Journal of Applied Research and Technology

www.jart.icat.unam.mx

ICAT Instituto de Ciencias Aplicadas y Tecnología

Journal of Applied Research and Technology 21 (2023) 256-264

Original

Dispatcher I-queue optimization in Join-Idle-Queue algorithm for improved load balancing in cloud computing

P. Goswami[a] • S. Narang[b]* • A. Jain[c]

[a]*SRM University, Sonepat, Haryana, India*
[b]*Ph. D. Research Scholar, SRM University, Sonepat, Haryana, India*
[c]*University of Petroleum & Energy Studies (UPES), Dehradun, Uttarakhand, India*

Received 05 24 2021; accepted 09 20 2021
Available 04 30 2023

Abstract: Objectives: Modern data centers serving web and mobile applications employ distributed load balancers. The Join Idle Queue (JIQ) algorithm is ideally suited for load balancing in a distributed setup. It attains fast response time by directing service requests to idle servers which can immediately process them. However, JIQ is not optimal in tracking idle servers leaving room for improvement. Methods: We observed that JIQ assigns idle servers non-uniformly to load balancers leaving some load balancers with no access to idle servers. We propose a variant of the JIQ algorithm, Join Idle Queue dispatcher I-queue Optimization (JIQ-DIO), which increases the probability of load balancers having access to idle servers without additional communication overhead leading to improved response time. Findings: We simulated JIQ-DIO on CloudSim Plus 3.0 and compared it with standard JIQ and its different variants. JIQ-DIO was found to increase the probability of incoming requests being directed to idle servers and lead to more than two-fold improvement in response time across a broad range of parameters.

Keywords: Cloud computing, Load balancing, Task scheduling, Join Idle Queue, Response Time, Virtual Machines

*Corresponding author.
E-mail address: sudhanarang@gmail.com (P. Goswami).

1. Introduction

Servers hosted on-premises have traditionally used a centralized load balancer. However, as modern web and mobile applications move to the cloud, the use of multiple distributed load balancers over a server farm has become the preferred implementation as it is more robust, programmable, and cost-effective (Lu et al., 2011).

A centralized load balancer oversees the entire incoming and outgoing traffic of tasks and the assignment of tasks to servers; hence it can keep track of the task queue at each server. Whereas in a distributed setup each load balancer, called dispatcher, is only aware of the tasks that it handles; hence dispatchers cannot track the task queues at servers. Additional communication between the servers and dispatchers is required for the dispatchers to have information on server queues for load balancing. However, as servers and dispatchers increase in number, information exchange between them becomes voluminous and an expensive time overhead making it impractical for the dispatchers to keep track of all server task queues (Mitzenmacher, 2016).

The Join Idle Queue (JIQ) algorithm was proposed to attain fast response time for distributed dispatchers with low communication overhead (Mitzenmacher, 2016). It was shown to perform better than other load balancing algorithms for distributed setups (Silva Filho et al., 2017) and has become the standard. Central to JIQ is decoupling the process of task assignment by dispatchers from the registration of idle servers at the dispatchers. Instead of dispatchers tracking server queues, idle servers register themselves with dispatchers. To facilitate this, each dispatcher maintains a data structure called the I-queue which contains a list of idle servers registered at the dispatcher. When a server becomes idle, it registers itself with the I-queue of one of the dispatchers which is selected at random. In this study, we observed that in a typical JIQ scenario many dispatchers persist with zero- length I-queues. Since the servers are blind to the I-queues of dispatchers, idle servers registerthemselves with the I-queues of randomly selected dispatchers. The distribution of idle machines across dispatcher I-queues is not uniform and allows zero-length I-queues to persist in some dispatchers. Dispatchers with empty I-queues assign incoming tasks to randomly selected servers leading to suboptimal average response time. We hypothesized that if dispatchers can make servers aware of their idle queues at the time of task assignment, the servers can have partial information on dispatcher I-queues and when idle they can register themselves with one of the dispatchers that have an empty I-queue. This will help to mitigate the persistence of zero-length I-queues while keeping communication overhead low. As the probability of dispatchers finding idle servers increases, the average response times will improve. In this paper we propose an improvement to the JIQ algorithm called *Join Idle Queue Dispatcher I-queue Optimization* (JIQ-DIO) which implements this idea.

