2 + d * \left(\frac{\sum_{i=1}^{k} V_i}{n} * C\right)^1 + e * \left(\frac{\sum_{i=1}^{k} V_i}{n} * C\right)^0, \end{aligned} \tag{7}$$

where C is a constant, which expresses how the input load is converted into a CPU load in percentages. And for the more general case, Eq. (7) is transformed into Eq. (8):

$$\begin{aligned} P_j = f(CPU) = f(s * C) = a * (s * C)^4 + b * (s * C)^3 + \\ + c * (s * C)^2 + d * (s * C)^1 + e * (s * C)^0 \end{aligned} \tag{8}$$

This formula is very simplified though. Depending on scheduler being used, the load of the system's nodes may be different.

There is another more generalized approach to evaluate power consumption of system nodes. Usually, servers are kept to be utilized at some average level CPU_{avg_j} that may be defined statistically for each system. It is actually more precise solution in comparison with analytical definition of CPU utilization that depends on scheduling being used and incoming load. For the whole system, total power consumption is then defined by Eq. (9):

$$P_\Sigma = \sum_{j=1}^{n} P_j\left(CPU_{avg_j}\right), \tag{9}$$

where $P_j\left(CPU_{avg_j}\right)$ is the individual functions of nodes' power consumption.

The mathematical models of total power consumption and loss probability mentioned above lead us to the optimization problem that can be formulated as follows:

To define an optimal number of active nodes of the distributed computing system n, the objective function (Eq. (10)) is minimized subject to Eq. (11).

$$P_\Sigma = \sum_{j=1}^{n} P_j\left(CPU_{avg_j}\right) \to min \tag{10}$$

subject to:

$$p_{loss} = p_0 * \frac{\left(\frac{\lambda_{Tmax}}{\mu_{avg}}\right)^{n'+kQ}}{n'!n'^{kQ}} < p_{loss\,SLA}, \; 1 < n < N \tag{11}$$

To solve this problem, we need to choose the time intervals T for which we will create the patterns. Since patterns are created statically on the basis of statistical workload, it is possible to define the time moments, when thresholds are achieved so that $p_{loss} = p_{loss\,SLA}$. When it happens there is a need to increase the number of nodes (thus, to change the pattern). In case the load is decreasing, it is possible to define the moment when P_Σ is not minimal anymore for the current statistical load.

The above problem solved for each time interval T would allow to determine the optimal number of processing nodes n for each time interval depending on the incoming load. Meanwhile, the following conditions will be fulfilled:

1. Energy consumption of the system will be minimal.

2. Job loss probability will meet the SLA requirements.

On the basis of the obtained values of n, T, it is possible to create patterns for the number of nodes of the system for periods of the day. These patterns may be created once for the regular workload and be corrected during further system operation. Since the dependencies of energy consumption and loss probability from the number of active nodes in system are already defined, it requires only minimal effort to adapt the system to a new workload pattern.

4.4 Deviations from patterns detection and system reconfiguration

In order to cope with unpredictable service workload deviations that can be caused by different reasons (e.g. holidays, special events, etc.), there is a need to detect significant deviations from the statistic workload curve, predict the workload for the next chosen time period and reconfigure the system, respectively.

The authors of the research [18] have proposed an approach to hybrid resource provisioning in virtualized networks. The main idea of this approach is as follows:

- To use workload statistics to create a baseline resource allocation scheme
- To monitor deviation of the actual workload from the statistical one
- To react dynamically to deviations from the base workload, which exceeds the permissible value
- To predict the workload for the next time interval T and adjust the resource allocation accordingly

This approach can be briefly explained using **Figure 7**, where an example of applying the approach is highlighted by red circle.

Within the current study, this approach may be adapted to serve the need of detecting deviations from the statistical workload in considered distributed systems and to adjust the number of active system nodes in accordance with defined optimization problem.

