Service Consolidation with End-to-End Response Time Constraints

Jonatha Anselmi, Edoardo Amaldi
Politecnico di Milano, p.zza L. da Vinci, Milan, Italy
Email: *jonatha.anselmi@polimi.it*

Paolo Cremonesi
Neptuny, via Durando 10, Milan, Italy
Politecnico di Milano, p.zza L. da Vinci, Milan, Italy

Abstract

In this paper, we address the service consolidation problem: given a data-center, a set of servers and a set of multi-tiered services or applications, the problem is to allocate services to the available servers in order to minimize the number of servers to use while avoiding the overloading of system resources and satisfying end-to-end response time constraints. Exploiting queueing networks theory, we describe a number of linear and non-linear combinatorial optimization problems related to the server consolidation problem. Since their solution is difficult to obtain through standard solution techniques, we propose accurate heuristics which quickly compute a sub-optimal solution and let us deal with hundreds of servers and applications. Experimental results illustrate the impact of the consolidation in data-centers and show that the heuristic solution is almost very close to the optimum.[1]

1. Introduction

An important problem arising in information technology (IT) infrastructures is the *consolidation* of data-centers resources (e.g., services, applications, servers), a complex task aimed to drop costs related to energy consumption, hardware investments, space costs, etc. To reduce conflicts between services, many enterprise data centers currently host most of their IT services on dedicated servers without taking into account the possibility of deploying multiple services on a same server. Therefore, servers are not used at their maximum capabilities and, in turn, high hardware investments are often required. However, recent technologies such as *server virtualizaton* and *server partitioning* provide a practical way to reduce conflicts between services and facilitate the migration and the deployment of services among servers. Within such technologies, a single server can be partitioned into multiple virtual servers and each virtual server is isolated from the others. Server partitioning is typically implemented with support at hardware level. On the other hand, server virtualization implements server partitioning by means of software environments called virtual machines which can be hosted on a same physical server. At the cost of increased CPU overheads, server virtualization allows more flexibility if compared to server partitioning. More recent approaches use a combination of both virtualization and hardware support to combine the benefits of the two solutions. Within these new technologies, many enterprises are undertaking a number of consolidation projects aimed to combine different services on a set of servers with the objective of minimizing data center costs without violating end-to-end response time constraints.

The first work addressing the use of optimization techniques for resource management problems in computing systems is the paper proposed by Chu [4] which exploits integer programming formulations for file allocation. The model he considers is related to a multi-dimensional bin packing problem. Rolia et al. [10] tackle the server consolidation problem with a dynamic approach and the optimization problem takes into account the dynamic behavior of the workloads, i.e. daily or weakly seasonal behavior. To solve the problem, the paper suggests the utilization of a genetic algorithm and a case study with 41 servers is presented. The main drawback of this approach is its limited scalability to large scale data–centers with hundreds or thousands of servers and applications. Bichler et al. [2] present a similar dynamic approach tailored for virtualized systems. The main difference of their approach is that the optimization problem is solved exploiting multi-dimensional bin-packing approximate algorithms [8]. This methodology does not take into account performance aspects, e.g. constraints on response times. Almeida et al. [1] formulate a complex

[1]This work has been accomplished thanks to the technical and financial support of Neptuny.

non-linear optimization problem tailored on computing systems working with admission-control mechanisms. The approach takes into account both the dynamic behavior of the workload and the loss of revenues due to service levels violations. Hühn et al. [6] present a simple optimization problem focused on maximizing the profit (i.e., the difference between revenues and costs).

In this paper, we tackle the service consolidation problem as a combinatorial optimization problem. Given a data-center specification, the objective of the problem is to find the best mapping between application tiers and servers which minimizes data-center costs while guaranteeing performance constraints on server utilizations and data-center response times. The estimates of such performance indices are obtained by exploiting the theory of queueing networks [7] because they are robust and versatile tools able to accurately capture the performance behavior of service systems (see, e.g., [11] for a recent work). The use of queueing networks models to tackle the consolidation problem is a novel approach and yields simple analytical formulation. We propose a number of combinatorial optimization problems related to the service consolidation. Since the response time analytic expression yields a non-linear formulation of the problem, we propose an efficient heuristic showing that it provides an upper bound of the objective function. Experimental results show that the heuristic almost finds the optimum. We also compare the impact of our consolidation with respect to trivial deployment schemes adopted by modern IT infrastructures. The main contribution of this paper with respect to the state of the art is the capability to address three previously unsolved issues: (i) multi-layered services, (ii) constraints on end-to-end response times, and (iii) load-balanced applications.

