Scheduling Cloud Workloads Using Carry-On Weighted Round Robin

Olasupo Ajayi(✉), Florence Oladeji, Charles Uwadia, and Afolorunsho Omosowun

Department of Computer Sciences, University of Lagos, Lagos, Nigeria
{olaajayi, foladeji, couwadia}@unilag.edu.ng,
afolorunshoi51@gmail.com

Abstract. Cloud Computing represents a paradigm shift in computing. It advocates the use of computing resources as a service rather than as a product. The numerous advantages which the Cloud offers has led to many users adopting it at a phenomenal rate. Providing service to this ever growing number of users in a fast and effective manner is a major challenge. Numerous researchers have proposed various approaches to scheduling user workloads, notable among which are the First-Come-First-Serve and Weight Round Robin (WRR), and have obtained varied levels of successes. Unfairness and excess allocation delay are some of the shortcomings of these approach. There is also the assumption that all Cloud users' workloads belong to a single class of requirement. This work proposes an efficient and fair Cloud workload scheduling algorithm called Adaptive Carry-On Weighted Round Robin (ACWRR), and also takes into consideration multiple workloads classes. Experimental simulations were conducted with ACWRR benchmarked against WRR. Results show that ACWRR performs better than WRR by at least 13% in terms of system latency and 38% for makespan.

Keywords: Cloud computing · Multi-queues · Queuing · Scheduling Weighted Round Robin

1 Introduction

Cloud Computing is a model for enabling ubiquitous and on-demand access to shared pool of computing resource [1]. These resources are made available to customers on a pay-per-use model, similar to utilities such as electricity and water supply. Cloud data centres are building where a large number of inter-connected servers are operated. Each server hosts a number of virtual servers on which user requests are deployed. Cloud computing is being adapted at a phenomenal rate, as recent reports have shown that there are over 140 million active Cloud users [2] and are growing at a rate of about 17% annually [3]. With this huge number of users allocation of requests to Cloud servers must be done efficiently. In [4], the authors grouped Cloud users into three group – Gold, Silver and Bronze and prioritized allocation of their requests to Cloud servers. The Gold class enjoyed highest priority and least delay however the Silver and particularly the Bronze class users were not treated fairly and suffered long delays.

V. Odumuyiwa et al. (Eds.): AFRICOMM 2017, LNICST 250, pp. 60–71, 2018.
https://doi.org/10.1007/978-3-319-98827-6_5

This work therefore proposes a user request allocation scheme that takes into consideration multiple user classes and allocated requests to servers in a fair manner. In the remaining sections of this paper, the terms Cloud servers and Physical Machines (PMs) are used interchangeably as well as the terms user request and user workload. The rest of this paper is organized as follows: related works on workload allocation in Cloud computing are discussed in Sect. 2. A detailed description of the proposed approach is given in Sect. 3. Experimental setup and performance analysis are done in Sect. 4. Sections 5 and 6 respectively presents the contributions and conclusion of this paper.

2 Related Works

Round Robin [5] though mostly suited for static environments has been proposed by numerous authors as a technique for workload allocation in Cloud Computing. It is a circular variant of First-Come-First-Serve (FCFS) allocation scheme, wherein workloads are allocated resources in time sharing manner. However, being a static approach, it does not take into consideration the status or current workload of the PMs into consideration when allocating workloads.

Round Robin with Server Affinity (RRSA) was proposed by [6]. The work introduced a variant of round-robin, which allocates workloads to PMs with a view of keeping a workload balance amongst the PMs. RRSA distributes workloads using the conventional round robin algorithm but introduces hash map and PM state list which store information about the last PM allocations and the current state of PMs respectively. Experimental results showed that when compared to the classic Round Robin, the response time improved as the number of data centres increased however processing time was only marginally better. There is also the high possibility of dip in performance during the process of searching the hash map.

In [7] an efficient version of the throttled allocation scheme was presented. Throttled allocation is a scheme that allocated workload to VMs with a view of keeping a balanced workload distribution amongst VMs. Prior to workload allocation, the scheduler checks its index-table for the status of all VMs. If a suitable VM is found for the workload, it is immediately allocated to the VM, otherwise it is enqueued. In the work, the proposed Efficient Throttled Algorithm (ETA) addressed the long execution time experienced by workloads when the using the classic throttled algorithm. The authors compared the performance of ETA with Round Robin, classic throttled and Equal Spread Current Execution algorithms and reported improvements in average processing time and overall response time. However the proposed approach was only minutely better than the classic throttled and at par with the two other algorithms on all metrics. The Round Robin and throttled algorithms are two of the default algorithms included in CloudAnalyst [8] - a popular Cloud research and simulation toolkit based on CloudSim.

