Consolidation of Multi-Tier Workloads with Performance and Reliability Constraints

Andrea Sansottera, Davide Zoni, Paolo Cremonesi and William Fornaciari
Dipartimento di Elettronica e Informazione Politecnico di Milano
via Ponzio 34/5, 20133 Milano, ITALY
Email: {sansottera,zoni,cremones,fornacia}@elet.polimi.it

***Abstract*—Server consolidation leverages hardware virtualization to reduce the operational cost of data centers through the intelligent placement of existing workloads. This work proposes a consolidation model that considers power, performance and reliability aspects simultaneously. There are two main innovative contributions in the model, focused on performance and reliability requirements. The first contribution is the possibility to guarantee average response time constraints for multi-tier workloads. The second contribution is the possibility to model active/active clusters of servers, with enough spare capacity on the fail-over servers to manage the load of the failed ones. At the heart of the proposal is a non-linear optimization model that has been linearized using two different exact techniques. Moreover, a heuristic method that allows for the fast computation of near optimal solutions has been developed and validated.**

I. Introduction

Traditionally, data centers run workloads on separate servers due to compatibility, reliability and security reasons. Since many workloads need only a fraction of the resources provided by modern servers, capacity is severely over-provisioned. In response to this situation, *server consolidation*, i.e. the practice of mapping a set of existing workloads on a set of servers in order to achieve optimality with respect to some criterion, while satisfying a number of requirements, has recently attracted the attention of both the academic community and large enterprises such as Google, which recently proposed a server consolidation problem for the ROADEF/EURO Challenge[1].

From a technological point of view, hardware virtualization, which is now a mature and powerful technology even on commodity x64 servers [1], allows to run multiple workloads on the same server satisfying compatibility, reliability and security requirements with only a small performance overhead. However, deciding the optimal workload placement is still a complicated problem and requires efforts in both problem formulation and in the development of algorithms to solve the problem. In fact, even consolidation problems with rather simple constraints have been shown to be NP-hard [19].

From a modeling point of view, server consolidation must account for the performance and reliability requirements of today data centers. In particular, service level agreements often require probabilistic guarantees on the response time of complex multi-tier applications, forcing consolidation solution to provide enough spare capacity on the servers [3]. Moreover, several workloads are deployed in high-availability clusters and need to be consolidated on separate physical servers [11], [5]. Hence, consolidation must be tightly controlled through appropriate modeling, because solving the over-provisioning problem should not lead costly service level agreement violations or reduced availability of critical workloads. While reducing the number of active servers, power consumption characteristics of the servers must be taken into account, for two reasons: (1) data centers are often populated by hardware from different generations and hence with widely different energy efficiency; (2) power efficiency strongly depends on resource utilization as shown by the publicly available results of the SPEC power_ssj2008 benchmark[2].

A. Main Contributions

In this paper, we present a new model for server consolidation that aims at minimizing server power consumption while satisfying both performance and reliability constraints. The main contributions of this work can be summarized as follow.

- *Multi-tier response time constraints*. Our model provides bounds on the average response time of multi-tier workloads – for up to two tiers – executed on multi-core architectures, modeling each server as a M/M/n queue with processor sharing. The natural formulation of these response time constraints is non-linear and required the adoption of linearization techniques to make the problem tractable by efficient solvers. The generalization to more than two tiers is trivial but computationally challenging and is therefore left as future work. Previous works have only solved such problems with loose response time approximations and M/M/1 queuing stations [3], which are not representative of modern multi-core systems.
- *Fault-tolerance constraints*. In active/active clusters, in the event of a server failure, the workload can be still processed by the surviving server in the cluster, but the load on this server increases [6]. We guarantee that instances of workloads running on such clusters are consolidated on separate physical servers and, differently from previous works [11], [5], we also ensure that, in the event of a single server failure, the remaining servers have enough spare capacity to handle the requests of the workloads previously running on the failed server. In other words, we guarantee stability conditions for the M/M/n queues

[1]http://challenge.roadef.org

[2]http://www.spec.org/power_ssj2008/

even in the event of a single server failure. Hence, we are the first to develop a model which considers the connection between performance and reliability constraints. As for the response time constraints, the formulation of fault-tolerance constraints required the application of linearization techniques.
- *Power optimization, performance and reliability*. To the best our knowledge, this is the first consolidation model which jointly considers power minimization as well as both response time and fault-tolerance constraints.

The problem is first formulated as mixed-integer non-linear program (MINLP) and then transformed into an equivalent mixed-integer linear program (MIP), using two different methods. Moreover, we provide an efficient heuristic for the fast computation of near-optimal solutions. An extensive comparison with the results of a state-of-the-art exact solver shows that the heuristic method is always able to find feasible high-quality solutions, approaching the global optimum in few seconds.

B. Structure of the paper

In Section II, we briefly cover the state of the art for server consolidation. Some necessary definitions and assumptions are given in Section III. The mathematical programming formulation of the problem is presented in Section IV and improved in Section V. The heuristic algorithm is addressed in Section VI. Section VII describes how the test instances were generated and Section VIII discusses the experimental results. Final remarks are drawn in Section IX.

II. Related work

In its simplest form, the server consolidation problem consists in finding the placement of workloads to server that minimizes the cost associated with server operation, subject to capacity constraints, considering the amount of resources consumed by workloads on each server. In [19], multiple resources, such as CPU, disk and network bandwidth, are considered for each server. This model is related to the multi-dimensional bin-packing problem [9].

Even if the main goal of server consolidation is cost saving, contractual service level agreements (SLA) represent another important aspect to be taken under control. Performance constraints on the response time distribution and availability requirements often have to be satisfied, modeling the impact of a server failure in a consolidated data center. In [3], Anselmi et al. present a server consolidation model considering end-to-end response time constraints for multitiered workloads. These constraints are expressed in a combinatorial optimization problem using queuing theory. In particular, the work presents a model with nonlinear constraints for the end-to-end response time of multi-tier applications and a solution technique based on an upper-bound heuristic approach. With respect to [3], we propose an exact linear optimization problem considering multi tier applications with response time constraints. Moreover our proposal can manage fault tolerance.

