A Review on Bin Packing Algorithm for Resource Auto Scaling

[1]Krishnapriya.S,[2]Sumathi.D
[1]PG Scholar,[2]Assistant Professor,
[1,2]Computer Science and Engineering, PPG Institute of Technology Coimbatore,641035,India

***Abstract*—One of the main features of cloud computing is automatic scalability of resource usage. There are web applications in which their usage changes from time to time. Application demand satisfaction depends on the underlying server capacity and the count of the application instances that are currently running on the servers. When the application demand is more, resource availability also needs to be increased to ensure requests satisfaction. Over provisioning of resources to handle peak load may cause some resources to lie idle during low demand. Automatic scaling of resources ensures better customer satisfaction during peak usage time by allocating more resources and scales down the resources when not needed. The scaling up and down of resource demands can be considered as item arrival and departure in bin packing algorithm. In this paper we are reviewing a few optimized versions of original bin packing problem. We have performed a comparison on the advantages and limitations of these bin packing problems when they are used for auto scaling of resources in cloud computing.**

***IndexTerms*—**Cloud Computing, resource auto scaling, online and offline algorithms, bin packing problem.

I. INTRODUCTION

In a traditional dedicated hosting environment, the usage of hardware resources is limited. Once the resource capacity reaches its maximum, the site/application suffers from performance issue. However, cloud computing helps in building a scalable server setup. If the application needs more computing power, Cloud has the ability to launch additional compute resources on-demand and terminate them when the additional demand is no longer required. The computing capacity available to an application depends upon the count of application instances running on the servers. The application demand satisfaction capacity is higher, if the servers are more powerful and there are more instances of the application [1].

Auto-scaling is a cloud computing service feature that automatically adds or removes compute resources like virtual machines (VM) and server capacities, depending upon actual usage. It adjusts the resource capacity in cloud environment based upon the current demand and condition. Auto Scaling ensures customer satisfaction by maintaining the performance by increasing the number of instances of the application during an increase in demand and decreasing it automatically during minimum demand. Also when the resource requirement is less, energy can be saved by cutting down the utilization of servers. Auto Scaling is well suited in situations where the application demand varies unpredictably.

To better understand about automatic scaling of applications, consider a web application which has varying usage. There may be occasions where usage of application is less or there is an increase in application requirement. One option to handle variable application demand is to include sufficient number of servers that can provide enough capacity to meet the application requirements. But the drawback of this option is that, there are cases in which this extra server capacity remains unoccupied and causes increase in cost. Another option is to have enough capacity to satisfy the average application demands. However, this may cause poor customer satisfaction, when the provided capacity fails to meet the demands on the application. The concept of automatic scaling helps in this scenario. By automatic scaling, new instances of the application are added whenever it is necessary and stops them when it is not required.

The adding and removing of application instances can be can be viewed as adding and removing of items to and from the bin in a bin packing problem. In the following section of this paper, we are reviewing few optimized packing problems used for achieving resource auto scaling in cloud computing.

II. BIN PACKING PROBLEM

In the original bin packing algorithm, a set of items with various sizes has to be placed in bins so that the number of bins (with limited capacities) is minimum. There are various modifications of original bin packing problems depending upon packing situations such as:

- Packing in which there is restriction in placing items in a bin in advance not to exceed a certain number.
- Restrictions on items which can be placed into the same bin.
- Packing in which the items being packed may be allowed to disappear during the packing process.
- Packing in which there is an ordering attached to some of the weights which limits the way those items can be packed.

In the bin packing problem, there is a finite set of items with n elements and a finite number of bins or containers. The objective is to place each item to a particular bin so that the total amount of elements in a bin does not cross the capacity of the bin and also the total count of bins utilized is as small as possible.

III. ONLINE AND OFFLINE ALGORITHMS

A problem is said to be online if the input to the problem is given one at a time, and the algorithm must commit to parts of the solution over time before the entire input is revealed to the algorithm [needs editing].

An on-line bin packing problem is one in which each element from the item set is given one at a time and must be assigned to a bin immediately on its arrival. The packing of each item ai is based on its size and the packing of the previous items a1, a2 , . . . ai−1, without having any information on neither the sizes of the subsequent items, nor their number [2]. The decisions of the algorithm are irrevocable, i.e. placed items cannot be replaced at a later stage and rejected items (in the knapsack problem) cannot be packed later.

An offline bin packing algorithm is completely aware about the list of items. So it can apply advanced strategies for packing. A Semi online algorithm is an intermediate class between on-line and offline algorithm. Semi online algorithms allow some relaxation on the on-line restriction by permitting certain additional operations in each step such as, packed items can be repacked at later stages, pre-processing the items by reordering them with respect to sizes or buffering a few items before packing those items [2].

Usually, in a Cloud environment, incoming requests are unpredictable. An online algorithm is capable of completing each request without the knowledge of the next request in the sequence. An optimal offline algorithm is one which has the capability to view the request sequence in advance. The ratio between the performance of an online algorithm and the performance of an optimal offline algorithm is called the competitive ratio of that online algorithm.

