БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ • BULGARIAN ACADEMY OF SCIENCES

ПРОБЛЕМИ НА ТЕХНИЧЕСКАТА КИБЕРНЕТИКА И РОБОТИКАТА, **60**
PROBLEMS OF ENGINEERING CYBERNETICS AND ROBOTICS, **60**

София • 2009 • Sofia

A Software System for Optimal Virtualization of a Server Farm[1]

Daniel Vatov

Institute of Information Technologies, 1113 Sofia
E-mail: daniel.vatov@gmail.com

1. Introduction

The term virtualization broadly describes the separation of a resource or request for a service from the underlying physical delivery of that service. With virtual memory, for example, computer software gains access to more memory than is physically installed, via the background swapping of data to disk storage. Similarly, virtualization techniques can be applied to other IT infrastructure layers – including networks, storage, laptop or server hardware, operating systems and applications. This blend of virtualization technologies – or virtual infrastructure – provides a layer of abstraction between computing, storage and networking hardware, and the applications running on it. The deployment of virtual infrastructure is non-disruptive, since the user experiences are largely unchanged. However, virtual infrastructure gives administrators the advantage of managing pooled resources across the enterprise.

A key benefit of virtualization is the ability to run multiple operating systems on a single physical system and share the underlying hardware resources – known as **partitioning**.

[1] This work was supported by the European Social Fund and Bulgarian Ministry of Education and Science under the Operation Programme "Human Resources Development", Grand BG051PO001/07/3.3-02/7.

Today, virtualization can apply to a range of system layers, including hardware-level virtualization, operating system-level virtualization, and high-level language virtual machines. Hardware-level virtualization was pioneered on IBM mainframes in the 1970s, and then more recently Unix/RISC system vendors began with hardware-based partitioning capabilities before moving on to software-based partitioning. For Unix/RISC and industry-standard ×86 systems, the two approaches typically used with software-based partitioning are hosted and hypervisor architectures. A **hosted** approach provides partitioning services on top of a standard operating system and supports the broadest range of hardware configurations. In contrast, a **hypervisor** architecture is the first layer of software installed on a clean x86-based system (hence it is often referred to as a "bare metal" approach). Since it has direct access to the hardware resources, a hypervisor is more efficient than hosted architectures, enabling greater scalability, robustness and performance.

Virtual infrastructure initiatives often spring from data center **server consolidation** projects, which focus on reducing existing infrastructure "box count", retiring older hardware or life-extending legacy applications. Server consolidation benefits result from a reduction in the overall number of systems and related recurring costs (power, cooling, rack space, etc.)

While server consolidation addresses the reduction of existing infrastructure, **server containment** takes a more strategic view, leading to a goal of infrastructure unification. Server containment uses an incremental approach to workload virtualization, whereby new projects are provisioned with virtual machines rather than physical servers, thus deferring hardware purchases.

Partitioning alone does not deliver server consolidation or containment, and in turn consolidation does not equate to full virtual infrastructure management. Beyond partitioning and basic component-level resource management, a core set of systems management capabilities are required to effectively implement realistic data center solutions. These management capabilities should include comprehensive system resource monitoring (of metrics such as CPU activity, disk access, memory utilization and network bandwidth), automated provisioning, high availability and workload migration support.

2. Problem description

While the large scale virtuaqliztion is gaining speed, in many organizations it is for first time when they decide to consolidate their server farms from physical deployment to virtual machines. The problems that they face are:

- Is it worth doing this in our case?
- How much will it cost to virtualize the department *X*? What about departments *Y* and *Z*?
- How much we will save in terms of maintenance cost and effort?

The overall efficiency of the solution and the confidence of the customer that was chosen the right strategy is a precondition for the successful spread of the virtualization in the data-centers. The optimization problem that should be solved can be defined as:

A set of physical machines with different OS is given and applications running on them, owned by different departments in the organization, with different importance for the successful daily operations and with different maintenance costs. How these physical machines can be consolidated and virtualized in optimal way, taking into account the organization's goals and specific constraints?

The strategy for virtualization that the company may choose can vary depending on its goals. Below we shall describe several strategies but the real life scenarios can be more numerous.

We define for each physical machine that should be virtualized a value *v*. We can have a scenario where *v* is:

$$v = \frac{\text{support effort}}{\text{usefulness}}.$$

The machines that are taking too much effort for support but are considered less valuable for achieving organization's goals receive higher *v*. In this scenario they will be virtualized first which will improve the overall value for the server farm

$$\sum_{i=1}^{M} (v_i),$$

where *M* is the number of machines in the farm.