Consider that the system has k scaling patterns that correspond to the daily workload (**Figure 8**). Pattern in this context means the number of active nodes that remains optimal for the certain time period, so the pattern may be denoted as a set $\{n, t_{start}, t_{end}\}$, where n is the number of active nodes and t_{start}, t_{end} show the interval of the day, during which these patterns remain optimal. These patterns may contain other configuration information as well, but it lies beyond the scope of the current study.

According to the adapted approach proposed in [18], the current workload should be monitored throughout the day, but the intensity of the monitoring may vary according to the deviation between the statistical workload and the current one. Let I_{base} be a baseline time monitoring interval. This interval should be determined empirically for each system. Monitoring interval then should be adjusted according to Eq. (12):

$$W(t) = I_{base} - K \sum_{j=t-h}^{t-1} \frac{\max(0; \lambda_{obs}(t) - \lambda_{pred}(t))}{h} I_{base}, \tag{12}$$

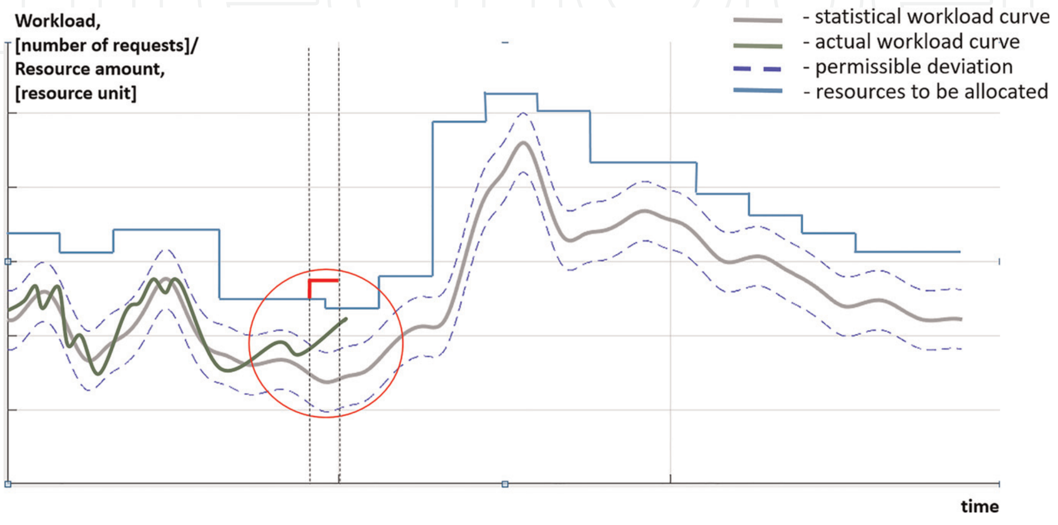


Figure 7.
The main idea of hybrid resource provisioning approach [18].

