A General Model for Virtual Machines Resources Allocation in Multi-tier Distributed Systems

Paolo Campegiani, Francesco Lo Presti
Università di Roma Tor Vergata
Dipartimento di Informatica, Sistemi e Produzione
Via del Politecnico 1, 00133 Roma
campegiani@ing.uniroma2.it,lopresti@info.uniroma2.it

Abstract

We propose a general model for resources allocation of virtual machines in multi-tier distributed environments. Our model describes each virtual machine and each physical host by a multi-dimensional resource vector, allowing the coexistence of both quantitative and qualitative resources, also handling different SLAs. As this model is a generalization of the classical 0/1 Knapsack Problem, we have also developed an heuristic to obtain very near optimum solutions in a timely manner.

1. Introduction

As many computer science paradigms, virtualization has come a long history, and after many years when it has been secluded to high end systems is now on the rage.

Virtualization could be defined as a two phase process. In the first phase, some resources of the same kind will be grouped together, hiding physical boundaries; in the second phase, a portion is carved out from this aggregated compound and presented to an user. There are many types of virtualization, depending on the type of the aggregated resource. VLAN are made for better network security, NAS and SAN system allows for cheaper data consolidation, backup and security, and Java JRE is instrumental in providing the same software to a wide variety of clients. By adding an intermediate layer between physical resource and resource demand, virtualization allows to demultipex and routing requests to a single management point, achieving better scalability, manageability, performances and security. In operating system level virtualization, on each physical host there is a Virtual Machine Monitor, also called hypervisor, that allows for many virtual hosts, called virtual machines, to share the physical resources. In fact, efficiency in resource allocation is the compelling reason that forces a data center to adopt operating system level virtualization, as network and storage virtualization have been in the last decades. The average server runs usually at $10 - 15\%$ of its capacity, so consolidating more systems inside it allows for costs reduction, being the operational costs on a per physical server basis. Each physical machine has an hypervisor that arbitrates accesses to resources stemming from virtual machines, offering a standard view of heterogeneous physical systems, allowing to consider a system as a standardized building block for large and complex distributed system.

In this paper we consider the problem of allocating a distributed, multi-tier system made up of virtualized physical machines, over a pool of physical hosts. Both hosts and virtual machines are characterized by a multidimensional resource vector capturing the associated resources (CPU, disk, memory,...). We formulate the problem as an integer programming problem which can be regarded as a generalization of the the knapsack problem [13]. Given the complexity of the exact problem, we also propose an efficient heuristic which provides good results in different scenarios. The proposed solution lends itself to implementation in systems with varying workload even over short time scale which requires continuos reconfiguration to optmize servers' performance and resources' usage.

Our approach differs from most of the prior works which tackle the problem as a monodimensional problem, i.e., by considering only the CPU as a resource to allocate, by accounting for all the resources in the system. This is important as the performance of a virtual machine stems from all the available resources, with different degree of intensity according to the virtual machine role. As an example, nodes of the front end tier require a lot of bandwidth for connections with external clients, and nodes comprising a database tier are bounded by available storage bandwidth. As shown in [9] all of these (and possibly others) parameters are required to have a correct estimation of service demand times, so a general resource allocation model should

necessarily be multi-dimensional.

The rest of the paper is organized as follows. In section 2 we briefly discuss related works. In section 3 we define the model both informally and as a linear programming problem, also showing that is a generalization of the knapsack problem. The heuristic we have developed is shown in section 4, and section 5 shows the results of the heuristic for two different datasets. We conclude with some indications for future works on section 6.

2. Related work

Many studies have been done where virtualization allows for an easier reconfiguration of a distributed system subjected to a changing workload. As mentioned earlier, in all these studies, the arbitrated resource is mono-dimensional, being the CPU shares or the number of physical servers, only in [12] the model is 2-dimensional, considering a load dependent resource as the CPU and a load independent resource as the main memory.

In [14] the CPU shares are dynamically allocated with the goal to optimize a global utility function, under varying workload levels. In [3] is defined a global QoS formula that takes into account deviations from average response times, throughput and probability of rejection with an autonomic controller that tries to minimize its value arbitrating the number of system processes devoted to serve the incoming requests. In [4] the proposed architecture involves different Application Environments (AEs), each one comprising several physical machines bounded together. Each AE serves different classes of transactions, and server could be moved from one AE to another, to optimize a global utility function which is based on the performance metrics of the AEs, like response time and throughput. The solver searches for the optimal number of physical servers for each AE, with a beam search algorithm.

