12th IFIP/IEEE International Symposium on Integrated Network Management 2011

VMSpreader: Multi-tier Application Resiliency through Virtual Machine Striping

Pradipta De, Sambuddha Roy
IBM Research, India Research Lab, New Delhi, India
pradipta.de@in.ibm.com, sambuddha@in.ibm.com

Abstract—**With the growing use of virtual servers for hosting applications, management of large IT infrastructure faces new challenges. In order to reduce the capital expenditure at data centers, virtual server consolidation approaches has been the focus of attention. However, server consolidation inadvertently introduces a new point of failure, which is the physical resource on which the virtual machines are hosted. A common practice is that for a multi-tier application, different tiers of the application are hosted from a single physical machine. Although this is beneficial in maintaining low cost of ownership when the number of hosted applications is low, with increasing number of hosted applications this leads to a skewed distribution of the applications. We propose that spreading the virtual servers of different applications across the available physical resources can lead to a more resilient design for multi-tier application hosting. We formulate the problem of striping the virtual machines across multiple physical machines as an optimization problem. We propose two MIP based techniques, where the first solution is fast but may be infeasible in many cases, while the other one is better in terms of accuracy but takes more time. The solution is tested on data sets representing typical hosting environments.**

Index Terms – Virtual Machine, Mixed Integer Problem, Optimization, Striping, Load Balancing, Resiliency

I. Introduction

A. Motivation

Management of a large data center serving ever growing business demands of a customer is a challenging problem. On the one hand the capital expenditure for infrastructure must be kept at a minimum, while servicing the growing requests. Typically, in order to keep the capital expenditure low a data center acquires just the right set of hardware infrastructure for hosting the applications. At the onset when a service delivery organization takes over the IT operations of a customer, the existing applications are moved to the new IT infrastructure. *A likely scenario is that several applications from a multi-tier software product suite are hosted on different virtual servers, but the virtual servers may reside on the same physical hardware.* This is beneficial in keeping the total cost of infrastructure low. Whenever, a new application suite is introduced, new hardware infrastructure is provisioned, and different tiers of the new application are hosted on virtual servers resident on the same physical machine. Over a period of time, this leads to a scenario where various tiers of an application are hosted from the same physical machine. In the event of a failure of the physical resource, every tier of the application suite faces downtime. In our view, *a desirable scenario would be if the physical server hosted applications from different application suites.*

Although intuitively it appears that the proposed setup leads to downtime for more application suites, in practice it could be quite effective. For instance, an ecommerce application, like Trade-6, has three tiers: web-tier for browsing, application tier, and the database tier for servicing the buy transactions. It has been observed that for most of the ecommerce sites the ratio of browse-to-buy is about 5%; a small proportion of the web-page visitors end up doing any transaction [1]. Hence if the database tier goes down it may not be disruptive to most of the visitors. *In essence, we are presenting a case for resiliency and faster recovery in the event of a hardware failure by striping the virtual servers of a multi-tier application over available physical machines; we term this as the VMSpreader problem.* Striping would spread the different tiers of an application across all the available physical machines.

In a cloud environment, when a user requests for a set of virtual servers, these are usually provisioned on a single physical resource. In Amazon EC2 [2], the virtual machines are provisioned on a single physical resource based on the creation parameters. For typical scenarios, it has been shown that a user's request is serviced from a single target host [3]. In order to reduce chance of complete service disruption to an user, it might be preferable to spread the virtual machines across multiple physical machines.

B. Problem Instance

We present a simple scenario to illustrate the problem further. While on-boarding a new client, an IT delivery organization on-boards only a single application, say App-1, for the customer. App-1 is a 3-tier application and is hosted on 3 separate virtual machines. However, the total resource requirement of the 3 virtual machines can be accommodated in a single physical machine. Later on the customer wants another 3-tier application, say App-2, to be hosted. The IT delivery unit provisions another physical resource for hosting the 3 tiers in 3 separate virtual machines, but uses the same physical server. All tiers of an application are thereby hosted from a single physical resource. We believe that this scenario is detrimental to application resiliency because it introduces a single point of failure. We claim that spreading the different tiers of the applications, App-1 and App-2, across the available physical resources is a more resilient design. Time to recover a single tier of an application will lead to a lower Mean-Time-to-

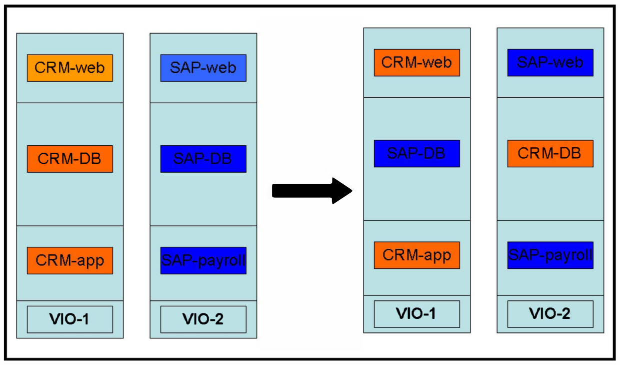

Fig. 1. A typical scenario in a data center showing desired transformation of application hosting. Here two applications types, CRM and SAP are shown. At the beginning, all 3 tiers of the two applications are resident on a single physical machine. After transformation, the tiers are balanced across the two physical machines.

