Cost Aware Fault Recovery in Clouds

Assaf Israel and Danny Raz
Technion Institute of Technology, Israel
{assafi,danny}@cs.technion.ac.il

***Abstract*—Maintaining high availability of IaaS services at a reasonable cost is a challenging task that received recent attention due to the growing popularity of Cloud computing as a preferred means of affordable IT outsourcing. In large data-centers faults are prone to happen and thus the only reasonable cost-effective method of providing high availability of services is an SLA aware recovery plan; that is, a mapping of the service VMs onto backup machines where they can be executed in case of a failure. The recovery process may benefit from powering on some of these machines in advance, since redeployment on powered machines is much faster. However, this comes with an additional maintenance cost, so the real problem is how to balance between the expected recovery time improvement and the cost of machines activation.**

We model this problem as an offline optimization problem and present a bicriteria approximation algorithm for it. While this is the first performance guaranteed algorithm for this problem, it is somewhat complex to implement in practice. Thus, we further present a much simpler and practical heuristic based on a greedy approach. We evaluate the performance of this heuristic over real data-center data, and show that it performs well in terms of scale, hierarchical faults and variant costs. Our results indicate that our scheme can reduce the overall recovery costs by 10-15% when compared to currently used approaches. We also show that fault recovery cost aware VM placement may farther help reducing the expected recovery costs, as it can reduce the backup machine activations costs.

I. Introduction

Recent advances in virtualization technologies and in data center management systems accelerated the deployment of variety of applications on shared computing environments (also known as clouds). The main motivation behind large scale resource sharing is the reduction of costs, which is achieved by lower infrastructure cost as well as by high resources utilization. In addition, cloud providers are continuously motivated to provide higher levels of VM availability, forcing heavy investments in hardware acquisitions and machine maintenance. While more expensive hardware reduces the chance of hardware faults, it does not eliminate it. Moreover, it does not protect against software or human errors. As a result, the de-facto assumption in cloud environments is that faults are bound to happen and it is up to the cloud operator to ensure a certain level of service availability in the present of failures.

Techniques of achieving high availability are usually divided into active/active or active/inactive techniques [1]. While active/active solutions like Xen Remus, VMware FT and KVM Kemari [2]–[4] may help protecting mission critical services, they require additional resources for maintaining shadow replicas of the protected instances. On the other hand active/inactive solutions like VMware HA [3] simply recover failed VMs from a backed up image onto a powered backup machine. While this technique provides recovery at reasonable speed for most services, it too does not take into consideration system wide operational costs to the provider. For example, a cloud provider may wish to implement a rapid VM recovery solution to protect against a a rack failure of 40-80 hosts. While the cold recovery solution may indeed help speed up the recovery process of the hosted VMs, it may also require an idle backup rack, raising the total maintenance costs considerably.

Powering off the backup machines reduces the operational costs, but may incur considerable delay when recovery is needed (i.e., when fault occurs). Barroso & Hölzle [5] analyzed one of Google's data-centers and showed that while 55% of boot events took less then 6 minutes, 25% of them took between 6 and 30 minutes with a total average of just over 3 hours. In their analysis, they related the long boot times to several causes, including file system integrity checks, machines hangs that requires semiautomatic boot process, software installation and testing. Furthermore, these techniques do not distinguish between VMs with different SLA requirements (in terms of failure tolerance). As some VMs may be more susceptible to failures, it is reasonable to prioritize their recovery to further minimize the expected cost caused due to SLA breaches.

In this paper we study the trade-off between the SLA related cost associated with the recovery process (termed VM recovery costs) and the maintenance cost of having powered-on back up machines (termed active cost). We define the *VM Recovery Problem (VMRP)* as a formal optimization problem that provides an optimal recovery solution taking into account both the cost of keeping machines active as well as the VM recovery cost, and observe that this problem is in NP-hard. We then develop a novel LP-based bicriteria approximation algorithm for this problem and formally prove that it has a guaranteed performance bound of $O(\log \frac{n+m}{OPT} + 1))OPT$ in terms of cost while allowed a factor of $(6 + \varepsilon)$ in terms of machines load. To the best of our knowledge this is the first theoretical proven approximation algorithm for this problem.

Our theoretical algorithm is somewhat complex and it is not practical to implement, so we turn to develop a more practical heuristic that can be easily implemented. We evaluate this approach using simulations based on data from one of IBM's research data centers and study how the performance depends on the container level (e.g., machine level or rack level) and on the VM SLAs. The simulation results indicate that this

This research was partly supported by the Israel Science Foundation (grant no. 1129/10)

heuristic approach can reduce the overall cost by 10-15% in many practical scenarios.

Moreover, this cost reduction also depends on the initial placement of the VMs in the data center, thus we also demonstrate that recovery cost aware VM placement can further reduce the cost of VM recovery. As mentioned before, efficient resource utilization is one of the key ingredients in the attractiveness of cloud computing. The main contribution of this paper is the development of both theoretically proven and practical algorithms that can be used by cloud providers to offer affordable high availability of services.

The paper is structured as follows. In Section III we define the *VM Recovery Problem (VMRP)* as an optimization problem. We present the bicriteria approximation algorithm and a proof of its bounds in Section IV, then in Section V we present our greedy heuristic and the simulation based evaluation. Finally, in Section VI we present the role of VM placement, and in Section VII we briefly discuss our conclusions.

II. Background & Related Work

During the last decade, advancements in the field of system management enabled increasing the efficiency of large data center. New management systems [6], [7] elevated hardware abstractions in the form of *OS Virtualization* and included monitoring capabilities that enabled secure, adaptive, on-demand resources provisioning in data-centers. Housing applications in Virtual Machines (VM) instances helped decouple applications from the underlying data center hardware, while advanced *Server Consolidation* techniques [8], [9] helped running the data center infrastructure at high utilization and thus bringing operating costs down significantly.