If we define *v* as

$$v = \text{importance}$$

probably it is wise to consider not to have too much virtual machines with high importance on one hypervisor. Thus in case of failure of a hypervisor the impact will be, hopefully, not fatal. On the contrary if the organization decides to address this risk by installing redundant hypervisors it will be better if the most important virtual machines are grouped together on less hypervisors thus reducing the cost of the solution.

Another consolidation strategy may require to separate the physical machines in groups depending on some predefined criteria. Such a criteria can be departamental ownership, geographical location, machine's architecture, operating system, etc.

3. Theoretical background

To get proper understanding of the consolidation algorithm, we shall introduce an important class of combinatorial problems, known as the knapsack problems. These types of problems are used to solve many practical problems like planning, scheduling, resource allocation. More details about the Knapsack Problems can be found in [2].

The classical 0-1 Knapsack Problem can be defined as follows. There are *n* objects and a knapsack. Each object has an associated value and a resource requirement. For example these can be price and weight and the knapsack can carry not more than *R* kilograms. The goal is to pick a set of objects in such a way so that the overall value of all the items in the knapsack to be maximized without exceeding the capacity of the knapsack.

Mathematically the problem is stated as follows:

$$\text{maximize } V = \sum_{i=1}^{n} x_i v_i$$

such that

$$\sum_{i=1}^{n} x_i r_i \le R,$$

$$x_i \in \{0, 1\},\ i = 1, 2, \ldots, n.$$

In the problem definition x_i is a variable for object i. The problem is called 0-1 knapsack problem because the variable x_i can either take a value of 0 implying object i is not picked, or a value of 1 implying object i is picked. Any set of picked objects that satisfy the knapsack constraints is called *feasible solution* of the problem. The solution of 0-1 knapsack problem is the feasible solution which maximizes the sum of the values of picked objects. If we relax the integrality constraints the resulting problem is a linear program. This is the LP relaxation of the knapsack problem (KP) denoted LP (KP).

$$\text{maximize } V = \sum_{i=1}^{n} x_i v_i$$

such that

$$\sum_{i=1}^{n} x_i r_i \le R$$

$$0 \le x_i \le 1, \qquad i = 1, 2, \ldots, n.$$

Since the constraints of LP(KP) are more relaxed than in 0-1 KP

$$V_{\text{LP(KP)}} \ge V_{\text{KP}}$$

it can be used to obtain upper bound for KP solution (See [2]).

The classical 0-1 KP can be generalized for multiple resource constraints. Such a problem is called *multi constraint knapsack problem* or *Multi Dimension Knapsack Problem (MDKP)*. In MDKP each object requires m resources and has value v. Resource requirements are given by the resource vector $r_i = (r_{i1}, r_{i2}, \ldots, r_{im})$. The amounts of the resources available in the knapsack are given by $R = (R_1, R_2, \ldots, R_m)$.

The MDKP problem is stated as follows:

$$\text{maximize} \quad V = \sum_{i=1}^{n} x_i v_i$$

such that

$$\sum_{i=1}^{n} x_i r_{ik} \le R_k, \qquad k = 1, 2, \ldots, m;$$

$$x_i \in \{0, 1\}, \qquad i = 1, 2, \ldots, n.$$

Another generalization of the knapsack problem that will be used is *Multiple Choice Multi-Dimension Knapsack Problem (MMKP)*. The problem definition is the same as for MDKP but the objects are grouped in different stacks (groups). In general every group has different number of objects. A feasible pick for the knapsack can contain only one object from a group.

The MMKP problem is defined as follows. The i-th group has l_i objects. Object j of group i has value v_{ij}, and requires resource $r_{ij} = (r_{ij1}, r_{ij2}, ..., r_{ijm})$. The amounts of available resources are given by $R = (R_1, R_2, ..., R_m)$,

$$\text{maximize } V = \sum_{i=1}^{n} \sum_{j=1}^{l_i} x_{ij} v_{ij}$$

such that

$$\sum_{i=1}^{n} \sum_{j=1}^{l_i} x_{ij} r_{ijk} \le R_k, \qquad k = 1, 2, ..., m,$$

$$\sum_{j=1}^{l_i} x_{ij} = 1, \qquad i = 1, 2, ..., n,$$

$$x_{ij} \in \{0, 1\}, \qquad i = 1, 2, ..., n;$$
$$j = 1, 2, ..., l_i.$$

The special case of MMKP, where there is only one resource constraint (i.e. $m = 1$), is called a *Multiple-Choice Knapsack Problem (MCKP)*. Variants of knapsack problems comprise an important class of combinatorial optimization problems. The classical 0-1 knapsack problem is one of the most studied problems in operations research or combinatorial optimization. Please refer to [2] for good surveys of algorithms for the knapsack problems and some of their variants.