Join Idle Queue was proposed in [9] as an approach that combines distributed dispatcher with Idle Queues (IQ). Each dispatcher has an IQ onto which idle PMs queue. On arrival of a task, the dispatcher checks its IQ for idle PM(s). If found, it dispatches the task to the first PM on the IQ else it is randomly dispatched to any PM. Idle PMs also enqueue themselves onto a dispatcher's IQ randomly or based on the shortest IQ. Though effective, the approach does not take workload imbalanced between busy PMs into consideration. In [10], the author analysed the performance of JIQ [9] using fluid limit approach and showed that it is advantageous for a PM to enqueue itself on a dispatchers IQ, while still actively servicing user requests. The use of early threshold was also recommended.

In [11, 12], a linear workload allocation scheme called based on Best Fit Descending (BFD) was proposed. The scheme, sorts all PMs in descending order of available resources then searches through the list of PMs for one which best matches the user workload requirements (VM). Though an efficient allocation scheme in terms of resource utilization, the linear search process used can increase allocation time especially in data centres with large number of PMs and user workloads with multiple resource requirements.

All the above reviewed works considered single user workload queue, implying that it is assumed that all user workloads belong to a single class of requirement. This is not true in reality, as factors such as users' purchasing power and socio-economic status play a significant role in daily lives and can therefore not be ignored. The authors in [4] therefore considered user workload classes and used these classes for workload allocation. The introduction of the Double-Depth Half Interval Scheduling (2DHIS) as a replacement to the BD scheme used in [11, 12], allowed the proposed algorithm finds suitable PMs for user workloads faster thus reducing allocation delay time. However, similar to [11, 12], a First-Come-First-Serve (FCFS) approach was also used for workload scheduling. Also the authors primarily focused on workloads within the Gold class with highest priority, thereby resulting in increased delay times for workloads of lower priority classes.

The authors in [13] introduced a modified Weighted Round Robin (WRR) [14], called Carry-on Weighted Round Robin (cWRR). It is a multi-queue scheduling algorithm like WRR but unlike WRR, in this work whenever a queue has less packets than its allocated service quantum, the excess quantum is carried over to the next active queue rather than being ignored as in the case of WRR. Unlike Deficit Round Robin (DRR) [15], the extra quantum is used in the current round, and not in the subsequent round. This in effect allows cWRR to better utilize the available bandwidth per round. Also the fact that cWRR refreshes each queues' service quantum ever round makes it a fair scheduling algorithm.

This paper adapts the cWRR by applying it as a workload scheduling algorithm for Cloud environments with multiple workload classes. The new scheme, simply called adapted-cWRR or ACWRR improves on the work of [4] with the introduction of cWRR which is able to schedule multiple user workload classes in an efficient and fair manner to all classes.

3 Proposed Approach

In this section, the Adapted-Carry-On Weighted Round Robin (ACWRR) is presented. ACWRR groups users' workloads into three (3) classes of Gold, Silver and Bronze, similar to [4]. However, rather than the first-come-first-serve used, a cWRR based scheduler is used to allocate user workloads to VMs/PMs in the Cloud Data centre (DC).

The ACWRR model shown in Fig. 1 can be broken into the following steps below, while a logical flow is depicted in Fig. 2.

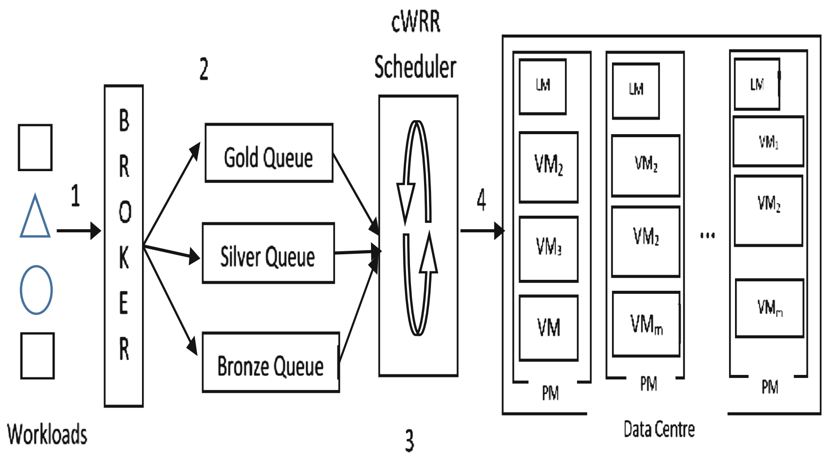

Fig. 1. The Adapted-Carry-On Weighted Round Robin (ACWRR) model

1. Workloads are submitted by users.
2. The workloads are grouped into three classes.
3. CWRR is used to select workload to schedule, this is shown in Algorithm 1.
4. Selected work are allocated to resources within the DC for processing.

