Virtual Machine Placement in Computing Clouds

Anjana Shankar (09305920)
Guided By: Prof.Umesh Bellur

April 10, 2010

Contents

1 Introduction to Virtual Machine Placement 3
1.1 Classification of Placement Algorithms 3
1.1.1 Power Based Approach 4
1.1.2 Application QOS Based Approach 4
1.1.3 Dynamic/Online and Static/Offline Approaches to Placement 5
1.2 Formalising Problem statement 5
1.3 Placement Approaches 5
1.3.1 Constraint Programming 5
1.3.2 Bin Packing 6
1.3.3 Stochastic Integer Programming 6
1.3.4 Genetic Algorithm 6

2 Constraint Programming based approach to Virtual Machine Placement 8
2.1 VM Placement problem as a Constraint Satisfaction Problem . . 8
2.1.1 Formalising the problem 9
2.1.2 Local Decision Module-Decisions made at each module locally 9
2.1.3 Global Decision Module - Decisions made at the global level 9
2.2 Performance Evaluation 11
2.3 Conclusion 11

3 Stochastic Integer Programming for Virtual Machine Placement 12
3.1 VM Placement problem in terms of Stochastic Integer Programming 13
3.2 Performance Evaluation 14
3.3 Conclusion 15

4 Bin Packing approach to Virtual Machine Placement 16
4.1 VM Placement problem as a Bin Packing Problem 16
4.1.1 Formalising the problem 16
4.2 Performance Evaluation 17
4.3 Conclusion 17

5 Genetic Algortihm for Virtual Machine Placement 18
5.1 VM Placement problem as a Grouping Genetic algorithm problem 19
5.1.1 Formalising the problem 19

5.2 Performance Evaluation 20
5.3 Conclusion 20

6 Virtual Machine Placement : Which algorithm works best? 21
6.1 Comparison of different techniques 21

Chapter 1

Introduction to Virtual Machine Placement

Virtual machine placement is the process of mapping virtual machines to physical machines. In other words, virtual machine placement is the process of selecting the most suitable host for the virtual machine. The process involves categorising the virtual machines hardware and resources requirements and the anticipated usage of resources and the placement goal. The placement goal can either be maximizing the usage of available resources or it can be saving of power by being able to shut down some servers. The autonomic virtual machine placement algorithms are designed keeping in mind the above goals.

~~Through this seminar, I am trying to understand~~ some of the algorithms that have been designed for tackling this placement problem.

1.1 Classification of Placement Algorithms

~~The~~ placement algorithms can be broadly classified into two categories on the basis of their placement goal.

- Power Based approach
- Application QOS based approach

This can be further classified as shown in figure below.