Recover (MTTR) than bringing all the tiers up. Also, note that in a typical IT delivery environment, there are separate teams handling each application suite. The scenario is illustrated pictorially in Figure 1, where the desired transformation is demonstrated.

C. Contribution

In this paper, we address the problem of planning a failure resilient hosting environment for different tiers of multiple n-tier applications. The planner helps in spreading the hosted applications across multiple physical machines, and can be used at the initial planning phase, and at the time of provisioning new applications. More specifically, our main contributions in this paper are:

1) We present an optimization model for striping the virtual machines dedicated to an application across available physical resources.
2) Given that the VMSpreader problem is an NP-hard problem, we present and evaluate heuristics for solving the problem.

The rest of the paper is organized as follows. In Section II we present similar work in VM consolidation. Section III presents a description of the problem and its variations. We describe the different constraints in the formulation of the problem in Section IV followed by the optimization formulation of the problem in Section V. Section VI presents the approaches to solving the problem and the results on a sample dataset. Finally we conclude in Section VII.

II. Related Work

The problem of VM placement in data centers and cloud environment has been studied with different objectives. Verma et al. proposes a solution for VM placement in a data center with the goal of minimizing the power consumption and number of migrations [4]. Tang et al. solves the problem of application placement in a set of machines in order to maximize the utility [5]. Similarly, Bobroff et al. proposes a dynamic approach to consolidate servers on physical machines [6]. In most of these solutions, which are cast as optimization problems, there is no restriction on placing a virtual machine on any physical machine. This can lead to the *undesirable* situation when different tiers of the same application gets consolidated on the same physical machine. In this work, our aim is to guard against the scenario where a physical server failure can lead to complete disruption of service for a multi-tier application. We formulate the problem such that different parts of a multi-tier application are never placed on a single physical machine.

In a recent work, Jung et al. proposes multi-tier application resiliency by creating replicas of the VMs, and ensuring that the replicas are placed on different physical machines [7]. This is similar in concept to our work, however, we address the problem of balancing the VMs according to their required capacities. We provide different notions of VM balancing by striping the VMs across physical machine. To the best of our knowledge, this problem has not been tackled before.

One effort which proposed a solution for the VM balancing problem is *Load Balancing of Virtual Machine (LBVM)* [8]. LBVM does not provide details of the technique used for balancing the VMs across physical servers, however, they do mention that it is an effective way to guard against the single point of failure introduced by placing multiple correlated VMs on the same physical server. We present a detailed and formal description of the problem, and propose effective solutions to the problem.

III. Problem Description

The typical setup in a data center has a collection of physical machines, which hosts a set of virtual servers. Each virtual server hosts a single tier of a multi-tier application. The collection of physical machines is denoted by PMs; the virtual machines that are hosted on these physical machines are denoted by VMs. Each virtual server has a pre-defined resource allocation based on the application workload. This is usually static allocation based on the peak load of the application. Thus, we can assume that there is an initial allocation of the VMs on the various PMs. Based on the application type that is running on the virtual machine, we assign a type to the VM. For example, a VM hosting one tier of the multi-tier application CRM will be tagged as being of type CRM. When a new multi-tier application is marked for hosting, new PMs are acquired, and VMs are allocated on those. In most cases, the set of VMs on a PM will be of the same type since at one time only one new application suite will be provisioned. Thus, over time the distribution of the various VM's by type on each PM gets skewed. Each PM will host VMs from a single application type. This introduces the single point of failure in terms of the PM. *We propose to reallocate the VMs across the different PMs such that there is a "balanced" distribution of VMs of all application types on the PMs.* It should be noted that as the application types increase, perfectly uniform balance becomes harder to acieve. Therefor, in our solution, we focus on minimizing the cost while achieving a balance as close to "uniform" as possible.