Related work including the JIQ algorithm, and its variants are described in Section 2. The JIQ- DIO algorithm is described in Section 3. Implementation and simulation of the JIQ-DIO algorithm in CloudSim Plus (Calheiros et al., 2011), a fork of popular tool CloudSim for simulating cloud computing infrastructures (Kunwar et al., 2018), is described in Section 4. The original JIQ algorithm and its other variants have also been simulated on the same platform for direct performance comparison with JIQ-DIO. Conclusions and future work are discussed in Section 5.

2. Related work

2.1. Load balancing with distributed dispatchers

Cloud computing delivers computing resources and services over the internet. An important aspect of research work in cloud computing is load balancing which refers to the optimal allocation of service requests to evenly balance the workload across multiple servers. Load balancing improves response time and throughput for the users and leads to optimal utilization of resources and low downtime at the servers. Traditionally load balancing is performed by a single dispatcher who makes all decisions on task allocation to servers. This classical problem of load balancing using a centralized dispatcher has been researched extensively (Khiyaita et al., 2012; Li, 2017) and analytical expressions for optimizing average response time, power consumption and cost- performance ratio in heterogeneous multi-server setups have been described (Rao et al.,2003).

However, lately the use of multiple distributed dispatchers over a server farm has been preferred over a centralized dispatcher. The scalability and performance advantages of decentralized load balancing were recognized in peer-to-peer file sharing systems (Gao & Min, 2009; Grosu & Chronopoulos, 2005; Yang & Garcia-Molina, 2003). Multiple heterogeneous servers distributing tasks among them using game theoretic algorithms in eithercooperative or non-cooperative fashion were found to be advantageous over centralized load balancing (Al-Fares et al., 2008; Duan et al., 2014). In data centers, multi-rooted tree architecture utilizing software algorithms andcommunication protocols for balancing data flows across a network was found to be superior to a centralized setup (Bharti & Pattanaik, 2013; Cheung & Leung, 2018; Harchol-Balter, 2021). Similarly, distributed load balancing was found to improve the resilience and scalability of data centers hosting cloud computing services (Badonnel &

Burgess, 2008; Ousterhout et al., 2013). Distributed load balancing can be achieved in many ways such as by servers sharing load with peers (Cardellini et al.,1999; Hong, Y et al.,2006), DNS based load balancing (Kingman, 1961), or by a strictly hierarchical architecture where dedicated load balancers called dispatchers direct requests to servers (Ousterhout et al., 2013).

This study considers cloud computing server farms having a hierarchical setup consisting of multiple distributed dispatchers and heterogeneous servers or virtual machines (VMs) as illustrated in Fig. 1. The server farm comprises of *p* physical servers hosting a total of $k_1 + k_2 + \cdots + k_p = K$ virtual machines (VMs). Service requests, or tasks, T1, T2, …, Tn are received by a router. Task arrival is modeled by a Poisson process and the processing times of tasks (or task lengths) are considered to be exponentially distribute (Narang et al., 2019). The router directs a service request to one of *m* dispatchers which assigns it to one of *K* VMs. The VMs maintain a task queue from which tasks are processed in a first-in-first-out (FIFO) order.

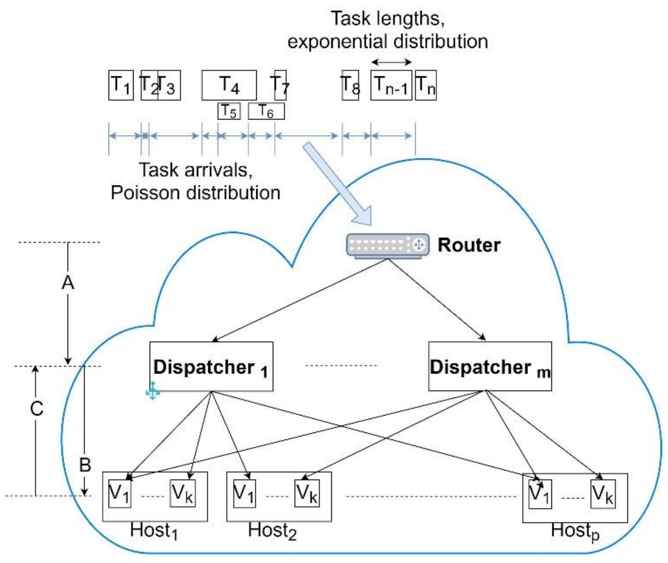

Figure 1. A cloud computing infrastructure with distributed dispatchers: (A) service requests (or tasks) are received by a router and directed to dispatchers, (B) the dispatchers assign tasks to VMs, and (C) the VMs send responses back to dispatchers.