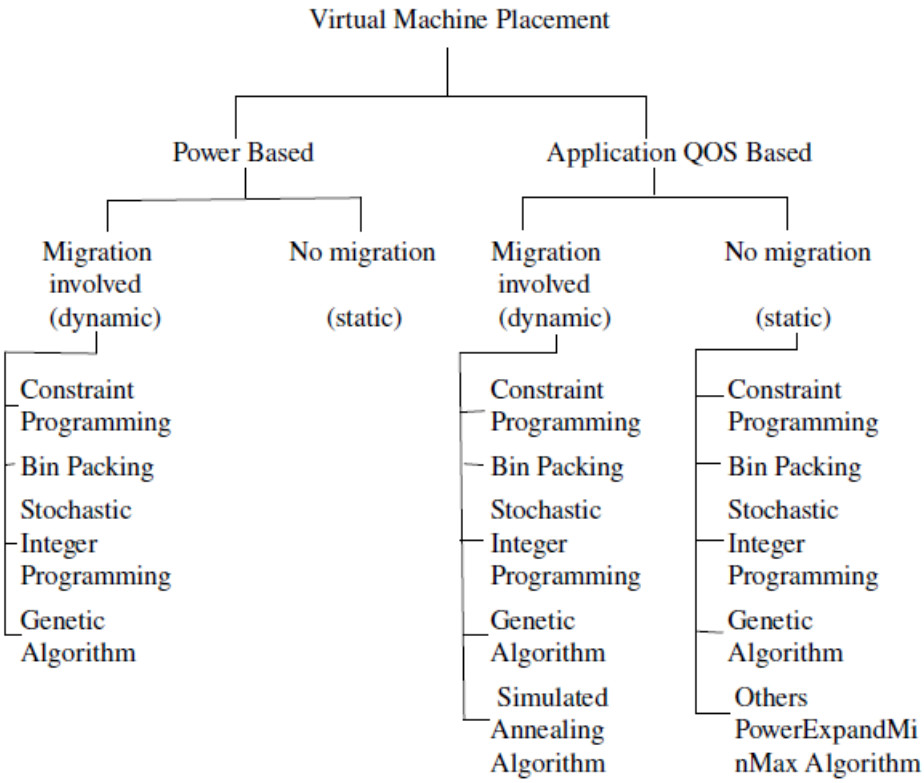

Figure 1.1: Classification of VM Placement Algorithms

1.1.1 Power Based Approach

The necessity of power management has become increasingly evident in computing environments. The need for power management is driven by two factors:

1. The increasing demands on power by both computing and cooling resources in a data center.

2. The rising cost of power.

The main aim of these approaches is to map virtual machines to physical machines in such a way, so that the servers can be utilized to their maximum efficiency, and the other servers can be either hibernated or shut down depending on load conditions.

1.1.2 Application QOS Based Approach

These algorithms manage the mapping of virtual machines onto physical hosts with the aim of maximizing the quality of service (QOS) delivered. By continuously monitoring virtual machine activity and employing advanced policies for dynamic workload placement, such algorithms can lead to better utilization of

resources and less frequent overload situations eventually leading to savings in cost.

1.1.3 Dynamic/Online and Static/Offline Approaches to Placement

The placement computations can be either dynamic or static. Static Algorithms typically use the data that is previously collected as an input. After initial calculations, the mapping may not be recomputed for long periods of time, such as several months. The computations are mostly done offline.

In contrast, dynamic allocation is implemented on shorter timescales, preferably shorter than periods of significant variability of the resource demand. The Placement algorithms run in the background of the application processes collecting data. [6]

1.2 Formalising Problem statement

A Virtual Machine placement problem is typically a combinatorial search problem. Given a set of N virtual machines and a set of M physical machines and the physical machines already host N_1 virtual machines. The virtual machine placement algorithms give a mapping of these $N + N_1$ machines on the M physical machines under a set of constraints. This may or may not involve migration of already placed N_1 virtual machines.

1.3 Placement Approaches

Some of the approches for virtual machine placement are explained in the subsequent paragraphs. The placement problem is a non deterministic problem. Following are some of the algorithms that have been used to solve the virtual machine placement problem.

1.3.1 Constraint Programming

Constraint programming is a programming paradigm. The solution should satisfy the constraints on relations between variables. It is useful for combinatorial search problems . [2]

It consists of three basic techniques:-

1. Declare the domains of variables.
2. Decide the constraints on the declared variables.
3. Search for the domain.

Example : N-Queens Problem
Constraint: No two queens on the same row, column or diagonal and $N > 2$.
Model:

1. Assume each queen is in a different column.

2. Assign a variable R_i (with domain 1...N) to the queen in the ith column indicating the position of the queen in the row.

The VM placement problem can be designed as a *constraint programming* problem as follows.

Constraint: Number of available physical machines and maximize the global utility function.

Model: Find VM allocation vectors [10]

The use of constraint programming allows mapping of virtual to physical machines that are better than those found by heuristics based on local optimizations and that are frequently globally optimal in the number of physical machines used.[7]

1.3.2 Bin Packing

The bin packing problem is a combinatorial NP-hard problem. In it, objects of different volumes must be packed into a finite number of bins of capacity V in a way that minimizes the number of bins used. Many variations of this problem are present, such as 2D packing, linear packing, packing by weight, packing by cost, and so on. The applications of these problems include, filling up containers, loading trucks with weight capacity, and creating file backup in removable media. Most of the efficient bin packing algorithms use heuristics to accomplish results. This provides a solution, which, though very good in most cases, may not be the optimal solution. For example, the first fit algorithm provides a fast but often nonoptimal solution, involving placing each item into the first bin in which it will fit. It requires (n log n) time, where n is the number of elements to be packed.[1]

The VM placement problem can be designed as a *bin packing* problem as follows. The Physical machines can be considered as bins and the VM's to be placed can be considered as objects to be filled in the bin.[6]

1.3.3 Stochastic Integer Programming

Stochastic programming is used for modelling optimization problems that involve uncertainty. Most of the real world problems include some unknown parameters. This can not be captured by deterministic integer programming. Stochastic programming models take advantage of the fact that probability distributions governing the data are known or can be estimated. The goal is to find some policy that is feasible for all (or almost all) the possible data instances and maximizes the expectation of some function of the decisions and the random variables.[4]

The VM placement problem can be designed as a *stochastic integer programming* problem as follows. The demands and prices are known (or can be estimated) and the objective is to minimize the budget of the user.

