Server Consolidation based on Hybrid Genetic Algorithm

Juan Zha*, Chang-Dong Wang†‡, Qiu-Liang Chen*, Xiao-Yan Lu* and Jian-Huang Lai†‡
*School of Mobile Information Engineering, Sun Yat-sen University, Zhuhai, P. R. China, 519082
†School of Information Science and Technology, Sun Yat-sen University, Guangzhou, P. R. China, 510006
‡SYSU-CMU Shunde International Joint Research Institute (JRI), Shunde, P.R. China, 528300
Email: zhajuan123@gmail.com, changdongwang@hotmail.com, chenqf6@mail2.sysu.edu.cn, zuixiangnian@qq.com, stsljh@mail.sysu.edu.cn

Abstract—Recently, with the rapid development of cloud computing, the number of cloud based-applications and cloud server providers have been increasing rapidly, which makes maximizing the efficient use of the cloud server an important research problem. The so-called server consolidation is a technology that uses limited server resources to improve the resource utilization of cloud computing data centres. Virtualization technique provides a way to integrate physical machines with low usage efficiency, assigning the tasks of the over-utilized servers to other servers, in order to achieve load balancing. However, due to the huge number of cloud servers, the consolidation problem is proved to be a NP-hard allocation problem. In this paper, the server consolidation allocation problem is taken as a three dimensional bin packing problem, and a hybrid genetic heuristic algorithm is proposed for addressing the limited server resource allocation problem. The experimental results show the high performance of the proposed method compared with the existing method.

Keywords—Server consolidation; Bin packing problem; Genetic algorithm; First-fit decreasing

I. INTRODUCTION

A. Background

Cloud computing [1] has become a hot technique recently. IBM first proposed cloud computing on 2007 technical White Paper [2]. In the following years, IBM launched their Blue-Cloud product. That product was first applied in their own company, organizing the company’s internal data center in the form of Internet Application, allocating the computing capacity to the resource organization which can be access across a local area network, instead of being limited to some local server cluster nodes or distal fixation.

As the essential part of cloud computation service, cloud server is a service platform which provides integrated services for all types of Internet users and access to a large pool of computation resources [3]. It integrates the traditional sense of the Internet application of the three core elements: compute, storage, network, which means applying Internet infrastructure services for the users of the public.

The benefits of cloud server consolidation are obvious, including but not limited to reduction of data-center footprint, indirectly reduction of power consumption and improvement of resource utilization. As we all know, power consumption concerns both the provider and consumer most. Server consolidation is in essence a cost-reduction activity, by significantly reducing the server footprint (by 30 to 50 percent or even more) [4] and allowing more than one virtual machines to run in one cloud server [5].

B. Motivation

Different from the traditional host service, server in the cloud has advantages such as steady performance, reliable qualities and so on. Additionally, cloud server can efficiently solve the problem caused by the difficulty of the traditional physical server management and shortage of the weakness of business scalability. But with the increasingly frequent use of the cloud, the cloud server access has increased, which leads to the challenging issues of how to consolidate and arrange cloud servers to get a better load balancing as well as save resources.

Meanwhile, as described in [6], different services require different amount of resources, which varies even within one day. With the help of server consolidation, we can make better arrangement and allocation in order to increase the utilization of physical server.

Server Consolidation in the cloud is an approach to take full advantage of resources we have and reduce consumption of energy. Now, there are three kinds of research on server consolidation. The first one is a multi-object server consolidation. This kind of research usually considers thoroughly a number of contradictory server consolidation goals, trying to get the solution which satisfies as many goals as possible at the same time; The second kind is a dependency server consolidation [7], which takes into consideration relations between virtual machines [8] as well as compatible or incompatible relations between virtual machines and physical machine; And the last one is a server consolidation combining load forecasting, considering the dynamic change of load in the virtual machine. We should notice that those three kinds of research are not completely independent of each other, which implies that there exist a certain overlap between them. The most important issue of the consolidation problem is how many and which servers are required for a given request set or tasks, and then how to make an optimal allocation of the request to the destination servers.

