Performance Comparison of Fault-Tolerant Virtual Machine Placement Algorithms in Cloud Data Centers

Christopher Gonzalez, Bin Tang
California State University Dominguez Hills, Carson, USA

Abstract

Fault-tolerant virtual machine (VM) placement refers to the process of placing multiple copies of the same VM cloud application inside cloud data centers. The challenge is how to place required number of VM replicas while minimizing the number of physical machines (PMs) that store them, in order to save energy consumption of cloud data centers. We refer to it as fault-tolerant VM placement problem. *In our previous work, we have proposed a greedy algorithm to solve this problem. In this paper, we compare it with an existing research that is based on well-known Welsh Powell Graph-Coloring Algorithm to place items into bins while considering the conflicts between items and items and items and bins. Via extensive simulations, we show that our greedy algorithm can turn off 40-50% more PMs than existing work and can place upto four times as many VM replicas as existing work, achieving much stronger fault-tolerance with less energy consumption. We also compare both algorithms with the optimal integer linear programming (ILP)-based algorithm, which serves as the benchmark of the comparison.*

1. Introduction

Cloud data centers serve as the digital infrastructure of our society and provide various Internet services including social media, video streaming, and search engines. Virtualization, the building block of cloud data centers, refers to the technology where multiple OS environments run on the same physical machine (PM). With virtualization, the hardware resources of PMs such as CPU cycles, memory, and bandwidth are divided into several smaller isolated computing units, called virtual machines (VMs). VMs can then be allocated and rented to cloud users in a pay-as-you-go manner (e.g., Amazon EC2 and Microsoft Azure).

To provide a seamless service to cloud users in spite of failures such as cloud outages and server failures, replicating VMs and placing their replica copies into different PMs becomes an important fault-tolerant technique in cloud data centers [7,11]. For example, Amazon S3 Replication can automatically replicate S3 (Simple Storage Service) objects across different AWS Regions [1]. For Microsoft Azure, the data in its geo-redundant storage is always replicated three times in the primary region [2]. In general, when considering fault-tolerant VM placement, there are three constraints that need to be satisfied.

- *Fault-tolerance constraint* of VMs: For the fault-tolerant purpose, it is preferred that multiple replica copies of the same VM application are placed into different PMs.
- *Resource capacity constraint* of PMs: Each PM in cloud data center has limited amount of cloud resources such as CPU cycles, memories and storages, and bandwidth.
- *Compatibility constraint* of VMs to PMs: Due to software and platform incompatibility, some VM applications (and their replicas) cannot be installed onto some PMs.

Meanwhile, it is well-known that energy consumption in cloud data centers contributes around 1.5% of the worldwide electricity usage [9] and the energy consumption of cloud servers (i.e., PMs) contributes around 40-60% of the total energy consumption of a typical data center. Therefore how to place multiple replica copies of VMs in cloud data centers while satisfying above three constraints and minimizing the number of active PMs (i.e., PMs that are turned on) becomes an important problem.

In our previous work [6], we have designed an optimal integer linear programming (ILP)-based algorithm and a time-efficient greedy heuristic algorithm to achieve fault-tolerant VM replica placement. In this paper, we further validate our algorithms and compare them with some existing work. Gupta et al. [8] proposed a server consolidation problem, in which multiple conflicting underutilized servers are consolidated into fewer servers to save energy. They introduced both item-item and bin-item incompatibilities, which are corresponding to the fault-tolerance constraint of VMs and compatibility constraints of VMs to PMs addressed in our paper. It proposed a two-stage item-to-bin placement heuristic based on the well-known Welsh-Powell graph coloring algorithm [10]. We refer to is as *Two-Stage Algorithm*. As Two-Stage is very related to ours as well as well-cited with more than 90 citations [3], comparing with it is an important step to validate the quality of our research.

Paper Organization. The rest of the paper is organized as follows. To make the paper self-contained, we introduce the problem formulation in Section 2 and our algorithms

in Section 3. We also describe in details the Two-Stage Algorithm. [8] in Section 3. We construct an example to illustrate how different algorithms work and give our insight why Two-Stage does not perform as well as our algorithm. In Section 4, we compare all the three algorithms and discuss the results in details. Section 5 concludes the paper with some possible future work.

