Reliable Service Allocation in Clouds

Olivier Beaumont, Lionel Eyraud-Dubois, Hubert Larchevêque

Reliable Service Allocation in Clouds

Olivier Beaumont, Lionel Eyraud-Dubois, Hubert Larchevêque
INRIA Bordeaux – Sud-Ouest
University of Bordeaux
Email: olivier.beaumont@labri.fr, eyraud@labri.fr, hubert.larcheveque@labri.fr

***Abstract*—We consider several reliability problems that arise when allocating applications to processing resources in a Cloud computing platform. More specifically, we assume on the one hand that each computing resource is associated to a capacity constraint and to a probability of failure. On the other hand, we assume that each service runs as a set of independent instances of identical Virtual Machines, and that the Service Level Agreement between the Cloud provider and the client states that a minimal number of instances of the service should run with a given probability. In this context, given the capacity and failure probabilities of the machines, and the capacity and reliability demands of the services, the question for the cloud provider is to find an allocation of the instances of the services (possibly using replication) onto machines satisfying all types of constraints during a given time period.**

In this paper, our goal is to assess the impact of the reliability constraint on the complexity of resource allocation problems. We consider several variants of this problem, depending on the number of services and whether their reliability demand is individual or global. We prove several fundamental complexity results (#P' and NP-completeness results) and we provide several optimal and approximation algorithms. In particular, we prove that a basic randomized allocation algorithm, that is easy to implement, provides optimal or quasi-optimal results in several contexts, and we show through simulations that it also achieves very good results in more general settings.

I. Introduction

A. Reliability Issues in Cloud Computing

This paper considers several reliability problems that arise when allocating Virtual Machines (VMs) onto Physical Machines (PMs) in a Cloud computing platform. Cloud Computing [1], [2], [3], [4] has recently emerged as a new paradigm for service providing over the Internet. Using virtualization, it is possible to run several Virtual Machines on top of a given Physical Machine. Since each VM hosts its complete software stack (Operating System, Middleware, Application), it is moreover possible to migrate VMs from a PM to another.

The problem of mapping VMs having heterogeneous computing demands onto PMs having heterogeneous capacities can be modeled as a multi-dimensional bin-packing problem. In this context, each physical machine is characterized by its computing capacity (*i.e.* the number of flops it can process during one time-unit), its memory capacity (*i.e.* the number of different VMs that it can handle simultaneously, given that each VM comes with its complete software stack) and its failure rate (*i.e.* the probability that the machine will fail during the next time period).

In order to deal with capacity constraints in resource allocation problems, several sophisticated techniques have been developed in order to optimally allocate VMs onto PMs, either to achieve good load balancing [5], [6], [7] or to minimize energy consumption [8], [9]. Most of the works in this domain have therefore concentrated on designing offline [10] and online [11], [12] solutions of Bin Packing variants.

The reliability constraints have received much less attention in the context of Cloud computing, as underlined by Walfredo Cirne in [13]. On the other hand, in the last few years, these questions have been addressed in the context of Peer-to-Peer networks. Indeed, efficient data sharing in such systems is complicated by erratic node failure, unreliable network connectivity and limited bandwidth. In this context, replicating data on multiple nodes can improve both availability and response time and the question is to determine when and where to replicate data in order to meet performance goals in large-scale systems [14], [15], [16], [17], [18]. Reliability issues have also been addressed by High Performance Computing community. Indeed, a lot of efforts is done to build systems capable of reaching the Exaflop performance [19], [20] and exascale systems are expected to gather billions of processing units, thus increasing the importance of fault tolerance issues [21]. The main solutions considered for fault tolerance in Exascale systems are based on replication strategies [22] and rollback recovery relying on checkpointing protocols [23], [24].

To assess the complexity introduced by reliability constraints, we will stick to a simple context, that nevertheless captures the main difficulties. First, we will consider that the applications running on the Cloud platform can be seen as a set of independent services, and that services themselves consist in a number of identical (in terms of capacities) and independent instances. Therefore, we do not consider the problems introduced by heterogeneity, that has already been considered (see for instance [6], [7]), since heterogeneity implies that allocation problems are amenable to bin packing problem and are therefore intrinsically difficult. Then, we consider static allocation problems only, in the sense that our goal is to find the allocation that optimizes the reliability during a time period, instead of relying on VM migration and creation to

ensure that a minimal number of instances of each service is running whatever the machine failures. Therefore, our work allows to assess precisely the complexity introduced by machine failures and service reliability demands.

The characteristics of the applications and their requirements have been negotiated between a client and the provider through a Service Level Agreement (SLA). In the SLA, each service is characterized by its demand in terms of processing capability (*i.e.* the minimal number of instances of VMs that must be running simultaneously) and in terms of reliability (*i.e.* the maximal probability that the service will not benefit from this number of instances at some point during the next time period). Our goal is to find an allocation of the instances of services so as to enforce capacity and reliability constraints, given the reliabilities of the machines and the capacities of machines. In general, the overall capacity of the machines is larger than the demands of the services, but the reliability constraint for the services is higher than the reliability of individual machines, so that we will allocate more services than the actual demand, *i.e.* use replication, so as to enforce reliability.

B. Paper Outline

In Section II, we present the model and the different problem formulations that will be addressed in this paper, in the context of a single service (in Section III), of several grouped services (in Section IV), or of several independent services (in Section V). In all three different situations, we assess the complexity introduced by reliability constraints by proving complexity results (NP-Completeness or #P'-Completeness results), we prove optimality results for special cases, and we propose heuristics and approximation algorithms that we compare through extensive average-case experimentations. Finally, we provide concluding remarks in Section VI.

II. Models

In this section, we introduce the notations that will be used throughout the paper and define precisely the optimization problems that we will consider. Our target cloud platform is made of m physical machines $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_m$. Machine $\mathcal{M}_j$ is able to store CAPA_j instances of services. On this cloud, our goal is to run n services $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n$. DEM_i identical and independent instances of service $\mathcal{S}_i$ are required. In practice, instances of services would run as Virtual Machines. Several instances of the same service can run concurrently and independently on the same physical machine. We will denote by $\mathcal{A}_{i,j}$ the number of instances of $\mathcal{S}_i$ running on $\mathcal{M}_j$. Therefore, $\sum_i \mathcal{A}_{i,j}$ represents the number of instances running on $\mathcal{M}_j$ and has to be smaller than CAPA_j. On the other hand, $\sum_j \mathcal{A}_{i,j}$ represents the overall number of running instances of $\mathcal{S}_i$ and is in general larger than DEM_i since replication (*i.e.* over-reservation) of services is used in order to enforce reliability constraints.

More precisely, each machine $\mathcal{M}_j$ comes with a failure rate FAIL_j, that represents the probability of failure of $\mathcal{M}_j$ during a fixed period a time (say, one day). During the time period, we will not reallocate instances of services to physical machines but rather provision extra instances for the services (replicas) that will actually be used if some machine fails. We will denote by ALIVE the set of running machines and by ALIVE_i the number of instances of service i running on alive machines after the time period (in the case of **OneService**, this last number is denoted by ALIVE_s).

On the other hand, each service $\mathcal{S}_i$ comes with some reliability requirement. The problem comes into two flavors, depending on the nature of the requirement: either each service $\mathcal{S}_i$ comes its own reliability requirement REL_i, or the whole group of services comes with some global reliability requirement REL.

OneService$(m, \text{CAPA}, \text{DEM}, \text{REL})$: Find an allocation $\mathcal{A}$ of instances of service $\mathcal{S}$ to machines $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_m$ such that $\mathcal{P}(\text{ALIVE}_s \geq \text{DEM}) \geq \text{REL}$, *i.e.* the probability that at least DEM instances are running on alive machines after the time period is larger than the reliability requirement REL.