Recently, many cloud providers started offering high-availability commitments as part of their standard or premium priced IaaS offerings. Some even commit for 100% availability [10]–[12], refunding the client a percentage of his investment for any service downtime period. These SLA models emphasis the importance of offering highly available services at a cost effective manner from the cloud provider standpoint.

In order to achieve a complete fault tolerant solution, all of the application resources usually needs to be replicated across availability zones, including the application instance itself. Addis et al. presented in [13] a resource allocation model that takes into consideration availability constraints. While their model aimed at maximizing the total profit, based on the differences between the SLA profits and the total machine costs, they also guaranteed certain levels of availability based on the services SLAs. The availability guarantees in their model were achieved by balancing between machine costs and the number of instance replications for each VM. In an effort to better profile application components and tailor the FT solution, Zheng et al. [14] devised a framework that ranks service components according to their significance and suggest a fault tolerant solution for each component based on its ranking. While this approach proved useful in simulations, it does not consider the infrastructure costs of the advised solution. FT solutions were assumed to be independent, and as such a component FT solution did not affect the cost or the feasibility on the next. Moreover, both of the above approaches considered only the active/active fault tolerant technique as means of achieving high levels of availability, ignoring the less expensive and yet effective active/inactive approach.

These methods save a periodical snapshot of an application state to disk. This enables restoring an application to its prior state in case of a disaster [15]. Using advanced deduplication techniques [16] it is possible to reduce the storage footprint of the VM snapshots using a small amount of cloud resources, making this solution highly cost effective. Finally, proactive techniques like those suggested by Nagarajan et al [17] use VM live migration from faulty machines prior to the actual fault. These techniques guaranty virtually no downtime assuming the health prediction mechanisms are effective.

III. Model

Let $H = (1, 2, \ldots, M)$ be the set of physical machines (PMs) that are currently available in the data center and let $X = (x_1, x_2, ..., x_M)$ be the indicators vector specifying which host is currently powered on $(x_i = 1)$ and which is powered off $(x_i = 0)$. The cost associated with keeping host i powered on is denoted by Cost-H$_i$. This cost may include power, maintenance and additional personnel costs that are caused due to the host activation.

The set of virtual machines $V = (1, 2, \ldots, N)$ which are currently maintained by the cloud provider is initially assigned to H according to a placement schema O which is described by the following indication matrix:

$$o_{i,j} = \begin{cases} 1 & \text{If VM } j \text{ is assigned to host } i \\ 0 & \text{Otherwise} \end{cases} \tag{3.1}$$

Once a physical host had failed, all of its resident VMs are unavailable until they are redeployed on another physical host, or until the failing host has recovered. The cost associated with the failure of the physical host is derived from the SLA of the guest VMs which are now unavailable, and the total time in which their service is denied.

Let $U = \{U_1, U_2, \ldots, U_k\}$ be a partition of the physical hosts to k *Virtual Components* (VC). We consider every VC as a potential faulty unit (e.g., rack, network segment) that is independent of other VCs, and assuming that at any point in time only one VC can fail.

Upon a VC failure, every one of it's guest VMs must be redeployed to some PM that is outside of the failed VC. A VM can be redeployed to an active host, thus shortening the expected down-time and lowering the expected cost to the customer. However, keeping a host active incurs maintenance and power costs [5] which can negate the gain of speeding up the VM recovery. Thus, it is important to balance between reducing VM recovery costs and active PM costs. Let Cost-V$_{i,j}(x_i)$ be the recovery cost of the j^{th} VM destined to the i^{th} host given the host's activation indicator x_i. These costs can be derived from various parameters such as VM

SLAs, redundancy levels, image profiles, expected VM, host boot times, etc. We assume that the recovery cost of VM j to an active host i is at most the cost of recovery of j to an inactive i, i.e. $\forall i \in H, j \in V \quad \text{Cost-V}_{i,j}(1) \leq \text{Cost-V}_{i,j}(0)$.

Next, we define the *Recovery Plan* that specifies the re-deployment targets of failed VMs. Let M be the recovery plan matrix (defined similarly to O) so that in case U_r had failed, after the recovery process will end, the new VM-to-host mapping O^r will be defined as follows:

$$o^r_{i,j} = \begin{cases} m_{i,j} & \text{If } \exists l (l \in U_r \wedge o_{l,j} = 1) \\ o_{i,j} & \text{Otherwise} \end{cases} \tag{3.2}$$

This means that only guest VMs of the faulty VC will be redeployed to their target physical machine (PM) according to the recovery plan M while other VMs will not be affected and will remain in their original placement. Note that we refrain from performing farther reshuffling of VMs, as this may impose farther costs that are hard to model and predict.

Finally, let $Y = (y_1, y_2, ..., y_M)$ be the *Activation Schema* of hosts following the recovery plan calculation, that is, $y_i = 1$ if and only if PM i is powered on.

IV. Approximating VM Recovery with Host activation costs

In order to specify a recovery plan for VMs and an activation schema for the hosts we use a variation of the algorithm presented by Khuller et al [18], with adjustments to suit our model where machines can have two distinct states (active & inactive), and VMs are partitioned according to their original placement and their respective VCs. Khuller et al. addressed the problem of scheduling jobs to machines with the goal of minimizing the aggregated cost that includes machine activation costs and job assignment costs. In their model, a collection of m machines (M) and n jobs (J) is given, where the processing time of job j on machine i is $p_{i,j}$. Each machine i has an activation cost a_i, and each assignment of job j to machine i incurs an assignment cost $c_{i,j}$. The objective is to assign jobs only to active machines with the minimal overall cost (machines activation and jobs assignment costs) while keeping maximum makespan smaller or equal to some constant T.

