A review on
“Application Performance Management in Virtualized Server Environments” (Khanna et at.)

By: Rakesh Kumar Mallick
ME, CSA
Sr. No.: 4710-410-091-06888

Abstract:
"Server Sprawl" is a term used for having many underutilized servers running on 10%-35% capacity leads to wasting resources, and increases total cost of ownership. To address this problem the concept of server consolidation using virtualization is introduced and it points out issues that arises in the area of application performance. Algorithms presented are attempt to minimize the cost of migration of VM, and to maintain acceptable application performance levels.

Introduction:
• Critical problem of Server-Sprawl, increased cost of ownership.
• An architectural approach to address this problem: virtualization.
• server consolidation: reduce the no. of servers in data-center by grouping together multiple applications.
• Each application is associated with Service level agreements(SLA) [eg: throughput or response time etc.]

Background:
A simple algorithm for server consolidation adapted by some industries:
1. For each server to be consolidated, compute av. CPU usage, memory requirements, disk I/O, and network bandwidth. Let us say there are X servers.
2. Choose target server type with compatible architecture, associated memory, access to shared disk storage and N/W communication.
3. Take each of these X servers, and construct a virtual machine image for it. So we will have X Vms.
4. int j=0;
   for(i = 0; i < X; i++)
   {
   if (sufficient required resources available in PM(j) for VM(i))
   map VM(i)--->PM(j);
   j++;
   }

In this paper, techniques are purposed to provide:
(a) migration of VM when necessary in terms to maximize the performance.
(b) if any of the VMs report an SLA violation, perform dynamic reallocation of VM.

Related Work:
• Schemes for virtualization and Server management
• Algorithmic approach of packing objects into bins (i.e. bin-packing problem)
  NP-Hard problem so only approximation solutions are possible

Problem Formulation:
• For simplification homogeneous physical servers are considered. For heterogeneous systems
scaling of migration cost is to be done.

- Objective: Design algo. that will resolve SLA violations by reallocating VMs to Pms.
- SLA determines the desired response time which determines threshold of utilization, crossing which triggers reallocation procedure.
- Input includes a function which maps the individual resource utilization of VMs to the combined utilization of local PM.
- Load in PM(k) in i'th resource is given by Li(t).
- System should operate below the knee in order to meet SLA requirement. Since Li(t) are modeled as random process we are looking for \[P(Li(t)< ni) < \text{confidence\_level}\].
- Cost of migration of VMj = Mc. Uj(t) where Mc: cost of migration of one unit vector; Uj(t): Resources Utilization vector
- Cost of introducing new PM = Nbc
- Goal of Algo: Keep variance of the vector R(t) [residual capacity = Cj-Li(t)] as high as possible.
- Finally goal is to maximize [variance - cost of migration of VMs - cost of introducing new PMs]. Here variance is an L2 norm of the variance vector.

**Design Challenges:**

- Mapping of VMs to PMs is NP-Hard, so heuristic algos. are possible

**Algorithm:**

- Initial allocation is already provided.
- VMs are ordered in non-decreasing utilization within each Pms.
- Constraints, as in eq(2), is always monitored in an interval, violating which triggers migration algo.
- An instance of utilization falling below to low mark also triggers algo for consolidation.
- Select VM from PM(k) [threshold constraints are violated here] with lowest utilization and move it to PM having least residual-capacity big enough to hold this VM.
- If algo. is unable to find a PM big enough to hold migrating VM, introduce a new PM and allocate VM to this new PM.
- Select VM with lowest utilization across all PMs, and move it to another PM. do this if migration results in higher variation.
- Repeat above step until variation starts decreasing or not enough residual space is left.
- Remove PM if all of it's VMs are migrated.

**Conclusion:**

- The ideas of consolidation is good and mechanism of adding and removing PMs makes it even more better.
- Eq(2) minimizes the cost of migration and no. of Pms.
- The heuristic algo. used is one of the safest one.
- Consolidation using variance is a nice idea.
- SLA is being taken care of during all the process by taking a knee.

-----------------------------------------------------------------------------------------------------------------------

**Questions:**

1. How VMs are migrated here: by stopping VM temporarily and then migrating or copying an image of VM to another machine while it is running.

**Things I like:**

1. Introduction of new PM as required, and to remove when not needed.
2. Variance of residual capacity is kept as high as possible. It will be highest when few machines are working with their full capacity and all others are switched-off leading to a very good consolidation.
3. Generalization of their problem statement in form of a eq.[eq(2)].
4. Searching destination PM takes O(log(m)) time.

**Things I don't like:**
1. Residual capacity of of a machine is kept as low as possible(say zero). This will lead to highly repeated migration of VM when resource demand increases continuously. Sandpiper is doing opposite to it(goes with worst fit of a VM in a PM).
2. Experimental results are not enough to support their research and claims.

**Future Work:**
• Their consolidation ideas can be merged with sandpiper's VM migration work where they try to provide extra resources for future needs (i.e. there worst fit strategy).
• In order to overcome hotspot, if more than one VM is required to be migrated then Sandpiper is making possible mapping of VM to PM before starting migration. But here, if not enough space is to migrate, a new PM is introduced. Migration of more than one VM could do better, because it might be cheaper to migrate 3-4 VMs than introducing new PM. [I could be wrong here, since all VMs could be best fit already in 2nd case, but we are not checking it every time].