3.1 Algorithm 1: CWRR

```
1. Initialization:
     a. ActiveList = NULL
     b. Bonus = 0
     c. Enqueuing module: on arrival of queue o
     d. InsertWorkload(Qi)
     e. i= ExtractFlow(p)
2. If(ExistsinActiveList(i) = = FALSE) then
     a. AppendToActiveList(i) // Create scheduler
        queue for flow i
     b. SetWeight(i)
3. Dequeuing module:
     a. While (TRUE){
     b. If NotEmpty(ActiveList) q = getqueueindex ()
     c. if(ExistinActiveList(q))      i = getQueue(q)
     d. Qq =Qq+Bonus  }
4. While ((Qq >  0) and Not Empty(i)){  // Process a
   workload from queue i
     a. p = Dequeue(q)
     b. Qq = Qq - 1                        }
5. If(Qq >= 0 And Empty(i)){
     a. RemoveinActiveList(i)
     b. Bonus=Qi
     c. Qi=0    }
6. If (Qi= 0 and Empty(i))    AppendToActiveList(q)
          }
```

3.2 System Process Flow

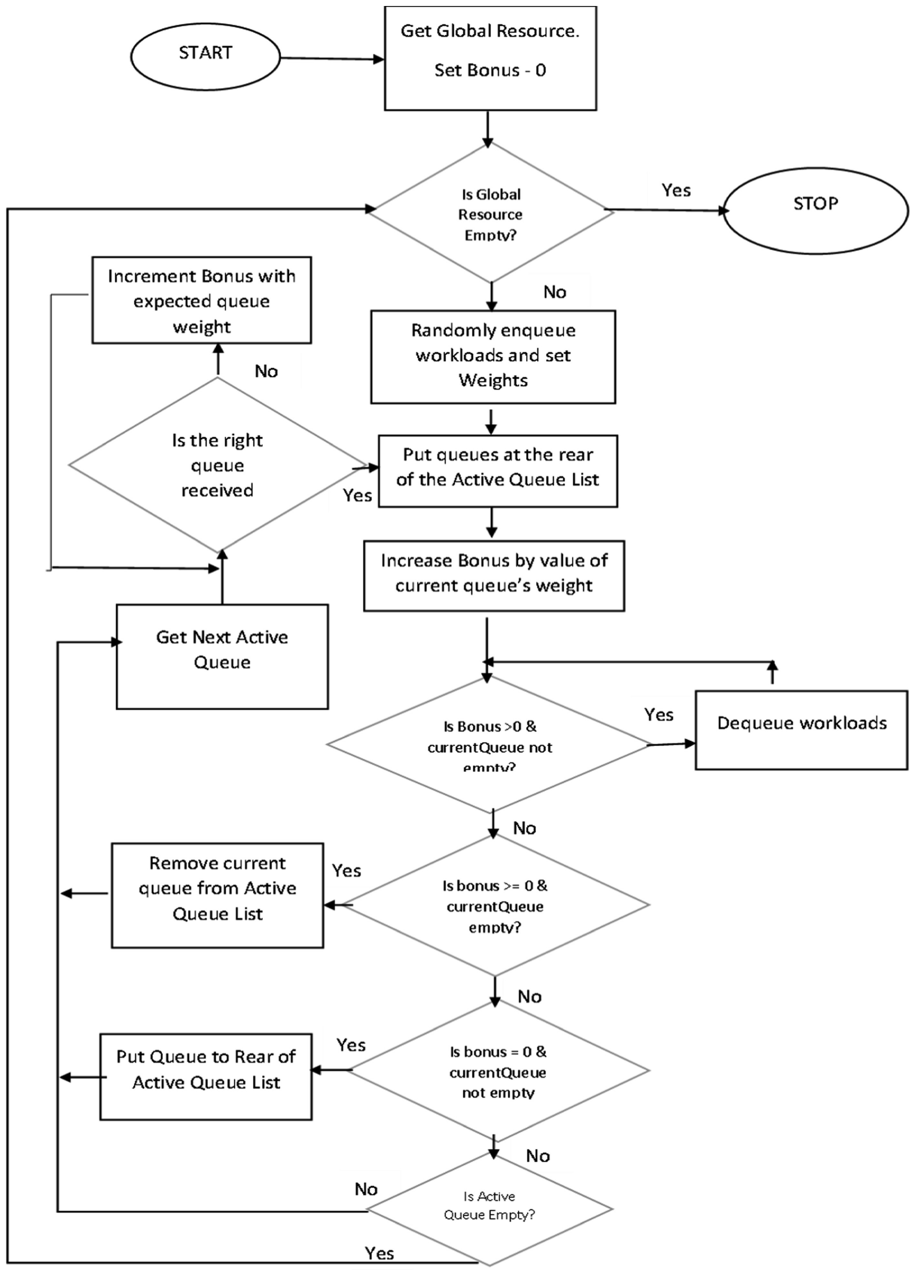
START
Get Global Resource.
Set Bonus - 0
Is Global Resource Empty?
Yes
STOP
No
Randomly enqueue workloads and set Weights
Put queues at the rear of the Active Queue List
Increase Bonus by value of current queue's weight
Increment Bonus with expected queue weight
No
Is the right queue received
Yes
Get Next Active Queue
Is Bonus >0 & currentQueue not empty?
Yes
Dequeue workloads
No
Remove current queue from Active Queue List
Yes
Is bonus >= 0 & currentQueue empty?
No
Put Queue to Rear of Active Queue List
Yes
Is bonus = 0 & currentQueue not empty
No
No
Is Active Queue Empty?
Yes

