International Research Journal of Computer Science (IRJCS)
Issue 12, Volume 2 (December 2015)
ISSN: 2393-9842
www.irjcs.com

Improved Performance for Cloud Services using Memory Allocation Techniques

Mohamed Al-Ibrahim, *Dr.*
Computing Dept. , KILAW

Naser Al-Ibrahim, *Eng.*
Computer Eng. Dept. , Kuwait University

Abstract— The technological advancements over the years towards the improvements of the processor were astonishing, however memory developments weren't as much. Memory allocation is a technique used to improve the memory assignment of programs in memory. The process of memory allocation is to assign either partial or complete portion of memory to the execution of processes. In this paper, we focus towards the concept of dynamic memory allocation in cloud computing environments where we present a cloud environment with a global memory and a stream of requests that requires to be assigned part of the global memory for a period of time. Different memory allocation schemes such as first-fit and best-fit have solved this problem. The proposed allocation scheme of this paper uses variant of sequential fit to produce slightly better results depending on the used environment. In this scheme, portals are proposed in order navigate and reserve memory. In order to demonstrate the efficiency of the proposed allocation scheme, we have simulated a pre-defined memory environment where we compared the performance of our technique with other techniques mentioned in the literature.

Keywords— cloud computing, memory management, sequential fit,

I. INTRODUCTION

Cloud computing has been a topic for interest for quite some time. Forrester Research [14] mentioned that the cloud computing market would grow up to 241$ billion by 2020 compared to 2010 where it was only 40$ billion. As an emerging topic of interest, cloud services are becoming the primary source of computing power for business and personal applications. Cloud computing operates by providing for their users a shared pool of servers in a secure data center where they offer subscription based and easily attainable computation in a scalable manner. Several cloud service models were proposed based on the generality of these resources: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [15]. A new cloud service model was recently introduced [16] named Resource as a Service (RaaS). The RaaS model operates by renting resources separately instead of a fixed bundle. This results in transforming memory into an important billed resource. Thus stimulating a new interest in memory allocation as part of memory management.

Memory allocation is a technique in memory management to efficiently assign memory to different computer process and programs. Memory allocation is divided into two types: Static and Dynamic. Static memory allocation operates at compile time where all memory assignments are done, however dynamic memory allocation functions at run time where memory assignments have to be done on the fly. Since dynamic memory allocation operates at run time, the crucial aspect of any dynamic memory allocation is speed because a slow allocator would become a major bottleneck. Moreover, as the memory is limited is space, other the goal is not waste any memory space.

There are three basic categories for any memory allocation algorithm: sequential fits, segregated free and buddy system [17, 18, 19]. The mechanism of sequential fits is to keep track of free memories (holes) as a linked list that can be searched later sequentially. There are different techniques to search the list such as: random, first-fit, or best-fit. The random technique operates by random selecting a hole that fits the requesters criteria. While first fit functions by selecting the first hole in the list that fits the criteria. On the other hand, best fit works by selecting the best hole that fits the given criteria. Fig. 1 illustrates the random, first fit and best fit strategies given that the requestors memory criteria is 10 GB.

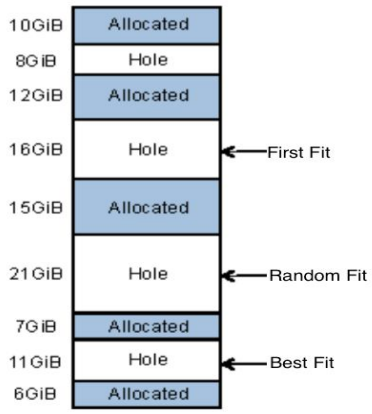

Fig. 1: Memory snapshot given that a users requesting 10GB [6]

In this paper, we introduce a new memory allocation technique that is based on the concept of sequential fits. The proposed solution exploits the concepts of portals to traverse memory in different manner. For simplicity we will assume that there exists a single portal in memory that can group different location of memory together. The scheme operates in a fashion similar to first fit until some user requesting memory gets blocked near a portal. In this case, it checks if it is possible to switch the portal into ON/OFF and check if it is possible to allow the user to reserve the memory space. We illustrate the resourcefulness of our approach by comparing it with other well versed approaches such as first fit, best fit and random memory allocation techniques. The rest of this paper is organized as follows: Section II presents the literature review of this field. The problem definition and analysis of the problem is defined in section III. In section IV we introduce our proposed technique where it is implemented and shown in section V. Section VI presents our results. We conclude in section VII with future work.