1.3.4 Genetic Algorithm

A genetic algorithm (GA) is a search technique used to find exact or approximate solutions to optimization and search problems. Genetic algorithms are

categorized as global search heuristics. They are inspired by evolutionary biology such as inheritance, mutation, selection, and crossover. A typical genetic algorithm requires:

- a genetic representation of the solution domain,
- a fitness function to evaluate the solution domain

The fitness function is defined over the genetic representation and measures the quality of the represented solution. The fitness function is always problem dependent. [3]

The VM placement problem can be designed as a *genetic programming* problem as follows. The solution domain can be represented as the physical machines with a resource provisioning capacity. The fitness function can be defined over the number of bins in the solution. The aim would be to deliver a solution that is nearly optimal in terms of the number of bins used and the efficiency of packing of the bins. [8]

The motivation behind all the algorithms that are designed for placing virtual machine on available physical machines is to reduce the costs involved. The costs can be reduced by optimally using the resources available and by shutting down the servers that have no load. The algorithm to be used varies as per the demands of the organisation.

Chapter 2

Constraint Programming based approach to Virtual Machine Placement

One of the aims of cloud providers is to automate management of virtual machines taking the quality of service requirements of applications into consideration. This problem can be formulated as a constraint proagramming problem. The goal of the constraint programming is to maximize a global utility function. This global utility function is chosen so as to take SLA fulfillment and operating costs into consideration. The utility functions mentioned in the paper [10] maps the current state of each application (workload, resource capacity, SLA) to a scalar value. This scalar value tries to quantify the applications' satisfaction with respect to the goals that are set by the automatic manager.[1]

The virtual machine placement using a constraint based approach can be thought of as a two-stage process.

Stage 1 : Local Decisions - It is associated with each application environment.

Stage 2 : Global Decisions - It takes as input the local decision from all applications and then tries to maximize the global utility function.

2.1 VM Placement problem as a Constraint Satisfaction Problem

Any application hosted in the cloud is referred as application environment(AE). We assume that one VM is associated with only one AE. A Local decision module is associated with each AE which computes an utility function. Given the current workload, this utility function gives us a measure of the application satisfaction with a specific resource allocation. This output is given to a global

[1]Reference for chapter : Autonomic virtual resource management for service hosting platforms, 2009

decision module which actually maps the VMs to PMs taking into account the CPU load of virtual and physical machines.

2.1.1 Formalising the problem

- $A = (a_1, a_2, \ldots, a_i, \ldots, a_m)$ denotes set of AEs
- $P = (p_1, p_2, \ldots, p_j, \ldots, p_q)$ denotes set of PMs
- There are c classes of VM available among the the set $S = (s_1, s_2, \ldots, s_k, \ldots, s_c)$ where $s_k = (s_k^{cpu}, s_k^{ram})$ specifies the cpu and memory capacity of the VM.

[10]

2.1.2 Local Decision Module-Decisions made at each module locally

These decisions are associated with a specific application environment. It takes into account a fixed service-level utility function mapping the service level to a utility value and a dynamic resource-level utility function mapping a resource capacity to a utility value. The result of resource-level utility function mapping is communicated to Global Decision Module in every iteration.
The resource-level utility function u_i for application a_i is defined as $u_i = f_i(N_i)$. N_i is the VM allocation vector of application a_i:

$$N_i = (n_{i1}, n_{i2}, \ldots, n_{ik}, \ldots, n_{im})$$

where n_{ik} is the number of VMs of class s_k attributed to application a_i.
Each application associates an upper bound on the number of VM of each class ($N_i^{max} = (n_{i1}^{max}, n_{i2}^{max}, \ldots, n_{ik}^{max}, \ldots, n_{im}^{max})$) and on the total number of VMs (T_i^{max}) that it is willing to accept. These application constraints are expressed as follows:
Constraints :

1. $n_{ik} \leq n_{ik}^{max}$, $1 \leq i \leq m$ and $1 \leq k \leq c$
2. $\sum_{k=1}^{c} n_{ik} \leq T_i^{max}$, $1 \leq i \leq m$

2.1.3 Global Decision Module - Decisions made at the global level

The global decision module works in two phases.

- VM Provisioning
- VM placement

Both these phases are expressed as Constraint Solving Problems.

