2013 IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing

Towards Energy-Aware Resource Scheduling to Maximize Reliability in Cloud Computing Systems

Hamid Reza Faragardi[1], Aboozar Rajabi[2], Reza Shojaee[2] and Thomas Nolte[1]
[1]Mälardalen Real-Time Research Centre, Mälardalen University, Västeras, Sweden
[2]School of Electrical and Computer Engineering, University of Tehran, Tehran, Iran
{hamid.faragardi@mdh.se, ab.rajabi@ut.ac.ir, r.shojaee@ut.ac.ir, thomas.nolte@mdh.se}

Abstract—**Cloud computing has become increasingly popular due to deployment of cloud solutions that will enable enterprises to cost reduction and more operational flexibility. Reliability is a key metric for assessing performance in such systems. Fault tolerance methods are extensively used to enhance reliability in Cloud Computing Systems (CCS). However, these methods impose extra hardware and/or software cost. Proper resource allocation is an alternative approach which can significantly improve system reliability without any extra overhead. On the other hand, contemplating reliability irrespective of energy consumption and Quality of Service (QoS) requirements is not desirable in CCSs. In this paper, an analytical model to analyze system reliability besides energy consumption and QoS requirements is introduced. Based on the proposed model, a new online resource allocation algorithm to find the right compromise between system reliability and energy consumption while satisfying QoS requirements is suggested. The algorithm is a new swarm intelligence technique based on imperialist competition which elaborately combines the strengths of some well-known meta-heuristic algorithms with an effective fast local search. A wide range of simulation results, based on real data, clearly demonstrate high efficiency of the proposed algorithm.**

Keywords- cloud computing; reliability; analytical model; resource allocation; quality of service; energy-aware scheduling.

I. Introduction

Cloud computing is widely referred as the next generation of computing systems in which dynamically scalable and often virtualized resources are provided as services over the Internet [1]. Service sharing and utility computing are the main characteristics of Cloud Computing Systems (CCS), which distinguish CCS from grid, cluster computing, and other types of distributed systems. Nowadays, a wide range of services are provided by the cloud providers such as computational resources for high performance computing applications, web services, social networking, and telecommunications services. The users can utilize cloud services from any corner of the world in pay-as-you-go manner. In CCSs, a Service Level Agreement (SLA) provides a facility to agree upon minimum requirements between end-user and cloud provider. The SLA contains Service Level Objectives (SLOs) that the services need to fulfill. Quality of Service (QoS) and dependability are the most important SLOs stipulated in the SLA [6]. QoS can be defined differently in various systems and from different perspectives. In this paper, QoS is defined as the meeting rate of service deadlines. Furthermore, only reliability is taken into account as a prominent dependability measure while other dependability measures such as availability, maintainability and security are out of scope of this paper. Nevertheless, improving the system reliability could also enhance system availability. If the SLOs are violated in the sense that the services are not executed within the negotiated QoS, agreed upon consequences (usually taking the form of penalty payments) go into effect. On the other side, fulfilling the SLOs results in an additional cost for the service providers (e.g., because the extra hardware cost, more energy consumption and the required cost to optimize the services). The amount of money that should be paid by the users to utilize a service is a considerable factor. If a service supplied by a cloud provider is more expensive than that provided by others, the users may not wish to utilize the services. Therefore, cloud providers should struggle to supply services within the admissible QoS with the minimum possible cost. Energy consumption is a dominant factor which directly affects the cost of services. Hence, in this paper energy consumption is also taken into account besides QoS, reliability and other system constraints.

There are several computing and communication components in a typical cloud computing system such as memory, disk, RAID disk controller, processor, communication link etc. Although each of which are carefully engineered, they may still fail. Due to large-scale service sharing, over a wide area network, using heterogeneous software/hardware components, having complicated interaction among, reliability in the CCS is a challenging issue and difficult to obtain [7].

A wide variety range of services are provided by the CCSs. Each service comprises a set of tasks. The service will be terminated once the execution of all corresponding tasks have finished. The tasks of a service can be allocated to multiple hosts and executed concurrently on different cores. We define two important reliability measures in CCSs, service reliability and system reliability. Service reliability is the probability that all tasks of a service run successfully and system reliability is the probability that all services in the system run successfully. If we assume that all software services are perfect, then successful execution of services only depends on the hardware reliability. Therefore, in this situation, the system reliability is equivalent to the hardware reliability during the execution of all services.

Redundancy and/or diversity are prevalent approaches to develop a highly reliable CCS and thereby making the system more fault tolerant [3][4]. However, they result in an extra hardware and/or software cost and thus further cost of services. Another alternative to enhance system reliability in such systems is an appropriate resource allocation. Due to the heterogeneity of CCSs, the hazard rate of various hosts and communication links are not identical. Consequently, this approach can be potentially applied to reach this goal without any additional cost. Proper resource allocation in CCSs has been extensively applied to achieve different purposes such as load balancing, energy management and QoS improvement. For example, Ai et al. [5] presented a resource allocation algorithm based on a random-key genetic algorithm to increase QoS in a hybrid Cloud. However, to the best of our knowledge, the resource allocation approach has not already been used to enhance system reliability in CCSs.

Maximizing reliability without paying attention to energy consumption may cause an increase of the cost of services as a result of higher energy consumption. It can occur when some of the more reliable hosts (i.e., the hosts with the lower hazard rate) are consuming more energy, for example due to different architectures and/or disparate operational environmental conditions. Thus, a reliability-aware resource allocation in which most of the services are allocated to the more reliable hosts may result in a higher energy consumption. Furthermore, increasing energy consumption leads to a higher operating cost and destructive environmental impacts [14]. Thus, we aim to evaluate the system reliability besides energy consumption and to analyze their impacts on each other.

The problem addressed in this paper can be described as finding a proper resource allocation to reach the equilibrium of the reliability and energy-awareness while at the same time satisfying application and resource constraints. QoS is taken into account as the application constraint. Memory and storage limitation of each server and the maximum communication load on each link are considered as the principle resource constraints.

Since optimal resource allocation in CCSs is known to be NP-hard in the general case [6], the mentioned problem is also NP-hard in the strong sense. The problem is formulated using Integer Linear Programming (ILP). There are two general options for dealing with the ILP problem; namely, offline and online. The online schedulers should be executed in a shorter time in comparison to offline schedulers. However, ILP-solvers typically take a long execution time. Thus, most of the proposed ILP-solvers used in the context are employed as offline schedulers. On the other hand, in order to cope with the dynamic nature of CCSs, an online scheduler is required. Consequently, we need a fast online ILP-solver while at the same time it should be able to find a high quality solution. In the paper, a new swarm intelligence technique based on the Imperialist Competitive Algorithm (ICA) is introduced as an efficient online scheduler. ICA was originally proposed by Atashpaz-Gargari and Lucas in 2007 to solve the continuous optimization problems [2]. In this paper we present an extended version of ICA intensified by an effective fast local search to tackle with the problem. Simulation results demonstrate the potential of ICA to decrease the energy consumption significantly along with substantial improvement of the system reliability. ICA is compared with GA which has been recently applied to minimize energy consumption [17]. It should be mentioned that in order to carry out a comprehensive evaluation, we equip GA in such a way that it also considers system reliability along with energy consumption. The results manifest that energy consumption is reduced by 17% and system reliability is increased by 9%.