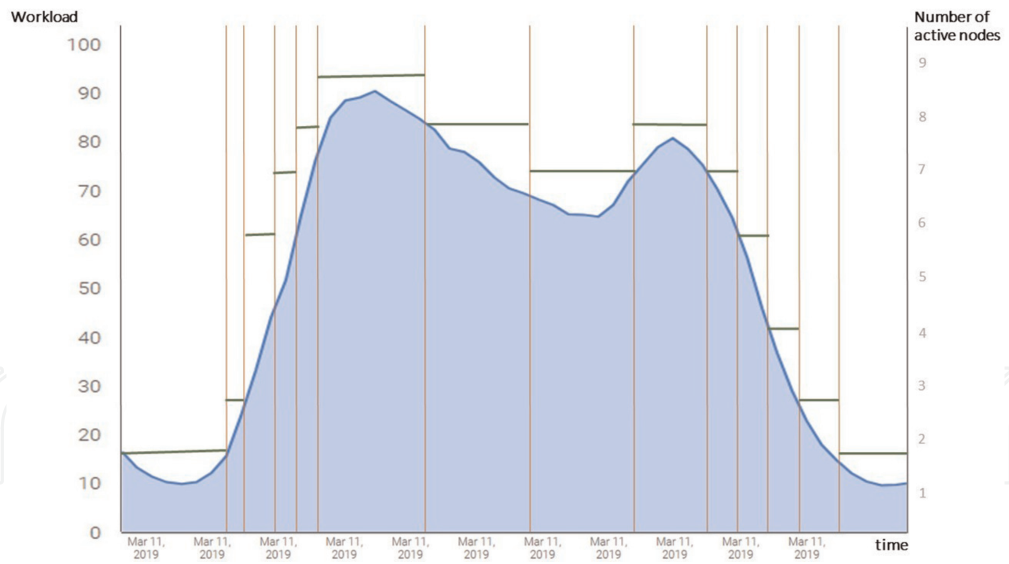


Figure 8.
Scaling patterns of the system that correspond to the daily workload.

where:
W is an new monitoring interval.
I_{base} is the baseline predefined monitoring interval.
K is the normalization constant that should be defined for each system individually.
$\lambda_{obs}(t)$ is the current load arrival rate during the time t.
$\lambda_{pred}(t)$ is the predicted load arrival rate during the time t (according to the statistical workload curve).
h is the number of preceding intervals considered by the algorithm.
If $\lambda_{basepred}(t)$ is a basic (statistical) load arrival rate for a period of time t and $\lambda_{obs}(t)$ is the current load arrival rate during the time t, the load predicted for the next interval should be adjusted according to Eq. (13):

$$\lambda_{pred}(t) = \lambda_{basepred}(t) + \sum_{j=t-h}^{t-1} \frac{\lambda_{obs}(j) - \lambda_{pred}(j)}{h} \tag{13}$$

This approach is described in details in [18].
Thus, having the predicted value of workload for the next time period, we can recalculate the pattern for that period and change the number of nodes in the system according to the calculated optimal number.
As a result of applying proposed steps, we get formed patterns $\{n, t_{start}, t_{end}\}$ for the static day workload and may adjust them according to the deviation between current workload and statistical one.

5. Modeling

Within the modeling the system with comparatively low workload and number of computing nodes was chosen for clarity, but such model can be scaled for larger systems as well. Matlab system was used to create the model.
Suppose we have a daily workload curve depicted in **Figure 9** (the value of workload is defined for each minute, so we also have value λ for each minute).
For this workload let us define static scaling patterns that describe the optimal numbers of active nodes and time periods, for which these numbers remain optimal.