Phase 1: VM Provisioning The aim of this phase is to find VM allocation vectors N_i for each application a_i, and at the same time, maximizing a global utility value U_{global}. The VMs allocated to all applications are bound by the constraints on the total capacity of physical servers.

$$\sum_{i=1}^{m}\sum_{k=1}^{c} n_{ik}.s_k^{cpu} \leq \sum_{j=1}^{q} C_j^{cpu}$$
$$\sum_{i=1}^{m}\sum_{k=1}^{c} n_{ik}.s_k^{ram} \leq \sum_{j=1}^{q} C_j^{ram}$$

where C_j^{cpu} and C_j^{ram} are CPU and RAM capacity of PM p_j

The global utility function is expressed as :-

$$U_{global} = maximize\sum_{i=1}^{m}(\alpha_i \times u_i - \in .cost(N_i))$$

where $0 < \alpha_i < 1$, and $\sum_{i=1}^{m}\alpha_i = 1$
$\in$ is a coefficient that determines the tradeoff between fulfillment of performance goal as opposed to cost of the required resources.
$cost(N_i)$ is a function of VM allocation vectors.

This phase outputs N_i that is a vector giving the allocation of VMs to applications.

Phase 2: VM Packing The input to this is the VM allocation vectors N_i from the VM provisioning phase. In this phase, GDM creates a single vector $\mathbf{V}$ from all N_i's. This vector lists all VMs that are running at the current time

$$\mathbf{V} = (vm_1, vm_2, \ldots, vm_l, \ldots, vm_v)$$

For each PM $p_j \in P$, the bit vector $H_j = (h_{j1}, h_{j2}, \ldots, h_{jl}, \ldots, h_{jv})$ denotes set of VMs assigned to p_j. Let $\mathbf{R} = (r_1, r_2, \ldots, r_l, \ldots, r_v)$ denote the resource capacity of all VMs. We can express these constraints as :-

$$\sum_{l=1}^{v} r_l^{cpu}.h_{jl} \leq C_j^{cpu} \quad 1 \leq j \leq q$$
$$\sum_{l=1}^{v} r_l^{ram}.h_{jl} \leq C_j^{ram} \quad 1 \leq j \leq q$$

The goal of this phase is to minimize the active number of PMs X. The function is as follows:-

$$X = \sum_{j=1}^{q} u_j \ ,$$

where $u_j = 1$ if $\exists vm_l \in \text{VM}|h_{jl} = 1$, else its 0

The GDM runs periodically, computes the difference between VM placement produced at the end of current round with previous round, this would tell the placement manager which VMs need to be migrated.

2.2 Performance Evaluation

The decisions are based on the α_i (Application's weight) and ϵ (Tradeoff factor) between fulfillment of performance goal and cost of resources. The constraints are :

- T_{max}, that is the total number of VMs and maximum number of VMs of each class that the application is willing to accept.
- The cost function used is Cost(CPU) = CPU_{demand}/CPU_{total}

With increase in ϵ, as the resource demands increases, the global utility value decreases. The operating cost of the application is boosted by this factor. Thus an application having a higher weight might end up not getting the resources, because of the higher tradeoff factor. The application weight denotes the resource allocation priority. Thus an application having a higher weight will get priority over an application having a lower weight. These weights should be critically chosen. Thus, in a time interval where demands for two applications are high, the resources will be provisioned to applications having a higher weight. The other application might have to face SLA violations. [10]

2.3 Conclusion

We need to find the solution to the problem in a time interval which is less than the time it takes for the constraint parameters to change. Else, the solution we get would be an expired solution. One way is to set a time limit on the time spent for searching a solution. Thus, the constraint solver need not explore the whole solution space.

We can define additional constraints that would take into account VM interference and produce a non-interfering VM to PM mapping. There may or may not be a provision to make changes to the global utility function at run time.

The choice of an appropriate application weight is very important. It can be done by both system administrator or autonomic virtual manager. If its set by a system administrator, it can be modified from time to time. However if it is set by autonomic virtual manager, the manager would have the additional task of checking that an application does not starve for resources.

Dependency of global utility function on a given architecture limits its applicability to different scenarios. But simultaneously, it is difficult to have a meta global utility function. A meta global utility function would have a high number of constraints thereby increasing computation and solution space, thus getting an optimal solution would be more difficult.