The remainder of the paper is organized as follows: In Section II, a brief survey on related works is presented. The problem is described in detail and assumptions are defined in Section III. Section IV presents a mathematical model to evaluate the reliability besides energy consumption and it formulates the system and application constraints. A new online resource scheduling algorithm is proposed in Section V. Section VI describes simulation and performance evaluation of the proposed algorithms. Finally, concluding remarks and future works are presented in Section VII.

II. Related Works

Reliability analysis is one of the challenging research areas in the classical distributed systems. Some prominent studies have been proposed based on meta-heuristic algorithms to find optimal or near optimal solutions by proper task allocation [26][27][28][31]. Also, in another work, maximizing reliability in real-time distributed systems was investigated [29].

On cloud computing, previous works can be categorized into two main groups. The former group has discussed about the reliability in CCS without considering QoS requirements. For example, Dai et al. [7] investigated various types of failures in CCS and they achieved a comprehensive picture about cloud service reliability. They also modeled the cloud failures using Queuing theory and Markov models. Another important effort on this category belongs to Lin and Chang [8] which evaluated system reliability for a typical CCS with imperfect nodes. They proposed an algorithm based on the branch and bound approach. Nevertheless, their algorithm is reasonable only for small-scale platforms. The latter group has investigated QoS in CCSs without considering reliability. For example, Cao et al. [9] introduced a new service-oriented model to support QoS in the cloud while releasing the resources. They utilized service-oriented QoS-Assured in a multi agent cloud computing architecture. In addition, Armstrong and Djemame [10] addressed QoS provisioning in the cloud and they have explained the differences of QoS issues in cloud and grid computing. Although an analytical model to evaluate reliability of CCSs along with QoS requirements was recently presented by Faragardi et al. [30], it has not considered energy.

The problem of energy-aware resource allocation in cloud environments has been investigated in several works. Buyya et. al [14] have expressed the problem, challenges and requirements. Also, they have proposed a green cloud architecture for CCSs to support the energy-awareness. A taxonomy and survey of energy-efficient cloud computing is published by [15] that explains reasons of energy consumption in clouds along with different power management solutions. The problem of scheduling with consideration of energy consumption has many constraints in the real world. Nevertheless, most previous studies have considered homogenous cloud systems [16][22][23], single core processors [24][32][33] or the deadline-free situations [17]. In addition, some papers have studied scheduling algorithms that use the DVS technique. For example, [17][18] have proposed two energy-efficient algorithms using DVS which calculate energy consumption in different ways and investigate a trade-off between make-span and energy consumption. The DVS technique is not used in our paper because working with the lower voltage levels can potentially result in a reliability reduction.

The main contributions of this paper can be stated as follows:

1. We present a reliability model for CCSs based on the resource allocation approach.
2. We contemplate reliability besides energy consumption, QoS requirements and other system constraints.
3. We propose a new online scheduler based on the imperialist competitive algorithm to find the right compromise between reliability and energy consumption as a bi-objective optimization problem.

III. Problem description

The cloud computing system used in this work includes several hosts which are connected together through a communication network. Each host consists of multiple components such as a multicore processor, hard disks handled by a RAID controller, memory modules etc., and failing each of which could result in a host failure. Due to unequal hazard rate of the underlying components, hosts' hazard rates may differ from one host to another. The communication links between the hosts may have different bandwidths and failure rates. In our model, we suppose other system components such as IO devices, hypervisors, operating systems and database systems are perfect. A fully connected mesh is considered as the network topology. It is worth noting that we assume that each pair of hosts communicate with each other just through the direct link. Consequently, if the direct link between a pair of hosts fails, then these hosts cannot continue to communicate with each other through an alternative path.

Each component of the system (processor, memory, hard disk, RAID controller and communication link) can be in one of two states: operational or failed. We ignore the degradation mode for components to simplify our model, and consider this as a potential future work. Additionally, the small fixed-size failure-notice messages are exchanged over the network in order to detect failed components. As the overhead of these messages is negligible, it can be disregarded. We define a critical failure as the one which causes a service failure. Failure of a component during an idle period is not considered as a critical failure because it can be replaced by a spare. It is should be noted that due to lack of any fast recovery mechanism (such as partial restart), transient and intermittent failures along with permanent failures are considered as critical failures. The failure of a component follows a Poisson process with a constant rate. Furthermore, failures of components are statistically independent. The reliability of the mentioned CCS depends on:

- The number of hosts composing the CCS and their individual likelihoods of failure.
- The likelihood of failure for each link between a pair of servers.

We presume that a service can be divided into a set of tasks which can be allocated to various hosts and executed concurrently on different cores. The tasks of the given service require resources including memory, storage space, computation power and a specific communication bandwidth. The tasks of a service can be represented as an undirected graph called Task Interaction Graph (TIG). Each node of the TIG displays a task and the arcs show the inter-task communication. There is a label on each arc that indicates the amount of data that should be transferred through the communication links between the tasks. Communication between services is not taken into account. Fig. 1 illustrates a TIG instance. In this system, task execution times are processor dependent, meaning that the execution time of a task may vary from one core to another. We assume that the data transmission mechanism between the cores on the same host is perfect. Hence, the failure rate of the links between the cores of a server is set to zero.

The precedence relation between the tasks is also out of scope of this paper. Each service has a certain deadline corresponding to its QoS, and its execution should be completed before the deadline in order to guarantee QoS requirements. Since, Earliest Deadline First (EDF) is an optimal real-time scheduling algorithm on a single processor [11], we assume EDF as the scheduler on each core. Although there is not any explicit deadline for the tasks in the mentioned problem, and the deadlines are considered in terms of services, we associate each service's deadline to all of its tasks. If all the tasks of a specific service complete their jobs prior to the service deadline, then the service definitely meets its deadline. Hence, EDF orders the tasks according to the deadline of corresponding services.

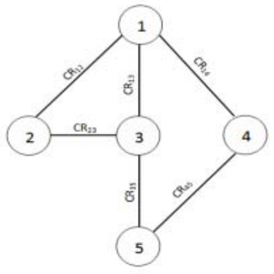

Figure 1. A TIG instance of a cloud service

If the goal is to guarantee QoS, then resource allocation should be performed such that the subset of tasks assigned to each core can be scheduled by EDF and no task misses its deadline. If some of the services do not have the deadline constraint, then we set their deadlines to the infinite value and the model is still valid.