The structure of the paper is as follows. In Section 2, we discuss the parameters characterizing the services and the data center and define the associated queueing network model. In Section 3, we incrementally introduce the consolidation problem and propose heuristics for its efficient solution. Section 4 presents experimental results numerically evaluating the efficiency and the accuracy of the proposed heuristics. Finally, Section 5 draws the conclusions of our work and outlines future research.

2 Data-center Queueing Network Model

2.1 Data-center Description

The data center is composed of M heterogeneous servers. The cost of using server j, which comprises energy consumption, maintainability costs, etc., and its *speed-up* are respectively denoted by c_j and ρ_j, $j = 1, \ldots, M$. The speed-up of a server is understood as its relative processing capacity obtained by the execution of suitable benchmarks with respect to a reference server.

The data center hosts R different applications (or services) and each application is deployed on multiple tiers (e.g. web-server tier, application-server tier, etc.). Application r sequentially spans T_r tiers, $r = 1, \ldots, R$, and when an application r job (or client) joins the data center, it initially executes tier 1 on some server, then it proceeds to tier 2 and so on till the T_r-th. For application r jobs, when the T_r-th tier is reached, the request is forwarded back to the $(T_r - 1)$-th for some further processing and so on till the first one. It is well-known that this behavior agrees with standard multi-tiered architectures. More than one application tier can be deployed on a given server and, for simplicity of notation, we denote by

$$T = \sum\nolimits_r T_r \tag{1}$$

the total number of application tiers.

In this paper, we assume that each tier of each application is deployed on exactly one server. However, we will show how our model can be extended to take into account the deployment of application tiers on multiple servers, i.e. the load-balancing. The deployment of a given application on multiple tiers is usually referred to as *vertical scalability* and it is important to provide a better performance handling larger workloads and to solve possible conflicts among different layers (different application tiers may use different technologies). On the other hand, the deployment of a given application tier on multiple servers is usually referred to as *horizontal scalability* and is needed to handle large workloads. The horizontal scalability is also important to guarantee availability issues: in fact, if a given application tier is deployed on multiple servers, then a failure on a single server does not prevent the availability of the application because the workload is rearranged among the available servers.

An other potential source of lack of system availability is the deployment of several tiers on a same server. Therefore, we assume that a maximum number of P_j application tiers can be deployed on server j. This assumption is also meant to avoid the modeling of non-negligible overheads in service times estimates (usually referred to as *virtualization overhead*) which would be introduced by the middleware management if the number of virtual machines running on a single server is large.

In agreement with the notation of basic queueing networks theory [7], we denote by $S_{j,r,t}$ the mean (total) *service time* (or execution time) required by a job which executes tier t of application r on server j when the network contains no other job.

In the following, if not otherwise specified, indices j, r and t will implicitly range, respectively, in sets $\{1, \ldots, M\}$,

$\{1,\ldots,R\}$ and $\{1,\ldots,T_r\}$ indexing servers, applications and tiers.

2.2 Queueing Network Model

Since the optimization problem we tackle takes into account performance constraints, we model the data-center with a queueing network. In fact, queueing network models are the most popular tool for evaluating the performance of computer and communication systems and, in the mathematical formulation of our problem, let us deal with simple analytical expressions of performance indices. In particular, we define a *product-form* (also known as *separable*) queueing network model (see [7] for an introduction). Product-form models are a robust tool able to capture the data center performance behavior and their effectiveness is due to the good compromise they provide between the accuracy and the computational effort needed by the model solution.

Since the data center hosts different applications (characterized by different service demands) and an arriving job can execute only one of them, the model we build is *multiclass*. For convenience, a job requesting the execution of application r is referred to as a class-r job.