While Khuller et al. considered simple capacity constraints, in the recovery version of the problem these constraints are too strict, since one host can be the recovery destination of several VMs originating from different VCs. Also, as oppose to the job assignment model presented in [18] our model permits the assignment of jobs (VMs) to inactive machines (PMs) at higher costs. To overcome this, we extend the set of hosts to include an additional set of "inactive" representations of the backup hosts $H^{\text{off}} = \{i' : i \in H \wedge x_i = 0\}$. Thus, currently-inactive PMs have two representations in our model. The "active" representation with a positive activation cost and lower VM assignment costs, and the "inactive" representation with a zero machine activation cost but with higher VM assignment costs. For simplicity, let $\bar{H} = H \cup H^{\text{off}}$ and $\bar{V} = V$ be the extended sets of PMs and VMs.

Let a_i be the activation cost of PM $i \in \bar{H}$, defined as follows:

$$a_i = \begin{cases} \text{Cost-H}_i & (i \in H) \wedge (x_i = 0) \\ 0 & (i \in H^{\text{off}}) \vee (x_i = 1) \end{cases} \tag{4.1}$$

Note that the activation cost of currently active machines is unimportant as we assume the decision of powering them has already been made by other processes, such as VM placement procedures or previous VM migrations that caused the current VM assignment and activation scheme. Thus the usage of vacant resources in currently active PMs as part of a recovery process does not incur farther activation costs for those PMs.

Let $c_{i,j}$ be the assignment cost of VM j on PM i, defined as follows:

$$c_{i,j} = \begin{cases} \text{Cost-V}_{i,j}(1) & i \in H \\ \text{Cost-V}_{i,j}(0) & i \in H^{\text{off}} \end{cases} \tag{4.2}$$

For each VM j and PM i we define $p_{i,j}$ to be the normalized size of j compared to PM's i free capacity. For every host $i' \in H^{\text{off}}$ and VM j, $p_{i',j}$ is equal to the normalized size of j in i' counterpart i, $p_{i,j}$.

Finally, we can express the VMRP as an Integer Program:

$$\begin{aligned} & \min \sum_{i \in \bar{H}} a_i y_i + \sum_{(i,j)} c_{i,j} m_{i,j} \\ s.t. \quad & \sum_{i \in \bar{H}} m_{i,j} = 1 \quad \forall j \in \bar{V} \\ & m_{i,j} \leq y_i \quad \forall i \in \bar{H}, j \in \bar{V} \\ & \sum_{j \in \bar{V}^r} o_{r,j} m_{i,j} p_{i,j} \leq y_i \quad \forall U_r \in U, i \in \bar{H} \\ & y_i + y_{i'} \leq 1 \quad \forall i' \in H^{\text{off}} \\ & y_i \in \{0,1\}, \quad m_{i,j} \in \{0,1\} \quad \forall i \in \bar{H}, j \in \bar{V} \end{aligned} \tag{4.3}$$

Where $\bar{V}^r$ denote the guest VMs of the U_r VC.

The first set of constraints ensures that each VM will be assigned exactly once. The second set of constraint ensures that VMs will only be assigned to marked machines. Note that the activation schema is derived from the identity of the marked machines (as described in Section IV-D). The third set of constraints is the capacity constraints on the machines, which bound the assignment of VMs, originating from the same VC, according to their target's capacity. The final set of constraints ensures that only one representative (active or inactive) of every host will be considered for the activation schema.

Next, we relax the IP into a LP by relaxing y_i and $m_{i,j}$ to be in the range $[0,1]$. This LP can be solved in polynomial time while achieving a fractional solution $\bar{m}, \bar{y}$ which we process and finally round in order to achieve an approximated discrete solution. Our rounding approach achieves an approximation factor of $O(\log \frac{n+m}{OPT} + 1)$ for the recovery cost while allowing a $(6+\varepsilon)$ factor for the PM's capacity. We start by solving the relaxed LP (Eq. 4.3). Let $\bar{m}, \bar{y}$ denote the optimum fractional solution of the LP. We define a bipartite graph $G = (\bar{H} \cup \bar{V}, E)$

where $\bar{H}$ and $\bar{V}$ are the PMs and VMs vertices respectfully. Edge $e = (i, j) \in E$ is defined if $\overline{m}_{i,j} > 0$, and it's weight is set to $\overline{m}_{i,j}$. The weight of a host node i is set to $\overline{y}_i$.

The algorithm consists of the following four main steps:

1) *Transforming the Solution*: We create two new bipartition graphs: 1) a "light" G_1 graph in which all nodes and edges are bound by some factor $\frac{y_i}{\gamma}$. This make it easy to omit certain edges in G_1 (in the *Cycle Breaking* step) and to solve the recovery assignment by looking at a collection of induced trees. 2) a "heavier" G_2 graph in which all nodes and edges have a minimal weight of $\frac{y_i}{\gamma}$. Thus we can focus on reducing the cost of recovery since a γ bound is guaranteed in terms of capacity. To create the graph partition we iteratively consider the graph G_1 and removing edges and/or vertices from it, and optionally placing them in G_2. During this entire phase, only the edges' weights change while the PM nodes' weights remain fixed.
2) *Cycle Breaking*: Manipulating every induced graph G_1^r (which will be described later) to a forest. This is done by either forcing or banning certain VM recoveries to selected PMs.
3) *Rounding:* In order to produce a discrete approximation we separate VM nodes from either G_1 or G_2, thus forcing each node to appear in only one of the graphs. We then exploit the characteristics of each of the graphs and perform the final rounding on each of them separately to come up with the VM assignments and machine activation vectors.
4) *Activation Schema:* The last phase forces a decision between "active" and "inactive" representations of backup PMs. This finalizes the recovery plan and activation schema.

Next, we elaborate on each of these steps.