In this scenario, a load balancing algorithm must address broadly three aspects:

A) Routing of a service request to a dispatcher, or *dispatcher selection*,

B) Assignment of a service request by a dispatcher to a VM, or *task allocation*, and

C) Communication between VMs and dispatchers for the *maintenance of task queues*.

2.2. A general model of Join Idle Queue

The Join Idle Queue algorithm addresses load balancing in the context of large data centers or server farms that maintain hundreds or thousands of servers managed by distributed dispatchers (Mitzenmacher, 2016). The large compute capacity ensures that any given time there are idle VMs available to immediately process incoming service requests. The problem of load balancing is then simplified to finding an available idle VM for an incoming service request. JIQ solves this problem with minimal communication overhead between the dispatchers and VMs. In a JIQ model each dispatcher locally maintains an Idle Queue or I-queue which stores a list of idle VMs that are registered with it. An idle VM may be registered with the I-queue of only one dispatcher at any time. When a task arrives, a dispatcher assigns it to an idle VM from its I-queue and removes the VM from its I-queue. Notably, instead of dispatchers attempting to find idle VMs to fill their I-queues, the idle VMs have the responsibility of registering themselves with the I-queue of a dispatcher. The different variants of JIQ present various strategies by which idle VMs select a dispatcher to register themselves with. For instance, the standard JIQ algorithm, also referred to as JIQ-Random, uses the following strategies for the three key aspects of load balancing:

A) Dispatcher selection: The router directs an incoming request to one of the dispatchers selected uniformly at random.

B) Task allocation: A dispatcher with non-empty I-queue selects an idle VM uniformly at random from its I-queue and then removes this VM from its I-queue. A dispatcher with an empty I-queue selects a VM uniformly at random among all VMs.

C) I-queue joining: When a VM becomes idle, it selects a dispatcher uniformly at random among all dispatchers and joins its I-queue.

2.3. Variants of JIQ

The standard JIQ or JIQ-Random uses a simple uniform-at-random strategy for dispatcher selection, task allocation as well as I-queue joining. This is easy to implement as there is no need to track the state of the system. However, its performance is not optimal. A few different variants of the JIQ algorithm have been proposed to improve performance (Lu, 2018). These are summarized in Table 1.

Table 1. Strategies used by different variants of JIQ algorithm: dispatcher selection refers to how the router chooses one among all dispatchers to which it directs an incoming request. Task allocation refers to how a dispatcher selects a VM to which it assigns the request. The selection strategy depends on whether or not the I-queue of the dispatcher is empty. I-queue joining refers to how an idle VM selects a dispatcher I-queue to register itself with. The random strategy implies choosing any one among the available alternatives uniformly at random. The SQ(d) strategy refers to randomly sampling any d out of all available alternatives and choosing the one with the shortest queue.

Algo	Strategy			
	Dispatcher Selection	Task Allocation		I-queue Joining
		Nonempty I-queue (Select from I-queue)	Empty I-queue (Select from all VMs)	
JIQ Random	Random	Random	Random	Random
JIQ-SQ(d)	Random	Random	Random	SQ(d), Shortest I queue
JIQ-PoD	Random	Random	SQ(d)	Random
JIQ-NE	SQ(d)	Random	Random	Random
JIQ-E	Random	Random	Random	SQ(d)

JIQ-SQ(d) makes a smarter choice in the I-queue joining strategy compared to JIQ-Random:

JIQ-SQ(d) I-queue joining: When a VM becomes idle, it chooses *d* dispatchers at random and among those registers itself with the dispatcher which has the shortest I-queue length.

JIQ-PoD, power-of-d-choices, improves upon the task allocation strategy:

JIQ-PoD task allocation: A dispatcher with non-empty I-queue selects an idle VM uniformly at random from its I-queue and then removes this VM from its I-queue. A dispatcher with empty I-queue selects a subset of *d* VMs from all VMs, and among these *d* VMs assigns the task to the VM with the shortest task queue.

JIQ-NE, non-empty, improves the strategy for dispatcher selection:

JIQ-NE dispatcher selection: The router chooses *d* dispatchers at random and among them selects the first one having a non-empty I-queue for routing a new task. If none of the *d* dispatchers has a non-empty I-queue, then the d^{th} dispatcher is chosen irrespective of its I- queue status.