Since the number of jobs populating the data center is not *a priori* known, the model we build is *open* and we denote by λ_r the mean workload (arrival rate) of class-r jobs, $r = 1,\ldots,R$. To meet the product-form assumptions we suppose that λ_r is the average of a poissonian arrival process. This assumption is often adopted, e.g. [9], and in our setting it is appropriate because data centers usually handle high workloads and many different applications.

The stations of the queueing network model the data center servers and, in the following, we use the term *station* when we refer to the queueing network and the term *server* when we refer to the data center. We assume that the stations service discipline is Processor Sharing (PS) because this is a reasonable abstraction of modern service centers and it is used in many works, e.g. [9].

Let also $D_{j,r}$ be the mean *service demand* [7] of class-r jobs at station j, i.e. the total average time required by a class-r job to station j during the execution of all its tiers and when the network contain no other job. Within this standard definition, we underline that the service demands include the processing times of jobs at servers when they visit stations passing from the first tier to the last one and returning back from the last tier to the first one. In other words, the service demands model the fact that jobs make exactly two visits at a given station and this is in agreement with multi-tier architectures. We assume that the service demands also include the time needed by a server to transfer a job to an other server, given that it is non-negligible.

The notion of service demand takes into account that it is possible to deploy more tiers of a given application on the same server. For instance, assuming that only tiers from 1 to $t_r \leq T_r$ of application r are deployed on server j, we have

$$D_{j,r} = \sum_{t=1}^{t_r} S_{j,r,t}. \quad (2)$$

In order to meet the product-form assumptions, we assume that $S_{j,r,t}$ is the mean value of a random variable having rational Laplace transform. This assumption is reasonable because this class of random variables is wide.

Within this queueing network model of the data center, we recall that the average utilization of station j due to class-r jobs, i.e. the *busy* time proportion of server j due to class-r jobs, is given by

$$U_{j,r} = U_{j,r}(\lambda_r) = \lambda_r D_{j,r}. \quad (3)$$

Formula (3) is known as *utilization law* [7]. Clearly, the total average utilization of server j is given by $U_j = U_j(\lambda_1,\cdots,\lambda_R) = \sum_r U_{j,r} < 1$. We also recall that the average response time of class-r jobs, i.e. the time interval between the submission of a class-r job into the data center and its receipt, is given by

$$W_r = W_r(\lambda_1,\cdots,\lambda_R) = \sum_j \frac{D_{j,r}}{1-\sum_{s=1}^{R}\lambda_s D_{j,s}}. \quad (4)$$

Since we deal with the *averages* of performance indices, when referring to an index we will drop the word average.

We now show a simple example to illustrate the queueing network model underlying the data center. Let us consider the case of two applications, i.e. $R = 2$, having both three tiers, i.e. $T_1 = T_2 = 3$ and $M = 5$ available servers, and let us also suppose that the application tiers are deployed on the servers as indicated in Table I. For instance, we have that tier 2 of application 2 is deployed on server 3. We notice that server 5 is not used. Since each tier of each application is deployed on exactly one server, all service demands are given by the sum of service times as in (2). The queueing network model underlying the deployment scheme of Table I is shown in Figure 1. We remark that the fact the queueing network model is represented as a pipeline is due to the fact that the service demands already model the fact that jobs return back to visit again application tiers.

Tier	Class 1	Class 2
1	1	2
2	1	3
3	2	4

Table 1. Deployment scheme of the example.

We note that the proposed queueing network model of the data-center does not explicitly take into account the notion of tier which is embedded in the notion of service demands.

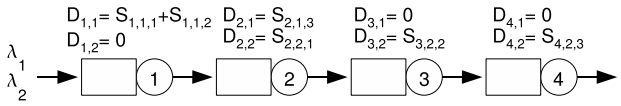

Figure 1. The queueing network model corresponding to the deployment of Table I.

3 Service Consolidation Problem

The objective of the service consolidation problem is to exploit the available servers to obtain a data center *configuration* able to satisfy, *in the average*, performance constraints on utilizations and data-center response times while minimizing the sum of servers costs.

The decision variable we include in our optimization models is

$$x_{j,r,t} = \begin{cases} 1 & \text{if tier } t \text{ of application } r \text{ is deployed} \\ & \text{on server } j, \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

Let us refer to *configuration* as a possible assignment of variables $x_{j,r,t}$ satisfying the issues discussed in previous section, i.e. a *feasible* deployment scheme. The goal of the optimization problem is to find the configuration of minimum cost which satisfies constraints on utilizations and response times.