2. Problem Formulation

System Models. In the data center there are p PMs denoted as $V_p = \{1, ..., |V_p|\}$. Initially a set of l distinct *original VMs* $V_m = \{v_1, v_2, ..., v_l\}$ have already been submitted by cloud users to the PMs to be executed. v_j $(1 \leq j \leq l)$ is stored at its *source PM* $s(j) \in V_p$. A source PM can store multiple original VMs. We denote the set of source PMs as $V_d \subseteq V_p$. We define the *incompatibility set* of VM v_j as the set of PMs that v_j and its replica copies cannot be placed upon, and denote it as $\mathcal{I}(j) \subset V_p$.

For each VM to run, it needs one unit of cloud resources (i.e., CPUs, memories, and disk I/O) for execution. Let rc_i denote the *resource capacity* of PM $i \in V_p$; that is, the total number of original or replica VM copies PM i can store is rc_i. Thus if i is a source PM of some original VMs, its available resource capacity becomes $rc_i - |\{1 \leq j \leq l | s(j) = i\}|$. Due to diverse fault-tolerance requirement from the cloud users, it requires to place $r_j \geq 0$ copies of v_j into the data center (when $r_j = 0$, it does not need to place any replica copies besides the original VM v_j). Let $R = \max\{r_1, r_2, ..., r_l\}$. Denote the k^{th} replica copy of $v_j \in V_m$, where $1 \leq j \leq l$ and $0 \leq k \leq r_j$, as $v_{j,k}$ ($v_{j,0}$ is the original copy v_j). Fig. 1 shows a small cloud data center with eight PMs $\{1, 2, ..., 8\}$ and three original VMs $\{v_1, v_2, v_3\}$. It also shows the incompatibility set of each VM, number of required VM replicas and their IDs.

We note that although rc_i is the resource capacity available at PM i, it is possible that not all of its resources can be utilized at i. The fault-tolerance constraint, which stipulates that multiple copies of the same VM be placed at different PMs in order to survive PM failures, incurs two consequences. First, any original VM and its replicas must be placed onto different PMs, thus it must be that $R + 1 \leq |V_p|$; otherwise, it is infeasible for the VM placement. Second, as there are l original VMs, a PM can store at most l VM copies, each from a different VM, even though its storage capacity could be larger than l. We thus define *effective resource capacity* of PM i, denoted as rc_i^e, as the maximum resource capacity of i that can be used to store VMs. rc_i^e is the smaller value between the available resource at i and the number of the VMs it can further store (besides its own stored original VMs if it has). That is, $rc_i^e = \min\{rc_i - |\{1 \leq j \leq l | s(j) = i\}|, l - |\{1 \leq j \leq l | s(j) = i\}|\}$.

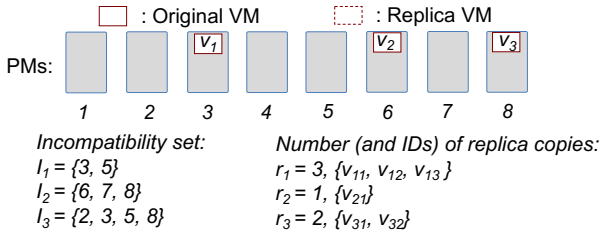

Figure 1. A small cloud data center with $p = 8$ PMs and $l = 3$ original VMs.

We define *active PMs* as PMs that have at least one VM copy (either original or replica) after VM placement thus have to be turned on. Otherwise, it is *inactive* therefore can be turned off. For any source PM $s(j) \in V_d$, $1 \leq j \leq l$, as some original VMs have already been submitted and are currently being executed there, they are considered as active PMs thus cannot be turned off.