Chapter 3

Stochastic Integer Programming for Virtual Machine Placement

Stochastic Integer programming is useful in cases where actual demands are not known but the distribution of demands is known or can be estimated. Most of the cloud providers offer two payment plans to users for resource provisioning. These are:

1. Reservation plan
2. On-demand plan

With the help of SIP (Stochastic Integer Programming), the future demands which are not certain can be taken into consideration, and along with this the difference in the prices of reservation and on-demand plans can also be considered. Taking into account all of this, a mapping of VMs to PMs is generated so as to minimize the cost spent in each plan for hosting virtual machines in a multiple cloud provider environment minimizing the problems faced by under-provisioning or overprovisioning.
The *tradeoff factor* is the On-demand cost versus oversubscribing cost. [1]

An algorithm is described in [9] which uses Stochastic Integer programming. It allocates resources in three phases.

Reservation : Cloud broker provisions resources without knowing the demand of users.

Utilization : The reserved resources are used.

On-demand : In case the demand exceeds the reserved resources, the additional resources can be requested in an on-demand pay plan.

Each phase has a corresponding cost associated with it.

[1] Reference for chapter : Optimal Virtual Machine Placement for Multiple Cloud Providers, 2009

3.1 VM Placement problem in terms of Stochastic Integer Programming

In the cloud environment, we have a number of cloud providers that supply a pool of resources to the user. The virtual machines can be divided into a set of classes with each class representing a different application type. The objective is to minimize all the costs, at the same time meeting the demands of users. The demands and prices are not fixed. However, we can estimate the demands and prices based on the distribution curve. Thus, SIP would be useful in finding a solution to the problem of VM placement.

The problem statement can be defined formally as:-

- $\nu = \{V_1, V_2, \ldots, V_{last}\}$ denotes the set of VM classes. A VM class represents an application type.

- $\rho = \{P_1, P_2, \ldots, P_{last}\}$ denotes set of cloud providers. Each cloud provider supplies a pool of resources to user.
  The supercripts (h) , (s), (n) and (e) correspond to the four resources provided by cloud providers, namely computing power, storage, network bandwidth and electric power respectively.

- $t_j^{(h)}, t_j^{(s)}, t_j^{(n)}$ denotes to the maximum capacity of corresponding resource which cloud provider P_j can supply to user.

- $r_i^{(h)}, r_i^{(s)}, r_i^{(n)}$ denotes amount of corresponding resource required by a single VM under class V_i.

- $c_j^{(h)}, c_j^{(n)}, c_j^{(s)}$ denotes the prices of corresponding resources in reservation phase for cloud provider P_j

- $\bar{c}_j^{(h)}$, $\bar{c}_j^{(n)}$, $\bar{c}_j^{(s)}$ denotes the prices of corresponding resources in utilization phase for cloud provider P_j

- The cost of resources in on-demand phase can be random. $\tilde{c}_j^{(h)}$, $\tilde{c}_j^{(n)}$, $\tilde{c}_j^{(s)}$ denotes the prices of corresponding resources in on-demand phase for cloud provider P_j

- Uncertainty of Demands and prices.
  - D_i=$\{d_{i1}, d_{i2}, \ldots, d_{i\nu}\}$ denotes set of maximum number of required VMs of class V_i. The total number of required VMs D will be cartesian product of all D_i over all i.
  - $\tilde{C}_j^{(h)}$, $\tilde{C}_j^{(n)}$, $\tilde{C}_j^{(s)}$ denotes set of possible prices of offered resources by provider P_j in on-demand phase.

- $v_i(d)$ denotes the number of required VMs in class V_i if demand d is realized

The algorithm in [9] consists of two stages.

Stage 1 This defines the number of VMs to be provisioned in reservation phase.

Stage 2 This defines the number of VMs allocated in utilization and on-demand phase.

The SIP formulation is expressed as

$$\sum_{V_i \in \nu} \sum_{P_j \in P} c_{ij} X_{ij}^{(r)} + \xi_\Omega[\wp(X_{ij}^{(r)}, \omega)]$$

- $X_{ij}^{(r)}$ denotes number of VMs provisioned in first stage.
- $\xi_\Omega[\wp(X_{ij}^{(r)}, \omega)]$ denotes cost in second stage.
  Here $\omega \in \Omega = D \times \prod_{P_j \in P}$ denotes realizations (set of demands and prices)

The *objective* is to minimize this function.
Taking the probabilities of demands and costs being realized, we can expand the above equation as follows.