During design time, all required information such as the number of tasks of each service, the average execution time of each task on each core, the amount of data that should be transferred between the tasks etc. can be gathered by the system designers by employing some profiling tools and compiler aids. In addition, the provided services by a cloud provider are usually constant. Hence, the assumption that all information is known in advance is reasonable and the proposed model can be practically applied.

IV. System modeling

A. Notation

The notations used to model the system are listed as follows:

- A represents the number of hosts.
- H_i represents ith host.
- φ_i is a set of processors (cores) on the ith host.
- n_i represents the number of processors (cores) on c_i.
- N represents the total number of processors in the system; $N = \sum_{i=1}^{A} n_i$.
- p_r represents rth processor $(1 \leq r \leq N)$. A general numbering is used for representation of the processors in the system. In this representation, the jth processor of ith host is denoted by p_r where $r = \sum_{l=1}^{i-1} n_l + j$.
- B represents the number of services.
- $S = \{ S_1, S_2, \ldots, S_B \}$ is the service set ordered according to service deadlines in ascending manner.
- D_i is the deadline of service S_i and all of its tasks should be finished before D_i.
- m_i represents the number of tasks which compose S_i.
- M represents the total number of tasks; $M = \sum_{i=1}^{B} m_i$.
- T_k represents the kth task $(1 \leq k \leq M)$. A general numbering is used for tasks representation in which the jth task of the ith service is denoted by T_k where $k = \sum_{l=1}^{i-1} m_l + j$.
- $\omega(k)$ is a function which indicates that T_k belongs to which service and it returns a service number.
- Γ_i is the hard disk hazard rate for c_i.
- Ψ_i is the hazard rate of RAID controller for c_i.
- ζ_i is the memory hazard rate for c_i.
- λ_i is the processor hazard rate of P_i.
- E_{ij} is the execution time of task T_i on processor P_j.
- DL_{ij} is a direct link between c_i and c_j.
- $X = [x_{ij}]$ is task to precessor assignment matrix where x_{ij} equals one, if and only if T_i is assigned to P_j; otherwise $x_{ij} = 0$.
- W_{ij} is the communication bandwidth of DL_{kl}.
- $\pi_{kl}(X)$ is the allocated communication load to DL_{kl} by assignment X.
- θ_{kl} is the maximum allowed load on DL_{kl}.
- CR_{ij} is the amount of transmitted data between T_i and T_j.
- TID_i is the task interaction density which indicates the number of collaborative tasks in S_i.
- μ_{ij} shows communication hazard rate for DL_{ij}.
- MEM_i is the amount of memory for ith server.
- mem_j represents the memory needed by task T_j.
- STG_i is the storage amount of the ith server.
- stg_j represents the storage needed by task T_j.
- $R_s(X)$ is the system reliability for assignment X.
- $R_{s'}(X)$ is the system reliability without considering failure of links.
- $R_{s''}(X)$ is the system reliability without considering failure of servers.
- $C(X)$ is the cost of assignment X (will be defined shortly).
- $TC(X)$ is the total cost of assignment X (will be defined shortly).
- P_i^{max} is the maximum power when H_i is fully utilized.
- P_i^{idle} is the power consumption of H_i at idle state.
- k_i is the fraction of P_i^{max} consumed by H_i at idle state i.e., $k_i = P_i^{idle}/P_i^{max}$.
- $TP(X)$ is the total power consumption of assignment X.
- $\Delta(X)$ is the total energy consumption.
- SP is the scheduling period.

B. Principle constraints

In this section, we outline the principal constraints of the system and we formally define them.

- *Memory:* The memory of each host is no less than the total amount of memory requirements for its all assigned tasks. Eq. 1 formulates this constraint.
  $\sum_{\forall j, p_j \in \varphi_k} \sum_{i=1}^{M} mem_i \, x_{ij} \leq MEM_k \quad 1 \leq K \leq A$ (1)
- *Storage:* The total amount of required storage for the tasks assigned to each host should be less than the available storage on that host.
  $\sum_{\forall j, p_j \in \varphi_k} \sum_{i=1}^{M} stg_i \, x_{ij} \leq STG_k \quad 1 \leq K \leq A$ (2)
- *QoS:* All of the services should be finished before their respective deadlines. We assume that EDF is used as scheduler in each processor and we assign the deadline of each service to all of its tasks. If all of the tasks of a service finish their jobs before the specific deadline of that service, it will result in meeting the deadline of that service. In other words, all of the tasks of S_i are executed before all of the tasks of S_j if $i < j$. Moreover, if T_z and T_c belong to the same service (i.e., have the same deadline) and $z < c$, T_z is executed before T_c. Accordingly, the QoS constraint can be formulated as follows:
  $\sum_{i=1}^{k} E_{ij} x_{ij} \leq D_{\omega(k)} \quad 1 \leq j \leq N, \ 1 \leq k \leq M$ (3)
- *Communication load constraint:* The total amount of transmitted data between servers through a specific link DL_{kl} should not exceed the maximum allowed load of that link. To formulate this constraint, the load of each link should be computed first. It is computed as follows:
  $\pi_{kl}(X) = \sum_{\forall z, p_z \in \varphi_k} \sum_{\forall r, p_r \in \varphi_l} \sum_{i=1}^{M-1} \sum_{j=i+1}^{M} x_{iz} x_{jr} \, CR_{ij}$ (4)
  Thus, the communication load constraint is formulated as:
  $\pi_{kl}(X) \leq \theta_{kl} \quad 1 \leq k < l \leq A$ (5)
- *No task redundancy:* This paper considers a model where a task must be allocated to exactly one processor, i.e., no task redundancy. That is, the following equality must hold with each task *i:*
  $\sum_{j=1}^{N} x_{ij} = 1 \quad 1 \leq i < M$ (6)

C. Reliability Evaluation

We formulate the system reliability in four steps. First, the reliability of each processor (core) is considered. Based on the reliability of processors, host reliability is computed. Afterwards, the reliability of links between hosts will be formulated. Finally, by using host and link reliabilities, system reliability will be modeled.