JIQ-E, empty, is like JIQ-SQ(d), however, the difference is in that JIQ-SQ(d) seeks the dispatcher with shortest length I-queue whereas JIQ-E seeks a dispatcher with an empty I- queue:

JIQ-E I-queue joining: When a VM becomes idle, it chooses *d* dispatchers at random and among those registers itself with the first dispatcher that has an empty I-queue. If none of the *d* dispatchers has an empty I-queue, then the d^{th} dispatcher is chosen irrespective of its I-queue status.

Also, JIQ-SQ(d) and JIQ-PoD probe *d* servers simultaneously whereas JIQ-NE and JIQ-E do the probing sequentially to reduce the communication cost.

In the same spirit, the present research preserves the general model of JIQ while improving upon the I-queue joining strategy.

3. Join Idle Queue Dispatcher I-queue Optimization

As seen in Table 1, in JIQ and its variants idle VMs register themselves with the I-queues of dispatchers selected by random or by the SQ(d) method. Through simulation studies we noted that this approach allows zero-length I-queues to persist in dispatchers which may lead to suboptimal assignment of tasks to VMs and hence increased response time. The proposed algorithm, JIQ dispatcher I-queue Optimization, aims to increase the number of dispatchers that have non-empty I-queues without increasing the overhead of communication between dispatchers and VMs. To achieve this, a list of dispatcher I-queue lengths is kept with the VMs. This list is updated opportunistically when dispatchers assign tasks to VMs. At the time of task assignment, a dispatcher also communicates its I-queue length to the VM which is stored by the VM in its list of dispatcher I-queue lengths. As different dispatchers and VMs communicate during the process of task assignment, the VMs gradually build up an approximate state of dispatcher I-queue lengths without incurring additional communication overhead. The VMs may never build a comprehensive view of all dispatcher I-queue lengths and the values may not be up to date, however the partial view suffices for VMs to register themselves with dispatchers with empty I-queues when they become idle. This leads to a reduction in the numbers of dispatchers with empty I-queues, hence increasing the probability of tasks finding a dispatcher with non-zero I-queue and being processed immediately. Hence, there is an improvement in the average response time.

JIQ-DIO compares with other JIQ algorithms listed in Table 1 as follows:

- Dispatcher selection: Random
- Task allocation:
- Non-empty I-queue: Random VM from the I-queue
- Empty I-queue: Randomly to any VM
- I-queue joining: Lookup the VM's list of dispatcher I-queue lengths. If there are dispatchers with empty I-queues in this list, select any one of them at random. Otherwise select any dispatcher at random.

3.1. Pseudocode of JIQ-DIO

The JIQ-DIO algorithm has been simulated in CloudSim Plus following the pseudocode presented below. In CloudSim terminology tasks are called cloudlets.

1.Cloudlet generation and processing module

```
    • Generate n cloudlets with lengths (task size) as per exponential distribution
    • Set the cloudlet arrival times as per Poisson distribution
    • For each cloudlet to be scheduled:
    o Select a dispatcher uniformly at random from all the dispatchers
    o IF (dispatcher has non-empty I-queue) THEN
    • Select at random an idle VM in the dispatcher I-queue
    • Assign the cloudlet to the selected VM for processing
    • Remove the selected VM from the dispatcher's I-queue
    • In the selected idle VM also set the length of the selected dispatcher's I-queue
    ELSE
    • Assign cloudlet to a random VM.
    • Capture how many times this happens, i.e., no idle VM found.
```

2.Listener: Cloudlet completion module

```
    • Decrement task queue length at VM
    • IF (task queue length == 0) THEN
    o VM is Idle, ready to be assigned to a dispatcher
    o Look up the list of dispatcher I-queue lengths (info embedded into VMs during cloudlet processing module)
    o IF (number of dispatchers with zero I-queue length > 0): THEN
    • Assign Idle VM to first dispatcher found with empty idle queue ELSE
    • Assign idle VM to a random dispatcher
```

4. Results

4.1. Simulation framework

The simulations in this study have been performed using CloudSim Plus, a fork of CloudSim. New modules were developed in CloudSim Plus to simulate JIQ-DIO as well as the other five JIQ variants listed in Table 1. The simulation model was described previously in (Narang et al., 2021), Section 7. In summary, cloudlets are generated dynamically with arrival times modeled by a Poisson process. The cloudlets are considered independent of each other and of the same priority. Cloudlet lengths are exponentially distributed. The VMs are modeled as having heterogeneous processing capacities in progressive increments of 1 MIPS step-ups while being homogeneous in other machine characteristics including RAM, bandwidth and storage. The parameters used in simulations in the present study are shown in Table 2.