A. Transforming the Solution

In this step we construct two new graphs, G_1 and G_2 based on G, and iteratively change the weights of edges in G_1 and move (or remove) edges and nodes to G_2. Initially, $V(G_1) = V(G) = H \cup V, E(G_1) = E(G), V(G_2) = H$, and $E(G_2) = \emptyset$. Throughout this process we maintain the set of the following invariants (similar to these presented in [18]):

(I) $\forall (i, j) \in E(G_1)$, and $\forall h : m^h_{i,j} \in (0, \frac{y_i}{\gamma}) \wedge p_{i,j} > 0$
(II) $\forall r, i \in H$, and $\forall h : \sum_j o_{r,j} m^h_{i,j} p_{i,j} \le y_i$
(III) $\forall (i, j) \in E(G_2)$, and $\forall h : \frac{y_i}{\gamma} \le m^h_{i,j} \le 1$
(IV) Once a variable is rounded to 0 or 1, it remains constant

Here $m^h_{i,j}$ denotes the weight of the edge (i, j) at the end of the h^{th} iteration, and γ is a constant (to be determined later).

After the initialization we continue by following Algorithm 4.1, which is based on two intertwined operations: *FixGraphs* and *RandStep*.

- FixGraphs removes edges or VM nodes from G_1 and optionally place them in G_2 in order to maintain the above invariants. We start by removing all host vertices of weight $y_i = 0$ from both graphs, and mark those hosts with $y_i = 1$. For edges that have $m^h_{i,j} = 1$ we assign VM j to host i in the recovery plan and remove j from G_1. In case $m^h_{i,j} = 0$ we simply remove the (i, j) edge from G_1. For all remaining edges of weight $m^h_{i,j} \ge \frac{y_i}{\gamma}$ we remove them from G_1 and place them in G_2, adding VM node j to G_2 in case it is not already there. For all (i, j) which have $p_{i,j} = 0$ we can simply set $m_{i,j} = \frac{y_i}{\gamma}$ and move the edge to G_2. It is easy to verify that following these steps maintains all of the above invariants.
  Since edge weights in G_2 do not change, we denote an edge weight in G_2 by $w_{i,j}$. After fixing G_1 and G_2 we can induce a new linear system as shown in Figure 1, which formulates the assignment and capacity constraints on G_1 edges at the $h + 1$ iteration, based on the weights of edges in G_1 at the h iteration and the edges of G_2. H' and V' denote the set of host and VM nodes with degree at least 1, at the beginning of the $h + 1$ iteration.
- RandStep changes the weights of edges in G_1 while forcing at least one edge weight $m_{i,j}$ to be in $\{0, \frac{y_i}{\gamma}\}$ thus having it removed from G_1 and optionally moved to G_2 after the next *FixGraphs* step. While the linear system $Am = b$ remains undetermined, it is possible to efficiently find a matrix r so that $Ar = 0$. Next, it is simple to find a strictly positive constraint parameters: α and β, so that $m_{i,j} + \alpha r_{i,j}$ and $m_{i,j} - \beta r_{i,j}$ belong to $[0, \frac{y_i}{\gamma}]$ for all (i, j) and for at least one (i, j) at least one of them belong to $\{0, \frac{y_i}{\gamma}\}$. Finaly, this procedure returns $m + \alpha r$ with probability $\frac{\beta}{\alpha+\beta}$, and $m - \beta r$ with probability $\frac{\alpha}{\alpha+\beta}$. As shown in [19] the solution guarantees that $E[m^{h+1}_{i,j} | m^h_{i,j}] = m^h_{i,j}$ and in particular $E[m^{h+1}_{i,j}] = \overline{m_{i,j}}$. Thus, after each *RandStep* the expected cost remains optimal at $\sum_{(i,j)} \overline{m_{i,j}} c_{i,j} + \sum_i y_i a_i$.

Note that after each call to *FixGraphs* the number of equations does not increase and the number of variables (i.e., edges in G_1) actually decreases thus we achieve progress after each step and finish after at most $M \cdot N$ iterations.

Algorithm 4.1 Transformation Phase

```
$m \leftarrow \bar{m}$
$LS \leftarrow FixGraphs(m)$
**while** $LS(m)$ is underdetermined **do**
    $m \leftarrow RandStep(A, m, b)$
    $LS \leftarrow FixGraphs(m)$
**end while**
```

B. Cycle Breaking

Let G_1^r be the induce graph of G_1 with $V(G_1^r) = \bar{H} \cup \bar{V}^r$ for every $U_r \in U$ at the end of the transformation step.

Lemma IV.1. *After the end of the transformation step the number of edges in every connected component c in G_1^r is smaller than the number of vertices. That is $|E(C)| \le |V(C)|$.*

Proof: Assume there is a connected component C in an induced graph G_1^r where $|E(C)| > |V(C)|$. Let LS_r be the

$$\forall j \in V', \quad \sum_{\substack{i\in H',\\(i,j)\in E(G_1)}} m_{i,j} = 1 - \sum_{\substack{i\in H',\\(i,j)\in E(G_2)}} w_{i,j} \tag{4.4}$$

$$\forall i, r \in H', \quad \sum_{\substack{j\in V',\\(i,j)\in E(G_1)}} o_{r,j} p_{i,j} m_{i,j} = \sum_{j\in V'} o_{r,j} p_{i,j} m^h_{i,j} - \sum_{\substack{j\in V',\\(i,j)\in E(G_2)}} o_{r,j} p_{i,j} w_{i,j} \tag{4.5}$$

Fig. 1: *FixGraphs* output linear system

sub-linear system of the last LS from the previous step which includes Eq. 4.4 for all $j \in \bar{V}^r$ and Eq. 4.5 for r. Since $\forall U_r, U_{r'} \in U$, if $r \neq r' \Rightarrow U_r \cap U_{r'} = \emptyset$ it is easy to see that LS_r is independent from $LS_{r'}$ and that $\bigcup_r LS_r = LS$. Having $|E(C)| > |V(C)|$ means that LS_r is undetermined. Since LS includes an independent undetermined linear system, it is in itself undetermined as well, and the condition in the while loop in Alg. 4.1 is true. ■

Corollary IV.2. *Every connected component in an induced graph G_1^r contains at most one cycle.*