1. Reliability of processor: The reliability of processing node p_j in time interval [0,t] can be modeled by:
   $R_{p_j}(t) = e^{-\int_0^t \lambda_j(\tau) d\tau}$ (7)
   where $\lambda_j(\tau)$ is processor hazard rate at time τ. Assuming a constant hazard rate, Eq. 7 reduces to:
   $R_{p_j}(t) = e^{-\lambda_j t}$ (8)
   As the total time for executing the tasks assigned to processor p_j by assignment X is $\sum_{i=1}^{M} E_{ij} x_{ij}$, the reliability of processor p_j can be expressed by
   $R_{p_j}(X) = e^{-\lambda_j \sum_{i=1}^{M} E_{ij} x_{ij}}$ (9)
2. Host reliability: for modeling host reliability, memory, hard disk, RAID controller and processor reliability of each host should be determined. Processor reliability was formulated by Eq. 9 and the reliability of other components of kth host can be computed as follows if we assume that they are statistically independent:
   $R_{H_k}(X) = e^{-\Gamma_k t_{c_k}(X)} \, . \, e^{-\Psi_k t_{c_k}(X)} \, . \, e^{-\zeta_k t_{c_k}(X)}$
   $= e^{-(\Gamma_k + \Psi_k + \zeta_k) t_{c_k}(X)}$ (10)
   where $t_{c_k}(X)$ is the required time to execute all of the assigned tasks to ith host for assignment X. The processing nodes of a server run tasks simultaneously. Hence, $t_{c_k}(X)$ is calculated as follows:
   $t_{c_k}(X) = \max_{\forall j, p \in \varphi_k} (\sum_{i=1}^{M} E_{ij} x_{ij})$ (11)
   The reliability of each host can be calculated by multiplying its processors and other components reliability, if we assume that they are independent. Therefore, the reliability of the kth host is:
   $R_{c_k}(X) = R_{H_k}(X) \, . \prod_{\forall j, p_j \in \varphi_k} R_{p_j}(X)$
   $= e^{-(\Gamma_k + \Psi_k + \zeta_k) \, t_{c_k}(X) - \sum_{\forall l, p_l \in \varphi_k} \sum_{i=1}^{M} E_{il} x_{il}}$ (12)

   Failure of the hosts can be supposed independent. Ergo, system reliability irrespective of failures of links can be modeled by simply multiplying the servers' reliabilities:
   $R_{s'}(X) = \prod_{k=1}^{A} e^{-(\Gamma_k + \Psi_k + \zeta_k) \, t_{c_k}(X) - \sum_{\forall l, p_l \in \varphi_k} \sum_{i=1}^{M} E_{il} x_{il}}$
   $= e^{-\sum_{k=1}^{A} ((\Gamma_k + \Psi_k + \zeta_k) \, t_{c_k}(X) + \sum_{\forall l \in \varphi_k} \sum_{i=1}^{M} E_{il} x_{il})}$ (13)
3. Reliability of Links: In a similar way, by assuming a hazard rate $\mu_{kl}(\tau)$ for link DL_{kl}, the reliability of DL_{kl} is:
   $R_{DL_{kl}}(t) = e^{-\int_0^t \mu_{kl}(\tau) \, d\tau}$ (14)
   Due to the constant hazard rate assumption, this relation is simplified as:
   $R_{DL_{kl}}(t) = e^{-\mu_{kl} t_{D_{kl}}(X)}$ (15)
   where $t_{D_{kl}}(X)$ is the duration in which DL_{kl} is used to transmit data between the tasks assigned to φ_k and φ_l and it is computed as follows:
   $t_{D_{kl}}(X) = \pi_{kl}(X) / W_{kl}$ (16)
   Accordingly, Eq. 16 is rewritten as:
   $R_{kl}(X) = e^{-\mu_{kl} t_{D_{kl}}(X)}$ (17)
   As we assume independent failure of different links, reliability of the system without considering the

failure of hosts can be computed by multiplying the reliability of the links:

$$R_{s''}(X) = \prod_{k=1}^{A-1} \prod_{l=k+1}^{A} e^{-\mu_{kl} t_{D_{kl}}(X)} = \mathrm{e}^{-\sum_{k=1}^{A-1} \sum_{l=k+1}^{A} \mu_{kl} t_{D_{kl}}(X)} \quad (18)$$

4. System reliability: Due to the independence of host and link failures, system reliability can be formulated as:

   $$R_s(X) = R_{s'}(X) \times R_{s''}(X) \quad (19)$$

   Using (13), (17), and (19), $R_s(X)$ can be calculated by

$$R_s(X) = e^{-\left(\sum_{k=1}^{A}((\Gamma_k+\Psi_k+\zeta_k)\, t_{c_k}(X) + \sum_{\forall l\in\varphi_k} \sum_{i=1}^{M} E_{il} x_{il}\right) + \sum_{k=1}^{A-1} \sum_{l=k+1}^{A} \mu_{kl} t_{D_{kl}}(X))} \quad (20)$$

D. Power Consumption Evaluation

The power consumption of a host consists of the power consumed by the CPU, memory, disk storage and network interfaces. It is shown by [19] that power consumption of the CPU dominates the overall power consumption of a host. Accordingly, power consumption of a host could be achieved by a linear model defined in Eq. 21 based on the CPU utilization.

$$P(u) = k * P_{max} + (1- k) * P_{max} * u \quad (21)$$

where $P(u)$ is the power consumption of the host when its CPU utilization is u, P_{max} is the maximum power of a fully utilized host and k is the fraction of power consumed by idle host. It is cost-effective to turn the host off when its utilization is equal to zero. However, we need a more accurate model to compute the power consumption in this situation. Indeed, the hosts still consume energy when they are off. It is observed by [20] that off-consumption, (i.e., the consumption of a plugged-host when it is off) is 15% of the idle consumption. Thus, off-consumption is also taken into consideration. Accordingly, Eq. 22 can model the power consumption of the *s*th host.

$$P_s(X) = \begin{cases} k_s * P_s^{max} + (1 - k_s) * P_s^{max} * u_s(X) & u > 0 \\ 0.15 * P_{idle}, & u = 0 \end{cases} \quad (22)$$

where $u_s(X)$ is the utilization of the *s*th host for assignment *X*, and it is computed by

$$u_s(X) = Min\{1, \frac{\sum_{i=1}^{M} \sum_{\forall j, p_j \in \varphi_k} x_{ij} E_{ij}}{SPn_i}\} \quad (23)$$

Accordingly, the total power consumption by the CCS for assignment *X* can be achieved by

$$TP(X) = \sum_{s=1}^{A} P_s(X) \quad (24)$$

Finally, because the CPU utilization is a function of time in the sense that it varies over time due to different assignments, the total energy consumption is calculated by

$$\Delta(X) = TP(X)\ .\ SP \quad (25)$$

As minimizing power consumption leads to a minimization of energy, we concentrate on Eq. 24.