Here, we are considering class constrained versions of packing problem which model resource allocation problem in computer science. The input to the packing problem is a set of items with each item having a unit capacity and color (class). The items have to be placed in identical bins. The output to the packing problem is a placement that constitutes a subset of items from each class to be placed in the bins.

Two optimization algorithms for Bin packing problem are Class Constrained Bin Packing Problem (CCBP) and Class Constrained Multiple Knapsack Problem (CCMKP) [3].

The Multiple Knapsack version tries to develop a placement which can maximize the number of packed items. In MKP problem, there is a set of n items and m bins such that each item has a profit p and a size s, and each bin j has a capacity c. The objective is to find a subset of items of maximum profit such that they have a feasible packing in the bins.

The items that need to be packed are grouped into classes or colors in class constrained bin packing problem. Each bin has a specific capacity (v) and can occupy items from a maximum of c classes. It is called "class constrained" since the class of items that can be placed in the same bin is constrained. The aim is this packing method is that, the number of bins used for packing must be less. That is, CCBP tries to reduce the number of resources needed for satisfying the requests.

Figure 1 represents a CCBP algorithm in which each bin (server) can accommodate items (applications) from a maximum of two classes simultaneously (i.e. c=2) and each bin has a capacity to occupy at most 5 items (i.e. v=5).

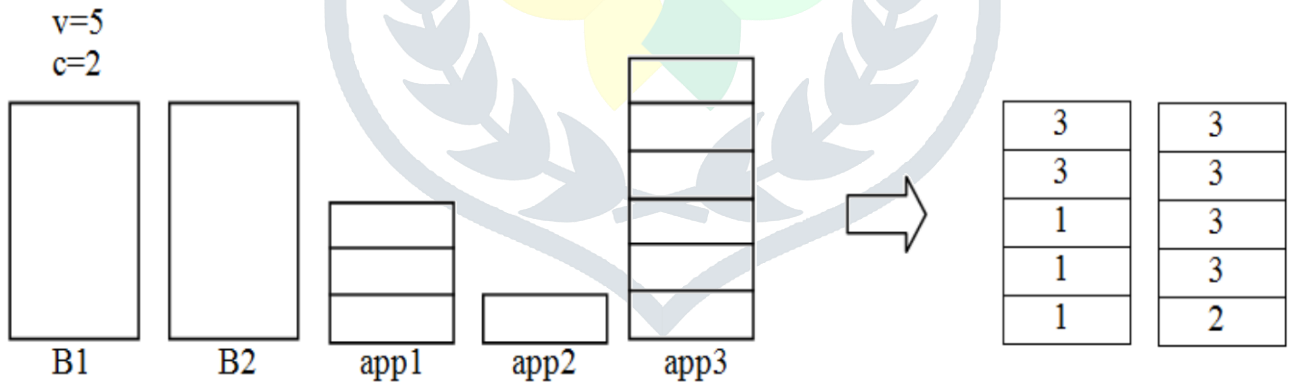

Fig. 1: Class Constrained Bin packing Algorithm [1]

IV. REVIEW OF PACKING ALGORITHMS FOR AUTO SCALING

Several modified bin packing algorithms have been developed to achieve auto scaling in cloud computing. In the algorithm presented in paper [4], a single machine is shared by multiple applications. Here a set of machines having constrained resources are considered. The problem in dynamic placement of applications is to decide the number of instances of each application that has to be run and where to place them. This problem of application placement can be viewed as a variant of Class Constrained Multiple Knapsack problem. In this algorithm the inputs given to the placement controller are the current placement matrix , the placement restriction matrix, the CPU/memory capacity of each machine and the CPU/memory demand of each application. The outputs of placement controller are the updated placement matrix and load distribution matrix . The goal is to allow the sharing of a single machine among various applications, and tries to improve the application demand satisfaction, reduce the number of application starts and stops, and to balance the load across machines.

The basic idea behind this algorithm is to optimize the placement solutions repeatedly in several cycles. The application instances can be classified as idle instance, underutilized instance or fully utilized instance. The maximum demand of application that the present placement can satisfy is calculated sequentially. The algorithm ceases, if all application demands are met. If not, it switches the load across machines. Then the idle instances are stopped and start the needed ones to improve satisfaction ratio. In each cycle, the algorithm first invokes the load shifting task and then invokes the placement changing subroutine. The switching of loads before altering the placement can make later modifications of placement solution easier.

Paper [1] proposes an efficient algorithm which is semi-online. This algorithm has good demand satisfaction ratio, reduces frequency of placement change, minimize request response time and saves energy by keeping idle servers on standby when the load is low. The proposed algorithm comes under the family of color set algorithm. Each class of the items is labelled with a color and organizes them into color set. The maximum number of distinct classes (colors) that a bin can accommodate in a color set algorithm is at most 'c'. The amount of resources available at a server for all its applications represents the capacity of a bin. The servers are assumed to be homogeneous with similar capacity and there is a finite number of servers available. The size of an item represents the amount of an application's load. Thus the capacity of a bin is the number of units of load that can be accommodated by a server. The resource requirement of an application represents the number of items in a specific class waiting to be packed.