Consolidating multiple underutilized servers into small number of servers is of great importance. For example, IT industry research and analysis firm Gabriel Consulting Group(GCG) conducted a Unix Server Consolidation Survey [9]. The results show that customers are increasingly tending
to put their applications onto mid-range and greater UNIX servers to reduce more important costs and achieve operational benefits.

Findings in [10] also show us that the cost is not the only reason which decides server consolidation projects in the industry. Several other factors such as improved performance, making less of management and improving technology are all key driving force behind those problems. According to the data of Gartner Inc, we can draw a conclusion that 94 percents of IT departments are either considering server consolidation or are currently consolidating.

C. Contribution

The server consolidation allocation problem can be modeled as an multiple dimensional vector packing problem [11]. In bin packing problem [12]–[14], we are given items of different sizes and bins with different capacity, we want to pack them into the minimum number of bins so that we can make best use of those bins. In server consolidation, items are the requests of different kinds and sizes as well as some existing requests or tasks, bins are the destination servers. So our goal is to allocate the requests to the minimum number of destination servers. At the same time, the requests need the computation resources such as CPU and memory, while the server has resource capacity. The resource types are taken as dimensions in the multiple dimension bin packing problem.

In this paper we will consider the CPU, memory, bandwidth, and thus map the server consolidation to three-dimensional vector packing problem, which is an extension of previous work. That is the first contribution of this paper. The packing problem is a typical NP-hard problem, even the one dimension problem is NP-hard, which is hard to calculate. Therefore, our second contribution is that we combine the classic genetic algorithm [15] [16] and First Fit Decreasing (FFD) [17] [18] to solve the bin-packing problem, which inherits the great global convergence of genetic algorithms and overcome the poor local search ability. Experimental results demonstrate that the proposed method can make a good solution to server consolidation problem.

II. THE PROPOSED METHOD

In what follows, we will formulate the server consolidation problem. We are given a set of destination servers, \( S \), with the capacity of server resources, i.e. CPU, memory and disk. And we are faced with a set of requests to be allocated, \( R \), with the need of each request. Suppose the servers are numbered as \( 1, 2, 3, \ldots, n \). The number of requested services is \( m \), which are numbered as \( 1, 2, 3, \ldots, m \). We need to consider three kinds of resource limitations, namely memory, CPU and bandwidth, and we assume that the need of resource and workload of request can be known by pre-analysis and the need is relative static. Then the Server Consolidation Problem can be modeled as follows.

\[
\begin{align*}
\min & \sum_{i=1}^{n} y_i \\
\text{Subject to:} & \\
\sum_{i=1}^{n} x_{ij} &= 1, & j = 1, 2, \ldots, m \\
y_i \cdot P_{\text{mem}_i} &\geq \sum_{j=1}^{m} N_{\text{mem}_j} \cdot x_{ij}, & i = 1, 2, \ldots, n \\
y_i \cdot P_{\text{CPU}_i} &\geq \sum_{j=1}^{m} N_{\text{CPU}_j} \cdot x_{ij}, & i = 1, 2, \ldots, n \\
y_i \cdot P_{\text{band}_i} &\geq \sum_{j=1}^{m} N_{\text{band}_j} \cdot x_{ij}, & i = 1, 2, \ldots, n \\
x_{ij}, y_i &\in [0, 1], & i = 1, 2, \ldots, n, & j = 1, 2, \ldots, m
\end{align*}
\]

Where:

- \( x_{ij} \) indicates the allocation of services to servers, i.e. \( x_{ij} = 1 \) means the service \( j \) is allocated to the server \( i \). Otherwise, the service \( j \) is not allocated to the server \( i \).
- \( y_i \) indicates whether the server \( i \) is being used or not, i.e. \( y_i = 1 \) means the server \( i \) is being used, otherwise, it is not being used.
- \( P_{\text{mem}_i} \) means the memory resource provided by the server \( i \), while \( N_{\text{mem}_j} \) means the memory resource needed by the service request \( j \).
- \( P_{\text{CPU}_i} \) means the CPU resource provided by the server \( i \), while \( N_{\text{CPU}_j} \) means the CPU resource needed by the service request \( j \).
- \( P_{\text{band}_i} \) means the bandwidth resource provided by the server \( i \), while \( N_{\text{band}_j} \) means the bandwidth resource needed by the service request \( j \).