In [5], Dhyani et al. aim at minimizing the server operating costs subject to utilization constraints on multiple resources, considering workload seasonality as in [17]. The model is complicated by a large number of rule-based constraints. These constraints take different forms and can be used to enforce compatibility and reliability requirements. In particular, the so called *candidate-candidate exclusion* (CCE) constraints, which require that workloads in a given set are placed on separate physical servers, are introduced to achieve fault-tolerance. In fact, a single workload can be represented by two or more identical workload instances and CCE constraints can be added to ensure that the workload is still available even if one of the servers fails.
Our approach differs with [5] in two main aspects. First, [5] does not consider response time constraints while our model does. Second, we argue that the approach based on CCE constraints is overly conservative, since it dedicates twice the required capacity to any fault tolerant workload. If two instances are required for each workload, the overall consumed capacity, costs and power consumption of the data centers are effectively doubled. To understand this argument, consider a simple scenario with two workloads and four identical servers. The two workloads require 60% and 50% of a server capacity, respectively. Both workloads require fault-tolerance. Without server consolidation, the only possible choice is to run two instances of each workload on separate servers, using a total of four servers. Using the approach based on CCE, we still require all the four servers, since the combination of the two workload instances exceed the capacity of a single server (110%). However, consider workloads that can be load-balanced among multiple computers (such as web servers and application servers). If two instances of such a workload are deployed on separate servers, not only the second instance acts as a fail-over, but it is also used during normal operating conditions, i.e. when all the servers are operating correctly. We can model such a situation using the approach based on CCE, splitting the workload into two instances which only require half of the total capacity (30% and 25% for the two workloads, respectively). This can lead to a very compact consolidation solution that only uses two servers, as shown in Table I. This solution works just fine under normal operating conditions, but whenever one of the two server fails, all requests are routed to the other servers and its utilized capacity exceeds 100%, leading to instability (queue lengths grow until requests are dropped). The model which we propose in this paper does not suffer from this issue and can provide a more compact solution than the standard, overly conservative approach. In the example under consideration, our method leads to a solution that only uses three server, as shown in Table II. We observe that the worst case occurs when the first server fails and the utilization of the second servers grows to 85%. Our model, however, is difficult to solve because utilization constraints are nonlinear. For simplicity but without loss of generality, in this paper we consider a single resource.

In [19], Speitkamp et al. propose a classic server consolidation model that is enhanced with a set of specific constraints that are used to enforce compatibility and reliability requirements. In contrast with [19], we explicitly model fault

TABLE I
TRADITIONAL CONSOLIDATION RESULTS (EXAMPLE). IN CASE OF FAILURE OF ONE SERVER, THE REMAINING HAS NO SPARE CAPACITY.

	Server1	Server2
Workload 1	30%	30%
Workload 2	25%	25%

TABLE II
OUR CONSOLIDATION APPROACH (EXAMPLE). IN CASE OF FAILURE OF ONE SERVER, THE CAPACITY OF THE OTHER SERVERS IS NOT EXCEEDED.

	Server1	Server2	Server3
Workload 1	30%	30%	0
Workload 2	0	25%	25%

tolerance and multi-tier workloads, formulating constraints on the response times under normal operating conditions. Moreover these constraints differ from those previously presented in literature (e.g., [3]) because we take into account the sum of response time of the different tiers. This introduces non-linearities in the response time constraints that must be linearized. Last, we aim at minimizing server power consumption, which accounts for a large part of the operating cost of a data center [2]. Hence, the solution provided by a bin packing approach, like that in [19], might not be the most power efficient.

III. DEFINITIONS AND ASSUMPTIONS

This section presents the problem and introduces the notation that will be used throughout the rest of this paper. Moreover, we describe the model used to estimate server power consumption and use queuing theory [13] to derive utilization and response time for a given workload placement. Section III-A details the basic notation and properties the model ensures for both servers and workloads, while Section III-B describes the power and performance models used in this work.

A. Servers and Workloads

We consider a base system without virtualization as a starting point for our problem formulation, containing a set of servers S and a set of workloads W. Our aim is to find the allocation of workloads to servers that minimizes the overall server power consumption while satisfying performance and reliability constraints. We consider two sets of workloads, single tier (W^{ST}) and multi-tier (W^{MT}), with $W = W^{\mathrm{ST}} \cup W^{\mathrm{MT}}$. Multi-tier workloads model client-server architectures in which the presentation, the application processing, and the data management are logically separate processes. In particular, this work considers only single-tier and two-tier workloads to assess our contribution. Generalization to more than two tiers is trivial but computationally challenging and is outside the scope of this paper. We model each workload in the system as a set of tiers, of cardinality one for single-tier workloads and of cardinality two for multi-tier workloads. Tiers can be either fault tolerant (T^{F}) or non fault tolerant (T^{NF}) and we let $T = T^{\mathrm{F}} \cup T^{\mathrm{NF}}$.

Considering single and multi-tier workloads with fault tolerant and non-fault tolerant tiers, we identify four different workload types, as depicted in Figure 1.

- Single-tier workloads without fault tolerance require a single server and are the set of workloads $w \in W^{\mathrm{ST}}$ such that $w = \{t\}$ with $t \in T^{\mathrm{NF}}$.
- Single-tier workloads with fault tolerance are the workloads $w \in W^{\mathrm{ST}}$ such that $w = \{t\}$ with $t \in T^{\mathrm{F}}$. These workloads run on two different servers in order to be available in the event a server failure.
- Multi-tier fault tolerant workloads are the workloads $w \in W^{\mathrm{MT}}$ composed of a couple of two fault tolerant tiers, i.e. $w = \{t_1, t_2\}$ with $t_1, t_2 \in T^{\mathrm{F}}$. Without sever consolidation, four servers are required to run the two fault tolerant tiers.
- Non fault tolerant multi-tier workloads are defined as a subset $w \in W^{\mathrm{MT}}$, where each workload is defined as $w = \{t_1, t_2\}, t_1, t_2 \in T^{\mathrm{NF}}$. Without server consolidation, the two tiers generally run on separate machines.

It is worth noting that, to enhance readability but without loss of generality, only a subset of the possible combinations of tiers have been defined for multi-tier workloads. In fact, we do not consider multi-tier workloads for which only one of the two tiers is fault tolerant.

When dealing with fault tolerant tiers, it is useful to use high availability (HA) clusters [6], i.e. distributed computing systems tightly coupled. The primary purpose of HA clusters is to provide uninterrupted access to the applications and their data when one of the server of the cluster fails. Considering two servers in HA clusters, it is possible to have active-passive and active-active configurations. In active-passive configuration one server performs a critical task while the other is a dedicated standby, ready node. On the other hand, in active-active configurations both servers are doing independent critical tasks and each host acts as a standby for the other. In this work we suppose an active-active configuration for each couple of servers that are managing fault tolerant tiers, forcing our system to manage each fault tolerant tiers $t \in T^{\mathrm{F}}$ on two distinct servers for reliability issues. This is necessary to face hardware fault, i.e. when a server fails the workload with fault tolerance capability must have at least a running virtual machine for each of its tiers.

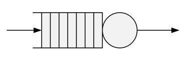
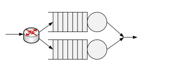
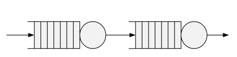
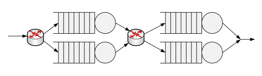
	Non Fault Tolerant (NF)	Fault Tolerant (F)
Single-Tier (ST)		
Multi-Tier (MT)		

Fig. 1. Workloads can have one or two tiers, which can be either fault tolerant or not.

We have to consider two different situations. When all servers are operating correctly, we want to ensure that the average response time is below a certain threshold for any workload in the system. When a server fails all the request directed to fault tolerant workloads running on the failed server are redirected to the appropriate server, on which the other part of the workload is still running. In this case our model ensures that stability conditions for all servers are satisfied, i.e. that queue lengths do not grow indefinitely. We assume that at most one server might be unavailable at any time, which is a reasonable assumption if the mean time to repair is small (thanks, for instance, to the presence of spare servers and the use of virtualization to quickly migrate workloads from the failed server to the spare server).