The cost of a transfer is determined by the number of transfers of VMs. Each *transfer* of a VM from one server to another is a costly operation and the goal is to minimize the *number* of such transfers. Thus, our objective function is to minimize the number of transfers of the VMs while achieving an as-close-to-balanced distribution of the VMs on the PMs as possible. In a natural variant of the problem, the costs of transfers of a VM from one PM to another may depend on the VM (and the original and destination PMs) in question. This can be thought of as the *non-uniform* version of our main problem.

We formulate the VMSpreader problem as a Mixed Integer Program (MIP). A mixed integer program is a mathematical program where some of the variables are forced to be integral, while some others may have *relaxed* constraints on them (for instance, $0 \leq x \leq 1$). Since integer programs can encode NP-hard problems like Vertex Cover, solving general MIPs in polynomial time is expected to be hard. A general reference for integer and linear programming is Schrijver [9].

A greedy solution approach will work easily only if one service type is present, but with multiple service types the problem is significantly harder. Another reason for not adopting a greedy heuristic is the following: while a greedy heuristic is computationally viable, the quality of solutions it produces might be much worse than the optimal number of transfers required to bring about balance. Also we make two observations – (1) new application suites are procured rather infrequently, and (2) the cost of transfers is usually high. Therefore, instead of using a greedy approach which might be computationally faster, we conclude that it is preferable to opt for a better quality (but computationally intensive) solution as provided by the MIPs, against a worse quality but computationally viable procedure (greedy approach).

In typical application farms, a transfer is an expensive operation – thus one would like to optimize this while maintaining balance. This is precisely what we aim to achieve by solving the MIPs discussed. A relevant point is that new application suites are procured rather infrequently, so it might be preferable to opt for a better quality (but computationally intensive) solution as provided by the MIPs as against a worse quality but computationally viable procedure (greedy approach).

A. Problem Assumptions

Prior to stating the problem using notation, we state the assumptions associated to the parameters and variables. It is assumed that a single VM cannot be hosted on multiple PMs since partitioning a VM into multiple parts is infeasible. It is also stipulated that when multiple VMs are allocated on a PM then all the necessary resources (like CPU, memory, network bandwidth) must be available. In our constraints, we capture the resource availability using a single variable for simplicity. The size of a VM is stated in terms of physical CPUs required. Since micro-partitioning [10] allows fractional CPU allocation to a VM, it is possible to have fractional allocation for a VM.

Another point to note is that, each PM may have a supervisor VM, called VIO, for which some resource has to be allocated. It is also possible that a PM is not perfectly packed at the beginning : this means that the total amount of resource of all the VMs (in the original configuration) on the PM may not consume the entire available PM resource.

Before we proceed, let us state the parameters of our problem. We will let M denote the total number of VMs, and let N denote the number of PMs, where $M \geq N$. Let TY denote the set of `Types` of the virtual machines. Each `Type` denotes one multi-tier application. In the rest of the discussion, the index i will be reserved to range over Virtual Machines (VM); the index j will be reserved to range over Physical Machines (PM); and the index k will be reserved for Types (TY), unless otherwise explicitly stated. We also denote the resource capacity of the PM_j as $\mathrm{Resource}_j$, and the size of the VM_i as Size_i.

There is also an *initial configuration* of the VMs on the PMs, which lists the set of VMs (along with their types) on each PM.

B. Hardness of the VMSpreader problem

In the following, we pinpoint the hardness of the VMSpreader problem. We will distinguish between several arguably natural cases of varying complexity. Each case in the following will be parametrized as below:

- The number of VM types k.
- The number of PMs N.
- The resource capacity of each PM.

Let us start off by considering the (apparently) simple case of $k = 1$, and $N = 2$. Furthermore, suppose that both the PMs have the same size. We reduce the (weakly) NP-hard problem of `Partition` to this case [11]. To elaborate, let us consider an instance of the `Partition` problem: we are given n numbers $a_1, a_2, \cdots, a_n$, and the `Partition` problem is to decide whether there is a *partition* of the n numbers into 2 (disjoint) sets such that the sum of the numbers in one set is equal to the sum of the numbers in the other. This is one of Karp's original list of 21 NP-complete problems.

We reduce the problem instance above of the `Partition` problem to a problem instance of the simple case of our VMSpreader problem: we will have n VMs of Type 1 on the first PM, with sizes $a_1, a_2, \cdots, a_n$ and the second PM is empty. Also let the capacities of the two PMs be equal to $\sum_i a_i$. Then there is a way to *perfectly balance* the VMs on the 2 PMs if and only if there is a solution to the `Partition` problem. Thus this simple case is already NP-hard. However, the `Partition` problem is *weakly* NP-hard: this implies that for a_i's that are of magnitude polynomial in the input size (i.e. n), there exists an efficient (viz. polynomial time) dynamic programming solution for the `Partition` problem [11]. Given a single type of VMs, we can easily provide a modified dynamic program to show that the case under consideration ($k = 1$, $N = 2$) may be efficiently solved in pseudo-polynomial time *. Details are omitted because of space considerations.