Problem Formulation. Recall that $v_{j,k}$ is the k^{th} replica copy of VM $v_j \in V_m$. We define *placement function* $r : V_m \times \{1, 2, ..., R\} \rightarrow V_p$, indicating that $v_{j,k}$ is placed to *destination PM* $r(j, k) \in V_p$. As the original VM v_j is located at $s(j)$, we have $r(j, 0) = s(j)$. Let $y_i = 0$ and 1 indicate that PM i is inactive and active respectively after VM placement r. The goal is to find r that minimizes the total number of active PMs, i.e., min $\sum_{i=1}^{|V_p|} y_i$, under fault-tolerance constraint of VMs, resource capacity constraint of PMs, and compatibility constraint of VMs to PMs.

3. Fault-tolerant VM Placement Algorithms

In this section, we first present Two-Stage Algorithm and then our own fault-tolerant greedy algorithm. Due to space constraint we omit the optimal ILP-based algorithm. Please refer to [6] for more details.

Two-Stage Algorithm [8]. It consists of two stages.

Stage 1. In the first stage it constructs a *conflict graph* of VM replicas. In particular, there exists an edge for any two VM replicas (i.e., replica-nodes) from the same original VM, as they cannot be put into the same PM. Next, it follows the well-known Welsh-Powell graph coloring algorithm [10] to color all the VM replicas such that any two conflicting replicas must have different colors (i.e., go to different PMs). The number of colors represents the least number of PMs it needs to store all the VM replicas. Stage 1 has the following four steps.

1. All the vertices initially are uncolored. Sort the vertices in the conflict graph in decreasing order of degrees.
2. Color the first vertex on the sorted list with color 1.
3. Traverse the vertices in the sorted list, and assign a vertex the same color if it is uncolored and if the vertex does not yet have a neighbor with the same color.

4. If there are uncolored vertices in the list, repeat Step 3) with a new color, until all the vertices are colored.

Stage 2. In their second stage, Gupta et al. [8] considers the *incompatibility constraint* between VM replicas and PMs. The incompatibility constraint specifies that all the VM replicas of a specific original VM cannot be placed onto a specific PM due to software and platform incompatibility. It first introduces pre-colored dummy nodes, each for a PM that has incompatibility with some VM replicas (i.e., PM-node). Then it augments the conflict graph in Stage 1 such that 'pre-colored' PM-node has an edge with any VM replicas that cannot be assigned to this particular PM.

In particular, it introduced a *modified Welsh-Powell Algorithm* that runs upon this newly augmented conflict graph. To proceed, it maintains two sorted lists. One contains the sorted 'pre-colored' PM-nodes in the decreasing order of the degree. The other contains the sorted uncolored replica-nodes in the decreasing order of the degree. It then takes place in below three steps.

1. Select the next pre-colored vertex from the pre-colored PM-node list.
2. Traverse the vertices in the uncolored list, and assign a vertex the color of the pre-colored vertex of Step 1) if it is uncolored and does not have a neighbor with the same color. Go to Step 1).
3. For the vertices (in the uncolored list) that remain uncolored at the end of the traversal process of the pre-colored list, color them using the usual Welsh-Powell Algorithm.

Discussions. The rational of Welsh-Powell Algorithm for graph-coloring is as follows. One undesirable situation in graph-coloring is that a new color needs to be created to color a node whenever all the created colors have already been used to color its neighbors. This occurs when a node with high degree is visited *after* many of its neighbors have already been visited and colored. To prevent this situation, we thus need to visit and color all high-degree nodes first before visiting and coloring their neighbors. Doing so decreases the chance of creating a new color thus saving colors. The Welsh-Powell algorithm is proven to use at most $\Delta(G)+1$ colors, where $\Delta(G)$ is the maximum degree of the conflict graph G [10].

Fig. 2 shows how the Two-Stage Algorithm works for the example in Fig. 1. In particular, Fig. 2(a) shows the augmented conflict graph of the example. Fig. 2(b) shows the sorted list of PM-node and the sorted list of replica-node, and how the color is assigned from the PM-node to the replica-node. Note that for clarity, the ID of the PM node serves as the color of the PM-node thus the color does not start with 1 as mentioned in the algorithm. Also note that PMs 1 and 4 are not on the sorted list of PMs as they do not have any conflict with any replicas. Fig. 2(c) shows the final placement of all the replicas in the PMs. It shows that PMs 2, 3, 6, 7, 8 have to turn on while PMs 1, 4, 5 can be turned off to save energy. The number of active PMs from the Two-Stage Algorithm is thus 5.