II. LITERATURE REVIEW

Memory allocation is a major topic of interest in the field of improving memory performance. Several works like Albers et al. [1] analyzed the first fit technique on the classical bin-packing problem where objects of different volumes must be placed inside fixed size containers while minimizing the number of bins used. They generalized the multi-dimensional Markov chain to demonstrate that the bin-packing algorithm is stable under pre-defined conditions. They created a tight bound to solve this NP-Complete problem. While Aron et al. in [2] proposed a new technique called cluster reserves to improve the performance of a webserver. They have evaluated a prototype implementation of their idea. Their attained results yield an increase in higher resource utilization and performance than the approaches used at the time.

In [3], Shore et al. published a through comparison on the effects of first-fit and best-fit memory allocation algorithms. Through simulation, the data obtained relative to the performance of the algorithms was acceptable. Their results demonstrated that with a small distribution both algorithms perform similarly while using an exponential distribution the first-fit algorithm outperformed best fit. They concluded that when the variation of the request distribution is greater than unity, first-fit would outperform best fit.

Stillwell et al. in [4] have proposed several algorithms to solve the resource allocation problem. They believe that the best approach to solve the problem is to perform a binary search over the yield. Among the different algorithms suggested, the Chose Pack vector- packing algorithm is the best. The disadvantage of the paper is that the workloads for which the number of instances per service do not change throughout time.

Costea et al. proposed an alternative resource allocator other than the generic allocation algorithms such as best-fit or first-fit for the miriaPOD platform in [5]. They claim that their allocator exploits the memory sharing features of the virtualization backend. They implemented a simulator that determines the performance of their proposed allocator. However, the proposed allocator is custom made for the miriaPOD platform.

In [6], Al-Yatama et al. proposed a novel technique that segregates the memory holes according to the size of the hole. After running their algorithm, they claim that their new algorithm is faster than traditional used techniques, it was also found that the proposed technique was distributing the data to the memory more fairly. Consequently, Younis et al. in [7] proposed a generic algorithm that handles segregated list of different size holes. They claim that their simulation results improved the performance of their system.

Lioa et al. have proposed another novel technique in [8]. The technique focuses on the energy consumption, which uses two heuristic algorithms that use Xan heaps. The simulation presented in the paper yielded 50% more energy efficiency than the standard used algorithms.

Elias et al. in [9] presented six different memory-allocating algorithms where they have compared them with each other in terms of execution time and memory usage. They claim that the Ptmallocv2 algorithm was superior in terms of time while the Ptmallocv3 algorithm was better in terms of memory usage.

In [10], Husagic-Selman et al. proposed a method that uses fuzzy logic in order to quickly handle real time memory allocation. Their simulated results yield a slight increase in speed by using their methodology. However, they claim that by changing and improving the fizzy pattern, the fizzy allocator is to be improved leading to better results.

Chung et al. in [11] proposed a new dynamic memory allocation scheme called Lazy-Fit. The scheme operates by using pointer increments as the main allocation method and conventional schemes such as First-Fit and Best-Fit are used as a backup. They claim that their proposed scheme if implemented properly could be faster than conventional schemes. However, the disadvantage of their scheme is the large increase of fragmentation that could be inadequate.

In [12], Hasan et al. performed a study of the different implementation of the Best-Fit memory allocator technique. In their study, they noticed that a 33% increase of performance is achieved if the algorithm is implemented by using the Doug Lea method. However, they should have included the First-Fit technique in their study to obtain even better results.

Orna et al. in [13], have proposed “Ginseng” a market-driven memory allocator. Their scheme inspires clients to bid their true memory needs upon actually requiring it.

By continuously gathering clients' bids, they claim that Ginseng finds an efficient memory allocator, re-allocates physical memory and present it to the clients. They state that an improvement of 83-100% of the optimum in aggregated client satisfaction is achieved when compared with other state of the art approaches in cloud memory allocation. Similar to the work done in [6], we have proposed a new memory allocation scheme explained in details in the following sections.

III. PROBLEM DEFINITION & OBJECTIVES

To improve the performance of computer programs, the physical memory should be utilized efficiently. Memory allocation is a technique used to further utilize the physical memory. In order to improve the overall performance of computer systems, a suitable memory allocation algorithm should be used.