E. Multi-Objective Optimization

According to the stated constraints and the reliability model, we can define our problem as an Integer Linear Programming (ILP) problem as follows:

Maximize $R_s(X)$ *and Minimize* $TP(X)$ (26)
Subject to (1),(2),(3),(5) and (6)

To incorporate this problem in our solution framework, it is more convenient that we integrate the constraints and objective function into a single cost function which is called total cost. In this way, the goal is only to minimize the total cost function. The violation of each constraint can be represented by a penalty function. The corresponding penalty functions for violation of memory, storage, QoS and communication load constraints are formulated in Eq. 27, 28, 29 and 30 respectively:

$$P_M = \sum_{k=1}^{A} Max(0,\ \sum_{\forall j\in\varphi_k} \sum_{i=1}^{M} mem_i\, x_{ij} - MEM_k) \quad (27)$$

$$P_S = \sum_{k=1}^{A} Max(0,\ \sum_{\forall j\in\varphi_k} \sum_{i=1}^{M} stg_i\, x_{ij} - STG_k) \quad (28)$$

$$P_Q = \sum_{p=1}^{N} \sum_{t=1}^{M} Max(0,\ \sum_{i=1}^{t} E_{ip} x_{ip} - D_{\omega(t)}) \quad (29)$$

$$P_C = \sum_{k=1}^{A-1} \sum_{l=k+1}^{A} Max(0,\ \pi_{kl}(X) - \theta_{kl}) \quad (30)$$

The QoS penalty function can be efficiently implemented using a dynamic programming approach in $O(NM)$. In addition, to take the reliability aspects into account, define a reliability cost function *C(X)* as follows

$$C(X) = \sum_{k=1}^{A}((\Gamma_k + \Psi_k + \zeta_k)\, t_{c_k}(X) + \sum_{\forall l\in\varphi_k} \sum_{i=1}^{M} E_{il} x_{il}) + \sum_{k=1}^{A-1} \sum_{l=k+1}^{A} \mu_{kl} t_{D_{kl}}(X) \quad (31)$$

From 20 and 31 we have $R_s(X) = e^{-C(X)}$. Therefore, maximizing the system reliability, $R_s(X)$, is equal to minimizing the reliability cost function, C(X). According to the definition of penalty functions and this cost function, we define the total cost of an assignment X as the weighted sum of reliability cost, total power consumption and all penalties:

$$TC(X) = \alpha.C(X) + \beta.TP(X) + \gamma_1.P_M + \gamma_2.P_s + \gamma_3.P_Q + \gamma_4.P_C \quad (32)$$

where coefficients α, β, γ_1, γ_2, γ_3 and γ_4 are used to show importance of each function. In fact, they should be selected in such a way that solving the above-mentioned problem is equal to minimizing $C(X)$ and $TP(X)$ while all constraints are met. In other words, they should guide the search towards valid solutions and away from invalid ones. Because of having the same importance of penalty functions in our model (since none of them can be violated), we assume that the penalty coefficients are equivalent. Thus, we replace γ_1, γ_2, γ_3 and γ_4 with a common value, τ, which yields:

$$TC(X) = \alpha.C(X) + \beta.TP(X) + \tau(P_M + P_s + P_Q + P_C) \quad (33)$$

Accordingly, the main goal is to minimize the total cost function. It is worth noting that another option to determine the penalty coefficients is to pay attention to the level of importance of each corresponding penalty function. For example, in a soft real-time system, where missing a low number of deadlines may be tolerable, γ_3 may be set to a lower value which results in a reduction of its importance in comparison to other constraints.

V. Solution Approach

In this section, we propose an online task allocation algorithm which is inspired by imperialistic competition. This section starts by introducing the algorithm and demonstrating how it can be applied to solve the problem.

A. Imperialist Competitive Algorithm

ICA was originally proposed from the work of Atashpaz-Gargari and Lucas [2] to solve continuous optimization problems. ICA is a socio-politically inspired optimization strategy which starts by an initial population similar to many other evolutionary algorithms. Population individuals called country are divided into two groups: colonies and imperialists. Imperialists are selected from the best countries (i.e. the lowest cost countries) and the remaining countries form the colonies. All the colonies of the initial population are divided among the imperialists based on their power. The power of an imperialist is inversely proportional to its cost. Therefore imperialists with lower costs (i.e. higher powers) will achieve more colonies. Each imperialist along with its colonies form an empire. The total cost of an empire is determined by the cost of its imperialist along with the cost of its colonies. This fact is modeled by the following equation.

$$TC_n = Cost(imperialist_n) + \varepsilon.mean\{Cost(colonies\ of\ empire_n)\} \tag{34}$$

where TC_n is the total cost of the *n*th empire and ε is the colonies impact rate which is a positive number between zero and one. Increasing ε will increase the role of the colonies in determining the total power of an empire. The competition among imperialists forms the basis of the algorithm. During the competition, weak empires collapse and the most powerful ones remain. This process continues until the stopping condition is met.

After the initialization of the countries, all of them are improved by a fast local search. The next step in the algorithm is moving colonies to their relevant imperialists. The movement is a simple assimilation policy which is modeled by a directed vector from a colony to the corresponding imperialist. If assimilation causes any colony to have lower cost of than that of its imperialist, they will change their positions. Then the revolution process begins between the empires. This is modeled by doing a simple local search to improve the imperialists.

In the imperialistic competition, the weakest colony of the weakest empire will be exchanged from its current empire to another empire with the most likelihood to possess it. The imperialist competition will gradually result in an increase in the power of the powerful empires and a decrease in the power of the weak ones. Any empire that cannot succeed in the competition to increase its power will ultimately collapse.

The final step in the algorithm is global war. If the best imperialist in the imperialistic competition did not get any better after a certain iteration time, the global condition is satisfied. This way a new empire will be formed with the same random amount of the initial population as in the initialization step. Then the best empires from the new existing empires will be selected and the algorithm repeats again. Global war can efficiently lead to escape from local optima. The algorithm stops when the stopping condition is satisfied. It can be simply defined as the time when only one empire is left.

ICA is used to find a near optimal solution for the problem which is modeled in Section IV and V under the following structures:

- Initial solution: It can be generated randomly or by using a heuristic method. ICA uses a Simple Heuristic (SH) algorithm to generate an initial solution. *SH* works as follows: We know that if the tasks with higher execution times are assigned to the nodes with lower hazard rates then the reliability will be improved. We relax deadline and communication load constraints and just memory and storage are taken into account. SH sorts the tasks with respect to their execution time in descending manner and it also sorts the nodes with respect to their hazard rates in ascending manner. Then the first task (the task with the longest execution time) is assigned to the first feasible node (the node with lowest hazard rate which has enough free space). After assigning the first task, the algorithm tries to assign the second task to the best feasible node. *SH* continues until all of the tasks have been assigned. Then a fast local search is applied to the solution. The pseudo code for the fast local search is given in Alg. I. The main loop repeats *M*δ* times and each time *N-1* neighbors are checked. To investigate each neighbor the *TC* function should be executed.
- Solution Representation: we represent each solution with a vector of *M* elements, and each element is an integer value between one and *N*. Fig. 2 shows an illustrative example for a solution. The third element of this example is two, which means that the third task is assigned to the second core. Furthermore, this representation causes satisfaction of the *no redundancy* constraint in the sense that each task should be assigned to no more than one core. Formally, $T_i = k$ implies that $x_{ik} = 1$ and otherwise $x_{ik} = 0$.

T_1	T_2	T_3		T_M
1	1	2	...	1

Figure 2. Solution representation

- Assimilation: It is modeled by choosing random tasks from the vectors of colonies (the number of tasks to be chosen is given by the factor α_{as}) and changing their assigned nodes to their corresponding values in the imperialist vector.
- Revolution: A local search is applied to the imperialists' vectors.
- Global War: If the best imperialists did not get any better after a specific number of iterations, global war is performed. A new population is generated randomly and the empires are formed. Then the worst existing empires are replaced by the best new empires and a new world is created.
- Stopping condition: The algorithm terminates after a predetermined number of global wars.