Constraints on the maximum utilization of each server are important to

1. avoid the saturation of physical resources letting the system handle unexpected workload peaks,
2. guarantee a low sensitivity of data-center response time in front of small workload variations (it is well-known, e.g. [7], that the response time curve grows to infinity according to a hyperbole when the utilization of a server approaches one),
3. increase the data-center reliability because if a failure occurs on a server, then the associated applications can be moved on different servers preventing a drastic growth of the data-center response time.

Constraints on the maximum response time are often used in many applications for ensuring, for instance, some quality of service (see, e.g., [3]).

In Section 3.1, we give an Integer Linear Programming (ILP) formulation of the problem which takes into account utilizations constraints only. Since for large data-centers the problem is characterized by several variables, i.e. hundreds of thousands, a heuristic is proposed for its solution. In Section 3.2, we extend the previous formulation introducing response times constraints. Since this new formulation makes the problem non-linear, to solve the problem we propose an upper bound.

3.1 Consolidation with Utilization Constraints

The first issue we take into account in our optimization problem is to limit the overall utilization of each server. In this section, we assume that each application tier is deployed on exactly one server.

Let $\hat{U}_j$ be the upper bound on the utilization of server j. Adopting binary variables y_j to take into account whether or not server j is included in the configuration, we formulate the following ILP problem

$$\mathrm{P}_1\text{: } \min \quad \sum\nolimits_j c_j y_j \quad (6)$$

$$\sum\nolimits_j x_{j,r,t} = 1, \ \forall r,t \quad (7)$$

$$\sum\nolimits_r \lambda_r \sum\nolimits_t x_{j,r,t} S_{j,r,t} \leq \hat{U}_j y_j, \ \forall j \quad (8)$$

$$\sum\nolimits_r \sum\nolimits_t x_{j,r,t} \leq P_j, \ \forall j \quad (9)$$

$$x_{j,r,t} \in \{0,1\}, \ \forall j,r,t \quad (10)$$

$$y_j \in \{0,1\}, \ \forall j. \quad (11)$$

Clearly, P_1 objective function minimizes the weighted sum of servers costs. Constraints family (7) ensures that exactly one server is dedicated to the execution of tier t of application r and we note that such constraints ensure that

$$\sum\nolimits_j y_j \leq T, \quad (12)$$

i.e. at most T servers are used, because in the worst case each tier of each application is deployed on a different server.

Furthermore, a direct consequence of (7), as described in Section 2.2, is that the service demands of the queueing network model are given by the following relation

$$D_{j,r} = \sum\nolimits_t S_{j,r,t} x_{j,r,t}. \quad (13)$$

Given (3), this allows us to state that the left-hand side terms of inequalities (8) represent the overall utilization of each server. Constraints (8), thus, limit the utilization of each server and introduce binary variables y_j which are raised to one if and only if there exists a tier t and an application r such that $x_{j,r,t} = 1$. Such variable essentially takes into account whether or not server j is chosen by the current solution.

Finally, constraints family (9) limits the number of applications and tiers to deploy on j. We remark that these latter constraints are of paramount importance to build a robust optimization model. In fact, if the number of applications deployed on server j is *large*, i.e. greater than P_j,

then it very likely happens that middleware management introduces significant computation overheads in the service times estimates so that the resulting model is not accurate.

Additional constraints can be introduced in P_1 to avoid the deployment of particular application tiers on some servers. For instance, this can be due to operating systems incompatibilities, e.g. Windows software on Linux servers. In this case, the constraint is given by simply imposing that $x_{j,r,t} = 0$ for some j, r and t.