$$\sum_{V_i \in \nu} \sum_{P_j \in P} c_{ij} X_{ij}^{(r)} + \sum_{V_i \in \nu} \sum_{P_j \in P} \sum_{m \in \bar{C}_j} \sum_{d \in D} p_j(m) p(d) (\bar{c} X_{ij}^{(u)}(m,d) + \tilde{c}(m) X_{ij}^{(u)}(m,d) + \tilde{c}(m) X_{ij}^{(o)}(m,d))$$

Subject to constraints

$$X_{ij}^{(u)}(m,d) \leq X_{ij}^{(r)}, V_i \in \nu, P_j \in P, m \in \bar{C}_j, d \in D$$
$$\sum_{P_j \in P} (X_{ij}^{(u)}(m,d) + X_{ij}(o)(m,d) \geq v_i(d), V_i \in \nu, m \in \bar{C}_j, d \in D$$
$$\sum_{V_i \in \nu} r_i^{(h)} (X_{ij}^{(u)}(m,d) + X_{ij}(o)(m,d) \leq t_j^{(h)}, P_j \in P, m \in \bar{C}_j, d \in D$$
$$\sum_{V_i \in \nu} r_i^{(s)} (X_{ij}^{(u)}(m,d) + X_{ij}(o)(m,d) \leq t_j^{(s)}, P_j \in P, m \in \bar{C}_j, d \in D$$
$$\sum_{V_i \in \nu} r_i^{(n)} (X_{ij}^{(u)}(m,d) + X_{ij}(o)(m,d) \leq t_j^{(n)}, P_j \in P, m \in \bar{C}_j, d \in D$$
$$X_{ij}^{(r)}, X_{ij}^{(u)}(m,d), X_{ij}^{(o)}(m,d) \in 0, 1, \ldots, V_i \in \nu, P_j \in P, m \in \bar{C}_j, d \in D$$

3.2 Performance Evaluation

Assume a setup where, the number of required VMs is same for all VM classes and all cloud providers offer same maximum capacity of storage and network band-width, the demand is varying and the price of resources in on-demand plan is varying. In such a setup, the varying parameters can be formulated as a stochastic integer problem. The solution would provide an optimum computation for number of VMs to be allocated in reservation and on-demand phase. The results would vary as the distribution of demand varies.

As the number of VMs in reservation phase increases, the cost in the first stage increases. The second stage cost decreases as the number of reserved VMs increases, because the number of VMs needed in the on-demand phase decreases. With increase in price of on-demand phase, the number of VMs in reservation phase increases. With the variance being higher, the number of reserved VMs increases. Thus the number of reserved VMs is highest when demand is varied as per test data and least when demand is varied with a normal distribution. [9]

3.3 Conclusion

Stochastic Integer Programming is helpful in estimating the variation in demand and costs, thereby, frequent recomputations are not needed. However, this means if there is error in the estimation, users might end up paying more.

The algorithm in [9] uses SIP to compute the number of VMs to be allocated in each phase, however it does not give a way to map these Virtual machines to available Physical machines. We will have to use some other approach, something like bin packing to do the actual mapping.

Chapter 4

Bin Packing approach to Virtual Machine Placement

The bin packing approach can be used to find the actual mapping of virtual machines to available physical machines. It is possible to minimize the cost of running data center by tightly packing the VMs required to be running at a time onto the least number of PMs possible. However, care should be taken so as to avoid overloading which results in SLA violations. [1]

4.1 VM Placement problem as a Bin Packing Problem

The Physical machines can be considered as bins and the VM's to be placed can be considered as objects to be filled in the bin. A 3-stage algorithm is presented in [6].

- Formulate a pattern of past demands.
- Forecast the future demand based on the pattern of past demands
- Map or Remap VMs to PMs, this is known as Measure-Forecast-Remap(MFR)

The algorithm aims at minimizing the number of hosts required for VMs, if the probability of overloading the server in the interval is fixed. The last stage of the algorithm uses bin packing.

The demand in the next interval is predicted on the basis of demand in the prior interval.The distribution of error is also taken into consideration. Based on these a heuristic is developed. Based on the solution to this heuristic, the VMs are packed onto the PMs. The paper [6] uses first fit approximation.

4.1.1 Formalising the problem

- Given N VMs and M PMs each having a capacity C^m.

[1] Reference for chapter : Dynamic Placement of Virtual Machines for Managing SLA Violations,2007

- Remapping interval is denoted by R.
- The resource demand of nth VM at start of interval i is denoted by U_i^n.
- The distribution of demand corresponding to time series U_i^n is $u^n(x)$.
- $f_{i,k}^n$ denotes the forecast demand k units of time ahead of current time i for VM n.
- The capacity needed to guarantee an error rate less than p is denoted by $c_p(\mu, \sigma^2)$ where μ and σ^2 is mean and variance of prediction error distribution.