ALGORITHM I. LOCAL SEARCH

```
Input: initial solution S, coefficient δ   Output: S_l
S_l = S
i = ⌊δM⌋
Select i random tasks; //R_0,R_1 ,…, R_i
for each random task R_j
    V= S[R_j]; //the node that task R_j is assigned to it.
    Select the best neighbor of S where task R_j is not assigned to
node V and this task of S is called S_n;
    if (TC (S_n) < TC(S_l))
        S_l = S_n;
    end if
end for
```

The pseudo code of the local search and the ICA are provided in Alg. I and II respectively. Furthermore, the initial values of ICA are determined in Tab. I.

B. *Online Scheduling Algorithm*

The online scheduler is called to allocate the services that have waited in the execution queue at the start of every scheduling period. Furthermore, unfinished tasks which are remaining from previous periods may be migrated from the current host to another based on the scheduler decision. The migrated tasks can continue their executions on the new hosts. As a result, the algorithm allocates new incoming services in each period besides a potential reallocation of older tasks that have not finished their executions yet. It should be mentioned that the migration overhead is ignored in this research similar to [14]. Because, it can be imagined infinitesimal for live migration supported by pre-copying task onto the destination host before starting its execution.

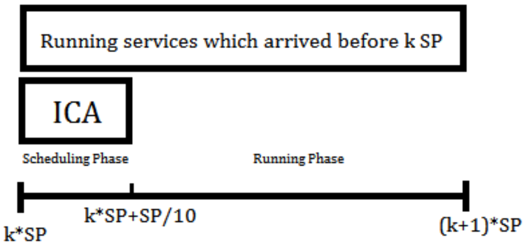

Figure 3. An scheduling period scheme

New incoming services are examined in terms of meeting deadlines before we put them in the execution queue. If the execution time of the longest task of the incoming service on the fastest processor, plus the remaining time until the next scheduling invocation, is greater than the service deadline, then the service is marked as a *strict* service. Furthermore, to expedite the scheduler and to make it more applicable as an online scheduler in real-world systems, a new stopping condition is considered. The algorithm is terminated once its execution time becomes more than ϑ% of the scheduling period. Therefore, the overhead of scheduling is strongly dependent on both the scheduling period and ϑ. On the other hand, limiting execution time of ICA may lead to a decrease in the solution quality or it may even not be able to satisfy all the constraints for heavy workloads. Fortunately, as it is seen in Section V, by selecting appropriate values for ϑ and *SP*, violating constraints occurs only in the few experiments and it is less than 3% for all experiments.

The scheduler can run with other services concurrently. Fig. 3 illustrates the kth scheduling period in the system. Each period is divided into two phases. In the former, ICA is run while the hosts are running the services which arrived in the *(k-2)*th period or earlier. ICA are assigned the services which arrived in the *(k-1)*th period to the hosts.

ALGORITHM II. ICA

```
1   Initialize the empires by SH heuristic;
2   Move the colonies towards their empires (Assimilation);
3   Randomly change characteristics of some countries (Revolution);
4   if there is a colony which TC_col < TC_imp then
5       Exchange the positions of that imperialist and colony;
6   end if
7   Compute the total cost of all empires (TC_emp);
8   Pick the weakest colony from the weakest empire and give to the
9   empire that has the most likelihood to possess it;
10  if there is an empire with no colonies then
11      Eliminate this empire;
12  end if
13  if there is only one empire then
14      Stop condition satisfied;
15  else
16      go to 2;
17  end if
```

Table I. Initial Values for ICA parameters

Algorithm	Description	Initial
N_{pop}	Initial Population	30
δ	Local Search coefficient	0.5
N_{imp}	Number of imperialists	5
N_{col}	Number of colonies	25
α_{as}	Assimilation factor	0.3
ε	Colonies impact rate	0.25
Z	Iterations before global war	30
I_{gw}	Global war flag	0

If a strict service arrives in this phase, it will be dropped and is not put in the execution queue. After finishing the first phase, the Running phase is started in which the scheduler allocates all tasks of the strict services to the feasible hosts as soon as they arrive. A feasible host is a one which has adequate available resources to execute the incoming service besides running other services which have been already allocated to it. Moreover, in both phases if a normal service arrives, it will be added to the execution queue. Consequently, the maximum waiting time for normal services is $SP + SP/10$. Additionally, strict services will certainly meet their deadlines or they will be dropped when they arrive in the system.

VI. Performance Evaluation

To evaluate the performance of the proposed algorithm based on real world data, we consider a cloud computing system composed of some common hosts in data centers similar to [25]. These servers are categorized into four different types. Specification of each kind is outlined at Tab. II. To extract the Power Consumption column, SPECpower [21] is deployed as reference. We also consider available storage for the hosts as the Disk column at the table.

In the problem, each host includes a certain amount of memory, storage and computing capacity. To provide computing power, each host uses a multicore processor. It should be mentioned that the hazard rates of the components are based on [13] in which 100,000 servers are examined for a 14 month period. Hazard rates of components and other system configuration are stated at Tab. III. The application specification is also expressed at Tab. IV.

As we mentioned before, we turn the host off if the CPU utilization of the host equals to zero in order to prevent negative impact of idle consumption. In this situation, the host consumes off-consumption which is 15% of its P_{idle}. As shown in the Tab. II various hosts consumed a different amount of energy at the idle state and thus making the decision about which host should be off is also a considerable issue which affect the total energy consumption.

Table II. Specification of the Hosts

Host	**Processor**			**Disk (GB)**	**Memory (GB)**	**Power Consumption**	
	CPU	MHz	Cores			@active idle	@100% utilization
1	Intel Xeon 7020	2660	8	800	16	520	833
2	Intel Xeon 7110	2600	8	800	16	575	732
3	Intel Xeon 3040	1860	2	500	4	86	117
4	Intel Xeon 3075	2660	2	500	4	93.7	135

In order to assess of the proposed algorithm, we implement ICA and express the results at Tab. V. In addition, as a solution based on the genetic algorithm has been recently applied in the literature to minimize energy consumption [17], ICA is compared with GA. Our GA solution is also modified in such a way that it considers system reliability along with energy consumption in order to comprehensive evaluation of the proposed algorithm. For each problem size, both algorithms run 20 times to reach 95% confidence interval. We suppose that the arrival rate of services follows a Poisson process and its parameter is denoted by the Service Arrival Rate per scheduling period in the Tab. V. We also consider 0.2 as the rate of strict service arrival in each period for all simulations. Reliability, energy and total cost are computed towards each set of hosts and the different service arrival rates for both algorithms. α and β are two coefficients which adjust the influence of reliability and energy respectively. Therefore the algorithm strive to maximize reliability irrespective of energy while α=1 and β=0. In order to achieve a fair trade off between reliability and energy, we should set the value of α and β to 0.5 and finally for considering energy without taking reliability into account, α and β are set to zero and one respectively.

