applied sciences

MDPI

Article

A Study Comparing Waiting Times in Global and Local Queuing Systems with Heterogeneous Workers

Inessa Ainbinder [1], Evgeni Temnikov [1] and Miriam Allalouf [2,*]

[1] Department of Industrial Engineering & Management, Azrieli College of Engineering Jerusalem (JCE), Jerusalem 9103501, Israel; inessaai@jce.ac.il (I.A.)
[2] School of Software Engineering and Computer Science, Azrieli College of Engineering Jerusalem (JCE), Jerusalem 9103501, Israel
* Correspondence: miriamal@jce.ac.il

Featured Application: Today's message queue (MQ) systems lack local queuing services. To enhance its functions, heterogeneity-aware policies, push control, and dedicated architecture should be applied to enable systems with human workers to use them.

Abstract: A virtual marketplace or service-providing system must ensure minimal task response times. Varying working rates among the human workers in the system can lead to longer delays for certain tasks. The waiting time in the queue is crucially affected by the queueing architecture used in the system, whether global or local. Studies generally favor global queue systems over local ones, assuming similar processing rates. However, system behavior changes when workers are heterogeneous. In this research, we used simulation to compare the waiting times of tasks assigned to three categories of processing rates in both architectures and with various routing policies in local queues. We found that when using random tie-breaking, there was a correlation between waiting time duration and the proportion of tie-breaking events. Performance is improved when controlling these events using scheduling awareness of the workers' processing rates. The global queue outperforms local queues when the workers are homogeneous. However, the push mechanisms that control the assignment processes and heterogeneity-aware algorithms improve local queue system waiting times and load balance. It is better than global queues when tasks are assigned to medium and fast workers, but it also enables specific slow workers' assignments.

Keywords: global queue system; local queues architecture; JSQ scheduling; waiting times in queue; workers with heterogeneous processing rates

Citation: Ainbinder, I.; Temnikov, E.; Allalouf, M. A Study Comparing Waiting Times in Global and Local Queuing Systems with Heterogeneous Workers. *Appl. Sci.* **2024**, *14*, 3799. https://doi.org/10.3390/app14093799

Academic Editor: Richard C. Millham

Received: 16 March 2024
Revised: 16 April 2024
Accepted: 20 April 2024
Published: 29 April 2024

1. Introduction

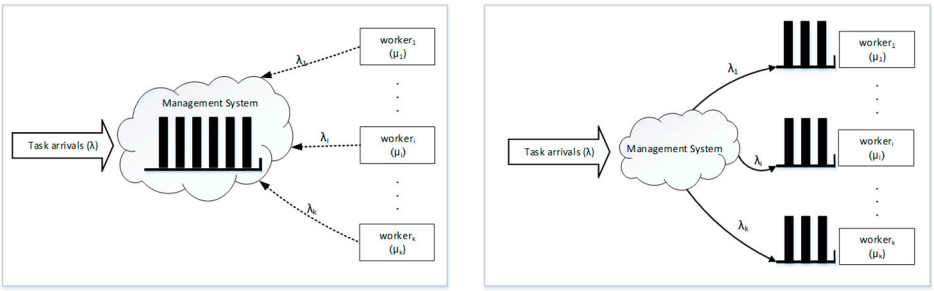
In an era dominated by digital services, from virtual marketplaces to call centers, the efficiency of queueing systems has emerged as a critical determinant of customer satisfaction and operational success. These systems, which orchestrate the allocation of tasks to service providers, are the backbone of service delivery in numerous industries. Central to the performance of these systems is their ability to minimize response times—the interval from when a task is initiated to its completion. The response time consists of the following two parts: the time the task waits for the server (the queueing time or waiting time) and the time the server is actively working on the task (the processing time). Traditionally, queueing systems have been designed under the assumption of worker homogeneity, simplifying the complex dynamics of task allocation and system performance. However, this assumption falls short in the face of human workers' diverse capabilities and efficiencies, necessitating a reevaluation of queueing architectures to accommodate worker heterogeneity.

Efficiency and speed can vary between workers in different industries, ultimately affecting productivity. To address this issue, managers can implement workload balancing,

route optimization, and incentive programs. In service-oriented industries, variability is the norm, and several papers have analyzed marketplace dynamics and workers' behavior [1–4]. In their comprehensive study of crowdsourcing and real-world marketplace data, Jain et al. [3] confirmed the variability of workers' processing rates. Efficiently routing jobs to servers with varying processing speeds is a challenge faced not only by human servers but also by digital servers. In computing environments, digital servers offer services at different speeds and have varying processing rates. Cloud computing environments use virtual machines or containers on servers with different capacities to allocate resources effectively. Servers in clusters or grids may have different hardware configurations or can be in different regions, resulting in variations in processing speed and network latency. Edge computing devices have varying processing capabilities limited by power constraints, size, and cost. Similarly, IoT systems consist of diverse devices with varying computing capabilities. Selecting the right queueing system architecture for these systems is crucial to managing their workloads.

The evolution of queueing systems from global to local architectures represents a significant shift in addressing the challenges of heterogeneous workforces. The following describes the two common architectures of queue structures: a single global queue (Figure 1a) and local queues (Figure 1b):

1. Global queue architecture (also called the centralized work pile, single, shared, or pooled queue) is the traditional queueing system in which the management system handles incoming tasks, and humans pull tasks from this single shared or pooled queue when idle.
2. Local queue architecture (parallel or dedicated queues) is a relatively newer queuing system developed when network bandwidth increases and local worker's storage allows it. In this architecture, each worker has a queue of tasks, and the management system pushes the tasks to the workers using routing or dispatching policies. Each worker fetches the next task from her dedicated queue and processes it. Join-the-Shortest-Queue (JSQ) is a policy where the next task is allocated to the worker with the lowest number of tasks in their local queue length. It is considered in many papers to be the policy that achieves lower response times in a system with parallel queues. JSQ flavors and other routing policies are considered in the literature and this paper.



(**a**) (**b**)

Figure 1. (**a**) Global queue architecture where workers pull tasks from this queue; (**b**) in local queue architecture, the management system pushes the tasks to the workers using policies such as JSQ.

Queue design is critical in many service industries, as reduced waiting times are positively linked to higher financial performance. Global queue systems, with their centralized task pools, offer simplicity and potential efficiency gains through task pooling, where workers immediately pull a task from a global queue whenever they become idle. This sharing reduces the mean and variance of waiting times in the single queue system but often overlooks the variability in worker performance. However, in many cases, a global queue may not be feasible or may not be the best option. For example, in the case of an extensive distributed system where the front-end servers receive the stream of arriving tasks and can perform scheduling and routing functionalities to route the job to different

workers' servers, they cannot manage a centralized queue mechanism. Recent research has shown that there may be better choices for systems with human strategic workers than a global queue system. A system with parallel and local queues can be more effective, as it motivates human workers to complete their tasks locally, resulting in better performance regarding workers' processing rates. This, in turn, leads to shorter queue lengths [5]. Local queue systems provide an opportunity to tailor task routing to individual worker capabilities, promising load-balancing improvements and reduced waiting times.