B. Power and Performance models

In order to estimate the power consumption of a server, a number of models have been proposed, based either on simulations [20], [18], [12] or on resource utilization, like Mantis [7]. Similarly to Mantis, we adopt a linear power model based on CPU utilization. Each server is characterized by the parameters p_s^{idle}, the power drawn when no request is being processed, and p_s^{busy}, the increase in power consumption when the server has maximum utilization. The expected power consumption for the server is $p_s = p_s^{idle} + p_s^{busy} u_s$, where u_s is the server s load fraction. The model has been parameterized using available results from the SPECpower benchmark[3], which measures the power consumption of one or multiple servers under a Java Enterprise workload of varying intensity. The roughly linear relationship between utilization and power consumption is demonstrated by the sample correlation coefficients [16] that range from 0.9827 to 0.9992.

Workloads are described using a Jackson Network model [14], where each workload $w \in W$ is characterized by the arrival rate λ_w and an upper limit on the average response time defined as $r_w^{\max}$. Since we consider multi-tier workloads and all the model described in Section IV is expressed using tiers (t), we define $\lambda_{w(t)}$ as the arrival rate for the workload w identified by its tier t as well as $r_{w(t)}^{\max}$ represents the response time for workload w, where the workload is identified using its tier t. Moreover each tier (t) can be assigned to a different server, so we define, for each tier $t \in T$ and for each server $s \in S$, the average CPU time required to process one request on server s, i.e. the service demand, as $d_{t,s}$. In general, the service demands might vary depending on the server, due to different CPU architectures. Given n_s, the number of CPU cores available on server s, according to the utilization law [14], the utilization of server s under normal operating conditions is

$$u_s = \sum_{t \in T_s^{\mathrm{NF}}} \frac{\lambda_{w(t)} d_{t,s}}{n_s} + \sum_{t \in T_s^{\mathrm{F}}} \frac{\lambda_{w(t)} d_{t,s}}{2 n_s} \ , \qquad (1)$$

where $T_s^{\mathrm{NF}}, T_s^{\mathrm{F}}$ are, respectively, the set of non fault tolerant and fault tolerant tiers allocated on server s. The arrival rate of fault tolerant tiers is halved, since the requests are evenly distributed on two servers. When server k is unavailable, the requests of fault tolerant tiers allocated on the server pair (s,k) are redirected to server s. Hence, the utilization of server s is

$$u_s^{\mathrm{fault}(k)} = u_s + \sum_{t \in T_s^{\mathrm{F}} \bigcap T_k^{\mathrm{F}}} \frac{\lambda_{w(t)} d_{t,s}}{2 n_s} \ . \qquad (2)$$

Clearly, $u_s \le u_s^{\mathrm{fault}(k)}$ for any $(s,k) \in S^2$. In order to meet stability conditions even when a server is unavailable, we require that

$$u_s^{\mathrm{fault}(k)} \le 1 \quad \forall (s,k) \in S^2 \ , \qquad (3)$$

which means that, at any time, the maximum utilization is not exceeded.

We model the CPUs of each server as a multi-server queue. We rely on the common assumptions of Poisson arrivals and exponentially distributed service demands. According to Kendall's notation [13], we are modeling the CPUs as a M/M/n queue. The expected response time for this queue can be easily obtained by solving a Markov chain. However, the resulting formula does not lend itself to being used in a mathematical programming formulation of the problem, since it presents polynomials of degree $n+1$ in the utilization. Hence, it would yield non-linear constraints even without considering fault tolerance. Rather than using the exact formula, we adopt the approximation proposed in [15]. This approximation models the M/M/n queue as a delay and a n-times faster M/M/1 queue. The delay makes the approximation exact when the utilization approaches zero.

Response times under normal operation have different formulations for single tier and multi tier workloads. For a single tier workload w the expected response time is

$$r_w = d_{t,s} \frac{n_s - 1}{n_s} + \frac{d_{t,s}/n_s}{1 - u_s} \quad \forall w \in W^{\mathrm{ST}}, w = \{t\} \ , \qquad (4)$$

where $s \in S$ is the server on which the tier t is deployed. We assume that each instance of a fault workload must satisfy this constraints without considering the other instance of the same workload. This approach provides a tighter bound, with small variance, on response time with respect to a formulation where the response time constraint is expressed as the mean of the response time values of the two instances. Response time for multi-tier workloads under normal conditions is defined as:

$$r_w = \left(d_{t_1,s_1} \frac{n_{s_1} - 1}{n_{s_1}} + \frac{d_{t_1,s_1}/n_{s_1}}{1 - u_{s_1}} \right) + \left(d_{t_2,s_2} \frac{n_{s_2} - 1}{n_{s_2}} + \frac{d_{t_2,s_2}/n_{s_2}}{1 - u_{s_2}} \right) \qquad (5)$$

$$\forall w \in W^{\mathrm{MT}}, w = \{t_1, t_2\} \ , \qquad (6)$$

where $s_1, s_2 \in S$ are the servers on which tier t_1 and t_2 are deployed, respectively. In order to meet Service Level Agreements (SLA), we require that, under normal operation, the average response times for all workloads are below a certain threshold:

$$r_w \le r_w^{max} \quad \forall w \in W \ . \qquad (7)$$

[3]http://www.spec.org/power_ssj2008/

IV. Mathematical Programming Formulation

In this section we present a mathematical formulation of the problem stated in Section III, based on nonlinear binary integer programming.

There are two sets of decision variables. For each combination of workload tier and server $(t, s) \in T \times S$, the variable $x_{t,s} \in \{0, 1\}$ is equal to one if and only if workload tier t is placed on server s. For each server $s \in S$, the variable $y_s \in \{0, 1\}$ is equal to one if and only if server s is active.

Using the decision variables $x_{t,s}$, we rewrite (1) in linear programming notation, using the binary variables $x_{t,s}$ to consider only the workloads allocated on the server:

$$u_s = \sum_{t \in T^{\mathrm{NF}}} x_{t,s} \frac{\lambda_{w(t)} d_{t,s}}{n_s} + \sum_{t \in T^{\mathrm{F}}} x_{t,s} \frac{\lambda_{w(t)} d_{t,s}}{2n_s} \ . \qquad (8)$$

We aim to minimize the total power consumption of the servers P. According to the linear power model described in Subsection III-B,

$$P = \sum_{s \in S} \left(p_s^{idle} y_s + p_s^{busy} u_s \right) \ . \qquad (9)$$

As expected, if a server is turned off its power consumption is zero, since $y_s = 0$ and $u_s = 0$.