There are some necessary explanations for the equations.

- Equation (1) is the objective function to finds the minimum appropriate number of servers;
- Equation (2) identifies each request is allocated one and only one server;
- Equation (3) (4) (5) make restraint on servers memory, CPU, bandwidth respectively, which can not exceed the max sources servers supply;
- Equation (6) respects that the value for \( x_{ij} \) and \( y_i \) can only be 0 or 1.

The above three-dimensional problem is a typical NP-hard problem, which cannot be resolved within polynomial time, especially when the scale of problem is large. However, in the field of cloud computing, the number of servers and service requests are of large scale, so it is very important to find an approximate solution in a relatively short time.
A. Relation of Server Consolidation and Three Dimensional Bin Packing Problem

And it must be pointed out that, because the server consolidation problem we consider of is a kind of three-dimensional problem, so we can make use of the method for three dimensional bin packing problem, however, it is not exactly the same as the typical and classical three dimensional bin packing problem. The typical three dimensional bin packing problem can be shown in Fig. 1.

![Three dimensional bin packing problem](image)

The main resource of three dimensional bin packing problem [19] is the volume of the bin, that is \( v = x \times y \times z \). But in our server consolidation problem, we can not take \( CPU \times mem \times band \) as our limitations, although they are distinct resources. For example, suppose the server has the CPU, memory and bandwidth which are all 100 units. If one task needs all 10 units of CPU, 90 units memory and 20 units bandwidth, then the server is almost used up. Thus if another task asks for 20 units CPU, 30 units memory and 30 units bandwidth, the server can not allocate those resources to it because of the limitation of memory. That’s why the constraints (3)(4)(5) are individual on CPU, memory and bandwidth.

Despite the constraint differences between server consolidation problem and three dimensional bin packing problem [20], they can be treated as the similar problem. Thus we can illustrate an algorithm solving server consolidation problem from the solution of bin packing problem. Due to the long history of bin packing problem, many researchers have thought out different kinds of solutions. For small scale problems, there are some exact solutions. Silvano Martello in [21] developed an exact Branch-And-Bound algorithm for the three dimensional bin packing problem, which also incorporates original approximation algorithms and discussed the lower bound. As for big scale problem, there are lots of approximate algorithms to solve this problem. The mainly-used ones are:

- First Fit Decrease (FFD) algorithm, namely to a certain bin, searching all the items which can be packed in this bin. If one particular item we want to find doesn’t exist in that bin, we change the bin. With a complexity of \( O(n^2) \), this method is used in [10]–[12].

- Least loaded (LL) algorithm, namely checking whether the next item can be fit in the bin being used, and finding a new bin if can’t. This method is used in [10] [12] [13].

- Best Fit Decreasing (BFD) algorithm, namely to a certain item, finding the bin which will be most filled when this item is packed in.

The three algorithms mentioned above are all implemented based on greedy algorithm. As we all know, the biggest problem of greedy algorithm is that it can not guarantee we will achieve a global optimal solution. To solve that problem, many scholars think out heuristic methods, such as genetic algorithm [10] [11], simulated annealing algorithm [12], tabu search algorithm [13] and so on. Those heuristic methods may show different performances depending on specific situations. When solving large scale problems, evolutionary algorithm like genetic algorithm shows better performance because of its excellent self-adaptive capability and learning capacity.

This paper uses classical genetic algorithm combined with FFD to solve the bin-packing problem of server consolidation.