We use the abbreviations GH, GHT, LH, and LHT to refer to the four systems we investigate in our study. The first letters (G or L) refer to the queueing architecture, with G for global and L for local. The other letter (H or HT) refers to server types, with H for homogenous and HT for heterogeneous.

Choosing between global and local queue architectures is not straightforward. Most of the studies assumed that all workers have similar processing rates. Analyzing a system with servers with processing rates can be challenging due to the large number of parameters involved (system load, workload distribution, arrival rate, number of workers, processing rates, and categories). Queuing architecture adds various configurations, such as queue management (FIFO or round-robin), dispatch policies (shortest or weighted queue), and routine policies. Papers that have analyzed waiting times in settings with heterogeneous workers have used different settings, such as considering a small number of workers or infinite system sizes and assuming low or high system loads. The comparative effectiveness of these systems in environments with varied worker speeds remains the subject of ongoing research. Almost no papers compare the performance of local vs. global queues in a setting where the workers are heterogeneous.

In this paper, we address this gap in the literature and use discrete event simulation (DES) to compare the two architectures in a system with homogenous Poisson arrivals and exponentially distributed service times. We compare the GHT and LHT systems in terms of their performance measured by waiting time and load distribution among the workers. We also compare the performance of the GHT and GH systems and of the LH and LHT systems. It is statistically acceptable to prioritize waiting times over response times when workers' task arrival and processing rates follow the Poisson distribution (M/M/K); the system load is above 0.5. This has been demonstrated in real-world marketplace data gathered through crowdsourcing by Jain et al. [3].

The main contributions of this work are listed below.

1. A literature review, including the following:
    - Identifying and describing papers that compare GH and GHT systems (Section 2.1). We found that most of the studies considered only systems with few servers.
    - Identifying and discussing papers dedicated to the LH and LHT systems (Section 2.2). Most of these papers assume very large-to-infinite system sizes and show that the JSQ routing policy effectively keeps waiting times low for the LH system. We found no papers directly comparing the waiting time performance of LH and LHT systems.
    - Identifying heterogeneity-aware routing policies discussed in the LHT literature. Our simulation studies the comparative performance of the Shortest Expected Delay (SED) and the speed-aware routing.
    - Reviewing the literature on handling tie-breaking (TB) events in local queueing architectures. A tie-breaking event occurs when several servers meet the same condition for task routing (for example, the same value for expected delay), and the scheduler needs to select one of the servers. With random TB routing, the server is chosen at random. The literature discusses other tie-breaking algorithms such as fastest TB, local JSQ random TB, and local JSQ fastest TB routing policies for homogeneous and heterogeneous settings. The local weighted JSQ with learning (WJSQL) is our version of SED.

- Identifying and discussing a limited number of papers comparing GH and LH systems and recognizing that there are even fewer papers comparing GHT and LHT systems (Section 2.3).

2. DES experiments are designed to compare various aspects of queuing architecture and local queue routing. Considering market behavior analysis [3], we chose three categories of processing rates as follows: slow, medium, and fast. We conducted tests on systems with 3, 6, 9, and 12 workers, equally divided into category groups when the systems' loads were 0.8 and 0.93, respectively. We defined accurate parameters to compare the homogenous and heterogeneous simulation environments for a certain system load. We adjusted the arrival rate parameters according to the processing rate categories and number of workers (Section 4.2). Our simulation results provide per-category waiting times and worker utilization, not only the total mean waiting time. The few papers that compare both systems did not provide these details.

Our findings are as follows:

3. Global queue architectures (GH, GHT) ensure consistent average task waiting times across all workers, regardless of speed settings. This happens because all tasks are in the same queue until the workers retrieve them, and it occurs for all numbers of workers or system load.
4. While GH and GHT outperform LH and LHT, on average, the push mechanisms that control the assignment processes and heterogeneity-aware algorithms improve the local queue system waiting times and load balance. It is significant and better than the global queues when tasks are assigned to medium and fast workers. Due to the control mechanism, slow worker's assignments are easier to control.
5. We observed a connection between the duration of waiting times and the proportion of TB events. In particular, the waiting times for tasks assigned to slow and medium workers tend to be excessively long when using a simple JSQ with random TB in a local queue architecture.
6. To conclude, the average waiting times (W) for GH, GHT, LH, and LHT for loads at 0.8 and 0.93 are ranked in the following order:
   - $W_{GH} \sim= W^{Slow}_{GHT}, W^{Medium}_{GHT}, W^{Fast}_{GHT} \sim< W_{LH} < W^{Medium}_{LHT} < W^{Slow}_{LHT}$
   - $W^{Fast}_{LHT} \sim< W_{GH}$

   The proportions of the TB events have a similar order. On the one hand, $W_{GH} \sim< W_{LH}$ follows previous works, but the behavior of the waiting times of tasks assigned to different categories when using GHT and LHT shows that heterogeneity-aware routing policies are required.
7. For both loads of 0.8 and 0.93 and the same number of workers, JSQ's fastest TB outperforms WJSQL and is like the waiting times in the global architecture. This finding is significant because simple JSQ is more practical to deploy, and the WJSQ is sometimes considered optimal. The WJSQL is a version of SED that we developed and tested. WJSQL learns the workers' processing rates and assigns the task to the faster worker in the shortest expected time. The WJSQL is more effective than the JSQ random TB and JSQ fastest TB routine policies for managing slow workers and reducing waiting times for their assigned tasks.

The paper is organized in the following way. Section 2 discusses the related work. Section 3.1 describes the global system, and Section 3.2 describes the local system. Section 3.3 describes the methodologies and simulation setup for the homogenous and heterogeneous environments. Section 4 describes the simulation results. Section 4.1 compares global and local systems with homogeneous workers (GH vs. LH). Section 4.2 demonstrates the correlation between the number of tie-breaking events and the waiting times when a random selection is used. Section 4.3 describes heterogeneity-aware routing policies for the local queue system, and Section 4.4 compares global and local systems with heterogeneous workers (GHT, LHT). Section 5 summarizes and concludes the paper.