The first set of constraints are the *allocation constraints*. Since our model can manage both fault tolerant and non-fault tolerant workloads, two different sets of constraints are needed. Tiers of non-fault tolerant workloads must be deployed on a single server, hence

$$\sum_{s \in S} x_{t,s} = 1 \quad \forall t \in T^{\mathrm{NF}} \ . \qquad (10)$$

On the other hand, tiers of fault tolerant workloads must be deployed on two different servers, hence

$$\sum_{s \in S} x_{t,s} = 2 \quad \forall t \in T^{\mathrm{F}} \ . \qquad (11)$$

It is worth noting that the formulation of these constraints for single-tier and multi-tier workloads are the same, since the constraint is applied to each workload tier individually.

In order to obtain an allocation of workloads resilient to failure, we require that the stability condition for any server $s \in S$ holds even when any other server $k \in S$ fails, causing all requests from fault-tolerant workloads placed on both servers to be reversed from server k to server s. In order to express this requirement in mathematical programming notation, we need a constraint for any pair of servers. The *stability constraints* are formulated as follows:

$$u_s + \sum_{t \in T^{\mathrm{F}}} x_{t,s} x_{t,k} \frac{\lambda_{w(t)} d_{t,s}}{2n_s} \le y_s (1 - \epsilon) \qquad \forall (s, k) \in S^2 : s \neq k \ , \qquad (12)$$

where ϵ is an arbitrary small value. On the left hand side, the first term is the server utilization when all servers are operating normally, defined in (8), while the second term represents the increase in utilization when server k fails and considers all the fault-tolerant workloads placed on both servers through the product of binary variables $x_{t,s} x_{t,k}$, which makes the constraint non-linear. The presence of y_s on the right hand side enforces that, if the server is turned off, utilization must be equal to zero and no workload can be placed on server s. The factor $1 - \epsilon$ is required because strict inequalities constraints are not allowed in mathematical programs and we want to ensure that the utilization is strictly smaller than 1.

The last set of constraints are the *response time constraints*, which require that the average response time of each workload w is below a given threshold $r_w^{\max}$. For fault tolerant workloads the constraint is applied on each possible request path: for single-tier workloads the constraint is applied to the response time of both VMs and for multi-tier workloads all the four possible paths are considered. These constraints must be satisfied only under normal operating conditions, i.e. when no server has failed. To express the response time constraints for single-tier workloads, we use (4), subtracting the delay term $d_{t,s} \frac{n_s - 1}{n_s}$ on both sides and multiplying both sides by $1 - u_s$. Hence, the constraints are formulated as follows:

$$\frac{d_{t,s}}{n_s} \le \left(r_w^{\max} - d_{t,s} \frac{n_s - 1}{n_s} \right) (1 - u_s) + L_{t,s} (1 - x_{t,s}) \qquad \forall (w, s) \in W^{\mathrm{ST}} \times S, w = \{t\} \ , \qquad (13)$$

where $L_{t,s}$ is a sufficiently large constant, which is added to the right hand side if and only if $x_{t,s} = 0$ in order to deactivate the constraint when workload tier t is not placed on server s. A suitable choice is $L_{t,s} = d_{t,s}(1 - \epsilon)$. For multi-tier workloads we use (5), subtracting the delay terms on both sides and multiplying both sides by $(1 - u_{s_1})(1 - u_{s_2})$. We obtain the following constraints:

$$\left((1 - u_{s_2}) \frac{d_{t_1,s_1}}{n_{s_1}} + (1 - u_{s_1}) \frac{d_{t_2,s_2}}{n_{s_2}} \right) \le \left(r_w^{\max} - d_{t_1,s_1} \frac{n_{s_1} - 1}{n_{s_1}} - d_{t_2,s_2} \frac{n_{s_2} - 1}{n_{s_2}} \right) (1 - u_{s_1})(1 - u_{s_2}) + L_{t_1,t_2,s_1,s_2} (1 - x_{t_1,s_1} x_{t_2,s_2}) \qquad \forall (w, s_1, s_2) \in W^{\mathrm{MT}} \times S^2, w = \{t_1, t_2\} \ , \qquad (14)$$

where L_{t_1,t_2,s_1,s_2} is a sufficiently large constant, which is added to the right hand side if and only if either $x_{t_1,s_1} = 0$ or $x_{t_2,s_2} = 0$. A suitable choice is $L_{t_1,t_2,s_1,s_2} = (d_{t_1,s_1} + d_{t_2,s_2})(1 - \epsilon)$. Formulating this condition in mathematical programming notation makes the constraint non-linear in the $x_{\cdot,\cdot}$ variables. Another source of non-linearity is the product $(1 - u_{s_1})(1 - u_{s_2})$, since, as defined in (8), u_s is a linear combination of the $x_{\cdot,s}$ variables.

V. Model Reformulation

The model described in Section IV allows the consolidation of multi-tier workloads with performance and reliability constraints. Unfortunately, because of the products of variables in (12) and (14), the formulation described so far is nonlinear and cannot be solved by efficient linear programming solvers, such as CPLEX[4]. This section describes two different exact techniques to construct a linear formulation of the proposed model. These two linearization techniques effectively reduce polynomial zero-one formulations to equivalent linear zero-one formulations leveraging the binary nature of the variables involved.

A. Usual linearization

Usual linearization is a linearization method independently presented by different authors [8], [4], [21] . It involves the addition of one binary variable and two linear constraints for each product of binary variables. This method is a general, problem independent solution to linearize polynomial binary problems. Given a set H, composed of h zero-one variables, the product $\prod_{j\in H} v_j^p$, for any positive p, can be replaced introducing an auxiliary binary variable $a_H \in \{0,1\}$ and imposing two additional constraints:

$$\sum_{j\in H} v_j - a_H \le h-1 \tag{15}$$

$$-\sum_{j\in H} v_j + h a_H \le 0\ . \tag{16}$$

Constraint (15) ensures that $a_H = 1$ when $v_j = 1$ for all $j \in H$. On the other hand, constraint (16) guarantees that $a_H = 0$ if $v_j = 0$ for any $j \in H$.

The first nonlinearity in our model can be found in the stability constraints (12) and is due to the presence of fault tolerant tiers T^{F}. The nonlinear term represents the increase in server utilization when another server fails. For every pair of servers $(s_1, s_2) \in S^2$, the *usual linearization* introduces the following auxiliary variables

$$q_{t,s_1,s_2} = x_{t,s_1} x_{t,s_2} \quad \forall t \in T^{\mathrm{F}},\ (s_1,s_2) \in S^2 : s_1 \ne s_2\ , \tag{17}$$

which is substituted in (12). Moreover, the following constraints must be added:

$$\begin{cases} q_{t,s_1,s_2} \in \{0,1\} \\ x_{t,s_1} + x_{t,s_2} - q_{t,s_1,s_2} \le 1 \\ -x_{t,s_1} - x_{t,s_2} + 2q_{t,s_1,s_2} \le 0\ . \end{cases} \tag{18}$$