Table 2. Parameter values used in simulations.

Parameter	Value(s)
Number of hosts	60
Number of virtual machines	180, 360, 600
VM processing capacity (MIPS) (heterogeneous)	20, 21, 22, …, 20+(n-1)
Number of dispatchers	10, 36, 60, 120
SQ(d)	0.3, 0.5, 0.6
Number of cloudlets	10000, 30000, 50000
Cloudlet length distribution	Exponential
Mean cloudlet length	800
Cloudlet arrival process	Poisson
Mean arrival time for cloudlets	1, 2

4.2. Performance metrics

The algorithms are compared on the following performance metrics which are key service level measures for cloud-based scenarios (Narang et al., 2021).

Average response time per task (cloudlet):

$$\textit{Average Response Time} = F - A + T_{delay},$$

where A is the arrival time of the task, T_{delay} is the transfer time of the task, and F is the time to complete the task. For simulations in this work $T_{delay} = 0$.

Makespan: It is the completion time of the last cloudlet, which implies that all submitted tasks (cloudlets) have been processed by the pool of VMs.

$$\boldsymbol{Makespan} = \max\,(\boldsymbol{Cloudlet\ completion\ times})$$

Resource utilization: It is the average utilization of VMs where the utilization of a VM is defined as the difference of its busy and idle times through the length of execution, i.e., makespan.

$$ru = \frac{R_{rt} - R_{it}}{makespan} \qquad Avg\ RU = \sum_{i=1}^{n} ru,$$

where R_{rt} and R_{it} are the VM running time and idle time respectively, and n is the number of VMs. Its value ranges between -1 (completely idle) and 1 (fully busy).

4.3. Simulation results

Simulation was performed in triplicates for each set of parameter values and the performance metrics were summarized by their mean and standard deviation over replicates.

Compared to standard JIQ and its variants, JIQ DIO had a consistently better response time which was even more pronounced as the number of VMs was increased (Fig. 2). In these simulations, JIQ-DIO led to almost 2-fold improvement in the response time with good statistical significance ($p < 0.05$). Interestingly, varying the number of dispatchers had little impact on the average response time as shown by lack of statistical significance ($p > 0.05$) in Fig. 3. It could be argued that performance improvement obtained with JIQ-DIO might be transient. To test this, the number of cloudlets increased from 10000 to 50000. Simulation results show that there is no drop in the performance of JIQ-DIO over time (Fig. 4).

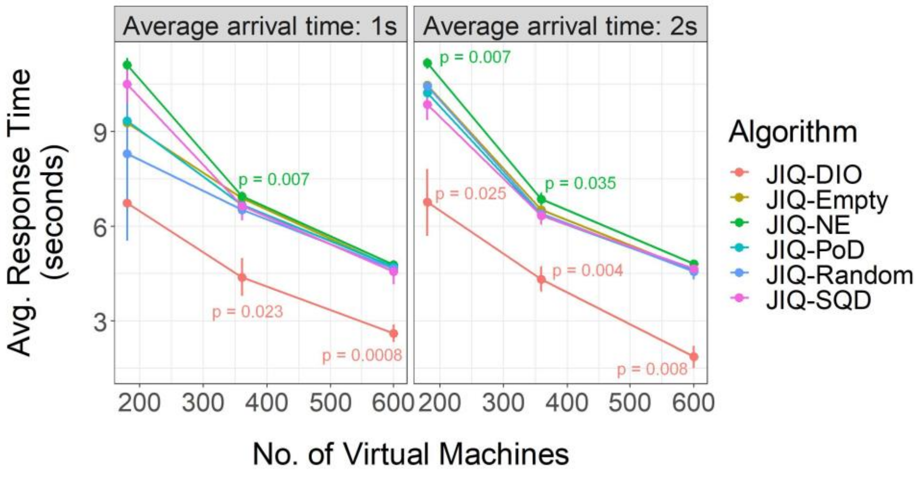

Figure 2. Average response time with varying number of VMs and frequency of cloudlet arrivals.
T-test was performed to identify significant differences in the average response times of various algorithms in comparison to JIQ-Random. Only significant p-values (p < 0.05) are shown.