GroupedServices$(m, \text{CAPA}, n, \text{DEM}, \text{REL})$: Find an allocation $\mathcal{A}$ of instances of services $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n$ to machines $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_m$ such that $\mathcal{P}(\forall i,\ \text{ALIVE}_i \geq \text{DEM}_i) \geq \text{REL}$, *i.e.* the probability that all services have at least DEM_i instances running on alive machines after the time period is larger than the overall reliability requirement REL.

IndependentServices$(m, \text{CAPA}, n, \text{DEM}, \text{REL})$: Find an allocation $\mathcal{A}$ of instances of services $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n$ to machines $\mathcal{M}_1, \mathcal{M}_2, \ldots \mathcal{M}_m$ such that $\forall i,\ \mathcal{P}(\text{ALIVE}_i \geq \text{DEM}_i) \geq \text{REL}_i$, *i.e.* the probability that a least DEM_i instances of $\mathcal{S}_i$ are running on alive machines after the time period is larger than the reliability requirement REL_i.

For each of those three problems we define the corresponding reliability estimation problem $\mathcal{P}_{OneService}$ (respectively $\mathcal{P}_{GroupedServices}$ and $\mathcal{P}_{IndependentServices}$) as the problem of computing the reliability probability of a given allocation for a given instance of $OneService$ (resp. $GroupedServices$ and $IndependentServices$), *i.e.* the probability that at least DEM instances are running on alive machines after the time period.

Note that in the case when there is only one service, the problem is somehow degenerate in the sense that finding the best allocation is trivial (simply allocate as much instances as possible on each machine). Nevertheless, we prove in Section III that estimating if the reliability of the resulting allocation is larger than the requirement REL is $\#P'$-complete. We prove in Section IV that the situation is in fact very similar in the case of grouped services, *i.e.* that finding the best (fractional) allocation is easy but estimating its reliability is $\#P'$-complete. At last, in Section V, we prove that in the case of several independent services, finding the optimal allocation is NP-Complete and estimating the reliability of an allocation is $\#P'$-

complete.

III. Case of a Single Service

A. Introduction

We consider the problem

OneService$(m, \text{CAPA}, \text{DEM}, \text{REL})$: Find an allocation $\mathcal{A}$ of instances of service $\mathcal{S}$ to machines $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_m$ such that $\mathcal{P}(\text{ALIVE} \geq \text{DEM}) \geq \text{REL}$, *i.e.* the probability that at least DEM instances are running on alive machines after the time period is larger than the reliability requirement REL.

Clearly, in order to maximize the reliability, one possible solution consists in allocating as many instances of $\mathcal{S}$ (up to DEM, obviously) on each machine. It is worth noting that we consider that the resources of the Cloud are assumed to be reserved, and we can use them completely to run $\mathcal{S}$ (or the set of services $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n$ in the following sections). The question of determining a minimal amount of resources is not meaningful in this context and is left out of the scope of this paper.

B. How to Estimate the Reliability of an Allocation?

In this section we consider the $\mathcal{P}_{OneService}$ problem. Let us assume that the allocation $\mathcal{A}$ is given. In order to compute what is the reliability induced by the allocation, it is necessary to estimate the probability of each possible configuration of machines states (dead or alive) after the time period.

A configuration is completely defined by the set of ALIVE machines. Its probability is given by

$$\prod_{j \in \text{ALIVE}} (1 - \text{FAIL}_j) \prod_{j \notin \text{ALIVE}} \text{FAIL}_j$$

and the corresponding number of alive instances is given by

$$\sum_{j \in \text{ALIVE}} \mathcal{A}_j.$$

Therefore, the reliability of an allocation $\mathcal{A}$ is given by the sum of the probabilities of the configurations (defined by their ALIVE set) satisfying $\sum_{j \in \text{ALIVE}} \mathcal{A}_j \geq \text{DEM}$ (other configurations are invalid), *i.e.*

$$\sum_{\text{ALIVE } s.t. \sum_{j \in \text{ALIVE}} \mathcal{A}_j \geq \text{DEM}} \left(\prod_{j \in \text{ALIVE}} (1 - \text{FAIL}_j) \prod_{j \notin \text{ALIVE}} \text{FAIL}_j \right)$$

Of course, there are 2^m possible ALIVE sets and therefore, above method to estimate the reliability of an allocation has an exponential computing cost. In Section III-C, we prove that in fact, the counting problem associated to the estimation of the reliability is $\#P'$-Complete. Nevertheless, we give in Section III-D a dynamic programming algorithm that computes the reliability in pseudo-polynomial time (and even in polynomial time for a large class of instances).

C. Complexity of Reliability Computation

When studying an **NP** decision problem, the question is to determine whether there exists a solution to a given instance. However, in many contexts, one is interested in knowing not only whether a solution exists, but also the number of solutions. For instance, in our context, we are interested in evaluating the (weighted) number of valid ALIVE sets in order to compute the reliability of an allocation. The $\#P$ complexity class, introduced by Valiant [25], captures this notion.

More precisely, since we are dealing with non integer values (outputs of our problems are probabilities in $[0, 1]$), we work with the $\#P'$ complexity class, derived from the $\#P$ class by Bodlaender et al. [26], that has been specifically designed to work with probabilities.

Several $\#P'$-completeness results have been established in the context of graphs and reliability. For instance, let an undirected, simple graph G be given with a number $p_e \in [0, 1]$ associated to each edge e of G and corresponding to the probability that the edge e is present in the surviving subgraph. Then, the problem of computing the probability that there is a path between two distinguished vertices s and t has been proven $\#P'$-complete [27], and the problem of computing the probability that all vertices remain connected has also been proven $\#P'$-complete [28]. More recently, these results have been extended to node failures in the context of allocations between clients and servers [26]. A problem closely related to our context, in which the complexity of computing the reliability of a task graph schedule in presence of task replication is studied, has been proved $\#P'$-complete by Benoit et al. [29], .

In what follows we prove that

Theorem 3.1: $\mathcal{P}_{OneService}$ is $\#P'$-Complete.

Proof: First, we have to prove that $\mathcal{P}_{OneService}$ belongs to $\#P'$. To do so, we consider the function h that takes as input an instance of $\mathcal{P}_{OneService}$, *i.e.* an allocation on a given instance of **OneService**, and that outputs the reliability probability of this allocation. We need to prove that h consists in a function $f \in \#P$, that takes as input the instance and that outputs an integer value, and a polynomial-time function g that takes as input an integer value (the output of f) and the instance, and that outputs non-integer value. In our case, the output of f is the number of valid ALIVE sets on a given instance of $\mathcal{P}_{OneService}$, and the output of g, which is also the output of h, is the reliability of the allocation, *i.e.* a probability in $[0, 1]$.

The failure probability FAIL_j of each machine during the time period is assumed to be encoded as $\frac{n_j}{d_j}$. A vector $x = (x_j)_{1 \leq j \leq m}$ defines the ALIVE set : if $0 \leq x_j \leq n_j$, then machine j fails, while if $n_j < x_j \leq d_j$, machine j does not fail.

The NP decision problem corresponding to $\mathcal{P}_{OneService}$ is the following : given an allocation $\mathcal{A}$, is there a vector x such that at least DEM instances of

the service are running on alive machines at the end of the time period? This problem belongs to NP since the vector x of size $O(m)$ constitutes the certificate. Indeed, to check whether x is a satisfying vector, we compute the number of instances of the service running on alive machines, that can be identified with x.