Response time constraints for multi-tier workloads (14) represent another source of nonlinearity due to two different terms at the right-hand side of the inequality. First, keeping in mind (8), it can be seen that the product of $(1-u_{s_1})(1-u_{s_2})$ is a second degree polynomial in the $x_{\cdot,\cdot}$ variables. The second nonlinear term is the binary quadratic product $x_{t_1,s_1}x_{t_2,s_2}$, used in conjunction with the big constant L_{t_1,t_2,s_1,s_2} to guarantee that the constraint is active only when the two workload tiers t_1 and t_2 are running on servers s_1 and s_2, respectively. The linearization approach introduces a set of binary variables, to be substituted in the constraint, defined as follows:

$$z_{t_1,s_1,t_2,s_2} = x_{t_1,s_1} x_{t_2,s_2}, \quad \forall (t_1,t_2) \in T^2,\ (s_1,s_2) \in S^2\ . \tag{19}$$

[4]http://www.ibm.com/software/integration/optimization/cplex-optimizer/

To control the linearization of the constraint (14), the following constraints have to be added in the model:

$$\begin{cases} z_{t_1,s_1,t_2,s_2} \in \{0,1\} \\ x_{t_1,s_1} + x_{t_2,s_2} - z_{t_1,s_1,t_2,s_2} \le 1 \\ -x_{t_1,s_1} - x_{t_2,s_2} + 2z_{t_1,s_1,t_2,s_2} \le 0\ . \end{cases} \tag{20}$$

B. Standard linearization

In [10], Glover and Woolsey proposed *standard linearization*, another general, problem-independent linearization method. Given a set H, composed of h zero-one variables, the product $\prod_{j\in H} v_j^p$, for any positive p, can be replaced introducing an auxiliary variable $a_H \ge 0$ and imposing $1+p$ additional constraints:

$$\sum_{j\in H} v_j - a_H \le h-1 \tag{21}$$

$$a_H \le v_j \qquad \forall j \in H\ , \tag{22}$$

where constraints (21) is equal to (15), while (22) sets p constraints forcing a_H to be equal to 0 if one of the v_j variable is zero. In particular this formulation adds $1+p$ constraints while *usual linearization* adds only 2 constraints, but its advantage is that the additional variables are not forced to be binary as for *usual linearization*, but it suffices that they are continuous and greater than zero. Moreover since we are dealing with quadratic binary product, $p = 2$, thus *standard linearization* introduces 3 more constraints for each linearization while *usual linearization* only 2. Starting from *usual linearization* method, we redefine all the auxiliary variables using Glover's approach as follow. For the q_{t,s_1,s_2} variables defined in (17), instead of the constraints defined in (18), the *standard linearization* imposes the following constraints:

$$\begin{cases} q_{t,s_1,s_2} \ge 0 \\ x_{t,s_1} + x_{t,s_2} - q_{t,s_1,s_2} \le 1 \\ q_{t,s_1,s_2} \le x_{t,s_1} \\ q_{t,s_1,s_2} \le x_{t,s_2}\ . \end{cases} \tag{23}$$

For the z_{t_1,s_1,t_2,s_2} linearization variables defined in (19), instead of the constraints defined in (20), the *standard linearization* imposes the following constraints:

$$\begin{cases} z_{t_1,s_1,t_2,s_2} \ge 0 \\ x_{t_1,s_1} + x_{t_2,s_2} - z_{t_1,s_1,t_2,s_2} \le 1 \\ z_{t_1,s_1,t_2,s_2} \le x_{t_1,s_1} \\ z_{t_1,s_1,t_2,s_2} \le x_{t_2,s_2}\ . \end{cases} \tag{24}$$

Both *usual linearization* and *standard linearization* introduce the same number of variables in the linear formulation but they present two differences. *Standard linearization* introduces one more constraint for every binary product with respect to *usual linearization*. On the other hand, *usual linearization* imposes a binary domain for every auxiliary variable, while *standard linearization* does not. We have implemented both of them in our model to evaluate their performance. Results are presented in Section VIII.

VI. Heuristic

In order to tackle the optimization problem formulated in Section IV, we not only tried to solve the linear formulations of Section V with a commercial branch-and-cut code, but also developed an ad-hoc heuristic algorithm that aims at finding high-quality feasible solutions within a small amount of time.

Our algorithm follows a randomized multi-start approach: until a user-defined time limit is exceeded, a greedy procedure is invoked multiple times, keeping track of the best solution found. Each run of the greedy algorithm is randomized, hence diversifying the exploration of the solution space.

The greedy procedure starts from an initial unfeasible solution in which no workload is allocated and, allocating workloads in random-order, tries to satisfy the allocation constraints (10) and (11) without violating the other constraints. Initially, all servers are turned off and therefore $u_s = 0$ and $p_s = 0$. Moreover, the variable u_s^{worst} holds the worst case utilization of server s in the event of a server failure, i.e. $u_s^{\text{worst}} = \max_{k \in S} u_s^{\text{fault}(k)}$. For efficiency reasons, the current allocation is not represented using the $x_{t,s}$ binary variables. Instead, for each tier $t \in T$, $servers_t$ contains the servers on which the tier is allocated (its size will be either 1, for non fault-tolerant workloads, or 2, for fault-tolerant workloads). Moreover, for each server $s \in S$, $tiers_s$ contains the tiers allocated on the servers. At each iteration, the algorithm looks for a feasible allocation of a workload. For multi-tier workloads, both tiers are allocated in the same iteration and two distinct physical servers are simultaneously selected for fault-tolerant tiers. Hence, up to four servers might be involved in the allocation of a single workload. Regardless of the type of workload considered, for each tier t to be allocated in an iteration of the algorithm, a priority sorted list of servers is built. The priority depends on two terms: the first term Δ_s is the increase in power consumption obtained by placing t on server s, including the cost of turning on the server if it is currently turned off; the second term is a random variable uniformly distributed in $[\min_{s \in S} \Delta_s, \max_{s \in S} \Delta_s]$. The random term is added for two reasons. First, we noticed that randomizing only the workload list did not yield a broad enough variation in the paths taken by the greedy heuristic and for some problem instances no feasible solutions were found. The second reason is that we are only considering power efficiency in the short term and, without the random term, we might excessively penalize servers which have a high idle power consumption but might yield a very good performance per Watt at high utilization. The actual allocation procedure differs according to the number of tiers and the fault-tolerance requirement of the workload w being evaluated. For the sake of conciseness, we only report, in Algorithm 1, the allocation procedure for single-tier workloads that require fault-tolerance. After sorting the servers, each pair of server s_1, s_2 such that $s_1 \neq s_2$ is evaluated, until a feasible pair is found. We observe that the complexity of this procedure with respect to the number of servers is $O(|S|^2)$ in the worst case. However, unless the current utilization of the data center is very high,