Fig. 2. ACWRR system flow chart

4 Performance Evaluation

In evaluating the efficacy of the proposed ACWRR model, simulated experiments were conducted using CloudSim [16]. The experimental setup used consist of a DC with 800 host machines of with two different processing powers in Millions of Instructions Per Second (MIPS) - 1,860 MIPS and 2,660 MIPS. Real workload traces from PlanetLab day 3 [17] were used, and consisted of 1078 workloads. The ACWRR model was benchmarked against the conventional WRR using the following metrics:

1. Average workload delay per queue – which is a measure of the time spent by a user workload on any given queue waiting to be scheduled; and given in (1).

$$\frac{\sum_i^n (D_i - A_i)}{n}. \tag{1}$$

   Where D_i is departure time of a workload i, A_i is arrival time of a workload i, and n is number of workloads on a queue.

2. Makespan per queue – a measure of the time it takes to schedule the last workload on a queue.

$$D_l - D_o. \tag{2}$$

   In (2), D_l is the departure time of the last workload on a queue, and D_o is the departure time of the first workload on the queue.

3. Average overall workload delay – average time spent by a workload across all queues. It is given in (3)

$$m * \sum_k^m \frac{\sum_i^n \left(D_{i,k} - A_{i,k}\right)}{n}. \tag{3}$$

   Where $D_{i,k}$ is departure time of a workload i on a queue k, $A_{i,k}$ is arrival time of a workload i on a queue k, n is number of workloads on queue k, and m is the number of queues.

4. Latency per queue – inverse of the number of workloads scheduled on a queue per unit time and given in (4)

$$\left\{\frac{\sum w_q}{T_q}\right\}^{-1}. \tag{4}$$

   Where w is number of workloads on a queue q and T is time taken to schedule all workloads on q.

5. Average system throughput – inverse of the average number of workloads scheduled per unit time across all queues.

$$\left\{ Q * \sum_{q}^{Q} \frac{\sum w_q}{T_q} \right\}^{-1}. \tag{5}$$

In (5), Q is the total number of queues, w_q and T_q are the workloads and allocation time on queue q.

5 Results

Implementation was in two phases. At the first phase, workloads were enqueued onto the three queues, while in the second phase, the enqueued workloads were selected/dequeued for allocation using one of three arrival schemes viz. random, equal and double. Using the random scheme, workloads were enqueued onto randomly selected queues. With the equal scheme, an even proportion of workloads to be dequeued were enqueued, such that at each iteration the arrival rate of workloads equal the departure rate. Finally with the double, departure rate is twice the arrival rate. For each of these schemes, a dequeueing ratio of 5:3:2 was maintained for all queues.