Given a physical memory of size X and a stream of requests of different instance types following a pre-defined distribution desiring to utilize the given memory. A memory allocation scheme will assign each request to a location in memory to utilize for a given period of time T. In order to improve the performance, the number of requests unable to utilize the memory given that the memory is full, must be minimized.

In brief, we do the following:

- Describe the new memory allocation scheme that improves the performance and postpone the life of the physical memory by reducing the fragmentation occurring in the memory
- Implement and simulate the proposed scheme
- Compare the proposed scheme with other well-known memory allocation schemes such as First-Fit and Best-Fit

Further, we define the work environment, and determine the representation of data.

A. *Defining Environment*

We assume a physical memory consisting of *M* memory units and *I* different instances, each having different memory requirements.

$$\forall\, i \in I\ \exists\ m_i \ll M$$

We will assume the smallest memory unit is u = 0.5 GiB. The amazon elastic cloud instances [20] t2.small and r3.8xlarge are equal to 2 and 244 GiB respectively. Thus the t2.small and r3.8xlarge will require 4 and 488 memory units respectively. We will define the different instances *I* according to the amazon instances. Table 1 shows the different amazon instances and their corresponding memory units.

TABLE 1
AMAZON CLOUD INSTANCES

Name	Memory Size (GiB)	Memory Units (u)
t2.micro	1	2
t2.small	2	4
t2.medium	4	8
t2.large	8	16
m4.xlarge	16	32
m4.2xlarge	32	64
m4.4xlarge	64	128
m4.10xlarge	160	320
r3.8xlarge	244	488

We will assume that each instance will arrive to our system following the Poisson distribution with λ_i requests per unit time. The normalized arrival rate for an instance can be obtained using the following equation:

$$\bar{\lambda_i} = \frac{\lambda_i}{\sum_I \lambda_i}$$

The holding time of requests in the system is assumed to follow the exponential distribution with a mean of 1 unit of time (e.g. 2 hours). Additionally, we assume that blocked requests will not enter the system again which implies that request-retrial rate is insignificant. We define the lost revenue in $ to be the sum of all blocking probability of requests multiplied by the requests demand for all $i \in I$. Then,

$$\text{Lost Revenue} = \sum_{i \in I}^{I} D_i \times b_i \quad \$$$

and the normalized lost revenue is

$$\text{Normalized Lost Revenue} = \frac{\sum_{i \in I}^{I} D_i \times b_i}{\sum_{i \in I}^{I} D_i} \quad \%$$

D_i is given by the following demand function $D_i = m_i \, x \, \lambda_i$ where m_i is the request memory unit size and λ_i is the arrival rate for that request type. The blocking probability b_i is the probability that a specific request type will be blocked from the system.

IV. DESIGN METHODOLOGY

The scheme is tested under different environments such that the arriving requests could be in uniform, exponential or bell distribution. The lost revenue is then calculated for each environment and compared with different memory allocation strategies in a graph.

Portal is used in the scheme to navigate the memory. Portal, simply, is different arrangement of the memory. Fig. 3 below shows how portals function in the memory.

	1
Portal Entrance	2
	3
Portal Exit	4
	5

→

1
2
4
5
3

Fig.3: Memory Snapshot of the portal closed and opened. The memory on the left shows the memory arrangement when the portal is closed while the memory on the right shows the memory arrangement when it is open

We notice from Fig. 3 that the memory in both cases is sequential listed. The algorithm exploits this feature by assigning users memory space in both situations if it is possible. The overall architectural design of this algorithm can be implemented by adding one extra bit in the hardware design. The underlying algorithm behind this technique is based on the first fit algorithm mentioned previously.

A. Algorithm Functionality

The algorithm functions initially by receiving a request to reserve a specific amount of memory. Next, it acquires the memory in a sequential linked list based on the state of the portal. Then , by using the underlying methodology, it navigates the memory and attempts to assign the user a part of that memory while saving the state of the portal in order to release it later on. If the attempt fails, the algorithm will check if it is possible to switch the portal state by checking surrounding memory locations around the portal. Assuming memory is free around the portal, the algorithm will switch the state of the portal and acquires the memory in a sequential linked list again but with the different state and attempts to assign the user some memory. Otherwise, the user is blocked until some memory is released. Fig. 4 shows a flow chart of the methodology.