Since the 0-1 knapsack problems are NP-hard [2], the worst-case computation time of the optimal solutions grows exponentially with the size of the problem. For this reason, there are two types of solutions proposed for the KP and its variants: optimal solutions, and near-optimal solutions. The objective of the near-optimal solutions is to provide solutions which are close to optimal values, but require computation times which are much shorter than those of the optimal solutions. Most of the optimal algorithms for the variants of the KP uses a general search method, called the *branch and bound method*.

3.1. Branch and bound method for MMKP

The *branch and bound* method provides a popular approach for many variants of the KPs. This section describes shortly an extension of the branch and bound method for MMKP which was proposed by Khan [1].

At any solution state of the MMKP, a group is either free denoting no item is picked from this group, or it is *fixed* denoting an item is picked from this group. The fixed/free status of the groups are indicated using the group status vector g. If group i is free, g_i is 0; otherwise, the group is fixed and g_i is 1. The solution state of a node is represented by a solution vector $x = \{x_{ij}\}$ where $i = 1, ..., n$ and $j = 1, ..., l_i$. For a

fixed group i, if $x_{ij} = 1$, it implies that item j is picked, and otherwise $x_{ij} = 0$ implying that item j is not picked.

The branch and bound algorithm for the MMKP involves the iterative generation of a search-tree. The basic algorithm works as follows:

Step 1. Start with a solution state where all groups are free. Compute the upper bound, select the branching group, and initialize the search-tree with this node as the only live node[2].

Step 2. Find the *e*-node[3] *e* which is the live node with the largest value of upper bound.

Step 3. If node *e* does not have any free group (i.e. all the groups are fixed), then this node represents the optimal solution, and the algorithm terminates.

Step 4. If node *e* has at least one free group, then this node is expanded by fixing the branching group *b*. Fixing group *b* involves the following steps for each item *j* of this group:

a) form a new node *t* where the picked items are the picked items of node *e* and the item *j* of group *b*,
b) compute the upper bound at node *t*,
c) select the branching group if there exists any free group in *t*,
d) if node *t* is feasible, put node *t* as a live node into the search-tree.

Step 5. Go back to Step 2.

The main distinction of this algorithm from the well known 0-1 branch and bound method lies in dealing with groups of items. In [1] a pseudo-code of the branch and bound algorithm for the MMKP is given .

3.2. Heuristics HEU

Since MMKP is an NP-hard problem, the computation time for any optimal algorithm, such as Bblp, may grow exponentially with the size of the problem instance in the worst case. This may not be acceptable for time-critical applications such as admission control and dynamic resource allocation in a multimedia system. These applications are forced to accept a near-optimal solution if the computational time for optimal solution exceeds real-time bounds.

Here are the main concepts of the heuristic:

- It starts with a solution where from each group the item with the smallest value (v_{ij}) is picked, and iteratively improve the solution by gradually replacing items of smaller values with those of larger values as long as the solution remains feasible.
- It uses Toyoda's concept of aggregate resource where the required resource vector of an item is converted to a scalar index using penalty factors taken from the current resource usage vector [1]. Here the main idea is to penalize the use of resources depending on the current resource state. It applies a large penalty for a heavily used resource, and a small penalty for a lightly used resource.

[2] A node which has been generated and whose children have not yet been generated.
[3] Called also *branching* or *expanding node*.

- To find the next item to be picked, it chooses the one which maximizes the savings in aggregate resource. But if no such item is found, it chooses the one which maximizes the value gain per unit of aggregate resource.

A detailed algorithm and pseudo-code procedure for HEU can be found in [1].

4. Software system description

In this description we will skip the trivial functionalities of the system and will just enumerate them. The focus will be put on the functionality concerning the consolidation and the application of the MMKP algorithms in this process. The system in which will be integrated the MMKP algorithms is VMWare's Capacity Planner[4].

At the core of VMware Capacity Planner[5] is the Information Warehouse, which contains a growing set of industry reference data. This information can be leveraged for comparative analysis and benchmarking to help guide system consolidation and capacity optimization decisions for the enterprise. VMware Capacity Planner is used as a business analysis, planning and decision support tool to direct the key phases within a variety of infrastructure assessment projects as described below:

- Assess the current workload capacity of an IT infrastructure through discovery and inventory of IT assets. Measure system workloads and capacity utilization across various elements of the IT infrastructure - including by function, location and environment.
- Plan for capacity optimization through utilization analysis, benchmarking, trending and identification of capacity optimization alternatives. Identify resources and establish plan for virtualization, hardware purchase or resource redeployment.
- Decide on the optimal solution by evaluating various alternatives through scenario modeling and “what-if” analysis. Determine which alternative best meets the pre-defined criteria.
- Monitor resource utilization through anomaly detection and alerts based on benchmarked thresholds. Help generate recommendations to ensure ongoing capacity optimization.