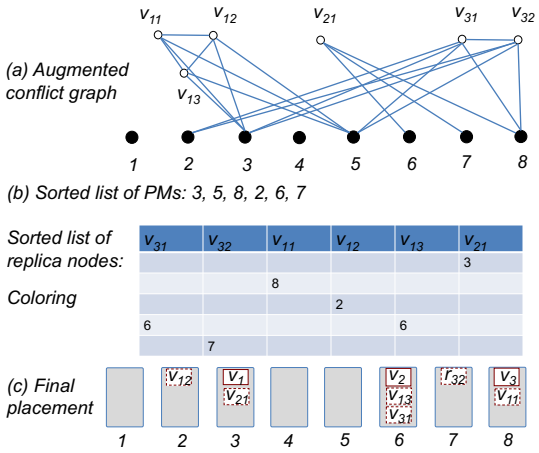

Figure 2. How Two-Stage works for the example in Fig. 1.

Greedy Algorithm. Next we present our own greedy algorithm Algo. 1 proposed in [6], which works as follows. As all the source PMs must be turned on, they are the initial set of active PMs. We sort all the replicas in the non-ascending order of the cardinalities of their incompatibility sets, and place the ones with largest incompatibility first. Replicas with largest incompatibility sets have less number of PMs to place upon, thus should be placed first before running out of options. It places each replica into the first available PM in the active PM set while satisfying fault-tolerance, compatibility, and resource capacity constraints. If not successful, we turn on another PM that is compatible with this replica, place this replica in it, and add it in the active PM set. It stops until all the replicas are placed in the data center. Sorting takes $l \cdot \log l$, placing replicas takes $l \cdot R$, $R = \max\{r_1, r_2, ..., r_l\}$. Thus the time complexity of this algorithm is $O(l \cdot (\log l + R))$.

Algorithm 1: Greedy VM Replica Placement Algorithm.
Input: An cloud data center instance $(V_p, V_m, rc_i, r_j, \mathcal{I}(j))$.
Output: The set of active PMs.

0. **Notations**:
   A: set of active PMs;
1. Sort $\mathcal{I}(j)$, $1 \le j \le l$, in the non-ascending order of their cardinalities $|\mathcal{I}(j)|$;
2. WLOG, let $|\mathcal{I}(1)| \le |\mathcal{I}(2)| \le ... \le |\mathcal{I}(l)|$;
3. $A = \{s(j)\}$, $1 \le j \le l$;
4. **for** ($j = 1$ to l)
5. **for** ($k = 1$ to r_j)
6. Let $\mathcal{C}(j) \cap A = B$;
7. **if** ($B == \phi$) // B is an empty set
8. Let x be the first element in $\mathcal{I}(j)$;
9. $A = A \cup \{x\}$;
10. **else**
11. Let x be the first element in B;

12. **end if;**
13. Place $r_{j,k}$ at x;
14. **end for;**
15. **end for;**
16. **RETURN** A. // Return the set active PMs

Fig. 3 shows how the Algo. 1 works for the example in Fig. 1. It shows that PMs 1, 3, 6, 8 have to turn on while PMs 2, 4, 5, 7 can be turned out to save energy. The number of active PMs is 4. Greedy performs better than Two-Stage by being able to turn off one more PM.

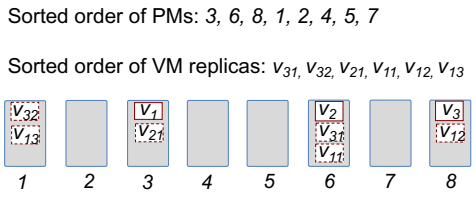

Figure 3. How Greedy works for the example in Fig. 1.