Algorithm 1 Allocation of single-tier, fault-tolerant workloads.

```
procedure ALLOCATE_ST_F($w$)
    $t \leftarrow$ the only tier of $w$
    $S' \leftarrow$ sort servers in $S$ by priority
    $found \leftarrow$ false
    $j \leftarrow 0$
    $k \leftarrow 0$
    while not $found$ and $j \leq |S|$ do
        while not $found$ and $k \leq |S|$ do
            if $j = k$ then
                continue
            end if
            $s_1 \leftarrow j^{th}$ element of $S'$
            $s_2 \leftarrow k^{th}$ element of $S'$
            $servers_t \leftarrow \{s_1, s_2\}$
            $tiers_{s_1} \leftarrow tiers_{s_1} \bigcup t$
            $tiers_{s_2} \leftarrow tiers_{s_2} \bigcup t$
            $u_{s_1} \leftarrow u_{s_1} + (\lambda_w/2) d_{t,s_1}/n_{s_1}$
            $u_{s_2} \leftarrow u_{s_2} + (\lambda_w/2) d_{t,s_2}/n_{s_2}$
            UpdateUWorst($s_1, s_2$)
            $p_{s_1} \leftarrow p_{s_1}^{idle} + u_{s_1} p_{s_1}^{busy}$
            $p_{s_2} \leftarrow p_{s_2}^{idle} + u_{s_2} p_{s_2}^{busy}$
            $affectedS \leftarrow \{s_1, s_2\}$
            if not CheckMoves($affectedS$) then
                $servers_t \leftarrow \{\}$
                $tiers_{s_1} \leftarrow tiers_{s_1} \setminus t$
                $tiers_{s_2} \leftarrow tiers_{s_2} \setminus t$
                $u_{s_1} \leftarrow u_{s_1} - (\lambda_w/2) d_{t,s_1}/n_{s_1}$
                $u_{s_2} \leftarrow u_{s_2} - (\lambda_w/2) d_{t,s_2}/n_{s_2}$
                $p_{s_1} \leftarrow p_{s_1}^{idle} + u_{s_1} p_{s_1}^{busy}$
                $p_{s_2} \leftarrow p_{s_2}^{idle} + u_{s_2} p_{s_2}^{busy}$
                UpdateUWorst($s_1, s_2$)
            else
                $found \leftarrow$ true
            end if
        end while
    end while
    return $found$
end procedure
```

the procedure will likely find a feasible pair of servers after few iterations. When performing consolidation, the load in the data center tends to be much lower than the available capacity; therefore, we expect the algorithm to perform well. The existence of particularly demanding workloads, might however suggest a different sorting of the servers, but this extension is left as future work. When evaluating a pair of servers, the utilization and power consumption is updated. Moreover, the worst-case utilization in the event of a server failure is updated by evaluating the fault-tolerant tiers in common between the two servers, as shown in Algorithm 2. Afterwards, the constraints are checked by the procedure CheckMoves, described below. If the some constraints are violated, the variables are restored to their previous values and another pair of servers is evaluated. Otherwise, the procedure terminates and the greedy algorithm moves to the next workload. The routine to allocate single-tier workloads that do not require fault-tolerance is even simpler and its complexity, dominated by the sort algorithm, is $O(\log |S|)$. On the other hand, allocating a multi-tier workload with fault-tolerance requires to evaluate two pairs of servers, one for each tier, and the worst-case complexity is $O(|S|^4)$. However, for the same reasons described above, we did not find this to be an issue, since, in practice, only few iterations were required to find a feasible choice of servers.

For efficiency reason, the procedure CheckMoves takes as an argument the list of servers affected by the recent moves. In this way, the number of constraints to be verified is greatly reduced. The routine works in three steps: (1) it verifies the worst-case utilization of all the affected servers; (2) it builds the list of the affected workloads, i.e. the workloads whose tiers are placed on any of the affected servers; (3) it checks the response time constraints for all the affected workloads. Since the number of affected servers is less than 4, the complexity with respect to the number of servers is constant. On the other hand, if we consider the number of workloads, the complexity is $O(|W|)$. This makes the worst-case complexity of each iteration of the greedy algorithm, taking into account multi-tier workloads with fault-tolerance, $O(|S|^4|W|)$. Considering that $|W|$ iterations are required, the worst-case complexity of the greedy algorithm is $O(|S|^4|W|^2)$.

Algorithm 2 Update of the worst case utilization after allocating a fault-tolerant workload tier on servers s_1 and s_2.

```
procedure UPDATEUWORST($s_1$,$s_2$)
    $u_{s_1}^{\text{fault}(s_2)} \leftarrow u_{s_1}$
    $u_{s_2}^{\text{fault}(s_1)} \leftarrow u_{s_2}$
    for $t$ in
        $tiers_{s_1}$ do
        if $t \in T^{\text{F}}$ and $t \in tiers_{s_2}$ then
            $u_{s_1}^{\text{fault}(s_2)} \leftarrow u_{s_1}^{\text{fault}(s_2)} + (\lambda_{w(t)}/2)d_{t,s_1}/n_{s_1}$
            $u_{s_2}^{\text{fault}(s_1)} \leftarrow u_{s_2}^{\text{fault}(s_1)} + (\lambda_{w(t)}/2)d_{t,s_2}/n_{s_2}$
        end if
    end for
    $u_{s_1}^{\text{worst}} \leftarrow \max\left(u_{s_1}^{\text{worst}}, u_{s_1}^{fault(s_2)}\right)$
    $u_{s_2}^{\text{worst}} \leftarrow \max\left(u_{s_2}^{\text{worst}}, u_{s_2}^{fault(s_1)}\right)$
end procedure
```

VII. Experimental Settings

All the computational tests were executed on the same computer, characterized by a Core i7 920 CPU and 12 GB of DDR3 memory. The model was described using the language AMPL[5] 8.1 and solved using CPLEX 12.2. CPLEX was configured to run in single-threaded mode and was run simultaneously on four different problem instances.

Our model was tested on a large number of realistic, randomly generated instances. We considered 18 different test cases, described in Table III. The first column contains the number that identifies each test case. We considered scenarios with 5 and 10 servers. Test cases are further differentiated by the overall data center utilization (0.3, 0.5 or 0.7) and the number of workload tiers (2, 3 or 4 times the number of servers). Half the workload tiers belong to multi-tier workloads, while the remaining half represent single-tier workloads. Moreover, about half the workload tiers are fault-tolerant. The last column of the table reports the number of workloads present in the different test cases. For each test case, 5 different instances of the optimization problem were generated and tested, for a total of 90 problem instances.

[5]http://www.ampl.com/

TABLE III
Test cases used to validate the heuristics.

Test Case	Utilization	Servers	Tiers	Workloads
1	0.3	5	10	7
2	0.5			
3	0.7			
4	0.3	5	15	11
5	0.5			
6	0.7			
7	0.3	5	20	15
8	0.5			
9	0.7			
10	0.3	10	20	15
11	0.5			
12	0.7			
13	0.3	10	30	23
14	0.5			
15	0.7			
16	0.3	10	40	30
17	0.5			
18	0.7			

TABLE IV
Server specifications. Power characteristics and speedup factors are based on published results of the SPECpower benchmark.