We note that constraints (7) can introduce a limitation in the problem because to handle large workloads it may be possible that a horizontal scalability is needed, i.e. that an application tier must be deployed on multiple servers. We now show the case in which tier t of application r must be deployed on exactly $m_{r,t} \geq 1$ servers. In this case, constraint (7) generalizes to

$$\sum_j x_{j,r,t} = m_{r,t}, \ \forall r, t \tag{14}$$

and assuming that the workload is equally partitioned among the $m_{r,t}$ (a priori not known) servers, the service demand expression (13) must include the replacement

$$S_{j,r,t} \leftarrow \frac{S_{j,r,t}}{m_{r,t}}. \tag{15}$$

It is easy to see that constraints (14) yield a configuration with increased availability. If $m_{r,t} \geq 2$, within some r and t, then a crash on a single server (say j) does not prevent the proper functioning of the system and the fact that servers utilizations are limited (by means of (8)) ensures that the workload to j can be rearranged among the $m_{r,t} - 1$ remaining servers without compromising the model stability.

Furthermore, consider also the case in which application tiers require disk storage. Let $d_{r,t} \geq 0$ be the disk quote needed by tier t of application r and let d_j be the amount of disk space available on server j. In this case, we take into account storage requirements by introducing, in P_1, the constraints family

$$\sum_r \sum_t d_{r,t} x_{j,r,t} \leq d_j, \ \forall j \tag{16}$$

which ensures that each application tier is reserved the proper amount of disk space.

It is easy to see that the number of binary variables adopted by P_1 is $M + MT$. Since large-scale data-centers are composed of hundreds of servers and applications, i.e. several thousands of variables $x_{j,r,t}$, the exact solution of P_1 requires a strong computational effort. Therefore, we now provide a simple heuristic aimed to find a good solution in a shorter time.

The heuristic we propose initially guesses the set of servers which yields the configuration of minimum cost and, with respect to this set only, checks whether or not a feasible configuration exists. If such configuration does not exist, then the guess is iteratively refined by adding the *best* server until a feasible solution is found. Algorithm 1 is the heuristic we propose for the efficient solution of P_1.

Algorithm 1 Heuristic solution for P_1

1: Solve the relaxation of P_1 when $x_{j,r,t}$ are continuous and y_j are binary;
2: **if** no feasible solution exists **then**
3: **return**
4: **end if**
5: $Y_1 := \{j : y_j = 1\}$;
6: $Y_0 := \{j : y_j = 0\}$;
7: Let $P_1^{'}$ be problem P_1 where
- the objection function (6) is removed;
- variables y_j are fixed to 1 for all $j \in Y_1$;
- variables y_j and $x_{j,r,t}$, for all $j \in Y_0$, r and t, are removed;

8: **while** a (binary) feasible solution of $P_1^{'}$ does not exist **do**
9: $j := \text{argmin}_{i \in Y_0} c_i/\rho_i$;
10: $Y_1 := Y_1 \cup \{j\}$;
11: $Y_0 := Y_0 \setminus \{j\}$;
12: **end while**

First, we solve P_1 assuming that $x_{j,r,t}$ are continuous variables. We note that the optimum of this problem can be obtained with a significantly smaller computational effort because the number of binary variables, which mainly affect the computational requirements, drops from $M + MT$ to M. Obviously, if a feasible solution of this problem does not exist, then a feasible solution does not exist also for P_1 and the algorithm ends. Otherwise, a lower bound on the configuration of minimum cost is obtained. We then define set Y_1 (Y_0) as the set of servers chosen (respectively, not chosen) by the optimal configuration of the relaxed problem and problem $P_1^{'}$ which takes into account the servers belonging to $\mathcal{Y}$ only. $P_1^{'}$ is thus composed of much less variables and constraints than P_1. Then, we search for a feasible solution of problem $P_1^{'}$ (Line 8). If this problem is feasible then a solution is found and the algorithm ends. Otherwise, we augment the space of feasible solutions by adding to Y_1 a server not included in the configuration computed by the relaxed problem in Line 1. This server is chosen as a trade-off between servers cost and speed-up. Then, we iteratively check the feasibility of $P_1^{'}$.

Since the optimum of the relaxed problem defined in Line 1 is a lower bound on the solution of P_1, we have that if the condition in the loop holds at its first evaluation, then it provides the optimum. In Section 4, we show that this

heuristic almost provides the optimum, i.e. Algorithm 1 very likely ends without performing iterations of the *while* loop. In general, if n is the number of iterations perfomed by the algorithm, then n is an upper bound on the difference between the number of servers adopted within the optimal solution and within the proposed heuristic. This because the objective function value corresponding to the optimal configuration of P_1 must be greater than or equal to the one obtained with the Line 1 relaxation of Algorithm 1.