*For details of pseudo-polynomial time algorithms, strong/weak NP-hardness, refer to [11]

We proceed to consider the situation where there are N PMs (instead of 2 as in the above case). We reduce the (strongly) NP-hard problem of `3-Partition` to this case. An instance of the `3-Partition` problem consists of $3m$ integers (for a number m), and the problem is to decide if there is a partition of the numbers into m subsets, such that the sum of the numbers in each subset is the same. Via a reduction similar to the above, we can easily reduce `3-Partition` to the case of N PMs. Thus, the VMSpreader problem with even 1 type (and same resource capacities for every PM) is already *strongly* NP-hard!

We conclude thereby that the fully general VMSpreader problem comprising of multiple types, and multiple PMs of possibly varying sizes may only be harder. The strong hardness results for the VMSpreader problem as outlined above mean the following: Given a problem instance of the VMSpreader problem it is NP-hard to even decide if there is a feasible solution that achieves *perfect balance* of the various VM types across all the PMs; note that we are talking *only* about the basic feasibility question here, without even approaching the (naturally harder) *optimization* question of *minimizing* the number (or cost) of transfers.

It is thereby clear that we will have to somehow cope with the inherent hardness of the problem in our design of practical, actionable solutions for the VMSpreader problem.

IV. Constraints in the VMSpreader problem

We describe the constraints involved in the VMSpreader problem. There are essentially *three* types of constraints: (i) the **allocation** constraints ensure that any single VM is *not* placed on multiple PMs; (ii) the **resource** constraints ensure that after allocation, the total available resource capacity of any specific PM is not exceeded; (iii) the **Load-Balancing** constraints guarantee that the application types are spread *appropriately* across all the PMs.

While the first two constraints are reasonably well-defined, the third **Load-Balancing** constraints requires some qualification. Given a problem instance with k types of VMs, and various PMs of varying sizes, it is not instantly clear as to what a natural definition of "balance" should be. (The reader may note that this situation is akin to "clustering" of data points [12], where the right notion/metric of *clustering* often depends on the problem at hand.) We offer two natural notions of balance; the motivation for these notions came from different business scenarios. It is perhaps worthwhile mentioning here, that the insight behind these notions of balance arose from such aforementioned metrics for clustering. Herein, we describe the notions of balance qualitatively, relegating the more quantitative versions to Section V.

The first notion of balance may be summarized as follows: Given k types of VMs, consider the fraction of VMs of any *specific* type over the VMs of *all* types. Load balancing implies then that, every PM ought to be assigned at most this fraction of the VMs of a specific type. Call this notion of balance StaticBalance.

We also consider a more delicate notion of balance. Given any configuration/allocation of VMs on PMs consider the fraction of (the sizes of the) VMs of any specific type on a single PM over (the sizes of) the VMs of that specific type over *all* the PMs. Then a reasonable notion of balance implies that this fraction should roughly be equal *across all types*. In contrast to the notion of StaticBalance, call this notion of balance DynamicBalance.

The reader is urged to note the subtle difference between the two notions of balance: the notion of StaticBalance does not need to consider any specific configuration to decide the fractions on each PM. The second notion of DynamicBalance considers any specific configuration to decide the fractions of VMs to allocate to each PM. Thus, while the definition of StaticBalance does not depend on the configuration, the notion of DynamicBalance imposes a *metric* on the specific configuration.

Apropos this discussion, we enumerate the desiderata for effective solution approaches.

1) The solution approach needs to overcome the "infeasibility" issues as outlined in Subsection III-B.
2) The solution approach should exploit the inherent linearity of the Allocation and Resource Constraints.
3) Solutions achieved should adhere to notions of *quality of balance* as described above.

Item (2) above indicates that a reasonably efficient way to solve the problem would be to somehow *linearize* the Load Balancing Constraints discussed above. This would enable us to formulate our problem as a Mixed Integer Program (MIP), rather than as substantially more complex nonlinear programs.

Without further ado, we proceed to formulate the VMSpreader problem as two MIPs, corresponding to the two notions of balance.

V. MIP formulations

As described in Section IV, we will consider two separate MIPs for either notion of balance. Corresponding to the notions of **S**taticBalance and **D**ynamicBalance, we will call the MIPs, the S − MIP and the D − MIP, respectively.

The MIPs will have some common parameters, as listed in Table I.