2. Related Work

This section summarizes the research on experienced delays and workers' utilization based on their processing rates in different queueing system architectures. The first part presents work dealing with global queue systems only. It compares waiting times when its workers have a homogenous average speed rate vs. a heterogenous speed rate (GH vs. GHT). The second part presents work dealing with local queue systems only and analyzes the waiting times regarding different routing policies and the heterogeneity of workers (LH vs. LHT). The last part compares waiting times in global and local queue systems, mainly when workers' speed rates are heterogeneous. A single queue architecture with K homogenous workers is signed M/M/K when the K workers' task interarrival times and service times follow an exponential distribution. When there are K heterogeneous workers, the system is signed M/Mi/K, where each worker has a different mean service time following the exponential distribution.

Parallel queues architecture with K homogenous multiple servers conforms to the M/M/K/JSQ/PS mode; when the task arrivals are Poisson and processing times of the K workers are exponentially distributed, the tasks are allocated using the Join-the-Shortest-Queue (JSQ) model. The worker handles tasks in their local queue in parallel. The M/Mi/K/JSQ/FIFO notation states that each worker can have a different processing rate while tasks are handled according to the FIFO order. Part of the following papers treats waiting time as synonymous with response time. Focusing on waiting time can simplify the analysis of theoretical models of the queue's dynamics. When the servers are identical, the distinction between waiting and processing times becomes less critical for the analysis.

2.1. Global Queue with Homogenous (GH) and Heterogeneous Workers (GHT)

Single queue architecture, where digital servers pull tasks from a single shared FIFO queue managed by a centralized system, was the dominant architecture until the 1990s. This architecture has been considered one way to reduce the average waiting time without increasing staffing levels because of its pooling benefit. However, research shows that the performance of a global queue system with heterogeneous workers decreased. The following studies deal with an M/Mi/K system where the workers are heterogeneous. Gumbel [6] analyzed an M/Mi/K system with one queue and *K* heterogeneous servers. They found that servers with different service rates cannot be replaced with an equal number of servers with the same rate without increasing the waiting time in the queue. They stated that the more significant the difference between the servers' processing rates, the longer the time waiting in the queue. Smith & Whitt [7] showed that in the tele-traffic world, combining multiple arriving streams with different service rates into one system may result in a higher waiting time. Grassmann & Zhao [8] studied the importance of selecting a server in a single queueing system with multiple heterogeneous servers since it changes with decreasing traffic intensity, increasing number of servers, and decreasing variation between interarrivals. Alves et al. [9] developed upper bounds for the average queue length and average waiting time in a single queue M/Mi/K system. They showed that heterogeneous systems can perform better than homogeneous systems when using the Fast Server First (FSF) allocation strategy. However, it can also result in longer waiting times with different numbers of servers. There is no condition under which the average waiting time in a queue of a heterogeneous system is better than that of a homogeneous one. Rubinovitch et al. [10] discuss the "Slow Server Problem" that arises when a service facility has both fast and slow servers, and the question is how to best utilize the slow server in the presence of a faster one. Rykov [11] generalizes the Slow Server Problem to include an additional cost structure. This approach provides a more nuanced analysis of activating an additional (slow) server in a system when already operating with servers of varying speeds might be beneficial. To enable customers to manage their waiting times, several works [12–14] proposed semi-parallel queueing by treating the global queue as several priority queues according to the processing rates. To conclude, there is no clear understanding of how waiting times behave in a global system with workers with different

processing rates. It is, therefore, difficult to compare the performance of such a system with one that has homogeneous workers. The waiting time in this kind of system mainly depends on the number of workers and their workload. However, studies have shown that the waiting times can be reduced if faster workers are given priority using techniques like FSF tie-breaking.

2.2. Local Queues with Homogenous (LH) and Heterogeneous (LHT) Workers

Parallel queue systems were developed for separated servers where a centralized management system could not handle all tasks locally. A centrally located dispatcher assigns tasks to several queues using different routing policies. The JSQ is an optimal routing policy that routes tasks to the server with the shortest queue. JSQ minimizes the average delay of jobs and the number of customers in the system consisting of identical servers [15] using Markov Decision Processes (MDP), assuming Poisson arrivals and exponential service times. This optimality result was extended to general stochastic arrival processes [16,17]. Gupta et al. [18] provided a formal model and analysis of JSQ when used in the routing policy for server farms with the same processing rate and general task-size distributions. Ainbinder et al. [19] found that JSQ achieves a better response time than the method that considers a worker's average queue length (AQL) with homogenous and heterogeneous workers (LH, LHT).

JSQ is good for identical servers with independent service times but not for heterogeneous systems with different processing rates. The SED routing policy is used in the latter case, which routes customers to the server with the shortest expected response time. However, SED requires knowledge of the entire job size distribution to estimate the remaining service time. It is difficult to analyze exact systems when service rates are not equal, and finding optimal delay schemes for heterogeneous systems remains an open problem due to infinite state spaces [20]. In this work, we simulate WJSQL, which learns the processing rates online and assigns them to the fastest worker with the shortest expected delay. Bhambay & Mukhopadhyay [21] introduced SA-JSQ (speed-aware JSQ), which is a variant of the JSQ scheme with a large number of processing rate categories. In SA-JSQ, each incoming task is assigned to a server with the highest speed among those with the minimum queue length. Ties among servers are broken randomly and uniformly. They proved that SA-JSQ is optimal in fluid limits when the system size is infinite. However, numerical simulations showed that assigning tasks based on SED performed better than SA-JSQ for small system sizes. Ellens et al. [22] investigate dynamic routing policies in a local queue system (LHT) with two servers, focusing on scenarios where only a subset of jobs was observable and controllable while others were assigned randomly, showing that heuristic routing strategies can significantly reduce the average response time, closely approaching the performance of the optimal fully observable system under heavy loads, as represented by the SED policy. Their problem is relevant because it shows the strength of a good routine policy.

Our study assumes a centralized dispatcher that assigns incoming tasks. As cloud infrastructure grows and server heterogeneity increases, traditional single-dispatcher load-balancing policies like JSQ become less feasible. There are low-communication policies with few dispatchers, such as JSQ(d), where each dispatcher picks d queues randomly and allocates the task to the queue with the shortest queue, and the Join-Idle-Queue (JIQ). Both architectures are unstable or perform poorly when the number is very large, and their speed is heterogeneous. Vargaftik et al. [23] introduced the Local Shortest Queue (LSQ) family of load-balancing algorithms with multiple dispatchers. Each dispatcher maintains its local view of the server queue lengths and uses JSQ on its local view, which is updated infrequently and possibly outdated. They showed how their LSQ policies often outperformed even JSQ due to the low communication and how the slow server problem was solved using advanced pull-based communication. Gardner et al. [24] formulated "power-of-d" versions of JIQ and JSQ policies, improving the mean response time and

queue length distribution analyses and outperforming other heterogeneity-aware policies, such as SED.