The sub-system that enables access to the gathered data and is used for managing the consolidation and assessment process is web based J2EE application. The assessment is configured in a five-step wizard where all the necessary information is gathered. The parameters that are configured are:

- General information as name, ownership, access.
- Input systems that should be considered for consolidation.
- Consolidation settings
  - Is it allowed systems to consolidate onto the same hardware when the systems have different departments, environments, functions, locations or operating systems?
  - Should the consolidated systems be virtualized?

[4] The opinions and ideas expressed here do not represent the official opinion of VMWare Inc.
[5] http://www.vmware.com/products/capacity_planner

- Is it allowed to consolidate systems with different processor architectures onto the same hardware?
- Should the systems be redeployed only on new hardware or to use the old hardware when possible.
- Level of details – summary or detailed information for each virtual machine

- The type of the hypervisor hardware and the specific parameters – memory, cpu, etc.
- The boundaries for the consolidation – consolidate over location, environment, function, etc.
- Minimal limits under which machines will not take part in the assessment process.

For the purpose of integrating MMKP algorithms in the consolidation engine this wizard will be extended to allow supplying also:

- Value for each machine. The semantics of the values will depend on the consolidation strategy chosen. Since number of servers may be quite big the value will be assigned through regular expression besides enumerating each server individually. For example – *v = 50 if (software inventory* ***contains*** *postgres* ***and*** *department* ***equals*** *"Accounting").*
- Threshold for the total value of the virtual machines running on the hypervisor servers. This threshold will be used for strategies that do not consolidate all the physical machines.
- Currently the attributes that are used for classification are: location, function, environment, department and OS. The classification is done to separate the input servers producing in this way several consolidation task - one per group. To allow grouping of the physical machines in the sense of MMKP problem the current groups will be extended with a free slot for a group with user chosen semantics – e.g. importance of the server.
- The current functionality providing alternative scenario generation will be extended to include the new parameters.

To illustrate usage lets consider the case where a virtualization have to take place in a company whose IT infrastructure consists of 200 servers owned by three departments. The servers are situated in two geographical locations. The software that runs on the servers vary from databases to intranet sites. The utilization of the servers also vary in terms of average and peak values.

Part of the assessment settings for this company are:

- Do not consolidate servers in different geographical locations;
- Group the machines by their departamental ownership.

The consolidation engine will solve two MMKP problems – one per a geographical location. For each MMKP problem the physical servers will be separated in groups according to their departamental ownership. Lets assume that the consolidation strategy selected by the company assigns higher values to the machines with lower importance for the organization but with higher maintenance costs. In this way the company will put the consolidation threshold to such a value that will ensure the expenses done for the new equipment and maintenance costs spread over one year period will be less than the current sell price of the old

hardware (or recycling costs) and its maintenance costs. Grouping the machines by departments ensures that from the savings will benefit all departments.

5. Conclusion

This article presented the straightforward application of MMKP problem to model the consolidation process. While using real-life scenarios the model is not as rich as it can be. Future extension to the model may include the information for the performance load during the different hours of the day. This will provide different optimal consolidation solutions for each time interval. There exist software products allowing dynamically, without stopping the virtual machine, to move it from one hypervisor to another. An algorithm trying to find the optimal or near to the optimal solution to the consolidation problem may benefit from this. On the other hand the model is quite simple and can be easily integrated with other legacy software systems. For example it can benefit from an ERP system to determine in automatic way the values for *v* for each machine in the server farm.

References

1. Md. Shahadatullah Khan, C. Flmd Shahadatullah Khan. Supervisors Dr. Kin F. Li, Dr. Eric, and G. Manning. Quality Adaptation in a Multisession Multimedia System: Model, Algorithms and Architecture. Technical Report, 1998.
2. Martello, S., P. Toth. Knapsack Problems: Algorithms and Computer Implementations. Technical Report, 1990.

Софтверная система для оптимальной виртуализации серверной фермы

Даниел Ватов

Институт информационных технологий, 1113 София
E-mail: daniel.vatov@gmail.com

(Резюме)

В работе представлено применение мультидименсионной задачи рюкзака с множеством выборов. Когда используются сценарии реальной жизни, модель не так богата. Расширение модели может включить информацию для поведения нагрузки в разных часах суток. Это дает оптимальные решения в каждом интервале времени. Существуют софтверные продукты, которые позволяют динамично, не останавливая виртуальную машину, переходить с одного хипервайзера к другому. Алгоритм, который ищет оптимального или вблизи оптимального, решения задачи, может воспользоваться этим. С другой стороны, модель очень простая и может быть легко интегрирована с другими софтверными системами.