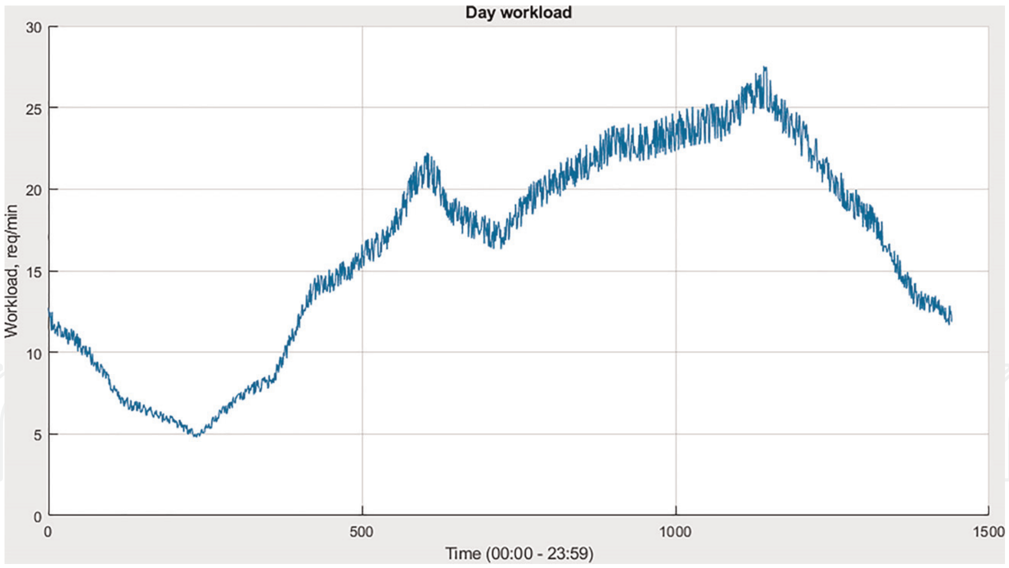
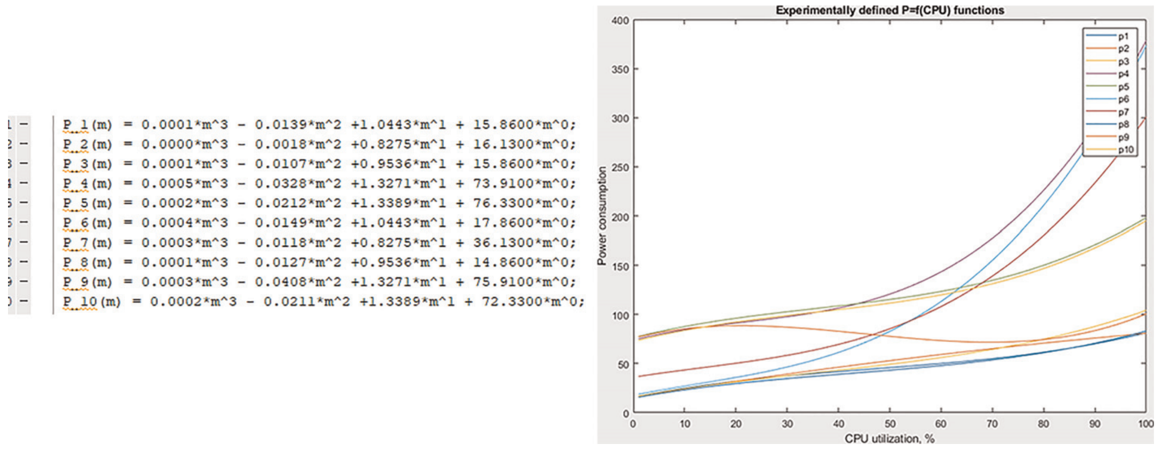


Figure 9.
Daily workload curve used for modeling.



```
P_1(m) = 0.0001*m^3 - 0.0139*m^2 +1.0443*m^1 + 15.8600*m^0;
P_2(m) = 0.0000*m^3 - 0.0018*m^2 +0.8275*m^1 + 16.1300*m^0;
P_3(m) = 0.0001*m^3 - 0.0107*m^2 +0.9536*m^1 + 15.8600*m^0;
P_4(m) = 0.0005*m^3 - 0.0328*m^2 +1.3271*m^1 + 73.9100*m^0;
P_5(m) = 0.0002*m^3 - 0.0212*m^2 +1.3389*m^1 + 76.3300*m^0;
P_6(m) = 0.0004*m^3 - 0.0149*m^2 +1.0443*m^1 + 17.8600*m^0;
P_7(m) = 0.0003*m^3 - 0.0118*m^2 +0.8275*m^1 + 36.1300*m^0;
P_8(m) = 0.0001*m^3 - 0.0127*m^2 +0.9536*m^1 + 14.8600*m^0;
P_9(m) = 0.0003*m^3 - 0.0408*m^2 +1.3271*m^1 + 75.9100*m^0;
P_10(m) = 0.0002*m^3 - 0.0211*m^2 +1.3389*m^1 + 72.3300*m^0;
```

Figure 10.
Analytical and graphical representation of the $P = f(CPU)$ functions defined experimentally.

As an input data for modeling, we also have the following:

1. $P = f(CPU)$ functions for $N = 10$ servers in the form of polynomials. These functions were defined experimentally and interpolated using fourth degree polynomials. Analytical and graphical representation of the $P = f(CPU)$ functions is presented in **Figure 10**.

2. Corresponding values of RAM volume, number of cores and performance for these ten servers.

3. Scheduler algorithm that is being used—PCPB algorithm (the details regarding PCPB algorithm modeling are provided within the previous research [11]).