In [18] a similar architecture has been considered for appliance-based autonomic provisioning. The architecture defines some Virtual Application Environments (VAEs): a VAE spans over one or more virtual servers, and each virtual server is defined inside a physical machine. Each VAE has a On-Demand Router that dispatches incoming requests to the less loaded virtual server inside the VAE, in a round-robin fashion. A global and utility-driven model solver founds the better configuration for the VAEs for the given and forecasted workload. The solver is also virtualization aware, as it takes into account the time required for virtual machine provisioning, i.e. the time required to activate a virtual machine and the time required for closing it once it's no longer needed; the overhead due to virtualization itself is assumed as one tenth of the available resources.

In [2], the model focuses on SLA violations, trying to minimize them. To get a solvable performance model, the probability of a service time bigger than the agreed value is bounded via the Markov Inequality, and the model is mono-dimensional.

Other approaches are possible, particularly based on control theory: in [15] it's exposed a control of CPU shares of two competing virtual machines over the same physical node.

3. The mapping problem

Our problem could be defined as: given a set of virtual machines and a pool of physical hosts, each one described by a multi-dimensional resource vector, which is the best mapping of the virtual machines to the physical hosts, according to a metric that maximizes resources efficiency usage? This problem could be put in the context of autonomic computing, where a performance model solver has determined the numerousness of each of the tiers comprising the architecture, and resources allocation must be done in a timely manner. We define this problem as the mapping problem. We formalize the model in paragraph 3.1 and showing that is a generalization of the knapsack problem in 3.2.

3.1. Formal model for the mapping problem

We formalize the mapping problem to allow for maximum generalization. To do so, we assume that each virtual machine could have different resource demand vectors, and for this group of available options we have to choose only one element that also returns us a profit. Profits and demand vectors are sound, so we can choose a more demanding service level in the sake of a bigger profit. We want to maximize the grand total of profits, while minimizing the number of hosts we have to use. It's possible that, for some or even all groups, we have only one virtual machine comprising it, meaning that we cannot do anything but instantiate that machine, and in such a case the problem is only to find where to instantiate it.

As the virtual machines are pooled in groups, we indicate each one of them by two indexes, the first denoting the group and the second the specific machine in the group (i.e. the service level). Following this convention, if X is a generic scalar (or vector), X^{ij} is the scalar (or vector) pertaining to the j-th machine of the i-th group. We stipulate that:

- G is the number of groups. Each group is composed of g_i different machines (it's possible that $g_i = 1$ for some or even all i's);
- each virtual machine is described by a K-dimensional demand vector $D^{ij} = (d_1^{ij}, d_2^{ij}, ...d_k^{ij})$;

- each virtual machine has an associated profit P^{ij};
- M is the number of physical hosts;
- each host is described by a K-dimensional resource vector:
  $R^l = (r_1^l, r_2^l, ..., r_k^l), 1 \leq l \leq M$;
- for each i, we have $D^{i1} \leq D^{i2} \leq ... \leq D^{ig_i}$, coordinate wise, and $P^{i1} \leq P^{i2} \leq ... \leq P^{ig_i}$.

The decision variable x_m^{ij} is 1 if the virtual machine ij is instantiated over the physical hosts m, otherwise is 0.

We want to choose one and one only virtual machine from each group, and allocate it on a physical host, with the constraint that we cannot exceed the available resources, maximizing the total profit earned and minimizing the number of physical machines that are used. To do so, we define the variables u^m to have the value 1 if the physical host m has at a least one virtual machines instantiated over it (so it's used), otherwise 0. Our objective function is:

$$P = \sum_{i=1}^{G} \sum_{j=1}^{g_i} \sum_{m=1}^{M} x_m^{ij} P^{ij} - C * \sum_{m=0}^{M} u_m \qquad (1)$$

which is the total profit of the virtual machines that are instantiated minus the number of physical hosts used times a convenient constant C. We assume C as a constant as the operational costs for running the infrastructure are usually proportional to the size of the the infrastructure itself: as we want to maximize their usage (by allowing for different service levels) we also want not to use more than the strictly necessary. We extend the $\leq$ operator from scalar to vectors in a coordinate wise fashion: if $X = (x_1, x_2, ..., x_n), Y = (y_1, y_2, ..., y_n)$ we say that $X \leq Y$ iff $x_i < y_i$ for each i s.t. $1 \leq i \leq n$, so constraints are formally defined as:

$$\forall i \,, \sum_{m=1}^{M} \sum_{j=1}^{g_i} x_m^{ij} = 1 \qquad (2)$$

$$\forall m \,, \sum_{i} \sum_{j} x_m^{ij} D^{ij} \leq R^m \qquad (3)$$

$$\forall m, i, j \,, \; u_m \geq x_m^{ij} \qquad (4)$$

Eq. 2 means that we choose only one virtual machine for each group, and eq. 3 means that, on each physical machine, we cannot allocate more resources than available ones, while eq. 4 relates variables u_m to x_m^{ij}.