Based on the forecast of demand of each VM, the VMs are sorted in descending order. Each VM is then taken off the list and an attempt is made to place it on the first PM where it fits, that is, after its placement, the sum of resource demands of all VMs on that PM does not exceed the total capacity of the PM. The first-fit algorithm minimizes the number of active PMs.

4.2 Performance Evaluation

The aspect of interest for us is the reduction in number of used PMs versus the SLA violations. In comparison to the number of PMs used for static allocation, the number of PMs used in dynamic allocation is reduced to half. For a given SLA violation rate, the average number of PMs used by the above algorithm is less than the static algorithms.

4.3 Conclusion

When the first-fit approximation is used, for finding a VM to PM map, the migration costs should also be taken into account. Since, the algorithm uses first-fit approximation, it might fail to deliver the optimal solution as all possible solutions in the solution space would not be checked.

The placement algorithm is dependent on the efficiency of the forecasting technique. If the error in the forecast is huge, the number of SLA violations can go beyond the accepted value.

This method does not take constraints other than capacity of resources into account. Thus it may end up putting two interfering VMs on one PM.

Chapter 5

Genetic Algortihm for Virtual Machine Placement

Genetic algorithm is a heuristic based search technique. It is particularly useful in problems where objective functions dynamically change.[1]

Typically a genetic algorithm starts with a population of solution, applies genetic operators on this which finally results in an optimal solution. [8]

A variation to genetic algorithm known as Grouping Genetic Algorithm can also be applied to the VM Placement problem. These algorithms can take into account additional constraints while optimizing the cost function. This is particularly useful in cases where we need to operate on groups. [5]

The genetic algorithm mainly involves

- Chromosome modeling - This is an encoding schema used to encode details of the problem that is to be passed from one generation to the other.
- Population Initialization - The feasibility of the solution is dependent on the feasibility of the initial solution.
- Crossover - It is a genetic operator used to vary the programming of a chromosome from one generation to next.
- Mutation - Mutation is a genetic operator that alters one ore more gene values in a chromosome from its initial state. This helps to prevent the population from stagnating at any local optima.
- Generation Alternation - This is where we select one of the solutions from the next generation of solution.

[8]

[1]Reference for chapter : Grouping genetic algorithm for solving the server consolidation problem with conflicts and Toward Virtual Machine Packing Optimization Based on Genetic Algorithm, 2009

5.1 VM Placement problem as a Grouping Genetic algorithm problem

The Grouping Genetic Algorithm can be thought of as a bin packing problem where the aim is to not only find a solution with highest packing efficiency but also to satisfy the constraints. The algorithm in [5] gives a mechanism to take into account VM interference.

5.1.1 Formalising the problem

- Y_i is a binary variable, equals 1 if server i is used, 0 otherwise.
- X_{ij} is a binary variable, equals 1 if server j is consolidated into target server i, 0 otherwise.
- $attr_{jk}$ denotes usage of the kth attribute for the jth server.
- $Attr_{ik}$ denotes usage of the kth attribute for the ith target server.
- K is a normalization parameter used to set priority between minimizing number of bins versus maximizing packing efficiency.
- i,$\bar{\mathrm{i}}$ are used as index new servers.
- j,$\bar{\mathrm{j}}$ are used as index for applications.
- J_a - A subset of applications that cannot be hosted on same target server.
- k used as index for serevr attributes.

The objective is to minimize

$$Obj = \sum_i Y_i - K * \sqrt{\sum_i \left(\sum_k \left(\frac{\sum_j attr_{jk} * X_{ij}}{Attr_{ik}} \right)^2 \right)}$$

Subject to constraints

$$\sum_i X_{ij} = 1 \qquad \forall j$$

$$Y_i \geq X_{ij} \qquad \forall i, j$$

$$\sum_j X_{ij} \geq Y_i \qquad \forall i$$

$$Y_i * Attr_{ik} \geq \sum_j X_{ij} * attr_{jk} \qquad \forall i, j, k$$

$$\sum_{j \in J_a} X_{ij} \leq 1 \qquad \forall i, a$$

$$X_{\bar{\mathrm{i}}\bar{\mathrm{j}}} = 0 \qquad \exists \bar{\mathrm{i}}, \bar{\mathrm{j}}$$

5.2 Performance Evaluation

Assuming, the normalising constant K is chosen so that the objective is to minimize the number of target servers.