4. Average serving rate for given servers: $\mu_{avg} = 25$ req/min.

5. $Q = 5$, the length of the queue.

6. Let the probability of loss be defined by SLA be $p_{loss SLA} = 10^{-6}$.

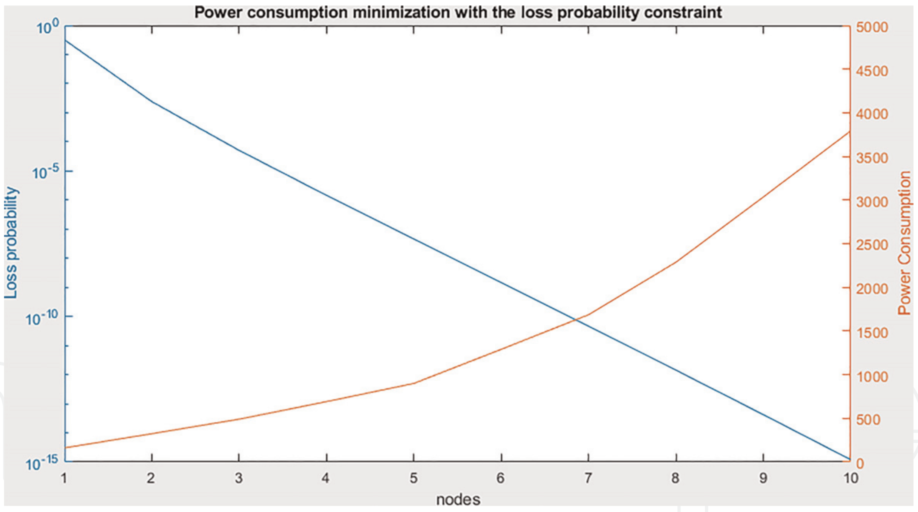


Figure 11.
Loss probability and power consumption calculations for the first workload value.

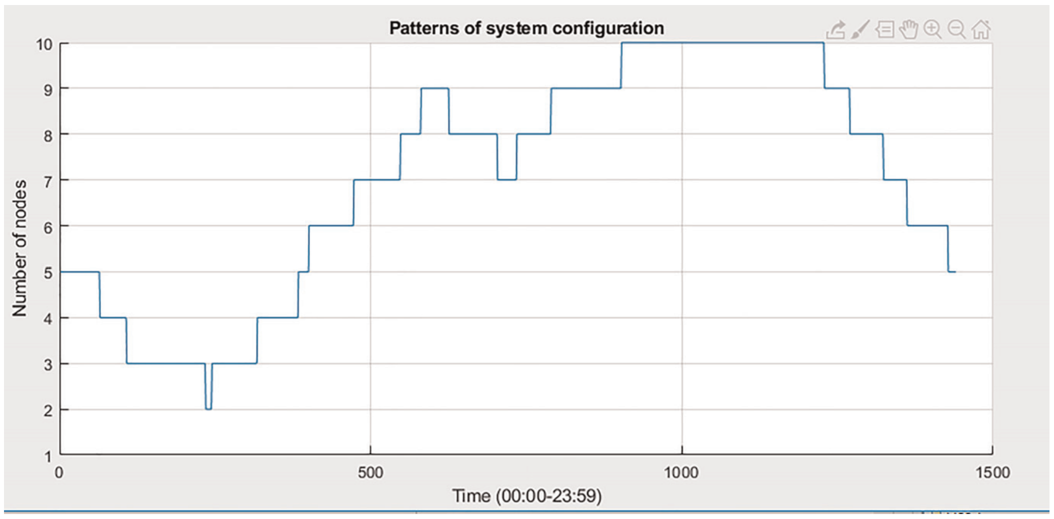


Figure 12.
Created scaling patterns for the given input workload.

In order to create patterns for certain time periods, it is worthy to run calculations according to Eq. (1) for each minute at first. Then, as soon as the thresholds of p_{loss} and P_{j_Σ} are defined, it is possible to derive $\{t_{start}, t_{end}\}$ values for each pattern.