In Fig. 3, the latency (inverse of throughput) of ACWRR and WRR are depicted. In this graph, for all queues, ACWRR outperformed WRR by at least 13.8% and implies that ACWRR is able to schedule at least 38.7% more workloads on the average than WRR.

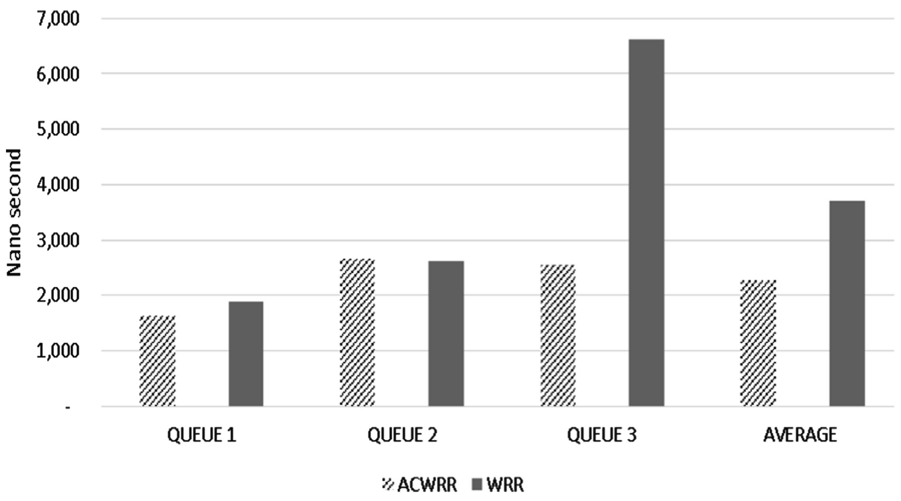

Fig. 3. Latency of ACWRR vs. WRR using random enqueue scheme

A comparison of the makepan is shown in Fig. 4 and though WRR is shown to have lower makespan for all three queues versus ACWRR, the values vary across these queues. For queue 1, WRR results in 30,420 ns, while ACWRR gave 37,030 ns. For queue 2, makespans of 33,058 ns and 36,839 ns were recorded for WRR and ACWRR

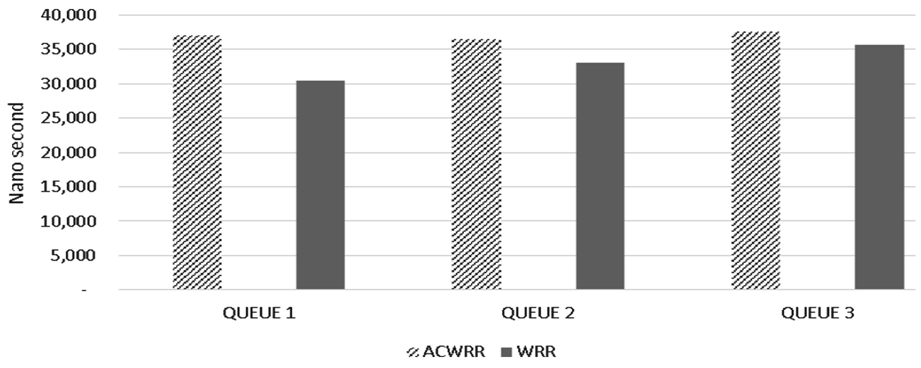

Fig. 4. Makespan of ACWRR vs. WRR using random enqueue scheme

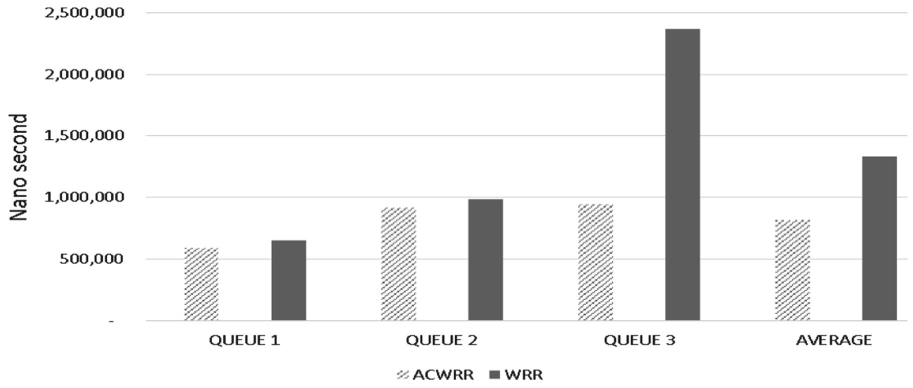

Fig. 5. Scheduling delay of ACWRR vs. WRR using random enqueue scheme

respectively. While for queue 3, WRR resulted in 35,672 ns versus ACWRR at 37,563 ns. These results show that the dequeuing ratio largely influences the makespan when WRR is used, resulting in queue 1 (with highest the dequeue ratio) having shortest makespan and queue 3 (with lowest dequeue ratio) having the longest makespan. However, when ACWRR is used, despite the dequeuing ratio (5:3:2), all queues experiences similar makespans, thus making ACWRR fairer to all its queues versus WRR.