3.2. The mapping problem as a generalization of the knapsack 0/1 problem

The mapping problem is a generalization of the well known knapsack problem [13]. The generalization stems from these considerations:

- the knapsack problem is mono-dimensional, whilst the mapping problem is multi-dimensional;
- the knapsack problem is with only one knapsack (physical machine), whilst the mapping problem deals with multiple knapsacks (physical hosts);
- the knapsack problem doesn't group items (virtual machines), the mapping problem does.

To the best of our knowledge, there are no published studies (in the field of computing performance modeling or operational research) that tackle all these generalizations together. Multi-dimensional knapsacks, called MDKP, are discussed in [5, 7]. Multi-knapsacks problems, are studied in [8, 6]. Knapsacks where the constraint is to choose only one item for each available group are called multiple-choice knapsacks (MCKP) and a minimal algorithm to solve them is shown in [16]. In [19] the algorithm is used in the context of QoS for web services. Some intersections of these problems have been evaluated: MMKP are multiple-choice, multiple-dimensional knapsack problems, and heuristics to solve them are discussed in [1, 10].

The only reference we have found, and only as a definition, of a multi-dimensional, multiple-choice, multiple knapsacks problem, that henceforth we call MMMKP problem, is in [17] in the context of an admission control system for multimedia servers, but in that case the only arbitrated resource was Internet bandwidth.

The MMMKP model will be simplified in a MMKP if we have only one machine for each group, i.e. $g_i = 1 \; \forall i$. The mapping problem appears in a complex architecture following the resolution of a multi-tier model, that determines the number of required nodes for each tier, usually assuming that workload would not experience transient surges. To try to accommodate for peaks in workload intensity, we could over-provision the architecture: it wouldn't change the tier's size nor the architecture, but it will improve efficiency in resources usage. This is even more realistic if the provider and the owner of the multi-tier system belong to the same organization, because in this scenario the solution of the mapping problem is the minimum required level of service, every extra computing power put in use (and therefore not wasted) will be appreciated.

Another point is that being the number M fixed or not will lead us to different problems: if M is fixed, we have a knapsack problem, otherwise we have a multiple-dimensional, multiple-choice bin packing problem, a generalization of the bin packing problem that as the MMMKP is almost unknown in the scientific literature (a survey of bin packing problems is [11]). We choose to work on a generalization of the knapsack problem, but with an objective function that tries to minimize the number of physical hosts used.

Last, we assume that the hypervisor technology that we adopt to manage the virtual machines suffers of no or little interference, meaning that is capable of perform a robust and fair physical resources sharing. If this is not the case, the mapping problem could be more easily defined as a generalization of the Generalized Assignment Problem.

As the MMMKP is a generalization of the classical knapsack 0/1 problem, it's an NP-hard problem. Due to the lack of space, we omit the demonstration that each classical knapsack problem could be formulated as an MMMKP.

4. An heuristic for the MMMKP problem

The computational complexity of the MMMKP problem forces us to develop an heuristic to solve it, as the problem is practically intractable even for small datasets. We develop it in two phases. First, we search for a basic solution, then we try to improve it.

4.1. Basic solution

If a solution of an MMMKP instance exists, then the same MMMKP has a basic solution (not necessarily optimal) defined as the solution where, from each group, we have chosen the less demanding virtual machine, i.e. the lowest available SLA. Also, the MMMKP has some resemblances with the Bin Packing Problem (with two major differences: first, in a multi-dimensional Bin Packing Problem it's possible to rotate and shuffle dimensions, second on the Bin Packing Problem it's possible that the set of items to be packed are not known a priori) so we consider three classical bin packing heuristics for the basic problem: First Fit, Next Fit and Best Fit ([11]).