2.3. Global vs. Local Queues Comparison

Global queue architecture is a way to reduce the average waiting time without increasing staffing levels. This is achieved through pooling benefits. Rudolph et al. [25] present a system with a collection of local task queues and a simple distributed load-balancing scheme well-suited for scheduling tasks in shared memory parallel machines. Typically, task scheduling on such machines has been performed through a single, globally accessible work pile. However, the scheme introduced in this paper achieves balanced utilization comparable to that of a global work-pile. Gupta & Walton [26] analyzed local queue architecture with JSQ with different load balancing rules in a contemporary scaling regime when workers were homogenous. They compared it with the global queue architecture and found that dispatching jobs to servers instead of storing them in a global queue costs 15%. The homogenous case is relatively easy since all the workers have the same mean rate, and the higher number of tie-breaking cases causes the price of dispatching jobs to the local queue. Thus, it does not reflect the real world since workers' service rates in most systems are not homogenous. Bhambay & Mukhopadhyay [21], who proposed a SA-JSQ scheme for both local queue settings and the global queue setting in scenarios where the system size is infinite, also obtained a lower bound on the mean response time for local and global queues and showed that this lower bound is better when the system uses global queue architecture. They did not express the mean response time of a task assigned to a specific category since they only discussed the lower bound.

The research studies described so far deal with systems where workers are digital processors or servers rather than human workers. The latest research shows that a system with human strategic workers performs better regarding workers' processing rates in a system with parallel queues rather than in a global queue system. These studies are about the change in workers' processing time rather than heterogeneous workers, but still, it is important to compare global and local queues. Shunko et al. [27] studied how queue design affects human worker productivity in service systems. The study found that a single-queue architecture or poor visibility of the queue length in local queues slows down servers. Delasay et al. [28] developed a general framework to analyze the influence of system load (queue length) on workers' service times. Do et al. [5] developed analytical models that incorporate human behavior for both types of queuing. They evaluated system performance based on the expected waiting time and total time. They established threshold values to identify situations where parallel systems outperform single queue systems for different workloads, system sizes, and speedup and slowdown effects. Sunar et al. [29] & Zhou et al. [30] studied the question of whether to operate a dedicated (local) system or a pooled (global) system by analyzing a model in which delay-sensitive customers could access their delay information (e.g., by observing the queue length or receiving real-time expected delay information) and make their joining or leaving decisions based on that information. They proved that pooling (global) queues can result in much worse performance than a dedicated system, even with identical servers and homogeneous customers.

To summarize, all the studies presented show that the superiority of a global queue system when workers are heterogeneous is not apparent. There are a lot of system and workload parameters, as well as behavioral terms, which show otherwise.

3. Models and Simulation Setup

Figure 1a,b illustrate the queueing architectures. The global and local models follow the M/M/K and M/Mi/K queuing distribution notations. Tasks arrive at a λ rate, each allocated to one worker. The systems examined in the current study are differentiated in the queue architecture, scheduling policies, and the characteristics of the resources (homogeneous and heterogeneous).

3.1. Global Queue Systems

In the global queue model (Figure 1a), the tasks are kept in the global FIFO pool of tasks where each idle worker pulls if the queue is not ready. With more than one idle worker, the task can be selected randomly by any idle worker. In a real (non-simulated) global queue architecture, the arrival rate λi of each worker i cannot be determined centrally. The worker's schedule of pulling times is determined by the system load and the worker's processing rate. An optimal worker's schedule is when workers' idle time is minimized. An optimal task assignment for workers is to minimize the time tasks, wait in a queue, and reduce the idle workers' free time. A TB event occurs when a task is assigned to a random idle worker out of others available at the same time. A "wrong" timing is when a task is chosen by one idle worker, causing another idle worker to wait, which may cause higher waiting times for the pending tasks. Thus, many TB events lead to different schedules of pulling times. Our simulation determines how different schedules of pulling times impact the arrival rate λi of each worker and tasks' waiting times. Specifically, we check a schedule where the task is pulled in FIFO order by a worker who is idle for a longer period than others.

In a homogenous environment, all workers have the same mean processing rates. In this work, we assume that the processing rates are distributed using an exponential distribution with an average of one. The schedule determined by the pulling times of homogenous workers in the global queue setting is considered optimal because when the queue is not empty while a worker pulls out a new task just after terminating the process of the previous task, the worker's wait time for the next task is minimized. There are very few TB events, and statistical changes cause them.

3.2. Local Queues Systems

In the local queues setting (Figure 1b), the dispatcher assigns the arriving tasks according to their order of arrival using a routing policy. The worker fetches the next task from her dedicated queue once it is ready. Routing policies target to keep each task's waiting time as short as possible and determine the arrival rate λi for each worker i. We expect the arrival rate λi to conform to the worker i efficiency level reflected by the processing time.

In a homogeneous environment, we expect the arrival rates λi to be equal. Assuming that the routing policy is JSQ, TB events may occur when the routing policy needs to be decided between workers, such as when there is more than one idle worker or when there are workers whose queue sizes are the same. Since there are more possible "wrong" decisions in local queue systems, the global queue systems have been shown in many studies to outperform local queue systems. Gupta et al. [18] provided a formal model and analysis of JSQ when used by the routing policy for homogenous server farms. They used a processor-sharing (PS) scheduler rather than a first-come-first-serve (FCFS) server farm in their model. They show the equivalence between M/G/K/JSQ/PS for *K* workers with local queues and Mn/M/1/PS for each worker in the system. It means that if each worker's assignment rate, λi, is conditioned by the queue length, as in JSQ, we can assume $\lambda i = \lambda/K$, which ensures the required load balance and schedule optimization we wish to have while allocating the tasks. Since our systems follow the M/M/K model, we can use Gupta's calculation to show that all the queues in the local queue system have the same number of tasks and similar waiting times. Green [31] describes the basics of queueing theory that are particularly useful in healthcare. For an M/M/K system, they observe the relationship between system workers' utilization and task waiting times using the coefficient variance (CV) metric. CV(A) and CV(S) are the ratios between the standard deviation and the mean of inter-arrival and service times, respectively. They show that if the coefficient of variation (CV) is close to one, which is the case for homogenous workers, the waiting times will be shorter than when the CV is far from one, as seen with heterogenous workers. We checked the CV(A) of the arrival rates when the workers were homogeneous and found that it was close to one.

3.3. Simulation Setup

We performed simulation tests using AnyLogic [32] simulation software (https://www.anylogic.com/, accessed on 15 March 2024) for global queue and local queue architectures and two environments: homogenous and heterogeneous. The following scenarios were tested for both architectures and homogenous and heterogenous workers:

1. A total of 3, 6, 9, and 12 workers with a system load of 0.8;
2. A total of 3, 6, 9, and 12 workers with a system load of 0.93.