Figure 5 shows the waiting time or delay of workloads in a queue. For queue 1, which has the highest dequeue ratio, workloads experience 10.5% less delay with ACWRR versus WRR. For the second queue, workload delays improved by 6.8% when ACWRR is used, while for the last queue, ACWRR resulted in significantly less waiting time with an improvement of 150.9% versus WRR. On the average, ACWRR resulted in 38.7% lower workload delay time than WRR.

Due to the unpredictability of the random allocation scheme and to have a fair benchmarking assessment, the equal and double schemes were also used. When using the equal scheme (equal arrival and departure times), WRR resulted in an average of

4.5% improvement over ACWRR with respect to latency and workload delays. However, it must be noted that when arrival rate equals departure rate, no queues would be formed hence queue management algorithms such as ACWRR would not be needed. For this reasons, these results are not shown, except for Makespan which is a measure of the fairness of a scheme to its queues. This is shown in Fig. 6.

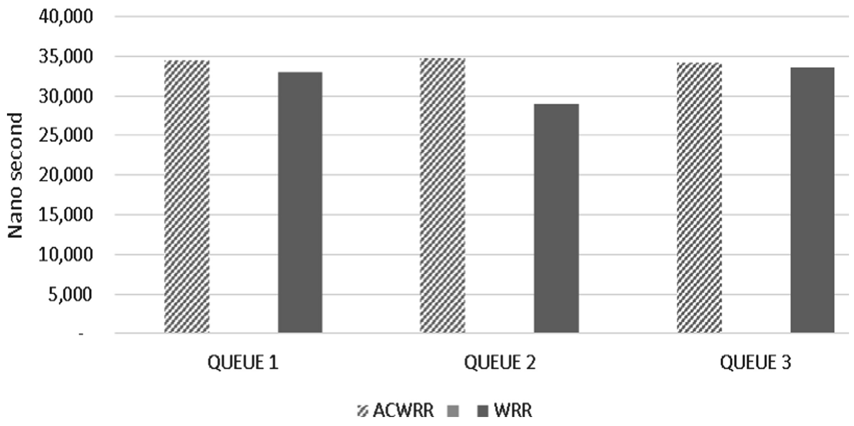

Fig. 6. Makespan of ACWRR vs. WRR using equal enqueue scheme

In Fig. 6, ACWRR is again shown to be fairer on all its queues versus WRR. For queue 1, 34,453 ns and 32,989 ns were reported for ACWRR and WRR respectively. For queue 2, makespans of 34,790 ns and 29,014 ns were reported for ACWRR and WRR respectively. For queue 3, 34,202 ns and 33,588 ns, were the makespans for ACWRR and WRR. As with the random enqueuing scheme, ACWRR is fairer to all its queues versus WRR.

In order to further ascertain the results, a last test was conducted using the double enqueue scheme, wherein, arrival rate is half the departure rate. In Fig. 7, the latency of ACWRR and WRR when using the double enqueue scheme are compared. The results show that under the same circumstances (enqueuing rate), ACWRR has significantly better latency (inverse of throughput) than WRR. On the average, ACWRR has a latency of 91 versus WRR's 1977.

Figure 8, shows the makespan of ACWRR versus WRR. For all measurements, lower values are desirable. Across all queues, and consistent with the other enqueuing schemes, ACWRR results in an even makespan across all queues. ACWRR results in makespans of 73,755 ns, 73,466 ns and 71,400 ns for queues 1, 2 and 3 respectively. Similarly WRR also results in an even spread of makespan across all queues at 353,479 ns, 353,322 ns and 352,875 ns.

Finally, and in tandem with the latency and makespan, Fig. 9, shows that ACWRR schedules workloads significantly faster than WRR across all queues, with an average delay of 66,968 ns versus 1,449,857 ns for ACWRR and WRR respectively. This implies that using ACWRR, workloads experience shorted delays than with WRR.