Discussions. An interesting question to answer is: Why Greedy performs better than Two-Stage for the example in Fig. 1? Let's take a look of the turned-off PMs in both cases. Greedy turns off both PMs 2 and 7, which are left on by Two-Stage. The reason PMs 2 and 7 are turned on by the Two-Stage is because it maintains a sorted list of pre-colored PM-nodes that conflict with VM replicas. This implies that the higher incompatibility a PM has with VM replicas, the better chance it is colored and selected to store VM replicas. PMs 2 and 7, which both have some incompatibility with some VM replicas, are colored and thus chosen to store other VM replicas. This example demonstrates that the idea of forcing PM nodes that are incompatible with some VM replicas to store other VM replicas does not seem to be a good strategy in order to turn off more PMs. In an extreme case wherein a PM is incompatible with all the VM replicas, for example, it is still colored for storing VM replicas even though it cannot store any. In contrast, our Greedy algorithm tries to put as many VM replicas into source PMs (which have to be turned on), and only turn on other PMs when needed, which has a clear focus on turning off more PMs.

4. Performance Evaluation

Simulation Setting. In this section we compare our designed algorithms with the existing work. We refer to our ILP-based algorithm as **ILP**, the greedy VM replica placement algorithm as **Greedy**, and the two-stage existing work as **Two-Stage**. We implemented ILP using LP solver *lpsolve* [4] and wrote our own Java simulator for other algorithms on a Windows10 computer with Intel Core i7-6500U 2.50 GHz CPU and 32 GB RAM. We compare all

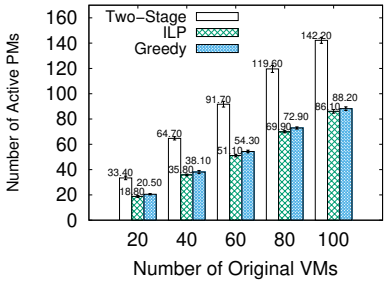

Figure 4. Number of active PMs by varying l, number of original VMs. Here, $k = 8$, $r_j = [5, 10]$, and $rc_i = 10$.

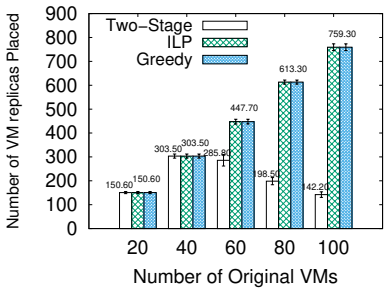

Figure 5. Number of placed VMs by varying l. Here, $k = 8$, $r_j = [5, 10]$, and $rc_i = 10$.

the algorithms in terms of active PMs; i.e., number of PMs that have at least one VM copy (original or replica) after executing each of the algorithms. We adopt the fat tree [5] for the data center topology. We consider a small $k = 8$ data center with 128 PMs and a large $k = 16$ data center with 1024 PMs, where k is the number of port of the switch in fat tree topology. Each data point in the plots is an average over ten runs. The error bars indicate 95% confidence interval. The number of replica copies of each VM r_j is a random number in [5,10] unless otherwise mentioned.

Effects of varying l, number of original VMs. Fig. 4 shows the performance comparison when varying l. It shows that as l increases, the number of active PMs of all three algorithms increases, as more VM replicas are placed inside the cloud data center thus less number of PMs can be turned off. We also observe that among the three algorithms, ILP, being the optimal algorithm, always results in the least number of active PMs, thus saving energy the most. Greedy however, performs very close to the ILP, giving less than 5% of more active PMs compared with ILP. Both ILP and Greedy outperform the Two-Stage, showing that Two-Stage is the least energy-efficient fault-tolerant VM placement algorithms. In general, we observe that Greedy can turn off 40-50% more PMs than Two-Stage.