For every r and a cycle C in G_1^r, we choose one edge (i', j) in C at random. If $m_{i',j} \geq \frac{1}{2}$ then we assign VM j to host i' in the recovery plan (and remove it from G_1). This increases the total VM assignments costs and load on PM i' by at most a factor of 2 since VM j was already partially assigned to it by at least a value of $\frac{1}{2}$. The same is true for the activation costs of PM i' since PM i' was already partially open with a value of at least $\frac{1}{2}$. Note that by the term "load" of host i we actually mean the maximum load of VMs in i originating from different VCs, or formally $\max_r\{\sum_{j\in\bar{V}^r} p_{i,j} m_{i,j}\}$. This is done due to the fact that assignment of VMs originating from different VCs to the same target does not mean aggregated capacities in the load calculation. For the same reason we can conclude that the order of which we choose r is unimportant. In case $m_{i',j} < \frac{1}{2}$ we remove (i', j) from the graph and scale up every (i, j) by a factor of 2. This means that the total assignment value of every VM remains at least 1, while the total costs associated with it (assignment and activation) is increased by at most a factor of 2.

C. Rounding

We would like to partition the VM nodes between the two graphs, G_1 and G_2 in the following way:

For every VM node j in G_2:

- If $\sum_{i:(i,j)\in E(G_2)} m_{i,j} < \frac{1}{\delta}$ remove every (i, j) edge and VM node j from G_2. Since we guaranteed that the total assignment of each VM is at least 1, we get $\sum_{i:(i,j)\in E(G_1)} m_{i,j} \geq 1 - \frac{1}{\delta}$.
- Otherwise, if $\sum_{i:(i,j)\in E(G_2)} m_{i,j} \geq \frac{1}{\delta}$ remove every (i, j) edge and VM node j from G_1.

Next we round separately the values in G_1 and in G_2:

1) Rounding G_2: Since the weight of each (i, j) edge in G_2 is at least $\frac{y_i}{\gamma}$, every feasible recovery plan in G_2 is a $\gamma-$approximation of the optimal solution in terms of capacity, thus we focus on minimizing the recovery costs. This problem is also known as the *non-metric uncapacitated facility location problem,* where the goal is to find a minimum cost assignment scheme of clients to facilities in which both the cost of assignments and the facility activation costs are fixed. By employing the approximation algorithm described in [20] we can find a recovery plan bounded by a factor of $O(log\frac{n+m}{OPT} + 1)$ from the optimal solution (here $n = |V(G_2)|$ and $m = |E(G_2)|$).

2) Rounding G_1: The rounding process on G_1 consists of $|U|$ phases of rounding. One for each G_1^r. Since VM assignments in different induced graphs are independent of each other, the order of this rounding is unimportant. For each $U_r \in U$, we traverse the G_1^r tree in a bottom-up order, and for every VM node j which is a child of PM node i: 1) If $m_{i,j} < 1/\eta$, we remove the edge from G_1. Since all VM nodes in G_1 initially maintain $\sum_{i\in H} m_{i,j} \geq 1 - 1/\delta$, after these edges were removed we have $\sum_{i\in H} m_{i,j} \geq 1 - 1/\delta - 1/\eta$ for every VM node j in G_1; 2) In case $m_{i,j} \geq 1/\eta$ we mark PM i ($y_i = 1$) and assign j to i in the recovery plan. This causes the capacity and costs associate with PM i to increase by at most a factor of η.

After the first phase of the rounding in G_1, for each $U_r \in U$ and VM node j in G_1^r that has not been assigned yet, we look at the disjoint star S_j, with j at it's center. For every PM i that has been marked in S_j we define an activation cost of 0. We assign j to the PM i_l in S_j with the minimum total cost $a_{i_l} + c_{i_l,j}$. Since only one VM will be assign to i_l in G_1^r at this phase, and because $p_{i_l,j} \leq 1$, the assignment increases the load on i_l by an additive 1 at most. Let $i_1, i_2, \ldots, i_{l_j}$ be the PM nodes in S_j. The contribution of every VM j to the costs in the optimal fractional solution is bounded by $\sum_{k=1}^{l_j} y_{i_k} a_{i_k} + m_{i_k,j} c_{i_k,j}$. Since $y_i \geq m_{i,j}$ for every (i, j) and due to the fact that for every VM node in G_1, $\sum_{k=1}^{l_j} m_{i_k,j} \geq 1 - 1/\delta - 1/\eta$ we can induce the following:

$$\begin{array}{rl} \sum_{k=1}^{l_j} y_{i_k} a_{i_k} + m_{i_k,j} c_{i_k,j} & \geq \\ \sum_{k=1}^{l_j} m_{i_k,j} a_{i_k} + m_{i_k,j} c_{i_k,j} & \geq \\ \sum_{k=1}^{l_j} m_{i_k}(a_{i_l} + c_{i_l,j}) & \geq \\ (1 - 1/\delta - 1/\eta)(a_l + c_{i_l,j}) & \end{array} \tag{4.6}$$

Thus the total cost is within a factor of $\frac{1}{1-1/\delta-1/\eta}$ from the optimum.

D. Activation Schema

Following the rounding step, every VM j is assigned to some host i. However, the activation schema is still not determined since some VMs can be assign to an "active" representation of the PM ($i \in H$) while other VMs will be assign to its "inactive" representation ($i \in H^{off}$). The final phase will determine which PM will be activated and which PM will not.

We activate every PM $i \in H$ that has been marked. For every such PM i that has been activated if there's an "inactive" representation of it, $i' \in H^{off}$ that has also been marked, assign all the VMs that were assigned to it, to the activate PM i. The load on PM i is increased by a factor of 2 at most, since i and i' have the same initial capacity. Since we assumed Cost-V$_{i,j}(1) \leq$ Cost-V$_{i,j}(0)$ and because the activation cost of PM i has already been paid for, there's no increase in costs. The remaining PMs from H^{off} or those from H that were not marked for activation will remain inactive.

We can now prove our main theoretical result.