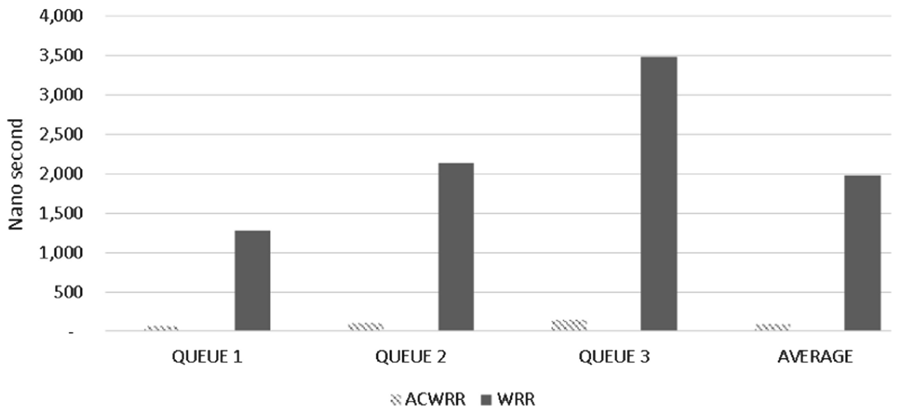

Fig. 7. Latency of ACWRR vs. WRR using double enqueue scheme

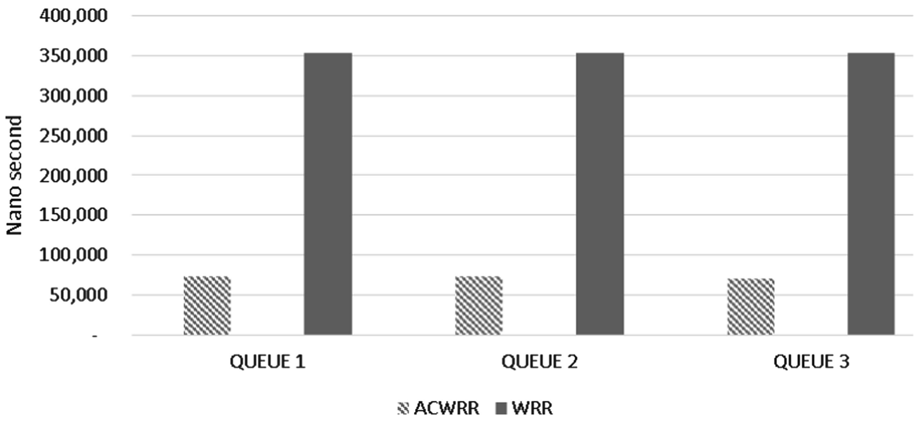

Fig. 8. Makespan of ACWRR vs. WRR using double enqueue scheme

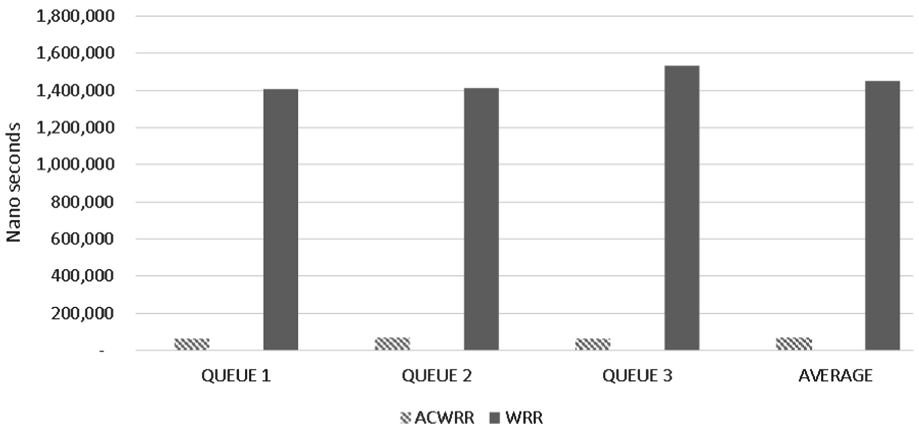

Fig. 9. Scheduling delay of ACWRR vs. WRR using double enqueue scheme

6 Contributions

The main contribution of this paper is in the development of a workload scheduling scheme for Cloud computing called ACWRR that:

a. Takes multiple user workload classes into consideration when scheduling.
b. Is fair to all queues by ensuring all queues experience similar makespan irrespective of their priority.
c. Reduces the delays experienced by workload waiting on queues to be scheduled allocation.