3.2 Consolidation with Response Times Constraints

The second issue we take into account in the consolidation is to limit the end-to-end response time of each application. Denoting by $\hat{W}_r$ the upper bound on the data-center response time of application r, we formulate problem P_2 which extends P_1 by adding the following constraints family

$$\sum_j \frac{D_{j,r}}{1 - \sum_{s=1}^{R} \lambda_s D_{j,s}} \leq \hat{W}_r, \ \forall r \tag{17}$$

where $D_{j,r}$ is given by (13).

Such constraints have been widely adopted in many contexts (see, e.g., [3]) and their main difficulty relies on their non-linearity which prevents the adoption of efficient techniques. Furthermore, the large number of applications characterizing real-world data centers is such that the number of such constraints is large and this makes the problem prohibitively expensive to solve. Hence, we initially approach its approximate solution proposing a heuristic which bounds from above the cost of the final configuration. In other words, this bound provides a pessimistic estimate on the configuration of minimum cost.

The intuitive rationale behind the heuristic is the following: if the objective function (6) tends to minimize the number of servers to adopt, then the optimal solution of P_2 is such that the utilization of each chosen server j very likely approaches to its limit value $\hat{U}_j$. This means that the (optimal) solution of P_2 is such that the denominator of summand j in the response time expression (17) becomes very close to $1 - \hat{U}_j$. Hence, consider the ILP problem $P_2^{'}$ which modifies P_2 considering the linear constraints

$$\sum_j \frac{D_{j,r}}{1 - \hat{U}_j} \leq \hat{W}_r, \ \forall r, \tag{18}$$

instead of (17). Clearly, $P_2^{'}$ is an ILP problem.

We note that the optimal solution of $P_2^{'}$ is an upper bound on the solution of P_2 because each term in the response time expression (18) is greater than or equal to the corresponding term in (17). This implies that the space of the feasible solutions of $P_2^{'}$ is a subset of the one of P_2. This implies that the value of objective function (6) chosen by solving $P_2^{'}$ must be greater than or equal to the one obtained by solving P_2.

The final solution can be easily obtained by applying Algorithm 1 taking into account constraints (18).

4 Experimental Results

In this section, we present some numerical results in order to evaluate the accuracy and the computational requirements of our approach. Experimental analyses have been performed by running the Ilog Cplex optimization solver on a 2.80GHz Intel Xeon CPU with hyperthreading technology. Algorithm 1 has been implemented in the AMPL language [5]. In the experiments, the utilization thresholds, the values of P_j and the arrival rates have been randomly chosen according to a uniform distribution among ranges $[0.6, 1]$, $[5, 15]$ and $[0.1, 10]$ jobs per time unit. Server costs are assumed to be unitary. The service times $S_{j,r,t}$ have been generated as follows. Since we want to evaluate the impact of the consolidation on costs, we assume that a data-center is given and that its (initial) configuration is such that each server hosts exactly a single application tier. We note that this trivial configuration adopting T servers is often employed by modern data-centers and, for simplicity, we assume that tier t of application r is deployed on server $f(r,t) \equiv \sum_{s=1}^{r} T_s + t$ (this essentially builds a bijection between application tiers and servers). This configuration corresponds to the situation in which no consolidation is performed. Within this initial data-center configuration, we generated the service times as follows

$$S_{f(r,t),r,t} = \frac{(1-\epsilon)\texttt{drand48()}}{\lambda_r} \tag{19}$$

where `drand48()` is the standard C function which generates a pseudo-random number in range $[0, 1]$ and $\epsilon = 0.01$ is a small correction factor which prevents the generation of service times which yield an average utilization of 100% (we assume that in the original configuration the servers are able to handle the incoming workloads). The remaining service times have been generated as follows. For each server j, we first generated its speed-up between 1 and 100 according to a uniform distribution. Then, we applied the following scaling factor

$$S_{i,r,t} = S_{f(r,t),r,t} \frac{\rho_{f(r,t)}}{\rho_i}, \ i \neq f(r,t) \tag{20}$$

which represents a reasonable approximation for the service times at server i. Formula (20) essentially scales the service times of the existing configuration according to the processing capacities of the data-center servers.