Theorem IV.3. *The algorithm for the VM recovery with host activation costs problem has a total cost of $O(\log \frac{n+m}{OPT} + 1))OPT$, with a PM load bound of $(6+\varepsilon)$.*

Proof: By combining all the above steps we can conclude a $2 \cdot \left(\max(\eta, \frac{1}{1-1/\delta-1/\eta}) + O(\log \frac{n+m}{OPT} + 1)\right)$ cost approximation, and a $2 \cdot (\max(2\gamma, 2\eta + 1))$ capacity approximation. By setting γ and η to be $\gamma = \eta = 1 + \frac{\varepsilon}{4}$ for any $\varepsilon > 0$, and $1/\eta = 1 - 1/\delta - 1/\delta(\log \frac{n+m}{OPT} + 1)$ we can get a total cost of $O(\log \frac{n+m}{OPT} + 1))OPT$, with a PM load bound of $(6+\varepsilon)$. ■

V. A Practical Approach

In the previous section we described a bicriteria approximation algorithm for the VM recovery problem. While this algorithm has guarantied upper bounds on cost and load, it is somewhat complex. Next we present a simple and effective greedy algorithm, named *Greedy Active (GA)* that while not having guarantied bounds, performs well under simulations based on real data-center data. The concept behind this heuristic is that if we had known which PMs are active and which are not we could calculate, using existing methods, a good approximation of the VMs recovery plan. Thus, the only thing left to decide is which PMs to activate and which will remain inactive.

A. Greedy Active Algorithm

At every iteration GA searches for the next inactive host that if activated will minimize the total cost of the recovery plan. Once found, the PM is activated and GA continues to search for the next PM to activate among the rest of the inactive PMs. The algorithm stops when it is no longer beneficial (due to active PMs costs) to activate any additional PM, or when all PMs have been activated.

Given the current PMs *Activation Schema* it is possible to find a recovery plan by iteratively solving the minimization version of the *Generalized Assignment Problem* (GAP), for each of the failing VCs. For each item i and a bin j, the item size $S_{i,j}$ and cost $C_{i,j}$ is defined. In Min-GAP the goal is to pack the set of items into bins so that the total cost of the packed items is minimized, and all items are packed under the bins capacity constraints. Both Min-GAP and it's counterpart Max-GAP (Where the goal is to pack a subset of items of maximum profit) have been thoroughly studied in the literature and constant approximation algorithms have been presented [21]–[23]. In the Greedy Active algorithm, virtual machines originating from the same VC are assigned to available backup PMs, considering the appropriate recovery costs and normalized size. By combining the GAP packing among all VCs we can generate a valid VM recovery plan.

Shmoys and Tardos presented an algorithm in [21] that achieved an optimal cost packing with a 2-approximation in bin load. This can be transformed into a feasible solution by reallocating excess items to secondary bins at the cost of higher VM recovery. A second option, is to use a Max-GAP transformation where items costs are transformed into profits (where an item i profit in bin j is $P_{i_j} = Max_{v \in V}\{C_{v,j}\} - C_{i,j}$) and a simple Max-GAP approximation is used (e.g., [23]). While none of these options maintains any approximation ratio (the latter also does not guaranty a feasible packing), we have found that when simulated using real data-center data, where the number of bins and items configurations is small, these techniques produced a near-optimal feasible packing. Thus, they may be considered "good enough" for any practical scenario.

Host	Capacity [GB]	Count [%]	Active State Cost [$]
Small	30	22	2900
Medium	62	68	3860
Large	126	8	5780
X-Large	254	2	9620

TABLE I: Physical hosts' configurations

VM	Demand [GB]	Count [%]	Annual Cost [$]
Small	1	10	276
Medium	2	37	552
Large	6	53	1104

TABLE II: Virtual machines' configurations

B. Performance Analysis

In order to assess the performance of the GA algorithm we have used a snapshot of a medium size IBM research cloud that contained a few hundreds physical hosts of 4 different configurations and a few thousands VMs of 3 different configuration (Table I and II list the exact PM and VM configurations). The PMs memory capacity was considered as the only resource capacity and the same was assumed regarding to the VMs requirements. All simulations were conducted on an Intel i3 CPU at 2.53GHz, 4GB RAM machine.

We compared our heuristic against two naive opposite approaches. The *Active* approach recovers VMs to only powered machines, thus while VM recovery is cheap, powering backup PMs increases costs considerably. The opposite *Inactive* approach does not power new PMs. While it may still

Availability [%]	SLA Fine [%]	SLA Fine [$]		
		Small	Medium	Large
99.999 - < 99.9999	10	27.6	55.2	110.4
99.99 - < 9.999	15	41.4	82.8	165.6
99.95 - < 99.99	20	55.2	110.4	220.8
< 99.95	50	138	276	552

TABLE III: VM Service-Level Agreement

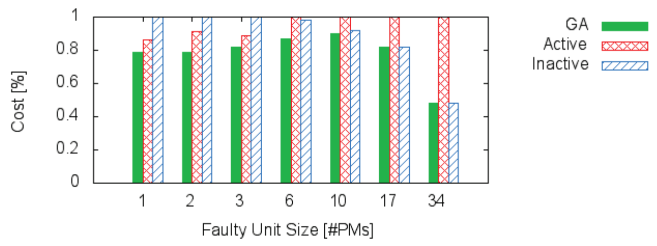

Fig. 2: Heuristic Total Costs Comparison

recover VMs to currently active PMs, using their residual capacity, backup PMs that are used in the recovery process remain inactive. While this approach saves costs in machine maintenance, recovery may be slow and thus expensive.

As the pricing model we used a SLA policy that is based on AWS EC2 SLA policy [24] and was modified to reflect higher availability guarantees at higher prices (Tables II and III). For example, the *Small* VM type has an annual cost of 276$ which is 4x the upfront cost of it's AWS EC2 counterpart. A downtime of 5 minutes would cost the cloud provider 10% of the annual VM cost, or in the case of the *Small* VM 27.6$. As mentioned in Section II, this scheme is based on common commercial offerings. The cost of active machines is in a range of 4000-12000$ (Table I) according to the PM size, following the typical costs presented in [5].