The corresponding $\#P$ problem consists in computing how many distinct vectors x correspond to valid allocations, what defines the function f evoked above. Moreover, there are $\prod_{1\leq j\leq m} d_j$ distinct vectors and each of them defines a possible final configuration at the end of the time period. We obtain the reliability of an allocation by dividing the number of successful vectors (*i.e.* the output of function f) by the total number of possible scenarios. This operation corresponds to the function g required to prove that $\mathcal{P}_{OneService}$ is in $\#P'$.

In order to prove that $\mathcal{P}_{OneService}$ is $\#P'$-Complete, we have to prove that $\mathcal{P}_{OneService}$ $\#P'$-Hard. To do so, we propose a reduction to the Knapsack problem.

Given a list of non-negative integer weights $w_1, \dots w_n$, and an integer capacity C, the counting variant of Knapsack consists in counting the number of subsets of items whose overall weight is strictly smaller than C. This variant is known to be $\#P$-complete [30].

To perform the reduction from the Knapsack counting problem, we consider an instance $w_1, \dots, w_n$ of the Knapsack problem, and we build an instance of $\mathcal{P}_{OneService}$ consisting in m machines such that $\forall 1 \leq j \leq m, \textsc{Capa}_j = w_j$ and $\textsc{Fail}_j = 1/2$. Let us set $\textsc{Dem} = C$, and let us consider optimal solutions such that all machines are filled at their maximum capacity.

Each vector x corresponds to a set of alive machines at the end of the time period. Therefore, since the whole capacity of each machine is used, the number of running instances is given by the sum of the capacities of alive machine. The number of vectors x such that at least $\textsc{Dem} = C$ instances of the service are alive is the difference between the overall number of possible x vectors and the number of vectors corresponding to a number of instances strictly smaller than $\textsc{Dem}$.

Since all failures rates are equal to $\frac{1}{2}$, all vectors x have exactly the same probability $\frac{1}{2}^n$, and therefore, this last term of the difference corresponds exactly to the solution of the Knapsack counting problem for this instance.

Since the Knapsack counting problem is $\#P$-Hard, computing the number of vectors x that provide enough alive instances of the service is also $\#P$-Hard. From [26], this implies that it is also $\#P'$-Hard, and thus that $\mathcal{P}_{OneService}$ is $\#P'$-Complete. ■

D. Pseudo-Polynomial algorithm via Dynamic Programming

Despite this $\#P$'-hardness result, it is possible to compute the reliability of an allocation with heterogeneous failure probabilities in pseudo-polynomial time, using dynamic programming. Indeed, let us denote by $F(i, s)$ the probability that at least s instances of the service are alive (*i.e.* run on alive machines) using the first i machines $(\mathcal{M}_1, \dots, \mathcal{M}_i)$ only. It is possible to derive the following recursive equation

$$F(i{+}1, s) = F(i, s){\times}\textsc{Fail}_{i+1}{+}F(i, s{-}\mathcal{A}_{i+1}){\times}(1{-}\textsc{Fail}_{i+1}),$$

that states the following property: in the allocations built from the set of the first $i + 1$ machines,

- under the condition that machine $i + 1$ is not alive (what happens with probability $\textsc{Fail}_{i+1}$), s instances of the service are available on the $i+1$ first machine if and only if they are available on the first i machines $(F(i + 1, s) = F(i, s))$;
- under the condition that machine $i+1$ is alive (what happens with probability $(1 - \textsc{Fail}_{i+1})$), having s instances on the first $i+1$ machines is equivalent to having $s - \mathcal{A}_{i+1}$ instances available with the first i machines only $(F(i + 1, s) = F(i, s - \mathcal{A}_{i+1})$.

With the initial conditions stating that $F(i, s) = 1$ for all $s \leq 0$ and $F(0, s) = 0$ for all $s > 0$, this equation allows to compute the values of $F(i, s)$ for all $1 \leq i \leq n$ and $0 \leq s \leq \textsc{Dem}$, in time $O(n \times \textsc{Dem})$. In particular, the value $F(n, \textsc{Dem})$ is the reliability of allocation $\mathcal{A}$. This complexity is not in contradiction with the result stated in Theorem 3.1. Indeed, the complexity of the dynamic programming algorithm is of order $n \times \textsc{Dem}$, whereas an instance of $\mathcal{P}_{OneService}$ is of size $n + \log \textsc{Dem}$, so that the algorithm has a pseudo-polynomial complexity only. Nevertheless, if the number of instances per machine is bounded by a constant, above algorithm provides the reliability of an allocation very quickly.

ï»¿

IV. Case of Several Applications with a Shared Reliability Goal

A. Introduction

In this section, we consider the case of several services, and our goal is to find an allocation $\mathcal{A}$ such that the probability of having at least $\textsc{Dem}_i$ surviving instances of any service $\mathcal{S}_i$ is at least $\textsc{Rel}$.

GroupedServices$(m, \textsc{Capa}, n, \textsc{Dem}, \textsc{Rel})$: Find an allocation $\mathcal{A}$ of instances of services $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_n$ to machines $\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_m$ such that $\mathcal{P}(\forall i,\ \textsc{Alive}_i \geq \textsc{Dem}_i) \geq \textsc{Rel}$, *i.e.* the probability that all services have at least $\textsc{Dem}_i$ instances running on alive machines after the time period is larger than the overall reliability requirement $\textsc{Rel}$.

In Section IV-B, we prove that if one can allocate a fractional number of a service on a machine, the proportional mapping strategy, that allocates on each machine a number of instances of $\mathcal{S}_i$ proportional to its overall weight (in terms of demand) provides the optimal solution. In the more realistic setting where only full instances of services can be allocated to machines, we prove in Section IV-C that the problem is $\#P'$-Complete even if the demand of each service is 1 and we provide in Section IV-D a deterministic algorithm and a randomized

algorithm inspired by the Proportional Mapping strategy and that achieve very good results in practice.

B. Fractional Solution

Let us assume that $\mathcal{A}$ is allowed to allocate a fractional number of services to a machine.

Theorem 4.1: Let us consider the proportional mapping where $\mathcal{A}^{\text{PM}}_{i,j} = \frac{\text{DEM}_i}{\sum_k \text{DEM}_k}\text{CAPA}_j$. Then, $\mathcal{A}$ is optimal, even in the fully heterogeneous case (where both the capacities and the failure rates of the machines are heterogeneous).

Proof: Let us consider any possible set of alive machines ALIVE. A necessary condition for ALIVE to be a valid set is that all services have at least DEM_i surviving instances, *i.e.*

$$\sum_{j \in \text{ALIVE}} \text{CAPA}_j \geq \sum_i \text{DEM}_i.$$

On the other hand, if $\sum_{j\in\text{ALIVE}} \text{CAPA}_j \geq \sum_k \text{DEM}_k$, then

$$\forall i, \sum_{j \in \text{ALIVE}} \mathcal{A}^{\text{PM}}_{i,j} = \sum_{j \in \text{ALIVE}} \frac{\text{CAPA}_j}{\sum_k \text{DEM}_k}\text{DEM}_i \geq \text{DEM}_i.$$

Therefore, the necessary condition for the validity of ALIVE is also a sufficient condition for $\mathcal{A}^{\text{PM}}$ to provide a valid allocation for ALIVE, what achieves the proof. ■

As in the one-machine case, it is therefore possible (in the fractional case) to determine an optimal allocation in polynomial time.

C. Complexity Results

As in the case of a single service, the problem of determining if an allocation satisfies the reliability constraint falls into the conditions of Theorem 3.1 and is therefore $\#P'$-complete.

Corollary 4.2: of Theorem 3.1 **GroupedServices** is $\#P'$-Complete.