Table III. System parameters and the corresponding value ranges

System Parameters	**Description**	**Value Ranges**
Γ_i	Hard Disk Hazard Rate	[100, 200]
ζ_i	Memory Hazard Rate	[100, 200]
Ψ_i	RAID Controller Hazard Rate	[0.00005, 0.00010]
λ_i	Core Hazard Rate	[0.0001, 0.0005]
CHR	Communication link Hazard Rate	[0.00015, 0.00030]
CBW	Link Bandwidth	[1, 4]
θ	Maximum link communication load	[20,50]

The results in Tab. V indicate that in terms of solution quality, the Imperialist-based algorithm works significantly better than the genetic algorithm. The energy consumption of ICA is reduced by 17% and system unreliability by 9% in comparison with GA. In addition, the results show that when reliability and energy are considered simultaneously (i.e., α=β=0.5), energy consumption is decreased by 83% in comparison with the none power-aware policy in which the hosts are not turned-off and consume maximum power.

Table IV. Application parameters and the corresponding value ranges

Parameters	**Description**	**Value Ranges**
E	Task Execution Time (sec)	[1, 15]
mem	Task Memory (MB)	[5, 50]
CR	Amount of Communication Data	[0.5, 2]
D	Service Deadline (sec)	[5, 75]
TID	Task Interaction Density	0.5
m	Number of Tasks in each Service	[4, 8]

Table V. Simulation Results

	Hardware Configuration		Service Configuration		Imperialist			Genetic Algorithm		
	# of hosts	# of cores	Service Arrival Rate (Per Period)	# of tasks	Reliability	Energy	Total cost	Reliability	Energy	Total Cost
α = 1 β = 0	12	60	4	20	0.99865	0.61164	0.00134	0.96978	0.71633	0.07961
	12	60	7	30	0.99461	0.85875	0.00538	0.89374	0.76328	0.69522
	12	60	8	40	0.98531	0.90409	0.01478	0.83909	0.81455	3.8884
	16	80	5	20	0.99881	0.44202	0.00118	0.98455	0.47783	0.06217
	16	80	6	30	0.99545	0.69307	0.00454	0.79068	0.66171	0.52066
	16	80	9	40	0.98774	0.85394	0.01232	0.75506	0.77033	1.4418
	20	100	4	20	0.99888	0.42475	0.00110	0.99518	0.44234	0.06144
	20	100	6	30	0.99596	0.5853	0.00403	0.83068	0.5732	0.43147
	20	100	8	40	0.98935	0.70277	0.01069	0.77739	0.67701	1.19941
α = 0.5 β = 0.5	12	60	4	20	0.99292	0.15448	0.08078	0.95531	0.17393	0.22281
	12	60	7	30	0.97066	0.19009	0.10993	0.87777	0.54773	0.72739
	12	60	8	40	0.93400	0.29057	0.17948	0.79101	0.67446	0.79544
	16	80	5	20	0.99449	0.14234	0.07392	0.96275	0.06731	0.18735
	16	80	6	30	0.97723	0.17116	0.09709	0.85368	0.32614	0.55302
	16	80	9	40	0.93625	0.19098	0.12842	0.78927	0.61829	1.05423
	20	100	4	20	0.99496	0.13463	0.06984	0.97581	0.05032	0.13241
	20	100	6	30	0.98015	0.08798	0.08798	0.87694	0.25143	0.45164
	20	100	8	40	0.95126	0.19942	0.12410	0.79762	0.45272	0.96161
α = 0 β = 1	12	60	4	20	0.98163	0.15274	0.15274	0.91387	0.07935	0.07935
	12	60	7	30	0.90289	0.17146	0.17146	0.85544	0.37014	0.37014
	12	60	8	40	0.84761	0.19080	0.19080	0.78411	0.67461	0.67461
	16	80	5	20	0.98228	0.14085	0.14085	0.96464	0.05124	0.05124
	16	80	6	30	0.94651	0.15397	0.15397	0.79038	0.17051	0.17051
	16	80	9	40	0.74391	0.16789	0.16789	0.73202	0.47911	0.47911
	20	100	4	20	0.96860	0.13363	0.13363	0.95662	0.06009	0.06009
	20	100	6	30	0.93179	0.14404	0.14404	0.89445	0.09009	0.09009
	20	100	8	40	0.86475	0.15442	0.15442	0.79101	0.25471	0.25471

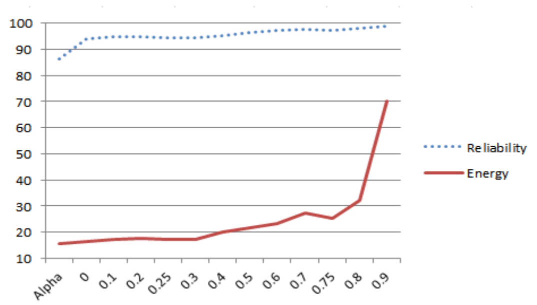

Figure 4. Reliability and Energy Consumption Percentage

The simulation results presented in Fig. 4 show the percentage of reliability and energy consumption of our proposed algorithm (ICA) for 20 hosts and 8 services. The results imply that by increasing reliability, more energy is consumed. This fact becomes crucial when the maximum reliability is desirable. Although ICA helps to keep energy consumption less than 32% of its maximum in most cases, it is noticeable that energy consumption increases by 40% when reliability grows up to 0.98. Consequently, system designers can choose the appropriate α value to reach the acceptable level of reliability which is stated at the SLA besides minimizing the energy consumption. In other words, they will be able to present negotiated QoS with the lowest cost based on the outcome of this research.

VII. CONCLUSION

Cloud providers supply a set of services to their end-users. Allocating the services on hosts of a CCS may result in different system reliability and energy consumption. In the paper, we proposed a mathematical model to evaluate reliability based on a proper service allocation approach. Processor, memory, hard disk, RAID controller and communication link failures were taken into consideration in order to present a holistic reliability model which is only based on a fully connected mesh topology. Furthermore, an energy model was also proposed. This model considers host energy consumption with respect to its CPU utilization. Off-consumption is also considered in order to present a comprehensive model. Moreover, service deadlines as QoS besides other resource constraints were formulated elaborately. By integrating the reliability model, the energy model and the system constraints, an ILP solution was suggested. Subsequently, an efficient online resource scheduler is introduced to find a solution with the right compromise between reliability and energy consumption while satisfying QoS and other system constraints. The proposed solution was compared with an alternative solution using genetic algorithms for a wide range of problem sizes. The simulation results indicate that the energy consumption of ICA is reduced by 17% and system unreliability by 9%. Also, results manifest that energy consumption is decreased by 83% in comparison with a none power-aware policy.