In a homogeneous environment, all workers have the same mean processing rates. In the heterogeneous environment, the workers were divided into three categories, each with different mean processing rates as follows: Poisson ($\mu = 4$), Poisson ($\mu = 2$), and Poisson ($\mu = 1$) for the fast, medium, and slow categories, respectively.

The mean arrival rate for each scenario was set according to the system load, number of workers, and processing rates. To set the arrival rate to the homogenous set, we needed first to find the arrival rate for the heterogenous set. The given parameters are the number of workers, groups, system load, and the mean processing rates for each group for the heterogenous and the homogenous sets. Here, there the formulated configuration for a given system load is given: K—total number of workers and M—number of groups. λ—Poisson distributed arrival rate.

In a heterogeneous environment, K workers are divided into M groups, each with $g = K/M$ workers with different processing rates of $\mu = 4$, $\mu = 2$, and $\mu = 1$ in fast, medium, and slow groups, respectively, such as $\sum_i^M \mu_i = 7$. Utilization of the heterogeneous system, $\rho(ht)$, is calculated in Equation (1).

$$\rho(ht) = \frac{\lambda(ht)}{g \times \sum_i^M \mu_i} \tag{1}$$

where $\lambda(ht)$ is the arrival rate for the heterogeneous system. Since the parameters $\rho(ht)$, K, M, $\mu_i \forall i = 1, 2, 3$ are set by us, the mean arrival rate of the heterogeneous system is $\lambda(ht) = \rho(ht) \times g \times 7$.

The mean arrival rate for the homogenous set is the same as the mean arrival rate that was found for the heterogenous set as in Equation (2).

$$\lambda(h) = \lambda(ht) \tag{2}$$

In a homogenous environment, each *K* worker has the same processing rate $\mu(h)$. A homogeneous environment has the same arrival rate as the heterogeneneous, which is $\lambda(h) = \lambda(ht) = \rho(ht) \times g \times 7$. The system load of the homogeneous is calculated by Equation (3).

$$\rho(h) = \frac{\lambda(ht)}{K \times \mu(h)} \tag{3}$$

where K is the number of workers. We wanted to evaluate both architectures under the same load, so $\rho(h) = \rho(ht)$.

Therefore, the processing rate of each worker in the homogeneous setting was calculated by Equation (4) to achieve the same utilization system rate.

$$\mu(h) = \frac{g \times 7}{K} \tag{4}$$

For example, in the case of $K = 3$ and $M = 3$, we obtained $g = 1$ (one worker in each group). For $\rho = 0.8$ in each environment, $\lambda = 0.8 \times 0.7$. In a heterogeneous system, the sum of the processing rates was 7; in a homogeneous system, the processing rate of each worker was 7/3 (the sum of processing rates is equal to 7).

We performed the simulation using AnyLogic platform version 8.5. Our implementation was based on AnyLogic's Process Modeling library objects, which include queues and service blocks. Within the simulation framework, tasks represented by the simulator's

agent units proceed through a process flowchart consisting of sequences of operational blocks, such as queues, delays, and resource utilization. The input parameters derived from the simulation included the group types, number of workers, load in the homogeneous system ($\rho(h)$, $\lambda(h)$, $\mu(h)$), load in the heterogenous system ($\rho(ht)$, $\lambda(ht)$, μ_i), queue architecture, dispatching policy, simulation run-time, and the number of replications runs. The λ, μ, and ρ for each simulation were calculated using Equations (1)–(4). The data samples of individual tasks, workers, groups, and the overall system performance were captured into statistics objects to calculate the statistical information. While certain parameters and functions were built-in within the simulator, we wrote custom Java functions that allowed dynamic routing based on the system load, queue lengths, and priority rules. We conducted a series of tests, each lasting 6000 time units after verification of the stabilization of the metrics. To achieve statistical validation and precision of the means, we repeated each test 20 times (replications) to ensure accuracy, resulting in an adjusted relative error of 0.09 (actual relative error of 0.1).

4. Simulation Results

4.1. Homogenous Workers' Simulation Results (GH vs. LH)

This section compares the global and local systems' waiting time and worker utilization when the workers were identical, given the same number of workers and load, and when using random TB. The following tables show the marginal difference in utilization, waiting times, and queue size between both architectures for loads 0.8 and 0.93 when there were 3, 6, 9, and 12 homogeneous workers and random TB.

Tables 1 and 2 demonstrate that all the workers are utilized in the same way. The arrival rate λi of each worker is similar and equals the workload rate. The results were the same for load 0.93 and conformed to the explanation and formalization in Section 2. For both loads, slightly higher task waiting times were demonstrated for workers in the local queue architecture.

Table 1. Global and local system with homogenous workers and load at 0.8.

	GH: Global Queues Homogenous Workers			LH: Local Queues Homogenous Workers		
Number of Workers	**Mean Utilization**	**Mean Waiting Time**	**Mean Queue Size**	**Mean Utilization**	**Mean Waiting Time**	**Mean Queue Size**
3	0.799	0.454	2.548	0.8	0.549	1.025
6	0.798	0.184	2.059	0.801	0.269	0.503
9	0.8	0.103	1.729	0.8	0.169	0.316
12	0.799	0.066	1.482	0.801	0.123	0.231

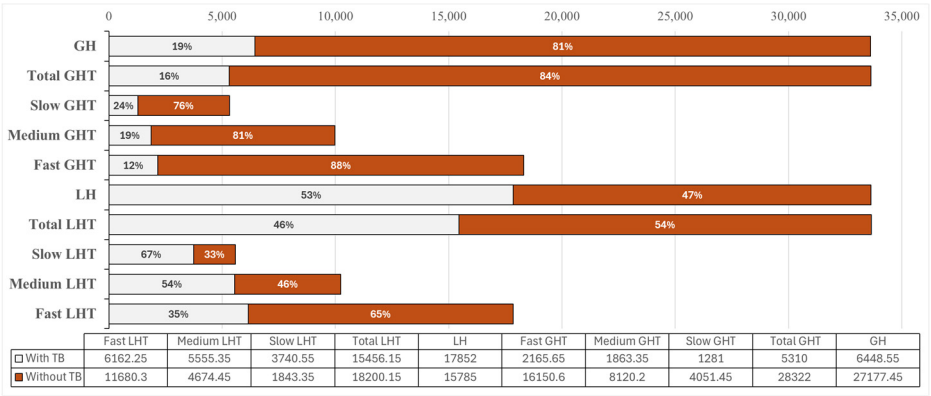
Table 2. Global and local system with homogenous workers and load 0.93.