4.2. Improving a solution

We improve the solution found insofar in an iterative way. At the generic step, we have chosen a specific item from each group. Assume that for the group i we have item j chosen on machine m, i.e. $x_m^{ij} = 1$ and $x_m^{i(j+1)} = 0$. If $j = g_i$ we already have the most profitable item from group i so we move on another group to check for possible increases. Otherwise, we remove item j from group i from solution, releasing associated resources on knapsack m, and we see if and where we could put in solution item $j+1$ of the same group. This requires considering all available knapsacks, finding the most suitable one to contain item $j+1$. If a generic knapsack k as enough free resources for the item, we evaluate the goodness of the mapping by this formula:

$$Goodness(i,k) = \frac{\|R\|}{P^{i(j+1)} - P^{i(j)} - (1-u_k) * C} \quad (5)$$

In eq. 5:

- the numerator is the vector norm of the residual amount of resources available on knapsack k after we put item $i(j+i)$ in it;
- the denominator is the increase in profit we have ($P^{i(j+1)} - P^{i(j)}$) minus the possibility that we may end up using a knapsack k that was not yet used;
- the equation makes sense only if the denominator is bigger than zero, i.e. only if we have some profit gain;

In each phase of the improvement process, we calculate Goodness(i,k) for each acceptable value of i (groups with item more valuable) and k (knapsacks with available resources). Lowest values of $Goodness(i,k)$ are better, so we choose the minimum positive one, and we perform the necessary corrections on the solution we are working on (i.e., we set $x_m^{ij} = 0$ and $x_k^{i(j+1)} = 1$). We repeat this process as long as we have made improvements on the current solution. Each pass has a computational complexity of $\Theta(G * M)$.

4.3. Randomization of data

The proposed heuristic is strongly based on the order by which groups and knapsacks are defined. We cannot stipulate that it exists an order of these variable such that the proposed heuristic could always find the optimal solution, but we are confident that if we permute the groups and the machines before actually building up a solution we could increase the final profit. We observe that there is not a general criterion to discriminate between good permutations that lead us to find better solutions and bad permutations, and also these good ones are less than statistically rare, unless that P=NP.

5. Experimental model and results

There are no analytical model of large distributed multi-tier systems characterizing their resources demand, so in defining our datasets we have been forced to make somewhat arbitrary choices. We have considered two different multi tier systems, both three tier as they are pervasive, modeling each node of them by two resources, number of CPU cores and GiB of RAM. We have compared the solutions from the heuristic with the optimal solutions obtained via the GNU Linear Programming Toolkit solver. Then, we performed some sensitivity analysis over the C parameter, to see how changing it forces the system to use fewer physical hosts.

The optimal solution for the smallest system is found within two minutes (with a Xeon 1.86 GHz processor),

while after two hours the computation for the larger system was not yet concluded. As the formal solver determines first the solution of the linear relaxation problem and then uses a branch and bound technique to find the solution of the integer problem, progressively displaying results, we use these two values as the range where the optimal solution of the problem lies.

5.1. Models

The first three tier system is made up of 10 nodes (2+6+2), the first tier of it has three service levels and others have two. The second system is made up of 15 nodes (4+7+3) with the first and second tier with three service levels and the third tier with two. First system could use up to 6 physical hosts, the second has 8 physical hosts available. On the second system we have introduced some heterogeneity in the hosts: three of them have 16 GiB of RAM and the others 8 GiB, which is the value for all the hosts in the first model. Each host for each model has 8 CPU cores.

Table 5.1 reports different resources demands for each SLA of the nodes of the first model and the corresponding profits, and table 2 for the second model. Table 3 characterizes the available physical resources from hosts for both models.

Tier	Nodes	CPUs	RAM Size	Profit
Web	2	1/2/3	2/4/4	2/4/8
Application	6	2/2	2/4	2/6
Database	2	2/4	2/4	2/4

Table 1. First model increasing SLAs and profits.

Tier	Nodes	CPUs	RAM size	Profit
Web	2	1/2/4	2/4/6	2/4/8
Application	6	2/4/4	2/4/6	2/4/6
Database	2	2/2	4/6	2/6

Table 2. Second model increasing SLAs and profits.

Model	Hosts	CPUs	RAM
First	6	48	48 GiB
Second	8	64	88 GiB

Table 3. Physical hosts characterization for both models.

5.2. Results

Table 4 and 5 summarize the results for the first and second model. For both models, we see that the heuristic is capable to find an optimal solution or a very close one when the value of C is small, whilst as this parameter increases the relative performances of the heuristic decrease. On table 5 we don't have the profit but only a range, where the first number is the value found after two hours of computation and the highest is the solution of the linear relaxation, rounded up to the nearest integer. We note that these range increases in the same qualitative way that the difference between the best solution found by the heuristic and the lowest value of the range itself increases. These two trends would probably indicate that when C increase the optimization problem is even harder.

C	Profit (optimal)	Profit (heuristic)
0	60	60
1	55	53
2	50	46
3	45	39

Table 4. Comparisons of solutions for first model.

C	Range of profit	Profit (heuristic)
0	$87 \div 96$	84
1	$79 \div 95$	69
2	$71 \div 93$	54
3	$63 \div 89$	39

Table 5. Comparisons of solutions for second model.