Proof: Clearly, the computation of the reliability of an allocation in **GroupedServices** is actually harder than in **OneService**, because every instance of **OneService** is an instance of **GroupedServices** consisting in only one service. ■

Nevertheless, and contrarily to what happens in the case of a single service, $\mathcal{P}_{GroupedServices}$ remains $\#P'$-Complete even if the demand for each service is 1, *i.e.* $\forall i, \text{DEM}_i = 1$). Therefore, the following result shows the intrinsic difficulty when several services are considered simultaneously.

Theorem 4.3: $\mathcal{P}_{GroupedServices}$ restricted to the case where $\forall i, \text{DEM}_i = 1$ is $\#P'$-Complete.

Proof: Following the proof of Theorem 3.1, $\mathcal{P}_{GroupedServices}$ belongs to $\#P'$. In order to prove that it is $\#P'$-Hard, we perform a reduction to Monotone-2SAT [25], by considering only instances where $\text{DEM}_i = 1$. Note that this reduction implies in particular that it is not possible to derive a pseudo-polynomial algorithm for $\mathcal{P}_{GroupedServices}$ just as we did for $\mathcal{P}_{OneService}$ in Section III-D.

In Monotone-2SAT, we are given a satisfiability formula $F = c_1 \wedge c_2 \wedge \cdots \wedge c_r$, $c_i = y_{i1} \vee y_{i2}$, where y_{ij}s are taken from a set of variables X, and the problem is to compute the number of truth assignments to variables that make the formula true.

Given an instance of Monotone-2SAT, *i.e.* a formula, it is possible to build the following instance of the $\mathcal{P}_{GroupedServices}$ problem:

- There is one machine for each variable in X, whose failure probability p_i is $\frac{1}{2}$;
- There is one service for each clause c_j, whose demand DEM_j is 1;
- $\mathcal{A}_{i,j} = 1$ if variable x_i appears in clause c_j and $\mathcal{A}_{i,j} = 0$ otherwise.

If we denote by x the $\{0,1\}$ vector where $x_i = 1$ if and only if machine $\mathcal{M}_i$ is alive, the set of vectors x such that $\forall j, \sum_i \text{ALIVE}_i \geq \text{DEM}_j$ is exactly the set of vectors x such that the original formula is true and all these vectors have exactly the same probability since the failure probability is $\frac{1}{2}$. Hence, computing this set of vectors is $\#P$-Hard and computing the reliability of $\mathcal{A}$, that is exactly the number of such vectors divided by 2^m, is $\#P'$-Complete. ■

D. Integer Solutions

In this section, we propose and study the behavior of two algorithms that compute integer allocations, with the goal of reproducing the good properties of the Proportional Mapping allocations described in Section IV-B.

In order to propose solutions that behave nicely in an on-line environment where services may stop or start over time, we focus our attention on randomized algorithms. Our first algorithm is **FullRandom**, in which all available slots of a given machine are considered separately: machine j is seen as CAPA_j slots of size 1. **FullRandom** mimics the Proportional Mapping scheme by allocating to each service $\mathcal{S}_i$ a number of slots proportional to its demand DEM_i. To achieve this, we cut the $(0,1]$ interval into n consecutive disjoint intervals of size proportional to each service demand DEM_i. Thus $(0,1]$ is cut into intervals $(0, \frac{\text{DEM}_1}{\sum_u \text{DEM}_u}]$, $(\frac{\text{DEM}_1}{\sum_u \text{DEM}_u}, \frac{\text{DEM}_1+\text{DEM}_2}{\sum_u \text{DEM}_u}], \ldots, (\frac{1-\text{DEM}_n}{\sum_u \text{DEM}_u}, 1]$. Then, for each slot we pick uniformly at random a number in $(0,1]$ and use it to define the service to allocate to this slot. Doing this, each slot is used by exactly one service, and is used specifically by service i with probability $r_i = \frac{\text{DEM}_i}{\sum_u \text{DEM}_u}$. The intuition behind this algorithm is that when the demand of the services are large enough, the actual number of slots alloted to service $\mathcal{S}_i$ on machine j is very likely to be very close to $\text{CAPA}_j r_i$, and thus the reliability of this allocation is close to the reliability achieved by Proportional Mapping.

Theorem 4.4: On instances with homogeneous machines (in terms of capacity CAPA and failure rate FAIL), **FullRandom** with a small resource augmentation guarantees to reach the required reliability.

Proof: Let us compute the probability that the allocation computed by **FullRandom** fails to achieve the required demands for each service. For a given service i, its allocation ratio is α_i, so the average number of allocated slots is $n\alpha_i$CAPA. Since machines fail with probability p, the average number of alive instances for service i is $n\alpha_i$CAPAp. Let us denote by $\mathcal{A}_i$ the number of alive instances for service i and by $\mathcal{M}$ the number of alive machines (both are random variables).For given positive ϵ_1 and ϵ_2, we can bound the probability that the actual number of alive slots deviates too much from this average using twice Chernoff inequality [31]:

$$\begin{aligned}
F &= P(\exists i, \mathcal{A}_i < n(\alpha_i - \epsilon_1)\text{CAPA}(p - \epsilon_2)) \\
&\leq P(\mathcal{M} < n(p - \epsilon_2)) + \\
&\quad P(\exists i, \mathcal{A}_i < n(\alpha_i - \epsilon_1)\text{CAPA}(p - \epsilon_2) | \mathcal{M} \geq n(p - \epsilon_2)) \\
&\leq e^{-\frac{1}{2}\epsilon_2 n^2} + \\
&\quad \sum_i P(\mathcal{A}_i < n(\alpha_i - \epsilon_1)\text{CAPA}(p - \epsilon_2) | \mathcal{M} \geq n(p - \epsilon_2)) \\
&\leq e^{-\frac{1}{2}\epsilon_2 n^2} + k e^{-\frac{1}{2}\epsilon_1 n^2 c^2 (p-\epsilon_2)^2}
\end{aligned}$$

By picking $\epsilon_2 \geq \frac{-2\log(r/(k+1))}{n^2}$ and $\epsilon_1 \geq \frac{\epsilon_2}{c^2(p-\epsilon_2)^2}$, we can ensure that

$$F \leq \frac{r}{k+1} + k\frac{r}{k+1} = r$$

Furthermore, if we assume that $\text{DEM}_i = \alpha_i n \text{CAPA} p$ (which is necessary for the original instance to be feasible), we can compute the number β of additional nodes required to compensate for this ϵ error. If we make sure that $(n+\beta)(\alpha_i - \epsilon_1)\text{CAPA}(p - \epsilon_2) \geq d_i$, then F is not smaller than the probability that the demand of at least one service is not satisfied, and since $F \leq r$, we can ensure that all services are satisfied with probability $1-r$. It is easy to see that it is enough to set

$$\beta = n\frac{p\epsilon_1 + \alpha_i\epsilon_2 - \epsilon_1\epsilon_2}{(\alpha_i - \epsilon_1)(p - \epsilon_2)}.$$

When n grows to infinity, both ϵ_1 and ϵ_2 are $O(\frac{1}{n^2})$, and hence $\beta = O(\frac{1}{n})$ is enough to guarantee the required reliability requirement.

■

Another, more deterministic approach is **RoundDown**: on each machine j, **RoundDown** first allocates $\lfloor r_i \text{CAPA}_j \rfloor$ slots to service $\mathcal{S}_i$. Then, the remaining slots (if any) are allocated using the same procedure as **FullRandom**.

To analyze and compare the behavior of these two heuristics, we present experiments in which we compare their performance to the optimal Proportional Mapping strategy. Specifically, we randomly generate instances with various parameter values (described below), and for each instance, we compute the "cost of integrity", *i.e.* how many additional resources are required for **FullRandom** and **RoundDown** to achieve the same level of reliability as Proportional Mapping.