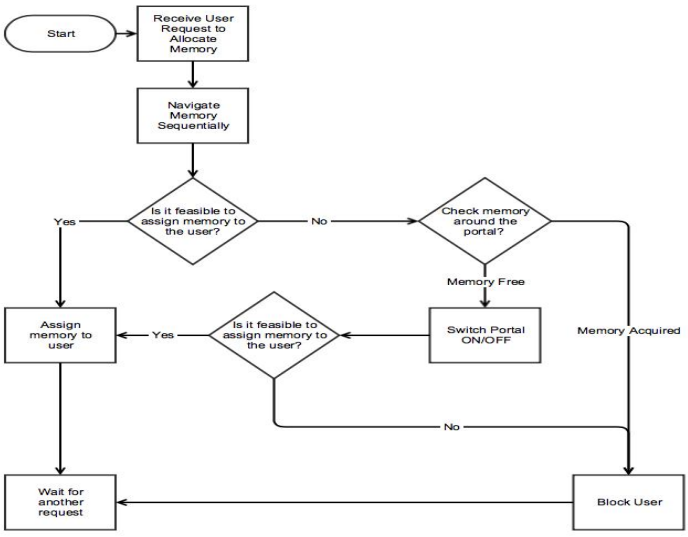

Fig. 4 Architecture Overview of the Proposed Memory Allocation Algorithm

B. Portals Placement

The key issue in the algorithm is where to place the portals in the memory? There are many ways to place the entrance and exit portals in the memory. We argue that in order to effectively benefit from the portals, certain conditions must be met. First condition is that the entrance portal must be place before the exit portal in order reassign parts of the memory correctly. This idea rises from the concept of a wormhole. The other condition is that the distance between portals have to be equal.

Since we are assuming one portal to be used here, we believe that placing the entrance portal after 25% from the beginning of the memory and placing the exit portal 25% before the end of the memory will give us optimal results. Fig. 5 shows an example of where the entrance and exit portals should be placed in memory given one or two portals. Since we are using one portal, we will use the memory arrangement shown in Fig. 5 Part A.

A	B	C	D
25%	20%	20%	20%
Entrance	Entrance 1	Entrance 1	Entrance 1
50%	20%	20%	20%
Exit	Entrance 2	Exit 1	Entrance 2
25%	20%	20%	20%
	Exit 1	Entrance 2	Exit 2
	20%	20%	20%
	Exit 2	Exit 2	Exit 1
	20%	20%	20%

Fig. 5 Shows different arrangement of the memory with different portal assignments. Part A shows only one portal in use while Parts B, C, D shows the possible portals placement in memory.

V. IMPLEMENTATION

In order to simulate the proposed scheme, Java Object-Oriented Programming Language was used to write a custom simulator. The generated data by the simulator was fed into Microsoft Excel to represent and plot the data in graphical shape.

We have assumed that the arrival of requests to our system in Section 3 to be followed through the Poisson distribution with the arrival rate lambda λ. Similarly, we stated that the dwell time of these requests follow the exponential distribution. This is an example of the M/M/1 system in queuing theory. To implement this, we exploit the special case of the Poisson distribution where the inter arrival time is memory less therefore following an exponential distribution. Now using only the exponential distribution; we can simulate the arrivals of requests by using the inter-arrival time and dwell time of those requests. In order to simulate this system with two variables, we have assumed that one of the variable "the dwell time of requests" to be 1-time unit under different arrivals rates. Fig. 6 shows the flow chart of the system.