	GH: Global Queues Homogenous Workers			LH: Local Queues Homogenous Workers		
Number of Workers	**Mean Utilization**	**Mean Waiting Time**	**Mean Queue Size**	**Mean Utilization**	**Mean Waiting Time**	**Mean Queue Size**
3	0.931	1.889	12.326	0.931	1.941	4.215
6	0.93	0.861	11.237	0.929	0.914	1.986
9	0.9333	0.556	10.900	0.928	0.654	1.418
12	0.932	0.390	10.210	0.935	0.555	1.211

The waiting times for the same load are smaller when the number of workers is increased from 3 to 12. We expected the waiting time to remain the same because the number of servers was scaled up with the arrival rate, so the average server utilization stayed the same. For example, the average waiting time for six workers was 0.861 in the global setting and 0.914 in the local setting. The average waiting time for twelve workers was 0.39 in a global setting and 0.555 in a local setting. These results follow the Sakasegawa approximation [33] for estimating the waiting time in an M/M/1 queue with variability in arrival and service processes. We can also see that the differences between waiting times in the global and local architecture are higher when the number of workers increases. To conclude, this section demonstrates that when the workers are homogenous, the waiting times and utilization are the same. The waiting times for LH are slightly higher than those for GH, which is consistent with previous research. The following section focuses on GHT and LHT systems with heterogeneous workers.

4.2. Correlation between Tie-Breaking Events and Waiting Times

This section shows the correlation between the number of TB events and task waiting times when the TB is random. We suspected that the high waiting times and the imbalanced throughput are caused mainly by wrong assignments between workers in the same condition, where a tie-breaking operation is required. Figure 2 shows the number of TB cases found in the different scenarios when the number of workers was three. For the GH case, 19% of the tasks (6448 out of 33,626) were involved in TB cases, and 15.78% were involved in a TB event for the GHT case. As explained before, it is a possible source for the slowdown of the system.



	Fast LHT	Medium LHT	Slow LHT	Total LHT	LH	Fast GHT	Medium GHT	Slow GHT	Total GHT	GH
With TB	6162.25	5555.35	3740.55	15456.15	17852	2165.65	1863.35	1281	5310	6448.55
Without TB	11680.3	4674.45	1843.35	18200.15	15785	16150.6	8120.2	4051.45	28322	27177.45

Figure 2. Number of tie-breaking cases when using random selection in cases of tie-breaking.

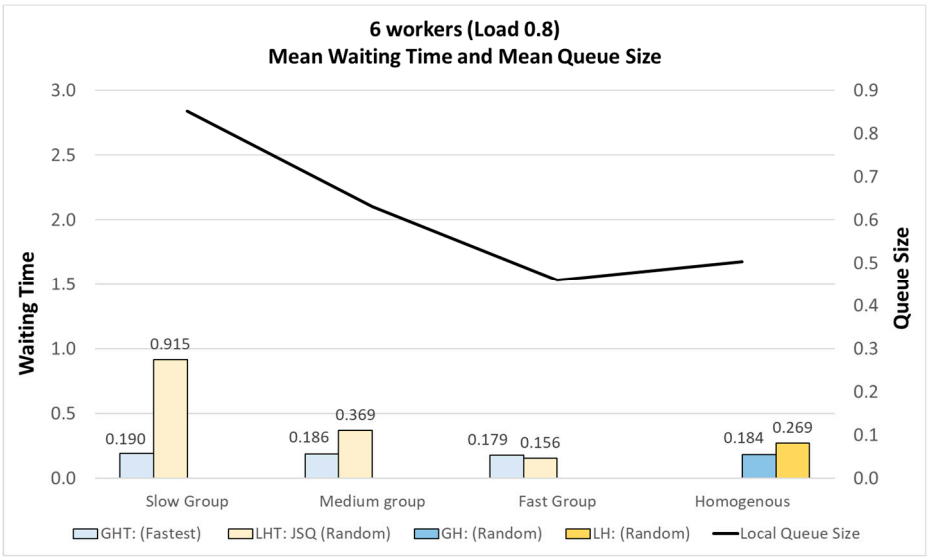
Figure 2 shows the number of TB cases in a system with local queues using the JSQ random TB, which is higher than in a system with one global queue (LH vs. GH). Additional TB events occur when the system is loaded in a local queue system. The length of the queues is the same for several workers, and the dispatcher must select which one will receive the task. Thus, for LH, 53% of the tasks were involved in tie-breaking cases, and 46% were involved in a tie-breaking event in LHT. The percentages of task assignments where the JSQ routing policy had to be chosen among workers with the same conditions were 67% of tasks assigned to the slow group involved in TB, 54% of tasks assigned to the medium group, and 35% to the fast group.

The number of tie-breaking events ($TB\#\ variable$) in each system (GH, GHT, LH, and LHT) and for each category are ranked in the following order:

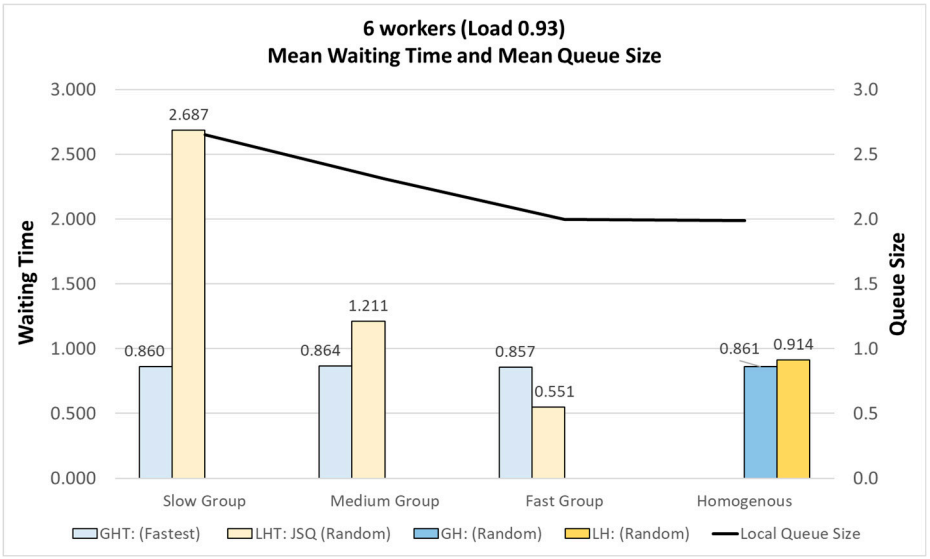
$$TB\#_{GH} \sim = TB\#^{Slow}_{GHT},\ TB\#^{Medium}_{GHT},\ TB\#^{Fast}_{GHT} < TB\#^{fast}_{LHT} < TB\#_{LH} < TB\#^{Medium}_{LHT} < TB\#^{Slow}_{LHT}$$

The following example in Figure 3a for a load of 0.8 and Figure 3b for a load of 0.93 shows the simulation results for the global random TB (GHT) and local JSQ random TB (LHT) when there are six workers. There are three categories of workers, with different

mean processing rates with a ratio of 4:2:1 and inter-processing times of 1, 0.5, and 0.25, respectively. There are two workers in each group. The right bar in the figures shows the waiting times when the workers are homogenous (as in Table 1 in Section 4.1). The global queue shows each group's average waiting times and utilization, including the results when running with homogenous workers (GH, LH). For load 0.8, the queue size is mostly 0 (0.19); hence, the waiting times for local queues are like those in the global queue.