Figure 2 shows a comparison between our heuristic and the two approaches for different faulty unit sizes (in terms of number of hosts). For example, when considering faults of a single PM, the local search heuristics activated only several of the cheapest hosts, and designated only the most expensive-to-recover VMs to these PMs. Thus achieving a 28% savings in costs versus having no active backup machines. On the other hand, when considering a fault of extreme proportions (i.e. the entire set of currently active machines) the local search heuristic converges to the same solution as the inactive approach of using a large set of inactive backup machines. In this scenario, the savings in recovery of the entire region to active hosts is negligible versus the cost of powering an entire backup region.

A closer look at the distribution of costs between the VM recovery costs and the active hosts costs reveals the reason the "Active" and "Inactive" heuristics performs differently under different faulty unit sizes and why GA performs better through the entire range. Since the "Active" heuristic always recover VMs to active PMs, the VM recovery costs portion slowly become overshadowed by the increased cost of machines' activation as more machines are needed to handle larger failures (Figure 3). In contrast to the "Active" heuristic GA manages to balance between the VM recovery costs and the active PM costs (Figure 4).

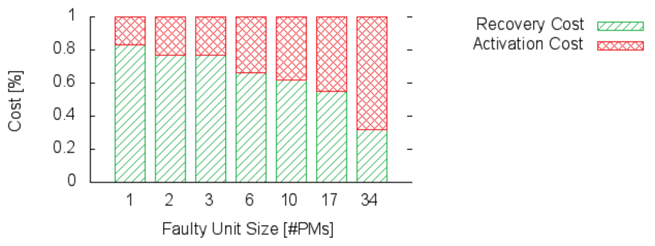

Fig. 3: Active Cost Partitioning

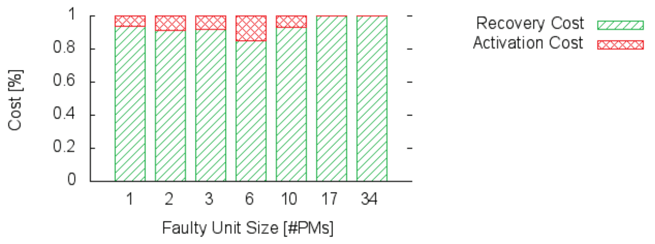

Fig. 4: Greedy Active Cost Partitioning

In order to assess the affect different SLA prices impose upon the recovery results of GA, two scenarios were considered. The first used a collection of low-end *Small* hosts (see Table I) as backup, while the second used higher-end *Large* hosts. Figure 5 and Figure 6 show the number of active and inactive hosts used for recovery of the same VM placement with each type of backup hosts, as a function of the normalized ratio between VM SLA costs and active PM costs. Here one is the original VM SLA and active PM costs as described in Tables I and II (i.e., a ratio of 2 used the original VM SLA costs but half the cost of having a powered PM).

While the total number of backup hosts used for recovery depends solely on the backup hosts configuration, the number of hosts GA chooses to activate depends on the cost ratio. The results show that GA allows some variance in this ratio before more or less active hosts are used for recovery. This allows cloud providers the ability to host more expensive-to-recover VMs or use cheaper backup machines while still maintaining similar levels of recovery costs.

VI. VM Placement

An important aspect of cost aware VM recovery that was not addressed so far is the original placement of VMs according to their recovery costs. In this section we show that considering the VM recovery costs as part of the VM placement may reduce the ultimate recovery costs and number of active machines needed for recovery.

As mentioned in Section III, VMs originating from different VCs can be designated to the same backup host when assuming only one VC fails at-a-time. A backup host is more likely

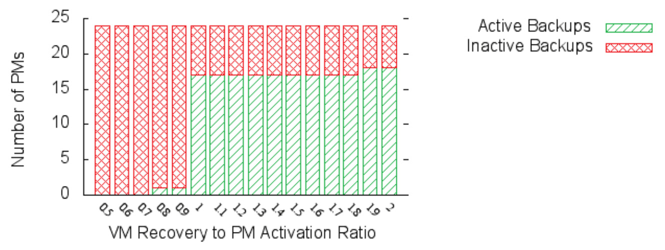

Fig. 5: Recovery using *Small* Backup Hosts

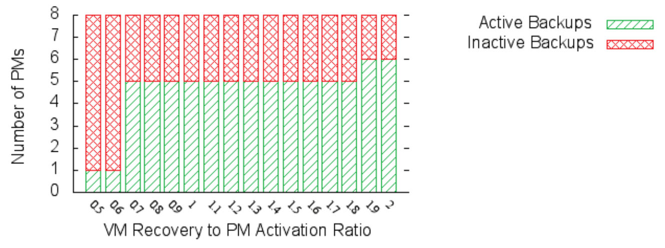

Fig. 6: Recovery using *Large* Backup Hosts

to be activated if more VMs can benefit from it's activation, raising the probability the recovery savings will pay for the host's activation. Also, the more recovery-expensive VMs are handled by the same active PM, the higher the chances that less extra active PMs will be needed for VM recovery, as cheaper to recover VMs will be assigned to inactive PMs. Thus, a distribution of VMs that may benefit more from an active backup host, across multiple VCs may reduce the number of active backup PMs in case of failures.

Figure 7 illustrates how a consolidation of costly VMs in a single host may result in a pricey recovery plan. The left scenario shows two hosts h_1 and h_2 assigned with two types of VMs each. Host h_1 holds 4 expensive-to-recover VMs (solid color) that will benefit most from recovery to an active host (b_1) in case h_1 fails. Host h_2 holds 4 VMs that are less susceptible to failures. These VMs will not benefit much from active host recovery, but since b_1 is already active it will be chosen for recovery. At the right hand scenario both VM types are evenly distributed among h_1 and h_2. Thus it is possible to recover the expensive VMs to a smaller, cheaper to maintain active host (h_3) and use an inactive host (h_2) to recover the remaining VMs.