The GGA algorithm significantly reduces the number of Physical machines used over the bin packing algorithmn under the same constraints. The results are consistent from one run of the algorithm to another.

5.3 Conclusion

GGA works well in most of the cases, irrespective of the number of constraints being high or low. It is important to choose a ~~feaisble~~ initial solution, because the genetic operators are applied to this solution. In order to get a solution in fixed time, we can put a restriction on the number of generations. Thus, we may end up not getting the optimal solution, but the solution will be better than the one we had begun with.

Chapter 6

Virtual Machine Placement : Which algorithm works best?

Automating the process of virtual machine placement has become important with the increase in size of the data centers. Some of the factors that are typically used to quantify the cost functions are :

- Processor usage
- Storage usage
- Memory usage
- Network usage
- Power usage.

The choice of which performance factors and how many performance factors are to be taken into consideration varies from cloud provider to cloud provider.

6.1 Comparison of different techniques

Constraint Programming is useful in cases where we have the input data with us, that is, before we compute the cost functions, we know the demands of the Virtual Machines.The techniques using constraint programming are easily extendable so as to take additional constraints into account. It is easy to specify constraints so as to handle the trade-off between multiple objectives which might conflict with one another. The time taken to generate an optimal solution is high as the number of constraints increases.

A **Bin-packing** approach can be considered as a subset of Constraint Programming approach. It is useful for dynamic VM Placement, especially where the demand is highly variable. It is a heuristic based approach. Thus it may not give us an optimal solution. However, it will always generate a good solution in considerable amount of time. A Bin-packing approach is really useful when all physical machines have the same amount of memory and processing capabilities.

However, we can even model bin packing algorithms with constraints. There is always a tradeoff between packing maximum number of virtual machines on a single physical machine versus distributing the load across all the physical machines.

Multi-dimensional bin-packing algorithm can also be applied to VM placement problem. In this the dimensions correspond to amount of memory and number of processing units.

Genetic Algorithm is a way to solve the bin packing problem with certain constraints. The Grouping Genetic Algorithm requires more computing time and higher computing resources as compared to bin packing. It is particularly useful for static placements, that is, in scenarios where the demands do not vary over a considerable period of time. GGA is also useful for specifying VM-VM and VM-PM interference constraints.

Stochastic Integer Programming is useful where the future demands and prices of resources are not known, but their probability distributions are either known or can be computed. This is the best technique to be used in the case where we have two or more uncertain parameters on which the cost depends.

Each of the virtual machine placement algorithm works well under certain specific conditions. Thus, it is important to choose a technique that suits the needs of the cloud user and cloud provider. Also, the parameters to these algorithms should be properly specified. The performance metrics are measured at both system level and application level. The system level metrics are measured in terms of CPU load and the application level metrics are measured in terms of response time of applications.

Bibliography

[1] Bin packing. http://en.wikipedia.org/wiki/Bin_packing.

[2] Constraint programming. http://en.wikipedia.org/wiki/Constraint_programming.

[3] Genetic algorithm. http://en.wikipedia.org/wiki/Genetic_algorithm.

[4] Stochastic programming. http://stoprog.org/index.html?spintroduction.html.

[5] Shubham Agrawal, Sumit Kumar Bose, and Srikanth Sundarrajan. Grouping genetic algorithm for solving the serverconsolidation problem with conflicts. In *GEC '09: Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pages 1–8, New York, NY, USA, 2009. ACM.

[6] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. pages 119 –128, 21 2007-yearly 25 2007.

[7] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia L. Lawall. Entropy: a consolidation manager for clusters. In *VEE*, pages 41–50, 2009.

[8] Hidemoto Nakada, Takahiro Hirofuchi, Hirotaka Ogawa, and Satoshi Itoh. Toward virtual machine packing optimization based on genetic algorithm. In *IWANN '09: Proceedings of the 10th International Work-Conference on Artificial Neural Networks*, pages 651–654, Berlin, Heidelberg, 2009. Springer-Verlag.

[9] Bu-Sung Lee Sivadon Chaisiri and Dusit Niyato. Optimal virtual machine placement across multiple cloud providers. In *2009, IEEE Asia-Pacific Services Computing Conference*, pages 103–110. APSCC, 2009.

[10] Hien Nguyen Van, Frederic Dang Tran, and Jean-Marc Menaud. Autonomic virtual resource management for service hosting platforms. *Software Engineering Challenges of Cloud Computing, ICSE Workshop on*, 0:1–8, 2009.