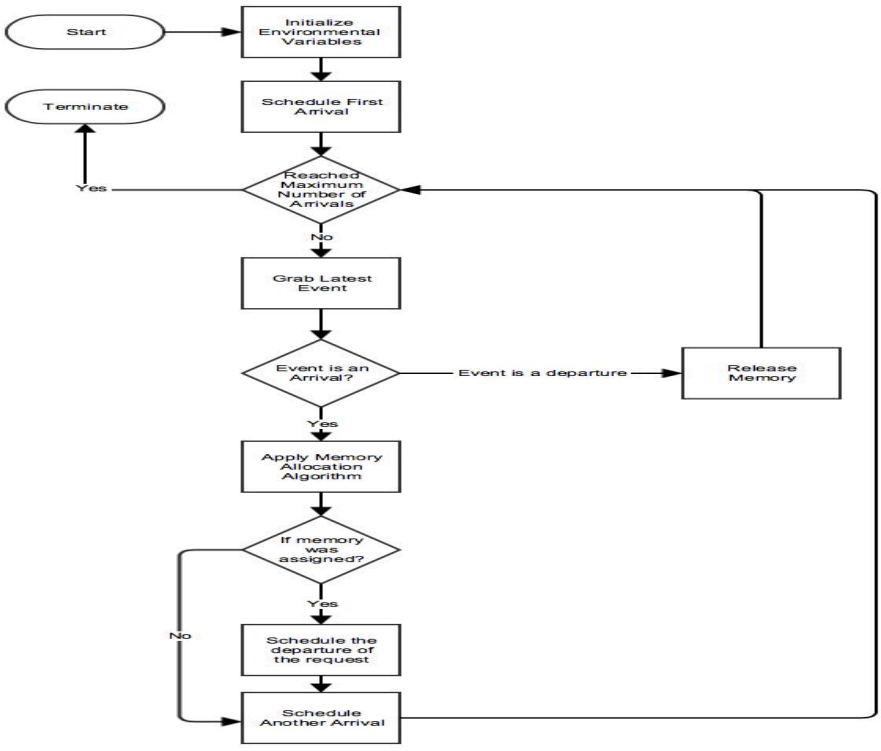

Fig. 6: Overall Flow Chart of the System

VI. RESULTS

The proposed memory allocation technique was simulated with the system previously discussed under three different environments. The first environment assumes a cloud where users select the different instances uniformly. Fig. 7 illustrates that our proposed scheme under this environment performed equally with the First-Fit and Best-Fit schemes and better than the Random scheme. The second environment under-take a scenario where cloud services are selected exponentially. This means that the lower instances such as the t2.micro instance has a higher chance than of being selected than the r3.8xlarge instance. The results shown in Fig. 8 shows that the proposed portal scheme performed slightly better than the other mentioned schemes. The final environment was tested on a cloud where users have a higher chance of selecting instances such as m4.xlarge & m4.2xlarge instead of t2.micro & r3.8xlarge thus leading to a normally distributed cloud. Fig. 9 shows that our proposed scheme performs slightly better than the other mentioned schemes. Using students t-distribution, we are 95% confidence of our results.