In addition to this, the MIPs also share the following **Decision Variables**. A *binary variable* v_{ij} indicates whether the VM i is finally to be placed on the PM j. Thus, $v_{ij} = 1$ implies that VM i resides on PM j in the solution configuration.

As discussed in Section IV, the S − MIP and the D − MIP have the same **Allocation** and **Resource** constraints. We proceed to formally enunciate these constraints, before moving on to the MIP-specific **Load Balancing** constraints.
The **Allocation** constraints (one for each VM i):

$$\forall i \in \text{VM} : \sum_{j \in \text{PM}} v_{ij} = 1 \qquad (1)$$

Notation	Arity	Explanation
C_{ij}	$i \in \mathrm{VM}$, $j \in \mathrm{PM}$	This encodes the initial state of the machine. $C_{ij} = 0$ if the VM i initially resides on PM j and is 1 otherwise.
Size_i	$i \in \mathrm{VM}$	Each virtual machine has a size associated with it.
Type_i	$i \in \mathrm{VM}$	Each virtual machine has a "type" associated to it; this is a symbolic string, for eg. CRM, SAP
$\mathrm{Resource}_j$	$j \in \mathrm{PM}$	Each physical machine has a limit on its resource.
Cost_{ij}	$i \in \mathrm{VM}$, $j \in \mathrm{PM}$	A transfer of virtual machine i to physical machine j incurs a cost.

TABLE I
PARAMETERS IN THE MIPS

Thus this constraint implies that every VM i is allocated, in the solution to precisely (a single) PM j.
The **Resource** constraints (one for each PM j):

$$\forall j \in \mathrm{PM} : \sum_{i \in \mathrm{VM}} v_{ij} \cdot \mathrm{Size}_i \leq \mathrm{Resource}_j \tag{2}$$

We proceed to discuss the cardinal **Load Balancing** constraints separately for the S – MIP and the D – MIP. In the following, we will also consider the corresponding *cost functions* for either MIP.

A. *The* S – MIP *formulation*

The **Load Balancing** constraints for S – MIP ensure that every machine receives an equal *proportion* of each *type* of virtual machine. Given a configuration of VMs on PMs, let us consider a specific PM j and calculate the sizes that VMs of various types should be placed on PM j. Define a parameter r_{jk} for every $j \in \mathrm{PM}$ and every $k \in \mathrm{TY}$ that evaluates to the number of units of VM of type k that should be placed on any specific PM. We can compute this parameter as follows:

$$r_{jk} = \frac{\sum_{i \in \mathrm{VM}:\mathrm{Type}_i = k} \mathrm{Size}_i}{\sum_{i \in \mathrm{VM}} \mathrm{Size}_i} \cdot \mathrm{Resource}_j \tag{3}$$

Given this, the constraint reads as follows:

$$\forall j \in \mathrm{PM}, k \in \mathrm{TY} : \sum_{i \in \mathrm{VM}:\mathrm{Type}_i = k} v_{ij} \cdot \mathrm{Size}_i \leq r_{jk} \tag{4}$$

Cost Function for S – MIP:
The objective function takes into account the total number of transfers across various physical machines and aims to minimize this total. Thus the objective function is modeled as:

$$\min \sum_{i \in \mathrm{VM}, j \in \mathrm{PM}} C_{ij} \cdot v_{ij} \tag{5}$$

Here, note that the vector of binary parameters C_{ij} encodes the initial configuration. We let $C_{ij} = 0$ if the VM i resides on PM j and 1 otherwise. Thus, the above cost function correctly encodes the *number* of transfers (i.e. where $C_{ij} = 1$).

This objective function is sufficiently generic. For instance, if we were to consider the problem where each transfer of a virtual machine to a physical machine has an associated *cost*, then we may appropriately transform our objective function as

$$\min \sum_{i \in \mathrm{VM}, j \in \mathrm{PM}} C_{ij} \cdot v_{ij} \cdot \mathrm{Cost}_{ij} \tag{6}$$

B. *The* D – MIP *formulation*

Building up to the **Load Balancing** constraints in D – MIP, we will keep a single "measure-of-balance" variable Z in addition to the other (binary) decision variables shared between S – MIP and D – MIP listed before. The *cost function* will now have *two* components as compared to the single component in S – MIP. We will also have a weighting parameter β in order to relatively weigh the two components in the cost function. The new constraints are as follows:

$$\begin{aligned} &\forall j \in \mathrm{PM}, k \in \mathrm{TY} : Z \geq \\ &\Big(\sum_{i \in \mathrm{VM}:\mathrm{Type}_i = k} v_{ij} \cdot \mathrm{Size}_i\Big) \cdot \Big(\sum_{i \in \mathrm{VM}:\mathrm{Type}_i \neq k} \mathrm{Size}_i\Big) - \\ &\Big(\sum_{i \in \mathrm{VM}:\mathrm{Type}_i \neq k} v_{ij} \cdot \mathrm{Size}_i\Big) \cdot \Big(\sum_{i \in \mathrm{VM}:\mathrm{Type}_i = k} \mathrm{Size}_i\Big) \end{aligned} \tag{7}$$