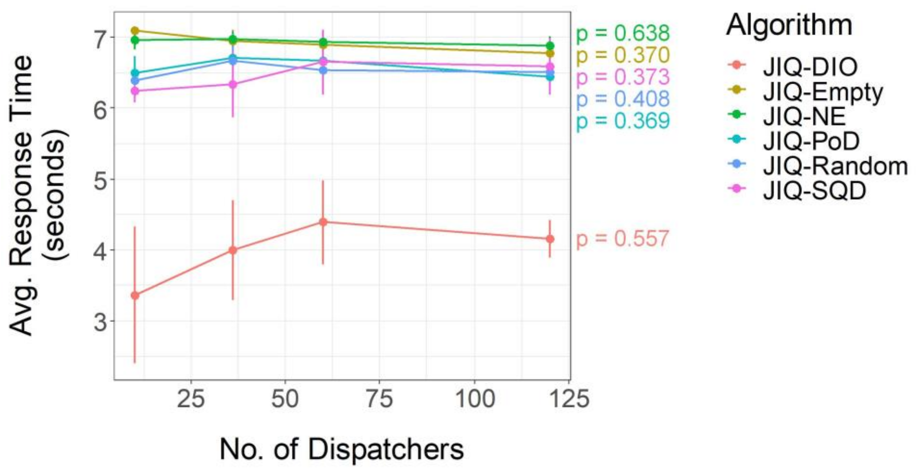

Figure 3. Average response time with varying number of dispatchers.
Mean cloudlet arrival time = 1s, no. of VMs = 360. Statistical significance (p-value) of the variation in the average response time with the number of dispatchers was estimated using the Kruskal Wallis test.

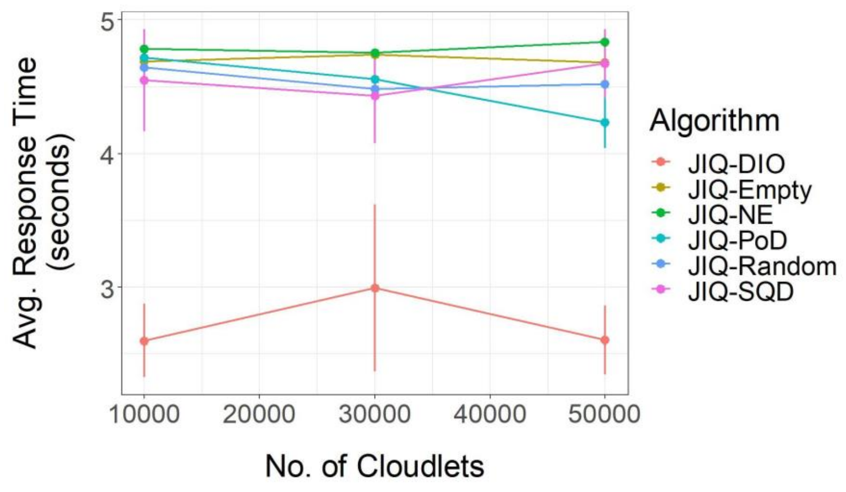

Figure 4. Average response time with varying number cloudlets.
Mean cloudlet arrival time = 1s, no. of VMs = 600, number of dispatchers = 60

Simulation results were used to compare the probability of dispatchers finding empty VMs in various JIQ implementations. It was mentioned above that our implementation of JIQ maintains a counter to track the number of times a dispatcher has an empty I-queue when a task is routed to it. The data collected from this counter is shown in Fig. 5. The JIQ-Empty and JIQ-NE encountered an empty I-queue 35,111 and 35,149 times respectively on average among 50,000 cloudlets which implies that there is greater than 70% probability of dispatchers having empty I-queues. The frequency (out of 50,000) is 1981 or 3.96% for standard JIQ, 1970 or 3.94% for JIQ-PoD, 200 or 0.4% for JIQ-SQ(d) and 8 or 0.0016% for JIQ-DIO. These results favor the hypothesis that JIQ-DIO decreases the probability of dispatchers with empty I-queues which in turn improves the average response time.

No significant differences were observed between the various JIQ variants in terms of the makespan and average resource utilization metrics.