7 Conclusion

In this paper, a workload scheduling scheme for Cloud Computing was developed. The scheme called ACWRR takes multiple workload classes into consideration and schedules workloads fairly across all the queues. It ensures that all queues experience similar makespan irrespective of their priority. The developed scheme was benchmarked against the well-known Weighted Round Robin scheme and it performed significantly better with respect to workload scheduling delays and makespan. The developed scheme can be applied to Cloud environment wherein prioritization or service differentiation is in use.

In future, the ACWRR scheme would be tested against the FCFS scheme used by numerous authors in order to test its performance with respect to workload delay and makespan. Also a test of violation of SLA as a result of queuing delay would be considered.

References

1. Mell, P., Grance, T.: The NIST Definition of Cloud Computing. NIST Special Publication 800-145. Whitepaper, NIST (2011). http://acm.org/citation.cfm?id=2206223
2. Cisco Inc.: Cisco Global Cloud Index: Forecast and Methodology, 2015–2020. Whitepaper, Cisco (2016). http://www.cisco.com
3. Gartner: Worldwide Public Cloud Services Market to Grow 17 Percent in 2016, Press Release, Gartner (2016). http://www.gartner.com/newsroom/id/3443517
4. Ajayi, O., Oladeji, F., Uwadia, C.: Multi-class load balancing for QoS and energy conservation in cloud computing. West Afr. J. Ind. Acad. Res. **17**, 28–36 (2016)
5. Sotomayor, B., Montero, R., Llorente, I., Foster, I.: Virtual infrastructure management in private and hybrid clouds. IEEE Internet Comput. **13**(5), 14–22 (2009). IEEE
6. Mahajan, K., Makroo, A., Dahiya, D.: Round robin with server affinity: a VM load balancing algorithm for cloud based infrastructure. J. Inf. Process Syst. **9**(3), 379–394 (2013)
7. Patel, D., Rajawat, A.: Efficient throttled load balancing algorithm in cloud environment. Int. J. Modern Trends Eng. Res. **2**(3), 463–480 (2015)

8. Wickremasinghe, B., Calheiros, R., Buyya, R.: Cloudanalyst: a cloudsim-based visual modeller for analysing cloud computing environments and applications. In: 2010 24th IEEE International Conference on Advanced Information Networking and Applications (AINA), pp. 446–452. IEEE (2010)
9. Lu, Y., Xie, G., Kliot, G., Geller, A., Larus, J., Greenberg, A.: Join-Idle-Queue: a novel load balancing algorithm for dynamically scalable web services. Perform. Eval. **68**(11), 1056–1071 (2011)
10. Mitzenmacher, M.: Analyzing distributed Join-Idle-Queue: a fluid limit approach. In: 2016 54th Annual Allerton Conference on Communication, Control and Computing (Allerton), pp. 312–318. IEEE (2016)
11. Beloglazov, A., Buyya, R.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers. Concurrency Comput.: Pract. Experience **24**(13), 1397–1420 (2012)
12. Farahnakian, F., Pahikkala, T., Liljeberg, P., Plosila, J., Hieu, N., Tenhunen, H.: Energy-aware VM consolidation in cloud data centers using utilization prediction model. IEEE Trans. Cloud Comput. 13 (2016). http://ieeexplore.ieee.org/document/7593250/
13. Oladeji, F., Oyetunji, M., Okunoye, O.: CWRR: a scheduling algorithm for maximizing performance of quality of service network router. Int. J. Comput. Appl. **41**(2), 30–34 (2012)
14. Shimonishi, H., Yoshida, M., Fan, R., Suzuki, H.: An improvement of weighted round robin cell scheduling in ATM networks. In: Global Telecommunications Conference, GLOBECOM 1997, vol. 2, pp. 1119–1123. IEEE (1997)
15. Shreedar, M., Varghese, G.: Efficient fair queuing using deficit round robin. IEEE/ACM Trans. Netw. **4**(3), 375–385 (1996). ACM
16. Beloglazov, A., Buyya, R.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurrency Comput.: Pract. Experience (CCPE) **24**(13), 1397–1420 (2012)
17. Park, K., Pai, V.: CoMon: a mostly-scalable monitoring system for PlanetLab. ACM SIGOPS Oper. Syst. Rev. **40**(1), 65–74 (2006). ACM