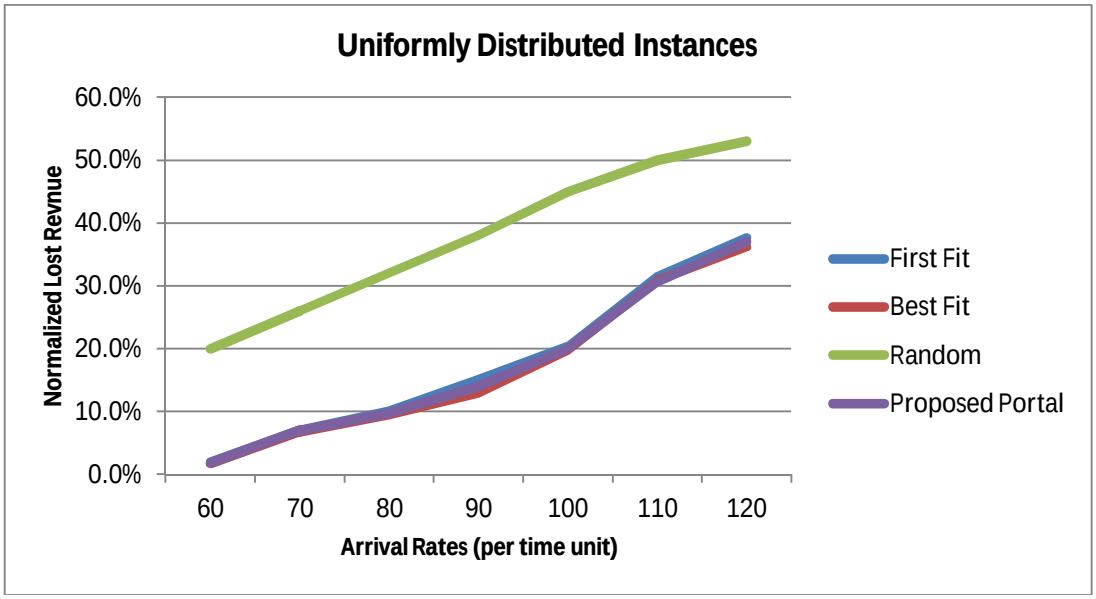

Fig. 7: Amazon Instances are selected based on the Uniformly Distribution

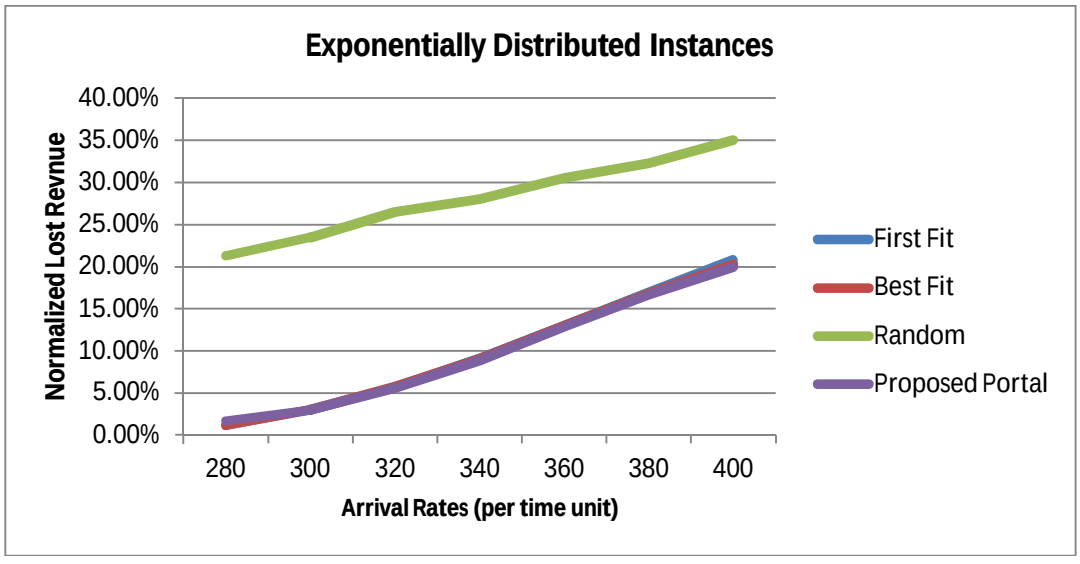

Fig. 8 Amazon Instances are selected based on the Exponential Distribution

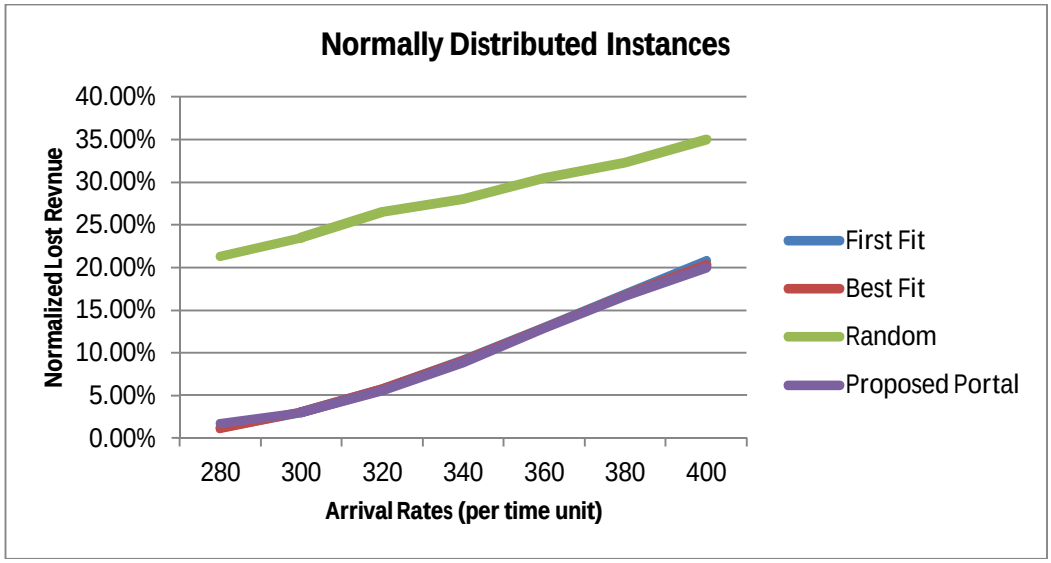

Fig. 9 Amazon Instances are selected based on the Normal Distribution (Bell-Shaped)

VII. CONCLUSION & FUTURE WORK

In this paper, we have proposed a new memory allocation algorithm in order to improve the performance of computer systems in cloud services. Additionally, we have simulated our proposed methodology and obtained some promising results compared with other widespread scheme. Future works will explore the potentials of adding a multiple portals and the different effects on the system, other works will explore a more improved version of the proposed scheme where the algorithm have some kind of intelligence in order to differentiate between different tasks.