(**a**)



(**b**)

Figure 3. The waiting times for GHT random TB and LHT JSQ random TB. (**a**) Load 0.8; (**b**) load 0.93.

For load 0.93, the queue size is mostly one, and the waiting times in the local setting are higher than in the global setting in each group. The waiting times and the queue size of the local queues using the JSQ random TB differ for each group. They are weighted proportionally to the arrival rates (2.68, 1.2, and 0.5) for the (slow, medium, and fast) group with a load of 0.93. The average wait time of the slow and medium groups of workers is much higher than that for the global queue and the local queue's homogenous setting (right bar). The queue size is empty mainly when the load is 0.8 and about two when the load is 0.93.

Hence, when the queue is mostly empty, and the load is at 0.8, the average waiting times for GH, GHT, LH, and LHT are ranked in the following order:

$$W_{LHT}^{Fast} \sim< W_{GH} \text{ and } W_{GH} \sim= W_{GHT}^{Slow},\ W_{GHT}^{Medium} \text{ and } W_{GHT}^{Fast} \sim= W_{LH} < W_{LHT}^{Medium} < W_{LHT}^{Slow}$$

When the load is 0.93, and the queue size is two on average, the average waiting times for GH, GHT, LH, and LHT are ranked in the following order:

$$W_{LHT}^{Fast} < W_{GH} \text{ and } W_{GH} \sim= W_{GHT}^{Slow},\ W_{GHT}^{Medium} \text{ and } W_{GHT}^{Fast} \sim< W_{LH} < W_{LHT}^{Medium} < W_{LHT}^{Slow}$$

The amount of time you must wait is dependent on the number of TB events that are presented earlier. When you use a simple JSQ, along with random TB in a local queue setup, the wait times for tasks assigned to slow and medium workers can be excessively long. As a result, it is important to consider alternative routing policies to ensure a balance between the waiting times for both slow and fast workers.

4.3. Local Queues Systems with Heterogeneity—Aware Routing Policies

We improved the performance by collecting a better routing policy to select the next worker in tie-breaking cases. First, we checked the FIFO tie-breaking; if several workers had the same conditions, ties were broken using the FIFO order. In this case, there was a timestamp for each worker's queue, which held the latest time when the queue changed its length. For example, assume there are three workers, W1, W2, and W3, with queue lengths of two tasks, and their timestamps are 1, 2, and 3 accordingly, showing the times their queue changed to two. In this case, the task will be assigned to W1. We discovered that there is no advantage to FIFO TB and that the waiting times in global random TB are lower than the waiting times. Random TB and FIFO TB policies improve the average waiting time of the homogenous local queue model. Still, when there are heterogeneous groups of workers, heterogeneity-aware routine policies are required to minimize the waiting time of each worker even though they are assigned to a slow worker.

For the heterogeneity-aware routine policies, we used similar algorithms described in the literature. In the global queue architecture, when there is more than one idle worker, GHT fastest selects the fastest worker among the other workers. This method is known in the literature as the Fast Server First (FSF) allocation strategy. We used the JSQ fast TB (Figure 4), WJSQ (Figure 5) and WJSQL (Figure 6) in the local queue architecture.

JSQ fast TB routing policy
The dispatcher should keep the length of each queue.
The routing policy is invoked at the arrival of a new task and finds the worker with the shortest queue length (as in JSQ).
If there are several idle workers,
    ties are broken by selecting the fastest worker among the fast, medium, or slow possible workers.
Otherwise, if several workers have tasks in the process and their queue is empty,
    ties are broken by selecting the fastest worker among the fast, medium, or slow possible workers.
Otherwise,
    The task is assigned to a worker with a minimal queue length.

Figure 4. JSQ fast TB routing policy.

WJSQ (Figure 5) is a weighted JSQ version that assigns the next task to the worker with the shortest weighted queue. The weighted queue i with queue length Li and the processing rate Si is Li divided by Si. Worker i has the shortest weighted queue if this weighted queue is shorter than the weighted queue of the other workers. Our WJSQ is slightly different than the traditional shortest expected delay (SED) since we added category awareness dispatching and online learning capabilities. Assuming three processing-rate categories to

be slow, medium, and fast, the processing rates for slow, medium, and fast workers in each group were Ps, Pm, and Pf, respectively (assuming Pf:Pm:Ps → 4:2:1). The processing rate ratio between medium and slow was Pm/Ps, the processing rate ratio between fast and slow was Pf/Ps, and the processing rate ratio between fast and medium was Pf/Pm. The queue length ratio between medium and slow queues was Lm/Ls, between fast and slow queues was Lf/Ls, and between fast and medium queues was Lf/Lm.

WJSQ routing policy
If there are several idle workers, ties are broken by selecting the fastest worker.
Otherwise, if several workers have tasks in the process and their queue is empty (Li = 0),
Ties are broken by selecting the fastest worker.
Otherwise (if all workers have a non-empty queue),
for each group of workers, find a worker with a minimum queue length: Ls, Lm, Lf.
Set the following ratios: Pm/Ps, Pf/Ps, Pf/Pm, Lm/Ls, Lf/Ls, Lf/Lm.
If Lf/Ls ≤ Pf/Ps, then the task is assigned to the worker with the shortest queue in the fast group.
Otherwise, if Lm/Ls ≤Pm/Ps, then the task is assigned to the worker with the shortest queue in the medium group.
Otherwise, the task is assigned to the worker with the shortest queue in the slow group.

Figure 5. WJSQ routing policy.

WJSQL routing policy
The dispatcher should maintain the length of each queue.
If there are idle workers, ties are broken by selecting the fastest worker.
Otherwise, if there are workers with tasks in the process and their queue is empty (Li = 0),
Ties are broken by selecting the fastest worker.
Otherwise (if all workers have a non-empty queue),
For each worker, i the system monitors and keeps its current queue length and average processing rate. The calculation Li/Pi is the required time for this worker to empty the queue. The task is assigned to a worker with a minimal processing rate.