B. Genetic Algorithm

Genetic Algorithm (GA), put forward by American professor J. Holland in 1975, is a kind of computing model which bases on Darwinism and imitates the natural selection as well as life evolution system. Always, it is used to find out the optimal solution through simulating natural revolution procession.

The procedure of solving a combinatorial optimization problem can be divided into five main steps: chromosome coding, fitness calculation, population selection, crossover and mutation.

C. Chromosome Coding

A problem of classical FFD algorithm is that different order of allocation may result in different output. If there are \( m \) service requests to be allocated, which one should be satisfied firstly? It is generally known that there are maybe up to \( m! \) types of orders, and as \( m \) becomes bigger and bigger, \( m! \) can be a huge number. Once we get the best order of the allocation, we can get an acceptable result by using the FFD algorithm. Though it is not the optimal result, it may be good enough. However, we can not simply test all the orders. In order to fixed that problem, we use genetic algorithm to get the global optimal solution.

The most important step of genetic algorithm is chromosome coding. Since our goal is to search for the best order of allocation, we use the chromosome coding as a representation of the order for service request. For example, if there are six service requests, one of the chromosome may be \( \chi_1 = \{2,1,3,4,6,5\} \), which means we first allocate request 2 and then request 1 and so on. Then the result of that chromosome is \( \chi_1 = \{2,1,3,4,6,5\} \) and \( \chi_2 = \{1,2,3,4,5,6\} \) calculated by FFD allocation algorithm may be different.
D. Fitness Calculation

Once we get a chromosome, we can define its fitness value. Considering that our objective function is to minimize the number of using servers, we can just take it as the fitness value at first, and optimize it later.

Because each chromosome means an allocation order, we need to calculate its fitness value for the GA algorithm. Actually, calculating fitness value for each chromosome is a hard problem too, in this paper we adopt FFD algorithm to do that. Here we use FFD algorithm.

Pseudocode is shown in Fig. 1

Algorithm 1 Process Allocation
1: Using: FFD Algorithm.
2: given: chromosome \( R = \{s_1, s_2, \ldots, s_m\} \)
3: initial: \( n = 1, \chi_1 = \{\} \)
4: for \( i := 1 \) to \( m \) do
5: for \( j := 1 \) to \( n \) do
6: if packOk(\( \chi_j, s_i \)) then
7: \( \chi_j \leftarrow \chi_j \cup \{s_i\} \)
8: break
9: end if
10: end for
11: if \( j = n + 1 \) then
12: \( n \leftarrow n + 1 \)
13: \( \chi_n \leftarrow \{s_i\} \)
14: end if
15: end for

There are some necessary explanations for the Pseudocode.

- Variable \( m \) in line(2) means the order of the request;
- The condition in line(3) means initial server number is 1, and no request allocated to that server;
- The condition in line(6) means if it’s satisfied, then the allocation is ok to make;
- The statement in line(7) means allocating the request of service \( s_i \) to the server \( j \);
- The statement in line(11) (12) means if for all the current servers, the request can not be fulfilled, then we will add more server rather than allocate resource to the request of \( s_i \).

E. Selection

Selection is an important step in genetic algorithm. After calculating the fitness value for each chromosome, we can take some actions to select from the population.

The common selection methods are Roulette Wheel Selection, tournament selection and so on. However, in order to avoid that the population fall into local convergence and degradation, we abandon tournament selection which may result in local solution. Instead we adopt a selective method according to fitness value order. At first, we calculate the adaptive value for each chromosome and sequence them by adaptive value from high to low, then keep the ones to next generation who have high fitness values. In this process, we must make sure that no duplicate chromosome will turn up. At the same time, keeping a small proportion of the worst solution which may have advantages in avoiding local convergence.

F. Crossover

According to the coding method displayed above, our main purpose is to change the genetic order of chromosome. This problem is similar to Travelling Salesman problem (TSP) [22], so we can use homologous crossover operator. Traditional crossover operators include single-point crossover, double-point crossover, partial crossover, sequential crossover and so on. Here we use the sequential crossover operator.