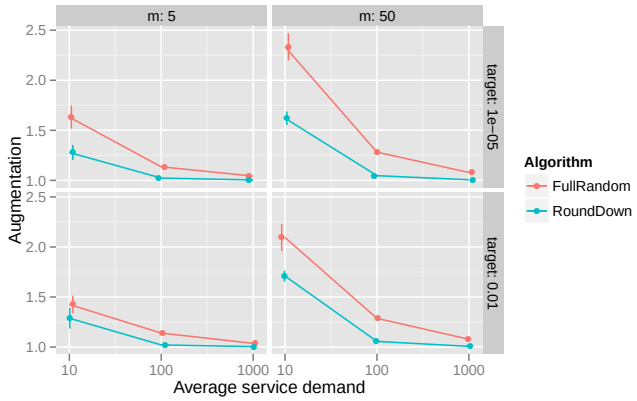

Figure 1. Augmentation ratios with respect to Proportional Mapping required for each algorithm to achieve the target reliability, with $n = 15$, $m = 5$ and 50 services.

Instances are generated as follows: we consider three parameters, namely the average service demand d, the number of services m, and a target reliability REL. In all experiments, the number of machines is fixed to $n = 15$ (since computing the reliability of an allocation takes time 2^n, it is difficult to consider significantly higher values), and the machine failure rates are set to $\text{FAIL} = 0.05$. From these parameters, we can compute an average machine capacity $c = \frac{dm}{n}$. Individual service demands $\mathcal{S}_i$ are then generated as uniform values between $d/2$ and $3d/2$, and similarly machine capacities CAPA_j have uniform values between $c/2$ and $3c/2$. For each instance, we compute for each algorithm $\mathcal{A}$ the smallest augmentation ratio $\alpha_{\mathcal{A}}$ such that the allocation produced by $\mathcal{A}$ when the capacity of all machines is multiplied by $\alpha_{\mathcal{A}}$ achieves the target reliability REL.

The results are depicted on Figure 1, where we plot the augmentation ratio of each algorithm with respect to Proportional Mapping (*i.e.* we plot $\frac{\alpha_{\mathcal{A}}}{\alpha_{\text{PM}}}$). We can see that the average service demand is the most critical parameter: as expected, when the average service demand increases, the performance of both algorithms gets closer and closer to the optimal behavior. **RoundDown** actually reaches this quasi-optimal performance as soon as $d \geq 100$.

The case when $n = 10$ and $n = 18$ are respectively depicted in Figure 2 and Figure 3, and they show very little difference with the case $n = 15$.

V. Case of Several Independent Applications

A. Introduction

In this section, we consider the context where we are given several services to run, each with a target reliability, and the goal is to find an allocation such that the reliability of each service is not smaller than its target.

IndependentServices$(m, \text{CAPA}, n, \text{DEM}, \text{REL})$: Find an allocation $\mathcal{A}$ of instances of services $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n$ to machines $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_m$ such that $\forall i,\ \mathcal{P}(\text{ALIVE}_i \geq \text{DEM}_i) \geq \text{REL}_i$, *i.e.* the probability that a least DEM_i

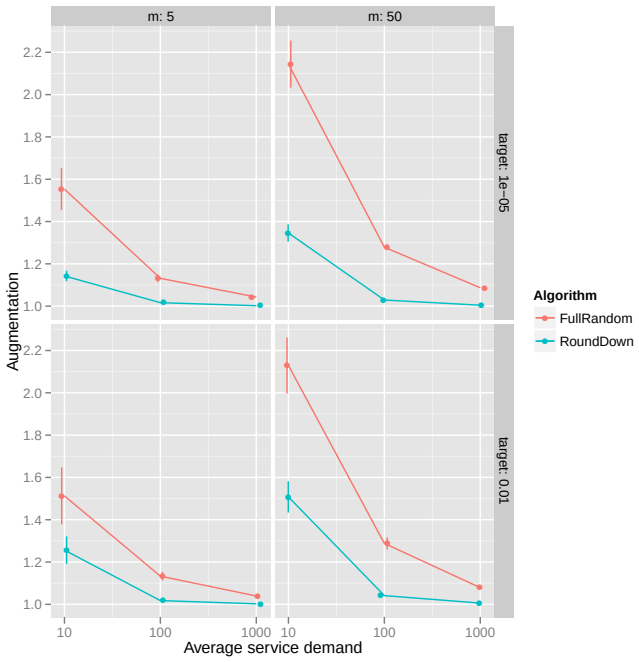

Figure 2. Same as Figure 1 with $n = 10$, $m = 5$ and 50 services.

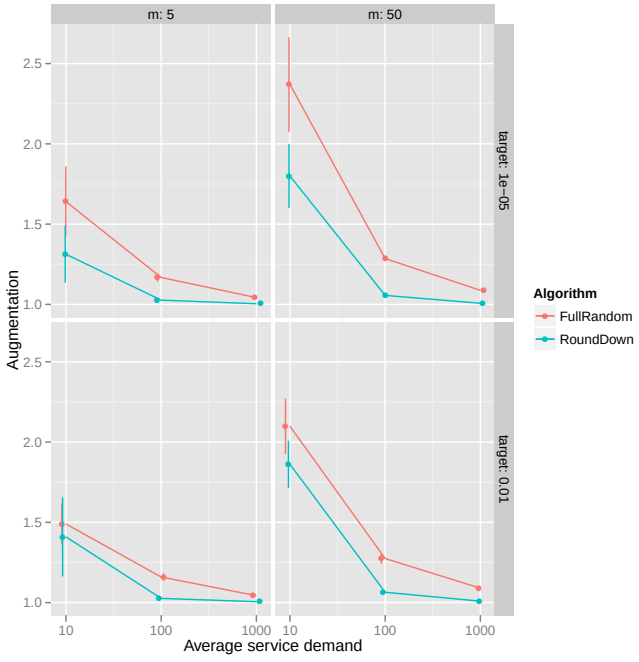

Figure 3. Same as Figure 1 with $n = 18$, $m = 5$ and 50 services.

instances of $\mathcal{S}_i$ are running on alive machines after the time period is larger than the reliability requirement REL_i.

This formulation strongly differs from the grouped case depicted in Section IV. As an example, let us consider a simple instance with two machines with identical capacity $c = 10$ and failure rate p, and two services with demand $d = 10$. An homogeneous allocation, as would be produced by Proportional Mapping, consists in allocating 5 instances of each service on each machine. The reliability of each service in that allocation is $(1-p)^2$ (both machines need to be alive). In contrast, allocating 10 instances of service 1 on machine 1, and 10 instances of service 2 on machine 2 yields a reliability of $1-p$ for each of the services, which is strictly better. Of course, in both cases, the grouped reliability, *i.e.* the probability that both services are satisfied, is $(1-p)^2$.

B. Complexity results

Theorem 5.1: **IndependentServices** is NP-Complete in the Strong Sense.

In this paper, until now, we have only proven #P-Completeness results: for $\mathcal{P}_{OneService}$ and $\mathcal{P}_{GroupedServices}$ (in the fractional case), finding an allocation with optimal reliability is not difficult. However, computing its reliability is a difficult problem, as assessed by #P-Completeness results.

In the case of **IndependentServices**, the problem is more difficult in the sense that computing an optimal allocation turns out to be an NP-complete problem. To prove Theorem 5.1, we will use a reduction to the **3-Partition** problem [10].

3-Partition: Given $3n$ integer numbers $a_1, \ldots, a_{3n}$ and an integer B such that

$$\forall i,\ \frac{B}{4} < a_i < \frac{B}{3} \text{ and } \sum_i a_i = nB,$$

is there a partition of $\{1, \ldots, 3n\}$ into n groups of exactly 3 elements, such that each group sums up exactly to B?