Cost Function for D – MIP:

$$\min \sum_{i \in \mathrm{VM}, j \in \mathrm{PM}} C_{ij} \cdot v_{ij} + \beta Z \tag{8}$$

Let us elaborate on the constraints given above and the extra term in the cost function. Suppose there is a configuration of the VMs on the different PMs that achieves perfect balance. We essentially want a function (of the different allocations) that is 0 precisely when this happens. Our constraints and the Z term in the objective function are a realization of this.

Note that a solution in v_{ij}s is perfectly balanced if the following holds for every type k:

$$\frac{\sum_{i \in \mathrm{VM}:\mathrm{Type}_i = k} v_{ij} \cdot \mathrm{Size}_i}{\sum_{i \in \mathrm{VM}:\mathrm{Type}_i \neq k} v_{ij} \cdot \mathrm{Size}_i} = \frac{\sum_{i \in \mathrm{VM}:\mathrm{Type}_i = k} \mathrm{Size}_i}{\sum_{i \in \mathrm{VM}:\mathrm{Type}_i \neq k} \mathrm{Size}_i} \tag{9}$$

The Load Balancing constraints in D – MIP are the *linearizations* of the above. We consider a deviation-from-balance term for each PM j and then we consider the *maximum* deviation-from-balance across all the PMs. This is encoded in the variable Z, which is to be thereby minimized, along with the original objective of minimizing the number (or cost) of transfers. We weigh these relative components via the parameter β: this parameter may be thought of as the penalty for *not* achieving perfect balance. Depending on the problem instance at hand, we may need to vary β in order to achieve reasonably effective feasible solutions to the VMSpreader problem.

C. Comparisons of the two formulations

Let us recall the three desiderata for solution approaches to the VMSpreader problem, as outlined in Section IV. By design, either of $\mathrm{S-MIP}$ or $\mathrm{D-MIP}$ achieves item (2) therein. However, notice that $\mathrm{S-MIP}$ suffers in that, it does not obey item (1). In fact, given the hardness results in Section III-B, we cannot expect $\mathrm{S-MIP}$ to always output feasible solutions.

For instance, consider the situation where we have two PMs each with 5 units of resource. Let there be two VMs, one of type CRM (of size 4 units) initially resident on PM_1, while the other VM is of type SAP (of size 4 units) resident initially on PM_2. In this case, no transfers are possible (since it is not allowed to transfer "part" of a VM). Thus the $\mathrm{S-MIP}$ returns as "infeasible".

The design of $\mathrm{D-MIP}$ was precisely to remedy this, via the useful artifice of *Lagrangian Relaxation*. Suppose that the variable Z in $\mathrm{D-MIP}$ had a hard constraint $Z = 0$ (encoding *perfect balance*). Lagrangian relaxation techniques imply that we may relax this constraint to $Z \geq 0$ and take it as a component in the *objective* function, with a corresponding Lagrangian relaxation parameter β. Addressing item (2) in the list of desiderata in Section IV, note that setting $\beta = \infty$ corresponds to infinite penalty for deviating from perfect balance; thus this leads to the (potentially infeasible) zone of "perfect balance". Lowering β from ∞ brings us to the feasible zone, while setting $\beta = 0$ corresponds to zero penalty for deviations from balance. An application at hand will have to consider varying β's; thus there is a tradeoff between quality of balance and cost of transfers. To reiterate, this is the best one can do, because of the hardness results in Section III-B.

However, one may now ponder as to why such a method was not adopted for $\mathrm{S-MIP}$ too. The issue is that there are quite a few constraints encoding "balance" in $\mathrm{S-MIP}$, which will all need to be taken into the objective function with corresponding Lagrangian parameters. While we could vary a single parameter β for $\mathrm{D-MIP}$, this would no longer be possible in this situation.