For example, if we have the following parents A and B, choose a coding substring from parent generation A randomly and put it on the corresponding position of child generation A. Then the rest positions of generation A are chosen from generation B by sequence of B (but don’t repeat with the existing code). Besides, child B can be got by the same method.

Father generation \( A \): 872 | 139 | 0546
Father generation \( B \): 983 | 567 | 1420

After cross:

Children generation \( A \): 856 | 139 | 7420
Children generation \( B \): 821 | 567 | 3904

G. Mutation

In the operation for variation, we can’t mutate a chromosome optionally, because we must make sure each service request appear only once. In order to avoid repeating visit, we just exchange genes in the same chromosome.

Figure 2 shows a chromosome before and after variation.

Fig. 2. Before and after variation

For clarity, Algorithm 2 summarizes the entire process of the algorithm.

There are some necessary explanations for the Pseudocode:

- The statement in line(6) means calculating the fitness of the individual;
- The statement in line(9) means selection operator;
- The statement in line(12) means we get the next generation of \( P(t + 1) \).
Algorithm 2 Process GA

begin
2: initialize P(0);
3: \( t = 0 \);
4: while \( t \leq T \) do
5: \( i := 1 \) to \( M \) do
6: Evaluate fitness of \( P(t) \);
7: end for
8: \( i := 1 \) to \( M \) do
9: Select operation to \( P(t) \);
10: end for
11: \( i := 1 \) to \( M/2 \) do
12: Crossover operation to \( P(t) \);
13: end for
14: \( i := 1 \) to \( M \) do
15: Mutation operation to \( P(t) \);
16: end for
17: \( P(t+1) = P(t) \);
18: end for
19: \( t = t + 1 \);
20: end while

III. Computational Experiments

We implemented the algorithm described above in C++. The computational experiments are run on our PC with 8G memory and 2.3Ghz CPU.

We first randomly initialize the test data. The test data are generated as follow: firstly, there are five categories, and the items(request) number are 100, 200, 500, 1000, 2000; secondly, in each category, we randomly generate three types of resources each request. And the capacity of the destination servers are set as 100, which is a constant. For each categories of data set, we run three times. We tests the data set by classical FFD algorithm [17] and our hybrid genetic algorithm (Hybrid GA) to compare to the results. Part of our test data are shown in Fig. 3 and the results are shown as table Fig. 4 below.

As the experiments result table Fig. 4 shows, the hybrid genetic algorithm we presented in our paper shows better performance in solving the server consolidation problem. However, it also indicates that the hybrid genetic algorithm cost more time than FFD algorithm. In the cloud computing industry, this may be a trade-off, if we want to get more optimal result and thus reduce the server number, we need to spend more time on computation. And, tens of seconds is an acceptable time consume.

IV. Conclusion and Future Work

Server consolidation plays an important role in reducing the costs of data center and cloud server. In this paper we presented a hybrid genetic heuristic algorithm to solve the consolidation problem [23] [24]. Although, the experiment mentioned above shows that our algorithm in this paper
performs well in use, we only focus on the physical server consolidation currently. So we still have further work to do.

Firstly, the problem we considered is static server consolidation problem because the need of resource and workload of request are static. Thus we can extend this problem to variable workload.

Secondly, we need to collect more testing data, especially some industry data to test our algorithm.

Thirdly, in our article, we suppose all the servers are in healthy condition and they have equal performance when dealing with data. But in fact, different servers can have different performance. Those elements can be taken into consideration in our future work, and we will take more work and improve our algorithm.

ACKNOWLEDGMENT

This project was supported by CCF-Tencent Open Research Fund, the PhD Start-up Fund of Natural Science Foundation of Guangdong Province, China (No. 2014A030310180), the Fundamental Research Funds for the Central Universities (46000-3161006), and Pilot Program of SYSU-CMU Shunde International Joint Research Institute.

REFERENCES