Let us build the curves described by Eq. (10) and Eq. (11) for the first minute workload. Resulting curves are depicted in **Figure 11**.

In **Figure 11** it is possible to see that in order to meet the SLA requirements $p_{loss SLA} = 10^{-6}$, we need to have at least five nodes in the system. We can also see that these five nodes will consume near 900 W of energy. As the nodes are sorted from the worst one to the best one, these five leftover nodes are the most energy efficient and productive among the others.

We need to do the same for the remaining part of workload curve. This process was modeled using Matlab. As a result, we got all the scaling patterns for the given input workload (**Figure 12**).

Let us introduce some deviation to the statistical workload in order to apply pattern adjustment. Introduced workload deviation is circled in red in **Figure 13**. Adjustment was made according to Eqs. 12 and 13 proposed in [18].

In **Figure 13**, the adjustment of the received patterns is shown in the right graph. As it is possible to see, the interval of patterns optimality and number of nodes were changed, respectively.

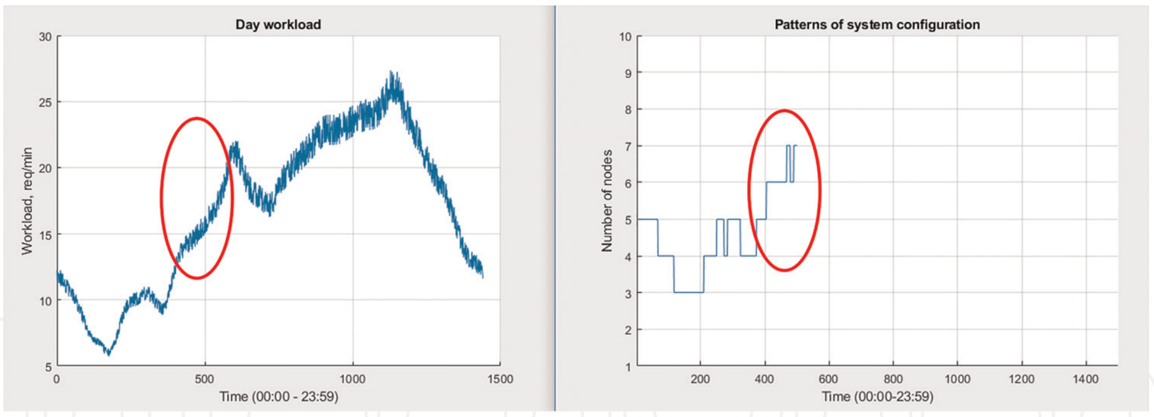


Figure 13.
Scaling patterns' dynamic adjustment result.

Thus, created model fully illustrates the process of proposed intelligent workload scheduling approach which is improved through the use of scaling approaches. The model shows the main idea of proposed approach; however, in real systems it is worthy to add some redundancy and always keep at least one extra server active in order to cope with unpredicted workload deviations.

6. Conclusion

In this chapter an intelligent approach to the workload scheduling in distributed computing environment was proposed. According to the proposed approach, energy-aware PCPB scheduling algorithm is combined with scaling approaches that allow to achieve an optimal balance between energy efficiency and performance while fulfilling the SLA requirements. In order to achieve these goals, we propose to create and dynamically adjust system scaling patterns while using energy-aware scheduling. An approach was modeled using Matlab. The simulation results showed that it is able to cope with the efficient processing of statistical load, as well as load deviations.

Within the future research, the system representation as a queueing system is going to be made more precisely, and proposed approach's efficiency should be proven by means of experiment.