Proof: In order to prove the NP-Completeness of **IndependentServices**, let us consider the following set of instances, that consist of $3n$ services $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{3n}$ to be run on n machines $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n$. Let a_i denote the demand of $\mathcal{S}_i$. At last, let both the failure probability of a machine and the reliability requirement for any service be p.

- Let us suppose that there exists a solution to the **3-Partition** instance. Then, let us allocato each group of a_is summing up to B to a single machine. Then, any service runs on exactly one machine whose reliability is p, and therefore, its reliability requirement is satisfied.
- Let us suppose that there exists a solution to the **IndependentServices** instance.
  Let us first remark that these instances are tight, in the sense that the overall demand of the services is exactly the capacity of the machines. In this case, if replication is used for one of the services,

then another one will not reach its demand, and its reliability will become $0 < p$. Therefore, no replication is allowed.

Let us now prove that a service can be run on at most one machine. Indeed, since no replication is allowed, if a service runs on $k > 1$ machines, then it will fail if any of the k machines fails, *i.e.* with probability $p^k < p$. Therefore, each service is associated to exactly a_i instances running on a single machine. Since $\forall i,\ \frac{B}{4} < a_i < \frac{B}{3}$ and since the whole capacity of each machine must be used (instances are tight), then each machine must hold exactly 3 services whose demands sum up to exactly B.

This achieves the proof of NP-Completeness of **IndependentServices**. ■

In fact, above proof enables us to state a more powerful result

Corollary 5.2: **IndependentServices** is not in APX.

Proof: Indeed, above proof states that for considered instances, it is impossible to determine in polynomial time (unless P=NP) if each service can be allocated to exactly 1 machine. Moreover, if a service is allocated to more than 1 machine, its reliability probability will drop by a factor $1-p$ (at least from $1-p$ to $(1-p)^2$). Therefore, determining if the best achievable reliability is $1-p$ or $(1-p)^2$ is NP-Complete in the strong sense, thus achieving the proof of the corollary. ■

C. Approximation Algorithm based on Resource Augmentation for Tight Instances

Nevertheless, in the case of tight instances, *i.e.* instances where the overall demand is exactly the overall capacity of the machines, and where all machines have the same failure probability p (but machines can have heterogeneous capacities and services can have heterogeneous reliability demands $(1-p)^k$, for any k, thus more general ones than those used in above negative result), we can obtain an approximation algorithm based on resource augmentation on the reliability of the services, by adapting the algorithm SEQ proposed in [7] in the context of client-server allocations.

The algorithm SEQ takes as input a set of servers SERV_i with heterogeneous bandwidth capacities b_i^{serv} and maximal degree d_i (the degree being the maximal number of services that can be connected simultaneously to SERV_i) and list of clients with their bandwidth demands b_j^{client}. The goal is to allocate clients to servers (a client can be allocated to several servers) so as to satisfy both capacity and degree constraints. Above problem is NP-Complete [7] but if such an allocation exist, then SEQ finds a valid allocation where the degree of servers is at most $d_i + 1$.

In order to adapt SEQ to the context of **IndependentServices**, let us associate each service $\mathcal{S}_i$ with demand DEM_i and reliability $\text{REL}_i = (1-p)^{k_i}$ to a server SERV_i with bandwidth capacity $b_i^{\text{serv}} = \text{DEM}_i$ and maximal degree $d_i = k_i$, and each machine with capacity CAPA_j to a client with bandwidth demand $b_j^{\text{client}} = \text{CAPA}_j$.

Then, if there exists a valid allocation, SEQ provides an allocation where the degree of the servers is at most $d_i + 1$, *i.e.* where the reliability of service $\mathcal{S}_i$ is at least $(1-p)^{k_i+1}$ (instead of $(1-p)^{k_i}$).

D. Heuristics and Simulation Experiments

In order to address the (more realistic) case of non tight instances, we have designed several heuristics for this problem, and more specifically for the case where machines are homogeneous (same capacity and same reliability). On the other hand, services can be heterogeneous, both in terms of demand and reliability requirements. In order to make it easier to compare the solutions, we formulate the problem as a minimization problem: given the demand and reliability requirements for the services, and given the number of machines, the output for each heuristic is the minimal capacity of the machines needed to find a correct allocation. Therefore, a smaller capacity means that the heuristic is more efficient.

Throughout this section, we will use the following instance as a toy example to explain the behavior the algorithm. The instance consists of 4 machines with capacity 7 and failure rate 0.1 and 2 services, $\mathcal{S}_1$ with demand 10 and reliability 0.9892 and $\mathcal{S}_2$ with demand 8 and reliability 0.81.

In what follows, we propose two types of heuristics:

- randomized heuristics based on the ideas developed in Section IV
- and deterministic heuristics.

Several heuristics are based on the (pre-)computation of the $P_{x,y}$ values, where $P_{x,y}$ is the probability that among y machines of failure rate p, at least x of them are alive. In fact, $P_{x,y} = \sum_{i=x}^{y} \binom{y}{i}(1-p)^i p^{y-i}$, and these n^2 values can actually be precomputed once n and p are known. $P_{x,y}$ values will be used in the heuristics to determine valid allocations, as stated in the following lemma.

Lemma 5.3: If $P_{x,y} \geq \text{REL}_i$, then assigning $\lceil \frac{\text{DEM}_i}{x} \rceil$ instances of service $\mathcal{S}_i$ to any set of y machines is enough to satisfy its demand and reliability constraints.

Proof: By construction, the probability that at least x machines are alive at the end of the period is at least $P_{x,y} \geq \text{REL}_i$, and the number of instances on these x machines is $\lceil \frac{\text{DEM}_i}{x} \rceil \times x \geq \text{DEM}_i$, what achieves the proof. ■

For our toy example, the set of valid pairs are $\{(2,4);(1,4);(1,3);(1,2)\}$ for $\mathcal{S}_1$ and $\{(3,4);(2,4);(1,4);(2,3);(1,3);(2,2);(1,2);(1,1)\}$ for $\mathcal{S}_2$.

1) Deterministic Heuristics:

- The **Homogeneous Allocation Heuristic** produces only homogeneous allocations. The same number of instances is allocated to a service on all n machines. For each service $\mathcal{S}_i$, we compute the largest value

k_i such that $P_{k_i,n} \geq \text{REL}_i$ and we set

$$\mathcal{A}_{i,j} = \lceil \frac{\text{DEM}_i}{k_i} \rceil$$

for each machine j. Lemma 5.3 ensures that resulting allocation is valid.

In the case of our toy example, $\lceil \frac{10}{2} \rceil = 5$ instances of $\mathcal{S}_1$ are allocated on each server and $\lceil \frac{8}{3} \rceil = 3$ instances of $\mathcal{S}_2$ are allocated on each server.

- The **Packing Heuristic** produces semi-homogeneous allocations, in the sense that, for any service $\mathcal{S}_i$, the number of allocated instances is either 0 or $\mathcal{A}_i$ (that does not depend on the processor j).
  - In the first step of the **Packing Heuristic**, we choose the pair (x_i, y_i) that minimizes the number of necessary instances among all possible semi-homogeneous allocations, *i.e.* among all valid pairs (x, y) such that $P_{x,y} \geq \text{REL}_i$, we choose the one that minimizes

    $$\lceil \frac{\text{DEM}_i}{x} \rceil \times y.$$

    In the case of our toy example, allocating 5 instances to all 4 machines or 10 instances to 2 machines are equivalent for $\mathcal{S}_1$, and allocating 8 instances of $\mathcal{S}_2$ to 1 machine or 4 to two machines is equivalent for $\mathcal{S}_2$. In both cases, we favor the more "spread out" allocations.
  - In the second step, we greedily allocate services to machines in non-increasing order of $\mathcal{A}_i = \lceil \frac{\text{DEM}_i}{x_i} \rceil$. Then, $\mathcal{A}_i$ instances of $\mathcal{S}_i$ are allocated to the y_i least loaded machines.