The accuracy of Algorithm 1 has been evaluated by measuring the number of iterations performed in its loop.

Clearly, if this number is equal to zero, then the proposed heuristic returns the optimal configuration. On the other hand, we remark that a non-zero number of iterations does *not* necessarily imply that the optimum is not found (see Section 3 for further details). A direct comparison with respect to the optimum of P_1 has not been carried out because of the prohibitively expensive computational effort needed by its exact solution. In Figure 2, we illustrate the average and the maximum number of iterations performed in the loop of the proposed algorithm by varying the number of applications from 20 to 140 with step 10 and keeping fixed to 3 the number of tiers (it is known that real-world applications usually span 3 tiers). Hence, the (variable) number of servers becomes $M = \sum_r T_r = 3R$. In the figure, each point is referred to the average or the maximum value of 50 randomly generated models. We note that the solution

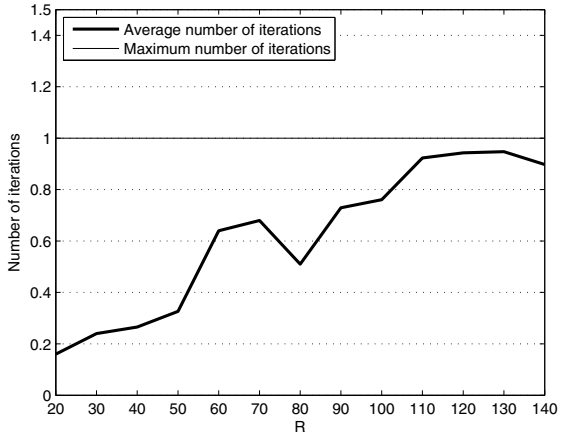

Figure 2. Average and maximum number of iterations performed by Algorithm 1.

identified by our heuristic is very close to the optimum: on a total of 650 randomly generated models, no more than 1 iteration has been performed by our algorithm. This lets us conclude that Algorithm 1 is very accurate. We also note that the bold line curve in Figure 2 can be clearly interpreted as an upper bound on the probability that Algorithm 1 does not return the optimum. If $R \leq 40$ such probability is less than 0.3.

Since the proposed heuristic performs one iteration in the worst case, the above result implies that the problem P_1 relaxation shown in Line 1 of Algorithm 1 can be adopted to immediately obtain (at design time) a very accurate measure of the maximum cost reduction resulting from the consolidation. We remark that the solution of such relaxation can be obtained very quickly because of the limited number of binary variables, i.e. T instead of $T + T^2$. With respect to the performed experiments, the computation times needed by the solver to compute the solution of this relaxation are of the order of a minute in the worst case. The computation times needed by the solver to find a feasible solution of the problem in Line 8 are of the order of 10 minutes in the worst case.

We now measure the improvement of the configuration found by Algorithm 1 with respect to the trivial configuration which simply assigns exactly one application tier to exactly one server and vice versa. This latter configuration is often employed in real-world data-centers. Within this setting, we increased again the number of applications from 20 to 140 while keeping fixed to 3 the number of tiers and keeping M equal to $3R$ as in previous cases. In Figure 3, we show the maximum, the average and the minimum values of ratio

$$\text{“Number of servers identified by Algorithm 1”}/M \quad (21)$$

which is a measure of the cost reductions adopted by our solution with respect to the solution which does not exploit the consolidation. In the figure, each point is associated to 10 randomly generated models. What we see in Figure 3

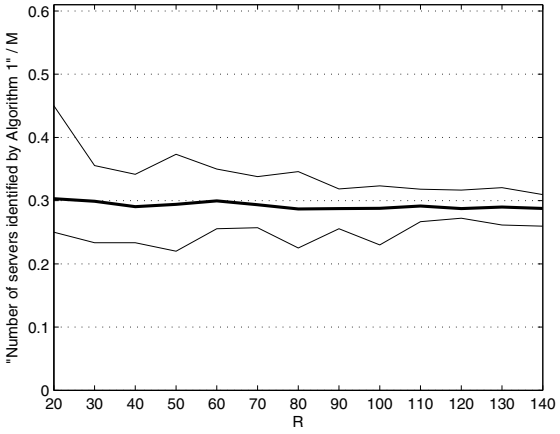

Figure 3. Maximum, average and minimum value of (21) for increasing the data-center sizes.