It is worth noting that the performance of JIQ and its variants reported in previous works (Mitzenmacher, 2016; Narang et al., 2021) is mostly considering a homogeneous set of VMs, whereas this work simulates a heterogeneous setup and uses a more generic simulation tool, CloudSim Plus. Hence, the simulation results concerning average response times presented herein might be more generalizable in comparison to those presented previously.

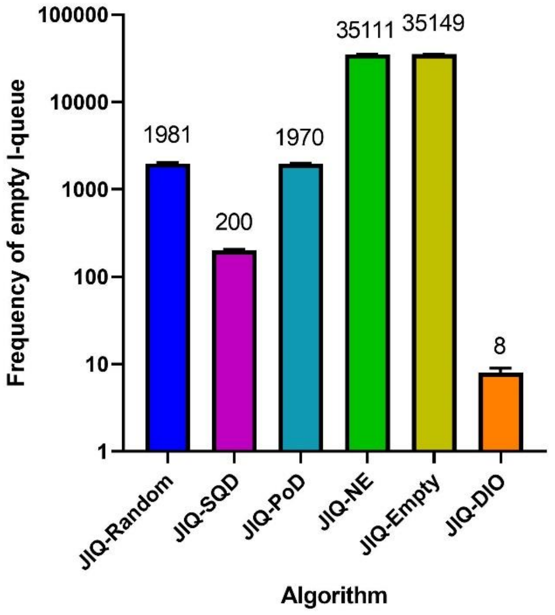

Figure 5. The number of times a dispatcher has an empty I-queue when a task is routed to it. The data is aggregated over all tasks and all dispatchers throughout a simulation. The bars show the mean value, and the error bars show the standard deviation over three replications of the simulation. Number of VMs = 600, number of dispatchers = 60, number of cloudlets = 50000, mean cloudlet arrival time = 1s.

5. Conclusion and future work

The Join Idle Queue algorithm is well suited for load balancing in distributed scenarios which are now commonplace in cloud computing. Over the years a few improvisations have been proposed to the initial work to yield better performance for specific scenarios. We proposed a new variant, JIQ dispatcher I-queue Optimization, and through simulations on a generic heterogeneous setup showed a two-fold improvement in response times across a broad range of parameters. We showed that JIQ-DIO results in a higher probability of dispatchers having access to idle machines leading to this performance gain. The reliability of the simulations was ensured by replication of the experiments followed by statistical analysis of the results. Rigorous mathematical evaluation of JIQ- DIO is left to a separate communication. In terms of future work, the model can be improved to attain a close to uniform distribution of idle machines among dispatchers to realize even further improvements to response times. Additional optimizations can be made to improve upon other performance metrics which would require taking into consideration additional parameters beyond the standard few used in this work.

Conflict of interest

The authors have no conflict of interest to declare.

Acknowledgments

We are grateful to our Institute for providing us with the platform to do research and explore new tools.

Funding

The authors received no specific funding for this work.

References

Al-Fares, M., Loukissas, A., & Vahdat, A. (2008). A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review*, *38*(4), 63-74. https://doi.org/10.1145/1402958.1402967

Badonnel, R., & Burgess, M. (2008). Dynamic pull-based load balancing for autonomic servers. In *NOMS 2008-2008 IEEE Network Operations and Management Symposium* (pp. 751-754). IEEE. https://doi.org/10.1109/NOMS.2008.4575205

Bharti, S., & Pattanaik, K. K. (2013). Dynamic distributed flow scheduling with load balancing for data center networks. *Procedia Computer Science*, *19*, 124-130. https://doi.org/10.1016/j.procs.2013.06.021

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R. (2011). CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, *41*(1), 23-50. https://doi.org/10.1002/spe.995

Cardellini, V., Colajanni, M., & Yu, P. S. (1999). Redirection algorithms for load sharing in distributed Web-server systems. In *Proceedings. 19th IEEE International Conference on Distributed Computing Systems (Cat. No. 99CB37003)* (pp. 528-535). IEEE. https://doi.org/10.1109/ICDCS.1999.776555

Cheung, C. M., & Leung, K. C. (2018). DFFR: A flow-based approach for distributed load balancing in Data Center Networks. *Computer Communications*, *116*, 1-8. https://doi.org/10.1016/j.comcom.2017.11.001

Duan, R., Prodan, R., & Li, X. (2014). A sequential cooperative game theoretic approach to scheduling multiple large-scale applications in grids. *Future Generation Computer Systems*, *30*, 27-43. https://doi.org/10.1016/j.future.2013.09.001