    In the case of our toy example, we therefore end up with 4 instances of $\mathcal{S}_2$ on Machines 1 and 2, and 5 instances of $\mathcal{S}_1$ on all machines. Because of the imbalance between the machines, this solution actually requires more capacity (9) than the completely homogeneous allocation.

The toy example used in this section also illustrates an interesting result: semi-homogeneous allocations are not a dominant class of allocations for this problem. Indeed, the reliability requirement of $\mathcal{S}_1$ can be reached by allocating 7 instances of $\mathcal{S}_1$ on two machines, and 3 on the two other machines. This leaves 4 instances available on two machines, which allows $\mathcal{S}_2$ to achieve reliability 0.81. However, there is no semi-homogeneous allocation that achieves reliability requirements on machines with capacity 7.

2) Random Heuristics: We also propose three randomized heuristics. For each heuristic, we first determine the ratio $\alpha_i \in [0, 1]$ of instances that should be allocated to each service $\mathcal{S}_i$. Then, the interval $[0, 1]$ is decomposed in pieces of length proportional to the α_is and, as in Section IV-D, for each slot we pick a random uniform number in $[0, 1]$ and assign that slot to the corresponding service. These three heuristics only differ in the definition of α_i.

- In the **Randomized Demand Heuristic**, α_i is proportional to the relative demand of $\mathcal{S}_i$, *i.e.*

  $$\alpha_i = \frac{\text{DEM}_i}{\sum_k \text{DEM}_k}.$$

- In the **Randomized Demand-Log Heuristic**, α_i is proportional to the relative demand of $\mathcal{S}_i$, weighted by its reliability constraints. More precisely, we set

  $$\alpha_i = \frac{\text{DEM}_i \log(\frac{1}{1-\text{REL}_i})}{\sum_k \text{DEM}_k \log(\frac{1}{1-\text{REL}_k})}.$$

- The **Randomized Allocation Heuristic** is based on $P_{x,y}$ values. For each service $\mathcal{S}_i$, as in the **Packing Heuristic**, we compute the pair (x_i, y_i) that minimizes the number of necessary instances among all possible semi-homogeneous allocations. Then, we set

  $$\alpha_i = \frac{\text{DEM}_i \frac{x_i}{y_i}}{\sum_k \text{DEM}_k \frac{x_k}{y_k}}.$$

At last, all three heuristics take as an additional parameter the machine capacity CAPA, and a binary search over this parameter is performed to identify the smallest value of CAPA such that the heuristic outputs a valid solution (the reliability of each service is computed using the dynamic programming algorithm described in Section III-D).

3) Experimental evaluation: To evaluate the behavior of those heuristics, we generate random instances in the following way: we first fix the number of machines n to 100 and their failure rate $p = 0.05$ (tests with larger values of n yield the same conclusions).

Then, we consider two different parameters: the number of services and the distribution of the reliability requirement for the different services (see Figure 4).

- The number of services m can be either 5, 10, 50 or 250. Once m is fixed, the number of instances associated to each service is chosen uniformly at random in $[\frac{1}{2}\frac{10n}{m}, \frac{3}{2}\frac{10n}{m}]$, so that the average load per machine is $\frac{m}{n}\frac{10n}{m} = 10$, throughout all the simulations.
- Different probability distributions for service reliability are considered.
  - in Constant p_1 (resp. Constant p_5), all services have the same reliability requirement $1 - 10^{-1}$ (resp. $1 - 10^{-5}$).
  - in Bi-valued, the reliability associated to a service is either very low $1 - 10^{-1}$ or very high $1 - 10^{-5}$ (both with probability $\frac{1}{2}$).
  - in Uniform, for each service, an integer i is chosen uniformly at random in $[1, 5]$ and the reliability of the service is chosen as $1 - 10^{-i}$.

Therefore, each entry in Figure 4 is associated to a couple (m,reliability distribution). For each such couple, and each heuristic, we compute (using binary search again on the size of the machines) the capacity of the machines that is necessary in order to enforce the reliability

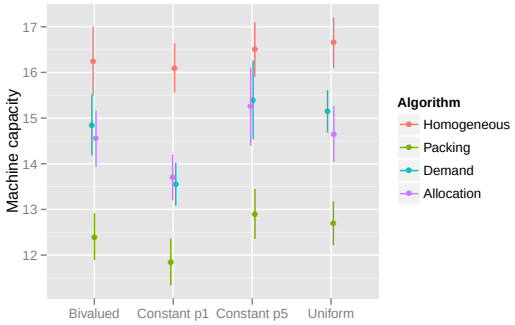

Figure 5. Machine capacity required for each algorithm for $m = 10$ and for different distributions of service reliability requirements.

constraint for each service. Each entry corresponds to the average value of the capacity with error bars (for 20 experiments). Therefore, the smaller the capacity is, the better is the allocation scheme, since it requires less resources to enforce the same reliability level.

We can see that when the number of services is low, most heuristics behave similarly, and only **Demand-Log** requires a much larger machine capacity. Actually, **Demand-Log** gives a very (too) high priority to services with high reliability requirements, and thus needs a large capacity to allocate enough instances for the other services. The **Homogeneous** heuristic is forced by design to allocate at least one instance of each service on each machine. Hence it requires a machine capacity of at least m, and when m becomes large, the minimum capacity of each machine also becomes large.

The behaviors of the **Demand** and **Allocation** random heuristic are very close to each other; actually the $\frac{x_i}{y_i}$ values are very close to 1, so that both heuristics use very similar α_i values. Similarly to what happens in the grouped case described in Section IV-D, when the average demand of the services is small (*i.e.* when m is large), these random heuristics need a larger capacity to reach the reliability requirements. They still largely outperform **Homogeneous** even for low values of m.

Finally, the performance of **Packing** is very good, even for high values of m. When the reliability requirement is low (see Constant p_1 with reliability = 0.9), then services are fully allocated on a single machine, what is optimal in terms of overall capacity. Then, a small resource augmentation is required when the demand of each service is small (*i.e.* when m is large) since the associated bin packing problem is easy (many small items) and more difficult when the demand is large (*i.e.* when m is small) since there are few larger items to pack.

A more detailed view of the previous graph, restricted to $m = 10$, is shown on Figure 5 and illustrates the fact that even for small values of m, random heuristics outperform **Homogeneous** and that **Packing** is clearly the best heuristic.

VI. Conclusion

In this paper, we considered several variants of service allocation problems in Cloud platforms under reliability constraints, and we analyzed their complexity. In order to assess precisely the difficulty introduced by the reliability constraints, we considered a simplified setting, where applications are services running as homogeneous independent instances and we considered the static allocation problem only, where the allocation is computed once for a given time period. On the other hand, we considered 3 different situations (one single service, several grouped services or several independent services), that correspond well to the typology of situations in Clouds. For each problem, we proved complexity results (NP-Completeness or #P'-Completeness depending on the context), we provided optimal or approximation algorithms for special cases and we proposed a set of heuristics for the most general settings, that we compared through extensive simulations.

An important conclusion of this paper is that although reliability introduces an extra complexity, several heuristic achieve good performance, especially in the case of large instances, *i.e.* the most realistic context for Cloud platforms. Moreover, many of the algorithms that we propose are randomized, and could therefore easily be adapted to more dynamic settings. At last, this paper opens many perspectives. For instance, the model of failures could be extended to non-independent failures (with massive failures occurring at the level of one rack or one site for instance), to more dynamic settings where task migration and task creation could be used to enforce the reliability constraint, and to services consisting of non-homogeneous memory footprint instances.