REFERENCES

[1] G. Gruman and E. Knorr, "What Cloud Computing really means," Technical report, Info World Inc., 2008.

[2] E. Atashpaz-Gargari and C. Lucas, "Imperialist Competitive Algorithm: An algorithm for optimization inspired by imperialistic competition," IEEE Congress on Evolutionary Computation, 2007.

[3] K. Church, A. Greenberg, and J. Hamilton, "On Delivering Embarrassingly Distributed Cloud Services," 2008.

[4] L. Pamies-Juarez, P. García-López, M. Sánchez-Artigas, and B. Herrera, "Towards the design of optimal data redundancy schemes for heterogeneous cloud storage infrastructures," Computer Networks, vol. 55, no. 5, pp. 1100–1113, Apr. 2011.

[5] L. Ai, M. Tang, and C. Fidge, "QoS-oriented Resource Allocation and Scheduling of Multiple Composite Web Services in a Hybrid Cloud Using a Random-Key Genetic Algorithm," vol. 12, no. 1, pp. 29–34, 2010.

[6] Q. Li, "Optimization of Resource Scheduling in Cloud Computing," 2010 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, no. 1, pp. 315–320, Sep. 2010.

[7] Y. Dai, B. Yang, J. Dongarra, and G. Zhang, "Cloud Service Reliability : Modeling and Analysis," 15th IEEE Pacific Rim International Symposium on Dependable Computing. 2009.

[8] Y. Lin and P. Chang, "Evaluation of System Reliability for a Cloud Computing System with Imperfect Nodes," pp. 83–94, 2011.

[9] B. Cao, B. Li, and Q. Xia, "A Service-Oriented Qos-Assured and Multi-Agent Cloud Computing Architecture," pp. 644–649, 2009.

[10] D. Armstrong and K. Djemame, "Towards Quality of Service in the Cloud," in Proc. of the 25th UK Performance Engineering Workshop, 2009.

[11] J. W. S. W. Liu, Real-time systems. Prentice Hall PTR, 2000.

[12] S. M. Shatz, J.-P. Wang, and M. Goto, "Task allocation for maximizing reliability of distributed computer systems," Computers, IEEE Transactions on, vol. 41, no. 9, pp. 1156–1168, 1992.

[13] K. V. Vishwanath, N. Nagappan, "Characterizing cloud computing hardware reliability", ACM Symposium on Cloud Computing, USA, 2010.

[14] R. Buyya, A. Beloglazov, and J. Abawajy, "Energy-Efficient Management of Data Center Resources for Cloud Computing : A Vision, Architectural Elements, and Open Challenges," Proc. of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, USA, July 12-15, 2010.

[15] A. Beloglazov, Y. C. Lee, and A. Zomaya, "A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems," vol. 82., 2010.

[16] Y. C. Lee and A. Y. Zomaya, "Energy efficient utilization of resources in cloud computing systems," The Journal of Supercomputing, vol. 60, no. 2, pp. 268–280, Mar. 2010.

[17] M. Mezmaz, N. Melab, Y. Kessaci, Y. C. Lee, E.-G. Talbi, a. Y. Zomaya, and D. Tuyttens, "A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems," Journal of Parallel and Distributed Computing, vol. 71, no. 11, pp. 1497–1508, Nov. 2011.

[18] Young Choon Lee; Zomaya, A.Y., "Energy Conscious Scheduling for Distributed Computing Systems under Different Operating Conditions," Parallel and Distributed Systems, IEEE Transactions on , vol.22, no.8, pp.1374,1381, Aug. 2011.

[19] Fan X, Weber WD, Barroso LA. Power provisioning for a warehouse-sized computer. Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA), ACM New York, NY, USA, 2007; 13– 23, 2007.

[20] A. Orgerie, L. Lefevre, and J. Gelas, "Demystifying energy consumption in Grids and Clouds," Green Computing …, 2010.

[21] The SPECpower benchmark, http://www.spec.org/power_ssj2008/

[22] K. H. Kim, R. Buyya, and J. Kim, "Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-enabled Clusters," Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07), pp. 541–548, May 2007.

[23] B. Rountree, D.K. Lowenthal, S. Funk, V.W. Freeh, B.R. de Supinski, M. Schulz, Bounding energy consumption in large-scale MPI programs, in: Proceedings of the ACM/IEEE Conference on Supercomputing, , pp. 1–9, November 2007.

[24] X. Zhong, S. Member, C. Xu, and S. Member, "Energy-Aware Modeling and Scheduling for Dynamic Voltage Scaling with Statistical Real-Time Guarantee," vol. 56, no. 3, pp. 358–372, 2007.

[25] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers," Concurrency and Computation: Practice and Experience, vol. 24, no. 13, pp. 1397–1420, Sep. 2012.

[26] H. R. Faragardi, R. Shojaee, and N. Yazdani, "Reliability-Aware Task Allocation in Distributed Computing Systems using Hybrid Simulated Annealing and Tabu Search," in 14th IEEE International Conference on High Performance Computing and Communications, 2012, pp. 1088–1095.

[27] R. Shojaee, H. R. Faragardi, S. Alaee, and N. Yazdani, "A New Cat Swarm Optimization based Algorithm for Reliability-Oriented Task Allocation in Distributed Systems," in Sixth International Symposium on Telecommunications (IST), 2012, pp. 861–866.

[28] H. R. Faragardi, R. Shojaee, M. Mirzazad-Barijough, and R. Nosrati, "Allocation of hard real-time periodic tasks for reliability maximization in distributed systems," in Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on, 2012, pp. 42–49.

[29] H. R. Faragardi, R. Shojaee, M. A. Keshtkar, and H. Tabani, "Optimal Task Allocation for Maximizing Reliability in Distributed Real-time Systems," in Computer and Information Science (ICIS), 2013 IEEE/ACIS 12th International Conference on, 2013.

[30] H. R. Faragardi, R. Shojaee, H. Tabani, and A. Rajabi, "An Analytical Model to Evaluate Reliability of Cloud Computing Systems in the Presence of QoS Requirements," in Computer and Information Science (ICIS), 2013 IEEE/ACIS 12th International Conference on, 2013.

[31] R. Shojaee, H.R. Faragardi and N. Yazdani. "From Reliable Distributed System toward Reliable Cloud by Cat Swarm Optimization", International Journal of Information and Communication, accepted in August 2013.

[32] A. Rajabi, H.R. Faragardi and N. Yazdani. "Communication-aware and Energy-efficient Resource Provisioning for Real-Time Cloud Services", 17th CSI Symposium on Computer Architecture & Digital Systems (CADS), Iran, 2013.

[33] A. Rajabi, V. Ebrahimirad, N. Yazdani. "Decision Support-as-a-Service: An Energy-aware Decision Support Service in Cloud Computing" 5th International Conference on Information and Knowledge Technology (IKT), Iran, 2013.