Gao, L., & Min, P. (2009). Optimal superpeer selection based on load balance for P2P file-sharing system. In *2009 International Joint Conference on Artificial Intelligence* (pp. 92-95). IEEE. https://doi.org/10.1109/JCAI.2009.165

Grosu, D., & Chronopoulos, A. T. (2005). Noncooperative load balancing in distributed systems. *Journal of parallel and distributed computing*, *65*(9), 1022-1034. https://doi.org/10.1016/j.jpdc.2005.05.001

Harchol-Balter, M. (2021). Open problems in queueing theory inspired by datacenter computing. *Queueing Systems*, *97*(1-2), 3-37. https://doi.org/10.1007/s11134-020-09684-6

Hong, Y. S., No, J. H., & Kim, S. Y. (2006). DNS-based load balancing in distributed Web-server systems. In *The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06)* (pp. 4-pp). IEEE. https://doi.org/10.1109/SEUS-WCCIA.2006.23

Khiyaita, A., El Bakkali, H., Zbakh, M., & El Kettani, D. (2012). Load balancing cloud computing: state of art. *2012 National Days of Network Security and Systems*, 106-109. https://doi.org/10.1109/JNS2.2012.6249253

Kingman, J. F. (1961). Two similar queues in parallel. *The Annals of Mathematical Statistics*, *32*(4), 1314-1323. https://www.jstor.org/stable/2237929

Kunwar, V., Agarwal, N., Rana, A., & Pandey, J. P. (2018). Load balancing in cloud—a systematic review. *Big Data Analytics: Proceedings of CSI 2015*, 583-593. https://doi.org/10.1007/978-981-10-6620-7_56

Li, K. (2017). Optimal task dispatching on multiple heterogeneous multiserver systems with dynamic speed and power management. *IEEE Transactions on Sustainable Computing*, *2*(2), 167-182. https://doi.org/10.1109/TSUSC.2017.2706425

Lu, Y., Xie, Q., Kliot, G., Geller, A., Larus, J. R., & Greenberg, A. (2011). Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services. *Performance Evaluation*, *68*(11), 1056-1071. https://doi.org/10.1016/j.peva.2011.07.015

Lu, Y. (2018). *Distributed join-the-idle-queue: new algorithm and analysis* (Doctoral dissertation, University of British Columbia). https://dx.doi.org/10.14288/1.0375787

Mitzenmacher, M. (2016). 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. https://doi.org/10.1109/ALLERTON.2016.7852246

Narang, S., Goswami, P., & Jain, A. (2019). Statistical Analysis of Cloud Based Scheduling Heuristics. In *Information, Communication and Computing Technology: 4th International Conference, ICICCT 2019, New Delhi, India, May 11, 2019, Revised Selected Papers 4* (pp. 98-112). Springer Singapore. https://doi.org/10.1007/978-981-15-1384-8_9

Narang, S., Goswami, P., & Jain, A. (2021). A Comprehensive Review of Load Balancing Techniques in Cloud Computing and Their Simulation with CloudSim Plus. *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, *14*(6), 1684-1694. https://doi.org/10.2174/2666255813666191218113350

Ousterhout, K., Wendell, P., Zaharia, M., & Stoica, I. (2013). Sparrow: distributed, low latency scheduling. In *Proceedings of the twenty-fourth ACM symposium on operating systems principles* (pp. 69-84). https://doi.org/10.1145/2517349.2522716

Rao, A., Lakshminarayanan, K., Surana, S., Karp, R., & Stoica, I. (2003). Load balancing in structured p2p systems. In *Peer-to-Peer Systems II: Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22, 2003. Revised Papers 2* (pp. 68-79). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-45172-3_6

Silva Filho, M. C., Oliveira, R. L., Monteiro, C. C., Inácio, P. R., & Freire, M. M. (2017). CloudSim plus: a cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness. In *2017 IFIP/IEEE symposium on integrated network and service management (IM)* (pp. 400-406). IEEE. https://doi.org/10.23919/INM.2017.7987304

Yang, B. B., & Garcia-Molina, H. (2003). Designing a super-peer network. In *Proceedings 19th international conference on data engineering (Cat. No. 03CH37405)* (pp. 49-60). IEEE. https://doi.org/10.1109/ICDE.2003.1260781