For each model and each value of C, we ran 10,000 times each of the three basic strategies (first fit, next fit, best fit) to see how they differ. Each run is randomized, and we rely on the GNU GSL scientific library to provide an easy and statistically independent way to permute data. It appears that, unregarding the employed basic strategy, the heuristic is capable of reaching the same high profit for each problem instance, but the relative frequencies of these favorably outcomes differ, as table 6 shows for the second model. They all go down as the value of C increase, but the Best Fit has the smallest drop.

C	First Fit	Next Fit	Best Fit
0	9.3%	13.3%	8.9%
1	10.9%	15.0%	8.9%
2	5.0%	8.2%	8.2%
3	4.9%	8.0%	8.2%

Table 6. Relative frequencies of best solutions for second model.

6. Conclusions and future work

Allocation of resources for a virtualized autonomic environment is a complex problem, where we also have stringent time constraints. The MMMKP model is capable of capturing all the complexities of a multi-tier distributed system, allowing for different SLAs to maximize resource usage and coexistence of both quantitative and qualitative resources. The proposed heuristic works surprisingly well when the pressure to minimize the number of used physical hosts is not too high, otherwise it could obtain sub optimal results. If we analyze the heuristic from a solution-space point of view, we see that it basically moves from a feasible solution to a close one, obtained by swapping the value of two different decision variables, with only one of them set to one. A general extension of this approach would be to consider more "distant" solutions, and this suggests us that a genetic algorithm could be of a great use to tackle this problem, as it allows to explore solutions with more than a single swap of difference from the current solution.

References

[1] M. M. Akbar, M. S. Rahman, M. Kaykobad, E. G. Manning, and G. C. Shoja. Solving the multidimensional multiple-choice knapsack problem by constructing convex hulls. *Comput. Oper. Res.*, 33(5):1259–1273, 2006.

[2] J. Almeida, V. Almeida, D. Ardagna, C. Francalanci, and M. Trubian. Resource management in the autonomic service-oriented architecture. *Autonomic Computing, International Conference on*, 2006.

[3] M. N. Bennani and D. A. Menasce. Assessing the robustness of self-managing computer systems under highly variable workloads. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing*. IEEE Computer Society, 2004.

[4] M. N. Bennani and D. A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*. IEEE Computer Society, 2005.

[5] D. Bertsimas and R. Demir. An approximate dynamic programming approach to multidimensional knapsack problems. *Manage. Sci.*, 48(4):550–565, 2002.

[6] C. Chekuri and S. Khanna. A ptas for the multiple knapsack problem. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2000.

[7] P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4(1):63–86, 1998.

[8] C. E. Ferreira, A. Martin, and R. Weismantel. Solving multiple knapsack problems by cutting planes. *SIAM J. on Optimization*, 6(3):858–877, 1996.

[9] G. Franks, S. Majumdar, J. Neilsony, D. Petriu, J. Rolia, and M. Woodside. Performance analysis of distributed server systems. In *Proc. of The 6-th International Conference on Software Quality*, pages 15–26, 1996.

[10] M. Hifi, M. Michrafy, and A. Sbihi. A reactive local search-based algorithm for the multiple-choice multi-dimensional knapsack problem. *Comput. Optim. Appl.*, 33(2-3):271–285, 2006.

[11] E. G. C. Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. pages 46–93, 1997.

[12] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi. Dynamic placement for clustered web applications. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*. ACM, 2006.

[13] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[14] D. A. Menasce and M. N. Bennani. Autonomic virtualized environments. In *ICAS '06: Proceedings of the International Conference on Autonomic and Autonomous Systems*. IEEE Computer Society, 2006.

[15] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. ACM, 2007.

[16] D. Pisinger. A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research*, 83:394–410, 1995.

[17] S. Shelford, M. M. Akbar, E. G. Manning, and G. C. Shoia. Distributed optimal admission controllers for service level agreements in interconnected networks. In *Proceedings of the 21st IASTED International Conference on Applied Informatics*, 2003.

[18] X. Wang, D. Lan, G. Wang, X. Fang, M. Ye, Y. Chen, and Q. Wang. Appliance-based autonomic provisioning framework for virtualized outsourcing data center. In *ICAC '07: Proceedings of the Fourth International Conference on Autonomic Computing*. IEEE Computer Society, 2007.

[19] T. Yu and K. Lin. Service selection algorithms for web services with end-to-end qos constraints. In *CEC '04: Proceedings of the IEEE International Conference on E-Commerce Technology*. IEEE Computer Society, 2004.