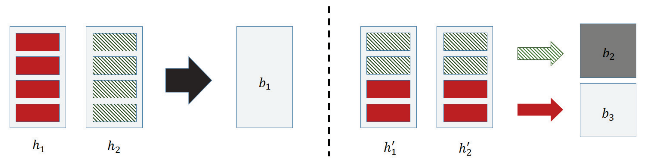

Fig. 7: Heuristic Total Costs Comparison

To validate this we have compared between two placement scenarios that were simulated using real IBM data center data (as described in Section V). In both scenarios the data center was partitioned into 10 VCs of equal size and both used the same set of small hosts as backup candidates. The first scenario (named *Compact*) defines two VCs as 5 times more "expensive" to recover. This means that the cost of recovery for all guest VMs was increased to 5 times their original recovery costs. The second placement scenario (named *Distributed*) scattered the previous expensive VMs throughout all VCs uniformly at random. Both Compact and Distributed placements used the same number of PMs achieving the same high levels of hardware utilization. Using the Greedy Active algorithm (Section V) a VM recovery plan and an activation schema were calculated for each of the two placement scenarios. While using the same number of backup PMs for the recovery plan in both scenarios Greedy Active managed to reduce the number of active backup machines by 60%. This experiment demonstrate how a distribution of expensive-to-recover VMs across several VCs may help to better utilize activated backup hosts as recovery targets.

VII. Conclusion and Future Work

In this paper we studied ways to maintain high availability of cloud services at a reasonable cost. Realizing that faults are prone to happen, we focused on the trade-off between maintenance cost and high availability. We presented two possible solutions for this optimization problem. The first is a bi-criteria approximation algorithm with proven worst case bounds, and the second is a much more practical greedy heuristic that performs very well in realistic scenarios compared to two naive alternatives. Our simulation results, over real data from an operational data center, indicate that our solution manages to balance between rapid VM recovery and machine activation costs, for the full spectrum of failure unit sizes.

As explained in Section III, this work mainly assumes faults due to machine failures. Thus, it would be interesting to refine the model by addressing failures according to their distinct attributes, like software problems or missconfigurations. Another interesting research direction is continuing along the lines of Section VI and developing a comprehensive VM placement solution that consider recovery costs as one of its optimization objectives.

VIII. Acknowledgment

We want to thank IBM Research - Haifa for the data-centers data.

References

[1] S. Loveland, E. M. Dow, F. LeFevre, D. Beyer, and P. F. Chan, "Leveraging virtualization to optimize high-availability system configurations," *IBM Syst. J.*, pp. 591–604, Oct. 2008.

[2] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: high availability via asynchronous virtual machine replication," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, 2008, pp. 161–174.

[3] *Protecting Mission-Critical Workloads with VMware Fault Tolerance*, http://www.vmware.com/files/pdf/resources/ft_virtualization_wp.pdf, 2009.

[4] *Kemari Fault Tolerance in QEMU*, http://wiki.qemu.org/Features/FaultTolerance, February 2011.
[5] L. A. Barroso and U. Holzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis Lectures on Computer Architecture*, 2009.
[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the 19th ACM symposium on Operating systems principles*, 2003.
[7] I. Habib, "Virtualization with kvm," *Linux J.*, no. 166, February 2008.
[8] C. A. Waldspurger, "Memory resource management in vmware esx server," *SIGOPS Operating Systems Review*, 2002.
[9] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," in *Proceedings of the 5th international conference on Virtual execution environments*, 2009.
[10] "Rackspace service license agreement," http://www.rackspace.com/managed_hosting/support/servicelevels/managedsla/, August 2012.
[11] "Hosting.com service license agreement," http://www.hosting.com/terms-conditions/sla, August 2012.
[12] "Terremark service license agreement," https://community.vcloudexpress.terremark.com/en-us/product_docs/w/wiki/service-level-agreement.aspx, August 2009.
[13] B. Addis, D. Ardagna, B. Panicucci, and L. Zhang, "Autonomic management of cloud service centers with availability guarantees," in *Proceedings of the IEEE 3rd International Conference on Cloud Computing*, 2010.
[14] Z. Zheng, T. C. Zhou, M. R. Lyu, and I. King, "FTCloud: A component ranking framework for fault-tolerant cloud applications," in *Proceedings of the 21st IEEE International Symposium on Software Reliability Engineering*, 2010.
[15] J. Duell, "The design and implementation of berkeley labs linux checkpoint/restart," Tech. Rep., 2003.
[16] W. Zhang, H. Tang, H. Jiang, T. Yang, X. Li, and Y. Zeng, "Multi-level selective deduplication for vm snapshots in cloud storage," *Proceedings of the IEEE 5th International Conference on Cloud Computing*, 2012.
[17] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive fault tolerance for HPC with xen virtualization," in *Proceedings of the 21st annual international conference on Supercomputing*, 2007.
[18] S. Khuller, J. Li, and B. Saha, "Energy efficient scheduling via partial shutdown," in *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, 2010.
[19] V. S. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan, "Approximation algorithms for scheduling on multiple machines," in *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, 2005.
[20] A. Srinivasan, "Improved approximation guarantees for packing and covering integer programs," *SIAM Journal on Computing*, 1999.
[21] D. B. Shmoys and v. Tardos, "An approximation algorithm for the generalized assignment problem," *Mathematical Programming*, 1993.
[22] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko, "Tight approximation algorithms for maximum general assignment problems," in *Proceedings of the 17th annual ACM-SIAM symposium on Discrete algorithms*, 2006.
[23] R. Cohen, L. Katzir, and D. Raz, "An efficient approximation for the generalized assignment problem," *Inf. Process. Lett.*, 2006.
[24] "Amazon ec2 service level agreement," http://aws.amazon.com/ec2-sla/, October 2008.