Server	CPUs	n_s	f_s	p_s^{idle} (W)	p_s^{busy} (W)
Sun Netra X4250	2 x Intel Xeon L5408	8	1.000	228.730	71.111
HP ProLiant DL385 G6	2 x AMD Opteron 2435	12	1.554	124.532	136.846
Dell PowerEdge R710	2 x Intel Xeon X5570	8	2.354	75.014	140.816
Dell PowerEdge R815	2 x AMD Opteron 6174	24	1.344	116.744	160.717
Dell PowerEdge R815	4 x AMD Opteron 6174	48	1.335	178.171	329.533
HP ProLiant DL380 G7	Intel Xeon L5640	12	2.105	68.708	101.635
Dell PowerEdge R710	2 x Intel Xeon X5670	12	2.643	77.813	145.670

Server specifications were randomly selected from a list based on real-world server models, represented in Table IV. Parametrization of the linear power model was performed applying least squares linear regression to the publicly available results of the SPECpower benchmark. The speedup factors f_s characterizing the different servers are based on the maximum throughput achieved in the SPECpower benchmark, i.e.

$$f_s = \frac{ssj\_ops_s n_{\tilde{s}}}{ssj\_ops_{\tilde{s}} n_s} \ , \tag{25}$$

where ssj_ops_j refers to the maximum throughput achieved by server j and the reference server $\tilde{s}$ is the Sun Netra X4250. It is important to observe that the speedup factor refers to a single CPU core.

The other parameters were generated according to uniform and discrete probability distributions. Let $\mathcal{U}_d\{h_1,\ldots,h_n\}$ denote the uniform discrete distribution over the set of values $h_1,\ldots,h_n$ and $\mathcal{U}(a,b)$ denote the uniform continuous distribution over the interval $[a,b]$. We generated the service demands of the workloads on the reference server as follows:

$$d_{t,\tilde{s}} \sim \mathcal{U}(0.1, 2.0) \quad \forall t \in T \ .$$

The service demands on the other servers were obtained using the speedup factors:

$$d_{t,s} = \frac{d_{t,\tilde{s}}}{f_s} \quad \forall (t,s) \in T \times S \ .$$

The relative maximum response time with respect to the reference server was selected from a uniform discrete distribution.

Hence the maximum response time for single-tier workloads was set as follows:

$$R_w^{\max} = \mathcal{U}_d\{1.1, 1.5, 2.0, 3.0\} d_{t,\tilde{s}} \quad \forall w \in W, w = \{t\} \ ,$$

where the coefficient 1.1 represent a very strict performance requirement, while the coefficient 3.0 represent a rather loose performance requirement. Analogously, for multi-tier workloads we considered the service demands of both tiers:

$$\begin{aligned} R_w^{\max} =& \mathcal{U}_d\{1.1, 1.5, 2.0, 3.0\} d_{t_1,\tilde{s}} + \\ & \mathcal{U}_d\{1.1, 1.5, 2.0, 3.0\} d_{t_2,\tilde{s}} \quad \forall w \in W, w = \{t_1, t_2\} \ . \end{aligned}$$

Let U^{total} represent the overall utilization of the data center and $G^{\text{total}} = \sum_{s \in S} f_s n_s$ be its total standard CPU capacity. In order to meet the desired level of data center utilization, the amount of consumed capacity was partitioned across workloads according to randomly generated weights $g_w \sim \mathcal{U}(0, 1)$, setting the arrival rates as follows:

$$\lambda_w = \frac{\frac{g_w}{\sum_{j \in W} g_w} G^{\text{total}} U^{\text{total}}}{\sum_{t \in w} d_{t,\tilde{s}}} \qquad \forall w \in W \ ,$$

where the denominator takes into account the service demands of multiple tiers where needed.

VIII. Experimental Results

This section presents results obtained using CPLEX 12.2 and the heuristic presented in Section VI to solve the proposed model using the instances detailed in Table III. The heuristic was implemented in C++ and it has been tested on the same datasets used to solve the model by CPLEX solver, thus it is possible to evaluate its accuracy. Since CPLEX has not found an optimal solution for the majority of the instances with 10 servers (see Table VI), the heuristic effectiveness on every instance has been tested comparing the heuristic solution and the best dual bound that CPLEX has found with either of the linearization methods. This is a pessimistic evaluation method, but it guarantees an upper bound on the error committed by the heuristic. Despite the evaluation method, results show the effectiveness of the heuristic for the proposed problem.

Results are organized in two different tables considering scenerios with both 5 (see Table V) and 10 servers (see Table VI). Both tables present three distinct parts labeled *Dataset Information*, *CPLEX* and *Heuristic*.

Dataset Information reports the scenario identifier and the linearization method used to solve the instance with CPLEX. Since we generated five random instances of the optimization problem for each test case, execution times and optimality gaps for both CPLEX and the heuristic are the average on the relevant instances.

The table part labeled *CPLEX* reports the number of infeasible instances out of the five randomly generated in the *inf* column. The number of optimally solved instances are reported in the fifth column, labeled *opt*, while column *opt time* details the average time CPLEX takes to find the optimal solution. Three other columns account for non optimally solved instances. In particular column *no-int sol* details the

TABLE V
RESULTS FOR 5 SERVERS.

Dataset Information		CPLEX) (tlim=24 hours)						Heuristic (tlim=5sec.)	
Test	lin. method	inf inf	no-int sol	opt	opt time (s)	sub opt	sub-opt gap (%)	opt	sub-opt gap (%)
1	standard	0	0	5	1.3	0	0	4	< 0.01
	usual	0	0	5	1.76	0	0		
2	standard	1	0	4	1.5	0	0	3	< 0.01
	usual	1	0	4	1.98	0	0		
3	standard	1	0	4	19.5	0	0	0	0.02
	usual	1	0	4	41.41	0	0		
4	standard	0	0	5	14.5	0	0	3	< 0.01
	usual	0	0	5	27.83	0	0		
5	standard	0	0	5	28.8	0	0	1	0.16
	usual	0	0	5	56.76	0	0		
6	standard	0	0	5	92.9	0	0	0	0.09
	usual	0	0	5	950.55	0	0		
7	standard	0	0	5	158.2	0	0	0	0.04
	usual	0	0	5	105.95	0	0		
8	standard	0	0	5	761.8	0	0	0	0.33
	usual	0	0	5	4567.84	0	0		
9	standard	0	0	5	18676.8	0	0	1	0.16
	usual	0	0	5	25878.6	0	0		

number of instances for which no integer solution has been found, while column *sub-opt* reports the number of instances for which CPLEX has found an integer non optimal solution. Obviously for every instances in *no-int* or *sub-opt* CPLEX has spent all the available time, so the average computational time, equal to the maximum available time, is not explicitly reported. The last column of the *CPLEX* part contains the average, over all the instances for which a sub-optimal solution was found, optimality gap between the objective value and the associated dual bound.

Heuristic collects, in both tables, results from the heuristic algorithm. In particular the number of optimally solved instances in the test case is reported in *opt* column, while *gap* column contains the average percentage optimality gap between the best dual bound found by CPLEX's (with standard or usual linearization) and the heuristic objective value, for instances with sub-optimal solutions.

A. Discussion

Table V reports results with 5 servers, that are all optimally solved by CPLEX with a time limited to 24 hours for both *usual linearization* and *standard linearization*. The heuristic can also solve optimally most instances with 5 servers with a time limited to 5 seconds. Moreover the instances for which only an approximate solution has been found exhibit an optimality gap always smaller than 0.4%.