REFERENCES

[1] Susanne Albers and Michael Mitzenmacher. *Average-case analyses of first fit and random fit bin packing*. In SODA, volume 98, pages 290–299, 1998.
[2] Mohit Aron, Peter Druschel, and Willy Zwaenepoel. *Cluster reserves: a mechanism for resource management in cluster-based network servers*. In ACM SIGMETRICS Performance Evaluation Review, volume 28, pages 90–101. ACM, 2000.
[3] John E Shore. *On the external storage fragmentation produced by first-fit and best-fit allocation strategies*. Communications of the ACM, 18(8):433–440, 1975.
[4] Mark Stillwell, David Schanzenbach, Fr´ed´eric Vivien, and Henri Casanova. *Resource allocation algorithms for virtualized service hosting platforms*. Journal of Parallel and distributed Computing, 70(9):962–974, 2010.
[5] Stefan Costea, Marian Barbu, Constantin Muraru, and Razvan Rughinis. *Resource allocation heuristics for the miriapod platform*. In Networking in Education and Research, 2013 RoEduNet International Conference 12th Edition, pages 1–6. IEEE, 2013.
[6] Imtiaz Ahmad Anwar Alyatama and Naelah Al-Dabbous. *Memory allocation algorithm for cloud service environments*. The Computer Journal
[7] Manal F Younis. *Memory allocation technique for segregated free list based on genetic algorithm*.
[8] Xiaofei Liao, Hai Jin, Shizhan Yu, and Yu Zhang. *A novel memory allocation scheme for memory energy reduction in virtualization environment.* Journal of Computer and System Sciences, 81(1):3–15, 2015.
[9] Diego Elias, Rivalino Matias, Marcia Fernandes, and Lucio Borges. *Experimental and theoretical analyses of memory allocation algorithms*. In Proceedings of the 29th Annual ACM Symposium on Applied Computing, pages 1545–1546. ACM, 2014.
[10] Alma Husagic Selman, Ali Aburas, and Suvad Selman. *Intelligent memory allocation based on fuzzy logic*. SouthEast Europe Journal of Soft Computing, 3(1), 2014.
[11] Chung, Y., & Moon, S. M. (2000, October). *Memory allocation with lazy fits*. In ACM SIGPLAN Notices (Vol. 36, No. 1, pp. 65-70). ACM.
[12] Hasan, Y., & Chang, M. (2005). *A study of best-fit memory allocators*. Computer Languages, Systems & Structures, 31(1), 35-48.
[13] Agmon Ben-Yehuda, O., Posener, E., Ben-Yehuda, M., Schuster, A., & Mu'alem, A. (2014). *Ginseng: Market-driven memory allocation*. ACM SIGPLAN Notices, 49(7), 41-52.

[14] Ried, S., Kisker, H., Matzke, P., Bartels, A., and Lisserman, M. (2011) Sizing the cloud—Understanding and quantifying the future of cloud computing. Technical Report, Forrester Research, Cambridge, MA . http://www.forrester.com/Sizing+The+Cloud/ fulletxt/E-RES58161.
[15] Manvi, S. and Shyam, G. (2014) Resource management for infrastructure as a service (IaaS) in cloud computing: a survey," Journal of Network and Computer Applications, 41, pp. 424-440.
[16] Ben-Yehuda, O., Ben-Yehuda, M., Schuster, A., and Tsafrir, D. (2014) The Rise of RaaS: the Resource-as- a-Service cloud," Communications of the ACM, 57, pp. 76-84.
[17] Sun, X., Wang, J., and Chen, X. (2007) An improve- ment of TLSF algorithm. Real-Time Conference, 15th IEEE-NPSS, 1, Batavia, IL.
[18] Masmano, M., Ripoll, I., Crespo, A., and Real, J. (2004) TLSF: a new dynamic memory allocator for real- time systems.16th Euromicro Conference on Real-Time Systems ECRTS, pp. 79-88.
[19] Peterson, J. and Norman, T. (1977) Buddy systems. Communications of the ACM, 20, pp. 421-431.
[20] Amazon Web Services Instance types matrix, Ama- zon EC2 instances, http://aws.amazon.com/ec2/ instance-types (August, 2014).