References

[1] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[2] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "Above the clouds: A berkeley view of cloud computing," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009.

[3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599 – 616, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X08001957

[4] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, Dec. 2008. [Online]. Available: http://doi.acm.org/10.1145/1496091.1496103

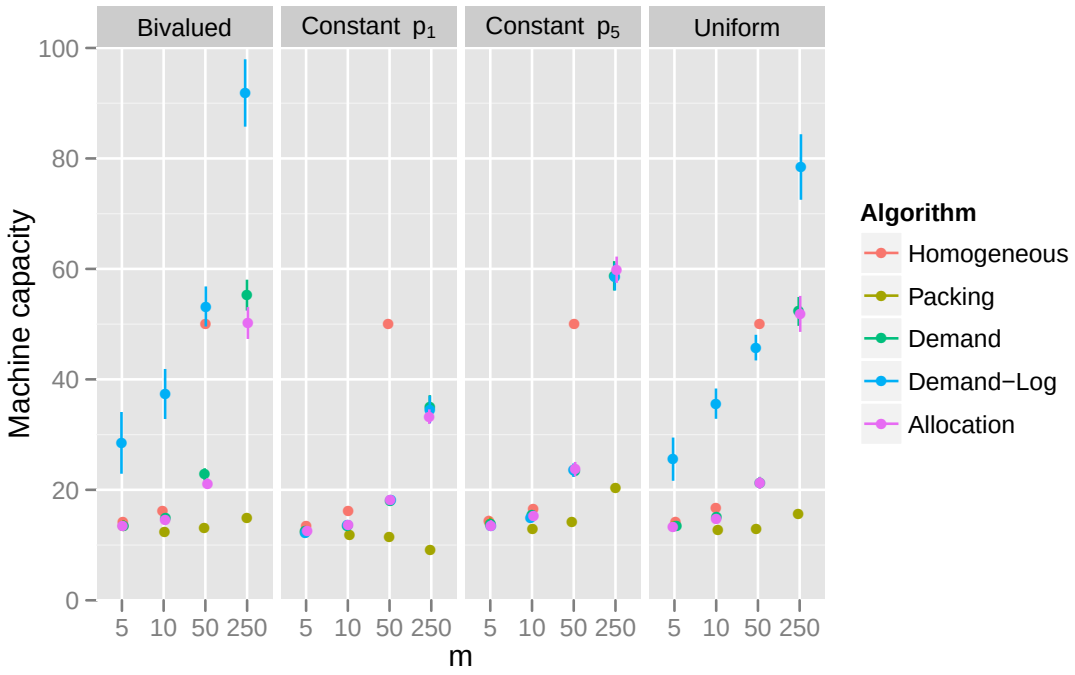

Figure 4. Machine capacity required for each algorithm for various values of m and different distributions of service reliability requirements.

[5] H. Van, F. Tran, and J. Menaud, "SLA-aware virtual resource management for cloud infrastructures," in *IEEE Ninth International Conference on Computer and Information Technology*. IEEE, 2009, pp. 357–362.

[6] R. Calheiros, R. Buyya, and C. De Rose, "A heuristic for mapping virtual machines and links in emulation testbeds," in *2009 International Conference on Parallel Processing*. IEEE, 2009, pp. 518–525.

[7] O. Beaumont, L. Eyraud-Dubois, H. Rejeb, and C. Thraves, "Heterogeneous Resource Allocation under Degree Constraints," *IEEE Transactions on Parallel and Distributed Systems*, 2012.

[8] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Dang, and K. Pentikousis, "Energy-efficient cloud computing," *The Computer Journal*, vol. 53, no. 7, p. 1045, 2010.

[9] A. Beloglazov and R. Buyya, "Energy efficient allocation of virtual machines in cloud data centers," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE, 2010, pp. 577–578.

[10] M. R. Garey and D. S. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[11] L. Epstein and R. van Stee, "Online bin packing with resource augmentation." *Discrete Optimization*, vol. 4, no. 3-4, pp. 322–333, 2007.

[12] D. Hochbaum, *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997.

[13] W. Cirne, "Scheduling at google," in *16th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), in conjunction with IPDPS 2012*, 2011.

[14] K. Ranganathan, A. Iamnitchi, and I. Foster, "Improving data availability through dynamic model-driven replication in large peer-to-peer communities," in *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on*, may 2002, p. 376.

[15] D. da Silva, W. Cirne, and F. Brasileiro, "Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids," in *Euro-Par 2003 Parallel Processing*, ser. Lecture Notes in Computer Science, H. Kosch, L. Böszörményi, and H. Hellwagner, Eds. Springer Berlin / Heidelberg, 2003, vol. 2790, pp. 169–180.

[16] M. Lei, S. V. Vrbsky, and X. Hong, "An on-line replication strategy to increase availability in data grids," *Future Generation Computer Systems*, vol. 24, no. 2, pp. 85 – 98, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X07000830

[17] H.-I. Hsiao and D. J. Dewitt, "A performance study of three high availability data replication strategies," *Distributed and Parallel Databases*, vol. 1, pp. 53–79, 1993, 10.1007/BF01277520. [Online]. Available: http://dx.doi.org/10.1007/BF01277520

[18] E. Santos-Neto, W. Cirne, F. Brasileiro, and A. Lima, "Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Springer Berlin / Heidelberg, 2005, vol. 3277, pp. 54–103.

[19] J. Dongarra, P. Beckman, P. Aerts, F. Cappello, T. Lippert, S. Matsuoka, P. Messina, T. Moore, R. Stevens, A. Trefethen *et al.*, "The international exascale software project: a call to cooperative action by the global high-performance community," *International Journal of High Performance Computing Applications*, vol. 23, no. 4, pp. 309–322, 2009.

[20] "Eesi, "the european exascale software initiative", 2011," http://www.eesi-project.eu/pages/menu/homepage.php.

[21] F. Cappello, "Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities," *International Journal of High Performance Computing Applications*, vol. 23, no. 3, pp. 212–226, 2009.

[22] K. Ferreira, J. Stearley, J. Laros III, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. Bridges, and D. Arnold, "Evaluating the viability of process replication reliability for exascale systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 44.

[23] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien, "Checkpointing strategies for parallel jobs," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*. IEEE, 2011, pp. 1–11.

[24] F. Cappello, H. Casanova, and Y. Robert, "Checkpointing vs. migration for post-petascale supercomputers," *ICPP'2010*, 2010.

[25] L. Valiant, "The complexity of computing the permanent," *Theoretical Computer Science*, vol. 8, no. 2, pp. 189 – 201, 1979. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0304397579900446

[26] H. Bodlaender and T. Wolle, "A note on the complexity of network reliability problems," *UU-CS*, no. 2004-001, 2004.

[27] L. Valiant, "The complexity of enumeration and reliability problems," *SIAM J. Comput.*, vol. 8, no. 3, pp. 410–421, 1979.

[28] J. Provan and M. Ball, "The complexity of counting cuts and of computing the probability that a graph is connected," *SIAM Journal on Computing*, vol. 12, p. 777, 1983.

[29] A. Benoit, L.-C. Canon, E. Jeannot, and Y. Robert, "Reliability of task graph schedules with transient and fail-stop failures: complexity and algorithms," *Journal of Scheduling*, pp. 1–13, 10.1007/s10951-011-0236-y. [Online]. Available: http://dx.doi.org/10.1007/s10951-011-0236-y

[30] P. Gopalan, A. Klivans, R. Meka, D. Stefankovic, S. Vempala, and E. Vigoda, "An fptas for #knapsack and related counting problems," in *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, oct. 2011, pp. 817 –826.

[31] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *The Annals of Mathematical Statistics*, vol. 23, no. 4, pp. 493–507, 1952.