Results obtained in test cases with 10 servers show that most of the instances cannot be optimally solved by CPLEX within a time limit fixed to 24 hours (see Table VI). Moreover the optimality gap reported in *lim-int gap* column has become very high, reaching values as large as 37% considering both the linearization methods. These results lead to two different considerations. First, there is no clear winner between the two linearization methods. In particular, we suppose that the impact of problem dimensions outweighs the difference of the two linearization approaches. Second, these results shows the real complexity of the formulated problem in terms of both constraint and variable numbers. Despite its time limit, set to only 30 seconds, the heuristic optimality gaps are comparable

TABLE VI
RESULTS FOR 10 SERVERS.

Dataset Information		CPLEX) (tlim=24 hours)						Heuristic (tlim=30sec.)	
Test	lin. method	inf inf	no-int sol	opt	opt time (s)	sub opt	sub-opt gap (%)	opt	sub-opt gap (%)
10	standard	0	0	3	3932.7	2	4.35	0	4.07
	usual	0	0	2	12364.4	3	5.22		
11	standard	0	0	3	11030.8	2	1.36	0	7.9
	usual	0	0	0	-	5	5.78		
12	standard	0	0	0	–	5	9.63	0	10.79
	usual	0	0	0	-	5	9.15		
13	standard	0	0	0	–	5	7.82	0	10.07
	usual	0	0	0	-	5	15.07		
14	standard	0	0	0	–	5	8.4	0	13.09
	usual	0	0	0	-	5	10.37		
15	standard	0	0	0	–	5	18.27	0	16.39
	usual	0	0	0	-	5	16.23		
16	standard	0	1	0	–	4	15.98	0	15.39
	usual	0	0	0	-	5	33.11		
17	standard	0	5	0	–	0	-	0	14.66
	usual	0	2	0	-	3	37.12		
18	standard	0	2	0	–	3	26.99	0	17.91
	usual	0	2	0	-	3	27.33		

with the ones found by CPLEX in 24 hours. While the heuristic optimality gap can be as large as $17,91\%$, we believe that the heuristic solution is very good and that the lower bound is extremely conservative, lending to a very pessimistic optimality gap estimate. In fact, CPLEX can only solve a few nodes of the branch and cut tree, due to the large amount of time required to solve the relaxation.

IX. CONCLUSIONS

In this work, we presented a novel model for server consolidation which takes into account two aspects neglected by previous literature. First, we considered workloads that, for reliability or performance reasons, are deployed on two separate servers. Requests were assumed to be routed to either of the two instances in a round-robin fashion. This introduced non-linearities in the response time constraints. Second, we ensured that stability conditions hold even if one the server fails. Hence, utilization constraints also became nonlinear. We dealt with both non-linearities with two different linearization techniques and compared them experimentally on a vast number of randomly generated realistic data sets. A randomized multi-start greedy heuristic was also developed and its effectiveness was validated by comparing its solutions with the lower bounds found by the exact solver.

In the future, we plan to look for more compact linear formulations to achieve lower bounds in a small amount of time even for large-sized problem instances. Moreover, we aim at further improving the performance of our heuristic, optimizing the code and making it parallel. Generalizations of our model to deal with multiple constrained resources and compatibility constraints are also in our plans.

REFERENCES

[1] K. Adams. A comparison of software and hardware techniques for x86 virtualization. In *in ASPLOS-XII: Proceedings of the 12th international conference on Architectural*, pages 2–13. ACM Press, 2006.

[2] U. E. P. Agency. Report to congress on server and data center energy efficiency, 2007.

[3] J. Anselmi, E. Amaldi, and P. Cremonesi. Service consolidation with end-to-end response time constraints. In *Software Engineering and Advanced Applications, 2008. SEAA '08. 34th Euromicro Conference*, pages 345–352, sep 2008.

[4] E. Balas. Extension de l'algorithme additif a la programmation en nombres entiers et a la programmation non linaire. *Comptes rendus de e l'Academie des Sciences (Paris)*, 1964.

[5] K. Dhyani, S. Gualandi, and P. Cremonesi. A constraint programming approach for the service consolidation problem. In A. Lodi, M. Milano, and P. Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *Lecture Notes in Computer Science*, pages 97–101. Springer Berlin / Heidelberg, 2010.

[6] S. Distefano, F. Longo, and M. Scarpa. Availability Assessment of HA Standby Redundant Clusters. In *2010 29th IEEE Symposium on Reliable Distributed Systems*, pages 265–274. IEEE, Oct. 2010.

[7] D. Economou, S. Rivoire, and C. Kozyrakis. Full-system power analysis and modeling for server environments. In *In Workshop on Modeling Benchmarking and Simulation (MOBS*, 2006.

[8] R. Fortet. L'algbre de boole et ses applications en recherche operationnelle. *Cahiers du Centre d'Etudes de Recherche Operationnelle 4*, pages 5–36, 1959.

[9] M. R. Garey, R. L. Graham, D. S. Johnson, and Andrew. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory*, 21:257–298, 1976.

[10] F. Glover and E. Woolsey. Converting 0-1 polynomial programming problem to a 0-1 linear program.

[11] R. Gupta, S. K. Bose, S. Sundarrajan, M. Chebiyam, and A. Chakrabarti. A two stage heuristic algorithm for solving the server consolidation problem with item-item and bin-item incompatibility constraints. *Services Computing, IEEE International Conference on*, 2:39–46, 2008.

[12] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L. K. John. Using complete machine simulation for software power estimation: The softwatt approach. *High-Performance Computer Architecture, International Symposium on*, 0:0141, 2002.

[13] L. Kleinrock. *Queueing Systems, Volume I: Theory*. Wiley Interscience, New York, 1975.

[14] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1984.

[15] D. A. Menasce and V. Almeida. *Capacity Planning for Web Services: metrics, models, and methods*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001.

[16] A. C. Rencher. *Linear Models in Statistics*. Wiley, 2000.

[17] J. Rolia, A. Andrzejak, and M. Arlitt. Automating enterprise application placement in resource utilities. In M. Brunner and A. Keller, editors, *Self-Managing Distributed Systems*, volume 2867 of *Lecture Notes in Computer Science*, pages 118–129. Springer Berlin / Heidelberg, 2003.

[18] H. Shafi, P. J. Bohrer, J. Phelan, C. A. Rusu, and J. L. Peterson. Design and validation of a performance and power simulator for powerpc systems. *IBM Journal of Research and Development*, 47(5.6):641 –651, sep 2003.

[19] B. Speitkamp and M. Bichler. A mathematical programming approach for server consolidation problems in virtualized data centers. *IEEE Trans. Serv. Comput.*, 3:266–278, October 2010.

[20] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye. Energy-driven integrated hardware-software optimizations using simplepower. *SIGARCH Comput. Archit. News*, 28:95–106, 2000.

[21] W. Zangwill. Media selection by decision programming. *Journal of Advertising Research*, pages 30–36, 1965.