VI. Results

Given the two MIP formulations in Section V, we can consider solving their *linear relaxations* as linear programs (LP); however the variables v_{ij}'s may thereby be fractional. A typical theme in LP-based optimization is to derive a "good" integer solution by *rounding* the variables (either deterministically, or in a randomized fashion) in the LP. A motivation for such rounding schemes is that LP's are guaranteed to have polynomial runtimes, whereas such a guarantee is perhaps not possible for general MIP's (under the commonly believed hypothesis that $\mathrm{P} \neq \mathrm{NP}$). Nevertheless, we proceed to elucidate as to why such rounding schemes may not apply to the VMSpreader problem. While it is easy to round the fractional variables v_{ij} to satisfy the **Allocation** constraints, we will not be able to guarantee that the rounded variables satisfy either the **Resource** constraints or the **Load Balancing** constraints (for either of $\mathrm{S-MIP}$ or $\mathrm{D-MIP}$).

PM Id	Resource on PM	VMs of Type-1 (size)	VMs of Type-2 (size)	Total Utilized Resource
1	16	5 (13)	0 (0)	13
2	16	6 (12.5)	2 (2.5)	15
3	16	1 (2.5)	7 (12)	14.5
4	16	3 (5)	0 (0)	5
5	16	0 (0)	4 (10)	10

TABLE II
TABLE SHOWING THE INITIAL DISTRIBUTION OF THE APPLICATION TYPES ON DIFFERENT PHYSICAL MACHINES

Under this scenario, we have to apply mixed integer programming to the VMSpreader problem. We code the problem with AMPL and use the CPLEX solver to solve the optimization problems in $\mathrm{S-MIP}$ and $\mathrm{D-MIP}$ [13]. As mentioned before, the issue with $\mathrm{S-MIP}$ is that it may be *infeasible* because of the Load Balancing Constraints:

$$\forall j \in \mathrm{PM}, k \in \mathrm{TY} : \sum_{i \in \mathrm{VM}:\mathrm{Type}_i = k} v_{ij} \cdot \mathrm{Size}_i \leq r_{jk}$$

Although $\mathrm{S-MIP}$ appears to be inferior with regard to the accuracy of the results, and potential infeasibilities, its benefit lies in the faster generation of results, as compared to $\mathrm{D-MIP}$. This can also be understood from the standpoint of the variable Z in $\mathrm{D-MIP}$ that considerably complicates the branch-and-bound procedures inherent in typical MIP solvers. Thus, while $\mathrm{D-MIP}$ always gives feasible solutions (subject to suitably varying β), $\mathrm{S-MIP}$ often provides *fast* rough-cut solutions, which may be reasonably good for the case at hand.

We proceed to present our evaluation results based on a typical data set from a large hosting environment. The data set contains two multi-tier applications, denoted Type-1 and Type-2, spread over 5 physical machines. There are 15 applications of Type-1 and 13 applications of Type-2. The size of the VMs range from 0.5 to 8. The size of each physical machine (PM) is 16. The PMs are not packed to the full capacity. The split of VMs on each PM at the beginning is shown in Table II. Aggregate total allocation at the beginning for each type on each PM is shown in Figure 2. It can be observed from the figure that there are some PMs which have no VMs hosting application of one type, leading to the skew in distribution.

We solve the optimization problem with the objective functions as shown in Equation 5 and Equation 8. The solution of $\mathrm{S-MIP}$ yields 5 as the number of transfers required. The problem is solved in just 0.015 sec. The distribution of the VMs is shown in Figure 3.

The solution of $\mathrm{D-MIP}$ is 8 transfers. The time taken is 529.89 sec. Noteworthy here is the difference in the striping of the two applications found by the two solutions. The solution of $\mathrm{D-MIP}$ is significantly better than that of $\mathrm{S-MIP}$ in terms of striping the VM's across the available physical resources. Figure 4 shows the distribution of VMs for the case with 8 transfers.

The result indicates that even in instances where the $\mathrm{S-MIP}$ is solvable it may not be the optimal in terms of

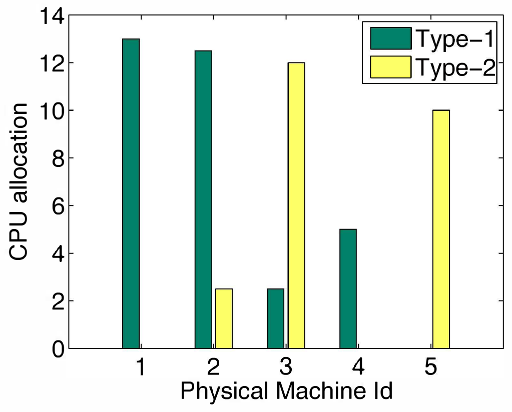

Fig. 2. Initial aggregate allocation for each type on individual Physical Machines

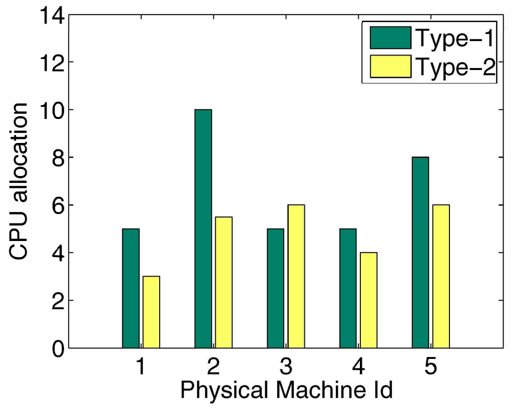

Fig. 3. Total size of Type-1 and Type-2 applications after distribution of VMs using $\mathrm{S-MIP}$. This takes 5 transfers, but fails to achieve good balance.