Author details

Larysa Globa[1]*, Oleksandr Stryzhak[2], Nataliia Gvozdetska[1] and Volodymyr Prokopets[1]

1 National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine

2 National Academy of Sciences in Ukraine, Kyiv, Ukraine

*Address all correspondence to: lgloba@its.kpi.ua

References

[1] Gartner. Gartner Top 10 Strategic Technology Trends for 2019 [Internet]. 2018. Available from: https://www.gartner.com/smarterwithgartner/gartner-top-10-strategic-technology-trends-for-2019/

[2] Andrae A, Edler T. On global electricity usage of communication technology: Trends to 2030. Challenges. 2015;**6**:117-157. DOI: 10.3390/challe6010117

[3] Beloglazov A. Energy-efficient management of virtual machines in data centers for cloud computing [thesis]. Department of Computing and Information Systems, The University of Melbourne; 2013

[4] Suleiman D, Ibrahim M, Hamarash I. Dynamic voltage frequency scaling (DVFS) for microprocessors power and energy reduction. In: Proceedings of the 4th International Conference on Electrical and Electronics Engineering. 2005

[5] Akram S, Sartor J, Eeckhout L. DVFS performance prediction for managed multithreaded applications. In: Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software-ISPASS. 2016. pp. 12-23

[6] Agarwal K, Nowka K. Dynamic power management by combination of dual static supply voltages. In: Proceedings of the 8th International Symposium on Quality of Electronic Design (ISQED 2007); 26-28 March 2007; San Jose, CA, USA. 2007

[7] Möbius C, Dargie W, Schill A. Power consumption estimation models for processors, virtual machines, and servers. In: Proceedings of the IEEE Transactions on Parallel and Distributed Systems. 2014. pp. 1600-1614

[8] Aldossary M, Djemame K. Performance and energy-based cost prediction of virtual machines auto-scaling in clouds. In: Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2018); 29-31 August 2018; Prague, Czech Republic. 2018. pp. 502-509. DOI: 10.1109/SEAA.2018.00086

[9] Liu W, Du W, Chen J, Wang W, Zeng G. Adaptive energy-efficient scheduling algorithm for parallel tasks on homogeneous clusters. Journal of Network and Computer Applications. 2013;**41**:101-113. DOI: 10.1016/j.jnca.2013.10.009

[10] Armenta-Cano F, Tchernykh A, Cortés-Mendoza J, Yahyapour R, Drozdov A, Bouvry P, et al. Heterogeneous job consolidation for power aware scheduling with quality of service. In: Proceedings of the Russian Supercomputing Days. RuSCDays'15; Moskow. 2015

[11] Schill A, Globa L, Stepurin O, Gvozdetska N, Prokopets V. Power consumption and performance balance (PCPB) scheduling algorithm for computer cluster. In: Proceedings of the 2017 International Conference on Information and Telecommunication Technologies and Radio Electronics (UkrMiCo 2017); Odesa, Ukraine. 2017. pp. 1-8

[12] Gvozdetska N, Stepurin O, Globa L. Experimental analysis of PCPB scheduling algorithm. In: Proceedings of the 14th International Conference the Experience of Designing and Application of CAD Systems in Microelectronics (CADSM); 21-25 February 2017; Polyana-Svalyava (Zakarpattya), Ukraine. 2017

[13] Bezruk V, Globa L, Stryzhak O, editors. Knowledge-Based Technologies

of Optimization and Management in Infocommunication Networks: Monograph. 1st ed. Kyiv: National Academy of Pedagogical Sciences of Ukraine; 2019. p. 194. ISBN 978-617-7734-02-3

[14] Google. Google Transparency Report [Internet]. 2019. Available from: http://www.google.com/transparencyreport/traffic/

[15] Mastroianni G, Milovanovic G. Interpolation Processes: Basic Theory and Applications. 1st ed. Berlin, Heidelberg: Springer; 2008. p. 446. DOI: 10.1007/978-3-15540-68349-0

[16] Sztrik J. Basic Queueing Theory. 1st ed. Debrecen, Hungary: University of Debrecen, Faculty of Informatics; 2011. p. 193

[17] Wikipedia. Round-Robin Scheduling [Internet]. 2019. Available from: https://en.wikipedia.org/wiki/Round-robin_scheduling

[18] Sulima S, Skulysh M. Hybrid resource provisioning system for virtual network functions. Radio Electronics, Computer Science, Control. 2017;(1): 16-23. DOI: 10.15588/1607-3274-2017-1-2