is that the consolidation yields remarkable costs reductions. In fact, the number of servers adopted in the configuration identified after the consolidation is, in the average case, $0.3M$. We also note that as the data-center size increases (recall that in these experiments we assumed $M = 3R$), the ratio (21) can be well-approximated by a constant, i.e. 0.3. In general, this suggests that even for very large data-center, the number of servers identified by our heuristic scales with the data-center size within a constant.

5 Conclusions

In this paper, we tackled the service consolidation problem aimed to find the best data-center configuration which minimizes costs while avoiding the overloading of servers and satisfying end-to-end response time constraints. A queueing network model of the data-center has been proposed to deal with performance indices and a combinatorial optimization model able to handle multi-tier applications, constraints on end-to-end response times and the load-balancing of applications has been formulated. Since the solution of the optimization problem through standard ILP techniques is prohibitively expensive with respect to significantly large data-centers, we approached its solution proposing efficient heuristics. Experimental results revealed that our approach yields configurations of the data-center yielding costs which are very close to the optimum. With respect to the trivial configuration which maps an application tier to exactly one server and vice versa, we numerically showed that the proposed approach yields configurations which approximately adopt the thirty percent of the available servers and it seems that this scaling holds in general even for larger data-centers.

We leave as future work the experimental analysis of the proposed approach when response time are considered. Within this setting, we showed that the proposed approach provides an upper bound on the final configuration. We also leave as future research the more difficult case in which the workload can partitioned among an arbitrary (not known) number of servers. This further feature would handle larger workloads and availability issues. This further feature would let the data center scale horizontally and it is also strictly related to availability issues. From a mathematical point of view, this can be accomplished by introducing the following constraints

$$\sum\nolimits_j x_{j,r,t} \geq m_{r,t},\ \forall r, t, \tag{22}$$

instead of constraints (7), where $m_{r,t}$ denotes the minimum number of servers in which tier t of application r must be deployed. Within this setting, we notice that the service demands expression (13) is more difficult because it becomes non-linear.

Acknowledgements

The authors thank Stefano Gualandi for his comments and his support on Cplex issues.

References

[1] J. Almeida, V. Almeida, D. Ardagna, C. Francalanci, and M. Trubian. Resource management in the autonomic service-oriented architecture. In *ICAC 2006 Proceedings (3rd International Conference on Autonomic Computing), 84-92, Dublin June 2006*.

[2] M. Bichler, T. Setzer, and B. Speitkamp. Capacity planning for virtualized servers. In *In Proceedings of the Workshop on Information Technologies and Systems, Milwaukee, Wisconsin, USA, December 9 - 10, 2006*.

[3] V. Cardellini, E. Casalicchio, V. Grassi, and R. Mirandola. A framework for optimal service selection in broker-based architectures with multiple qos classes. In *SCW '06: Proceedings of the IEEE Services Computing Workshops*, pages 105–112, Washington, DC, USA, 2006. IEEE Computer Society.

[4] W. W. Chu. Optimal file allocation in a multiple computer system. *IEEE Trans. Comput.*, 18(10):885–889, 1969.

[5] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, November 2002.

[6] O. Hühn and C. Breitbarth. Performance modelling for sla-compliant but cost-effective it-service provisioning. In *Proceedings of Workshop on Information Technologies and Systems (WITS 2007) at the International Conference on Information Systems (ICIS 2007), Montreal, Canada, 2007*.

[7] 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.

[8] S. Martello, D. Pisinger, and P. Toth. New trends in exact algorithms for the 0-1 knapsack problem. 1997.

[9] D. A. Menasce, L. W. Dowdy, and V. A. F. Almeida. *Performance by Design: Computer Capacity Planning By Example*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[10] J. Rolia, A. Andrzejak, and M. F. Arlitt. Automating enterprise application placement in resource utilities. In *DSOM*, pages 118–129, 2003.

[11] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. Analytic modeling of multitier internet applications. *ACM Trans. Web*, 1(1):2, 2007.