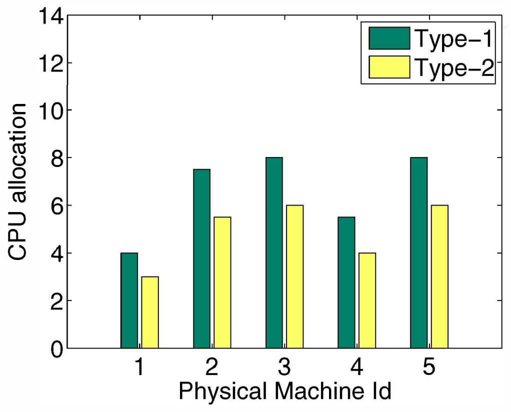

Fig. 4. Total size of Type-1 and Type-2 applications after distribution of VMs using $\mathrm{D-MIP}$. This takes 8 transfers, but achieves near optimal balance.

load balancing the VM's, as discussed in Section V. While the solution of $\mathrm{D-MIP}$ is typically closer to optimal, it takes much longer to solve. Thus, depending on the business scenario, it can be left to the user to tradeoff between a better balanced solution and the time taken to solve.

VII. Conclusion

The rise in the use of Virtual Machine technology in hosting applications in data centers present new set of challenges in service management. Virtual servers are used for hosting different tiers of multi-tier applications. As all applications are not provisioned at the same time, it is common to minimize the cost of infrastructure by hosting all the VMs of an application on the same physical machine. This introduces a single point of failure. Even when new n-tier application is added, the same procedure repeats. We propose to alleviate the problem by spreading the different tiers of the applications across the available physical resources. We modeled the striping of Virtual Machines as an optimization problem where the goal is to spread the n-tiered applications proportionally across the Physical Machines while minimizing the number of VM transfers. The Mixed Integer Problem (MIP) formulation of the problem is NP-hard. In order to reach a solution, we proposed two heuristics. We presented results on a data set taken from a real data center environment to show the performance of the solutions.

In the current approach, no restriction is imposed on the placement of VM's from two different types on a PM. However, one may want to prevent (for instance) all database servers to be co-located on a PM. This can be easily incorporated in our formulation.

A future direction to this work would be to allow for probabilistic failure of PM's and adjust the VM placements accordingly. For each PM, the Mean Time To Failure (MTTF) can be taken into account, and VM's can be preferentially placed on PM's with higher MTTF.

References

[1] D. A. Menasce, V. A. Almeida, R. Fonseca, and M. Marco A, "A methodology for workload characterization of e-commerce sites," in *In Proc. of 1st ACM conference on Electronic commerce*, 1999.

[2] "Amazon elastic compute cloud (amazon ec2)." [Online]. Available: http://aws.amazon.com/ec2/

[3] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off my cloud: exploring informatuon leakage in third-party compute cloud," in *In Proc. of Conference on Computer and Communications Security*, 2009.

[4] A. Verma, P. Ahuja, and A. Neogi, "pmapper: Power and migration cost aware application placement in virtualized systems," in *In Proc. of Middleware*, 2008.

[5] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data center," in *In Proc. of 16th international conference on World Wide Web*, 2007.

[6] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *In Proc. of Integrated Network Management*, 2007.

[7] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu, "Performance and availability aware regeneration for cloud based multitier applications," in *In Proc. of Conference on Dependable Systems and Networks*, 2010.

[8] "Load balancing of virtual machines." [Online]. Available: http://lbvm.sourceforge.net/

[9] A. Schrijver, *Theory of linear and integer programming.* New York, NY, USA: John Wiley & Sons, Inc., 1986.

[10] S. Berube, "Understanding micro-partitioning," *IBM Systems Magazine*, 2010.

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

[12] A. Jain, M. Murty, and P. Flynn, "Data clustering: A review," *ACM Computing Survey*, vol. 31, 1999.

[13] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*. Danvers, MA, USA: The Scientific Press (now an imprint of Boyd & Fraser Publishing Co.), 1993.