Fig. 5 shows the number of placed VM replicas of the three algorithms. When l is small ($l = 20, 40$), all three

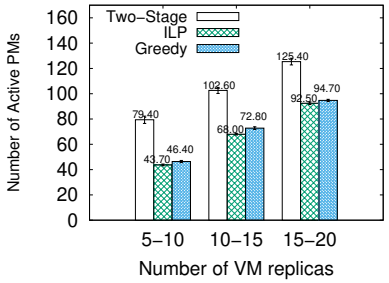

Figure 6. Number of active PMs by varying r_j, number of replica copies of each VM. Here, $k = 8$, $l = 50$, and $rc_i = 10$.

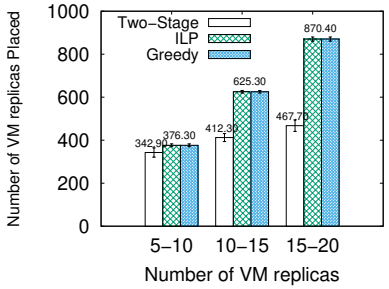

Figure 7. Number of placed VMs by varying r_j. Here, $k = 8$, $l = 50$, and $rc_i = 10$.

algorithms are able to place all the VM replicas required for each original VM. However, When l gets larger ($l \geq 60$), Two-Stage is no longer able to place all the replicas while Greedy and ILP are still able to place all the replicas. This shows that our algorithms are more fault-tolerant than Two-Stage. In some case ($l = 100$), Greedy can place around four times as many VM replicas as Two-Stage.

Effects of varying r_j, number of replica copies of each VM. Fig. 6 and 7 show the performance comparisons when varying r_j. First, with the increase of r_j, both the number of active PMs and number of placed VM replicas for all three algorithms increase, as more replica copies are placed to achieve stronger fault-tolerance. Again, we observe that Greedy performs very close to the ILP in terms of both active PMs and placed VM replicas, while both outperform the Two-Stage in the entire parameter range.

5. Conclusion and Future Work

We presented the performance comparison of fault-tolerant virtual machine placement algorithms in cloud data centers. We compared our greedy algorithm with the existing one based on Welsh Powell graph-coloring algorithm in terms of active PMs and VM replica copies placed. We showed that our greedy algorithm can turn off 40-50% more PMs and can place upto four times as many VM replicas as the existing work, thus achieving stronger fault-tolerance with better energy efficiency. We also compared both algorithms with the optimal ILP-based algorithm, which serves as the bench mark of our compared algorithms. In the future, we plan to systematically find out if fault-tolerant VM placement can be achieved for any given problem instance. In the future we will consider that different VMs consume different amounts of resources, and investigate how this affects the existing fault-tolerant VM placement algorithms and their comparisons.

Acknowledgement

This work was supported by NSF Grant CNS-1911191.

References

[1] Amazon s3 replication. https://aws.amazon.com/s3.

[2] Azure storage redundancy. https://docs.microsoft.com/en-us/azure/storage/common/storage-redundancy.

[3] Google scholar. https://scholar.google.com/.

[4] Linear programming solver lp_solve. https://sourceforge.net/projects/lpsolve/.

[5] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, 2008.

[6] C. Gonzalez and B. Tang. Ft-vmp: Fault-tolerant virtual machine placement in cloud data centers. In *Proc. of the IEEE ICCCN*, 2020.

[7] H. Goudarzi and M. Pedram. Energy-efficient virtual machine replication and placement in a cloud computing system. In *Proc. of IEEE Cloud*, 2012.

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

[9] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J. M. Pierson, and A. V. Vasilakos. Cloud computing: Survey on energy efficiency. *ACM Comput. Surv.*, 47(2), 2014.

[10] D.J.A. Welsh and M.B. Powell. The upper bound for the chromatic number of a graph and its applications to timetabling problems. *The Computer Journal*, 11:41–47, 1967.

[11] Z. Zheng, T. C. Zhou, M. R. Lyu, and I. King. Component ranking for fault-tolerant cloud applications. *IEEE Transactions on Services Computing*, 5(4):540–550, 2012.

Address for correspondence:

Christopher Gonzalez
Computer Science Department
California State University Dominguez Hills
Carson, California 90747
cgonzalez393@toromail.csudh.edu