A review on different versions of bin packing problems namely, (I) Offline CCBP approximation algorithm, (II) Strict online CCBP algorithm, (III) Semi – online CCBP algorithm in Paper[1], (IV) Class Constrained Multiple Knapsack algorithm and (V) Placement algorithm in Paper[4] has been carried out. Table 1 shows the advantages and limitations of these algorithms.

Table.1: Analysis of various packing algorithms

Algorithms	Merits	Demerits
I	Algorithm is aware about the input which is an added advantage in producing a better output	No minimization of shuffling of already packed items, do not support item departure, assumes that input sequence is known in advance
II	Input sequence of items are not known in prior which is more realistic in cloud environment	Decisions are irrevocable, placed items cannot be replaced at a later stage and rejected items cannot be packed later
III	Supports items departure, low placement change frequency, short request response time, good energy saving	Assumes that servers are homogeneous with uniform capacity
IV	Maximizes total number of packed items i.e. maximizes the number of satisfied requests.	Rejected items cannot be packed later, does not attempt to minimize the number of knapsack used, hence it does not support green computing when the system load is low
V	Attempts to maximize the total satisfied application demand minimize the number of application starts and stops, and to balance the load across machines.	Developed for enterprise application where the applications are trustworthy unlike cloud computing, also it does not support green computing.

V. CONCLUSION

Resource elasticity is one of cited service of cloud computing. Many Internet applications can benefit from an auto scaling property where their resource usage can be scaled up and down automatically by the cloud service provider. Efficient management of resources is of prime interest to both Cloud Providers and Cloud Users. Resource allocation is similar to placement of items in bin packing problem. In this paper we have performed a review on different approximations of bin packing algorithm and analyzed various properties of these algorithms.

REFERENCES

[1] Zhen Xiao, Qi Chen and Haipeng Luo, "Automatic Scaling of Internet Applications for Cloud Computing Services," IEEE Transactions On Computers, VOL. 63, NO. 5, May 2014.

[2] L. Epstein, E. Kleiman, "Resource augmented semi-online bounded space bin packing," Discrete Applied Mathematics 157 (2009) 2785–2798.

[3] H. Shachnai and T. Tamir, "Tight bounds for online class constrained packing," Theor. Comput. Sci., vol. 321,no. 1, pp. 103–123, 2004.

[4] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in Proc. Int. World Wide Web Conf. (WWW'07), May 2007, pp. 331–340.

[5] L. Epstein, C. Imreh, and A. Levin, "Class constrained bin packing revisited," Theor. Comput. Sci., vol. 411, no.34–36, pp. 3073–3089, 2010.

[6] M. Armbrust et al., "Above the clouds: A Berkeley view of cloud computing," EECS Depart., Univ. California, Berkeley, CA, Tech. Rep. UCB/EECS-2009–28, Feb. 2009.

[7] E. C. Xavier and F. K. Miyazawa, "The class constrained bin packing problem with applications to video-on-demand," Theor. Comput. Sci., vol. 393, no. 1–3, pp. 240–259, 2008.

[8] G. Galambos and G. J. Woeginger, "On-line bin packing-a restricted survey," Physica Verlag, vol. 42, no. 1, pp. 25–45, 1995.

[9] E. Caron, L. Rodero-Merino, F. Desprez, and A.Muresan, "Autoscaling, load balancing and monitoring in commercial and opensource clouds," INRIA, Rapport de recherche RR-7857, Feb. 2012.

[10] G. Gambosi, A. Postiglione, and M. Talamo, "Algorithms for the relaxed online bin-packing model," SIAM J. Comput., 30:1532–1551, 2000.

[11] Steven S. Seiden, "On the online bin packing problem," Journal of the ACM, 49(5):640–671, 2002.

[12] J. Famaey, W. D. Cock, T. Wauters, F. D. Turck, B. Dhoedt, and P. Demeester, "A latency-aware algorithm for dynamic service placement in large-scale overlays," in Proc. IFIP/IEEE Int. Conf. Symp. Integrat. Netw. Manage. (IM'09), 2009, pp. 414–421.

[13] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi, "Dynamic placement for clustered web applications," in Proc. Int. World Wide Web Conf. (WWW'06), May 2006, pp. 595–604.

[14] G. Gutin, T. Jensen, and A. Yeo, "Batched bin packing," Discrete Optimization., 2:71–82, 2005.

[15] E. F. Grove, "Online binpacking with lookahead," In Proc. 6th Symp. on Discrete Algorithms (SODA), pages 430–436.ACM/SIAM, 1995.

[16] L. Epstein and R. van Stee, "Online bin packing with resource augmentation" In Proc. of the Approximation and Online Algorithms, Second International Workshop, WAOA, pages 23–35, 2004.

[17] J. Csirik and G. J. Woeginger, "Resource augmentation for online bounded space bin packing," Journal of Algorithms, 44(2):308–320, 2002.