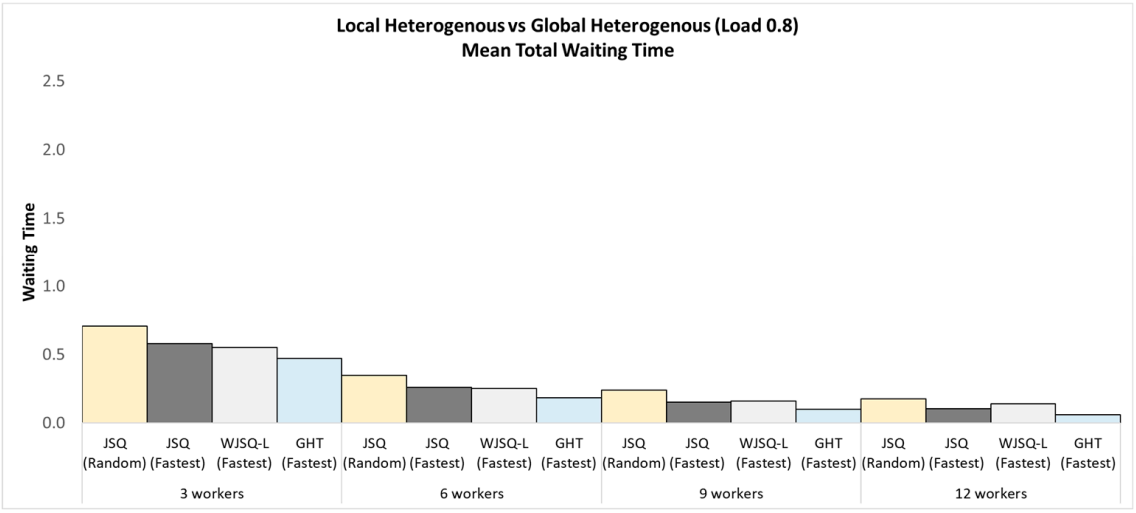
Figure 6. WJSQL routing policy.

The WJSQ policy cannot be used as is since there is no way to know workers' processing rates in advance or assume their distribution. In the following method, termed WJSQL (weighted JSQ with processing rate learning), each worker is monitored, and their average work rate is measured over time and considered appropriately at the time of the dispatcher's selection. The WJSQL routing policy, which is invoked at the arrival of a new task, is presented in Figure 6.

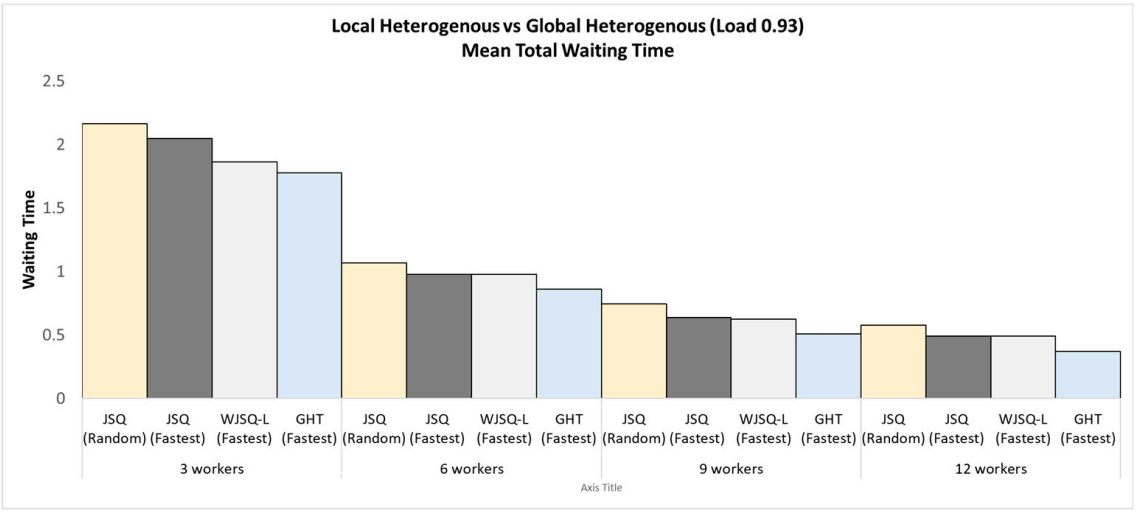
4.4. Heterogenous Workers Simulation Results

This section presents simulations of a system comprising workers divided into three processing rate categories, with a 4:2:1 ratio and inter-processing times of 1, 0.5, and 0.25, respectively. The objective of these simulations is to compare the average utilization of workers belonging to the group and the waiting time for tasks assigned to them across four different systems: global fastest TB, local JSQ random TB, local JSQ fastest TB, and local WJSQL (Figure 6). The performance of a good system is defined by lower waiting times for all workers, regardless of their processing rate.

The graphs in Figure 7a,b show the mean waiting times of all the tasks in the simulation run for each architecture and policy and for 3, 6, 9, and 12 workers when system loads are 0.8 (6a) and 0.93 (6b). GHT fastest demonstrates a slightly lower mean waiting time for any number of workers. The mean waiting times are reduced for each policy when the number of workers is increased. Next, we compare the waiting times for each category separately.



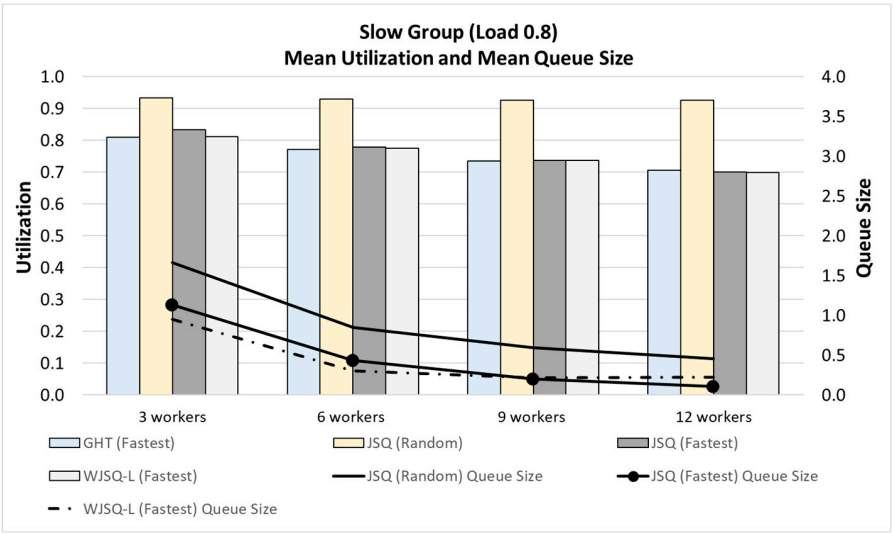
(**a**)



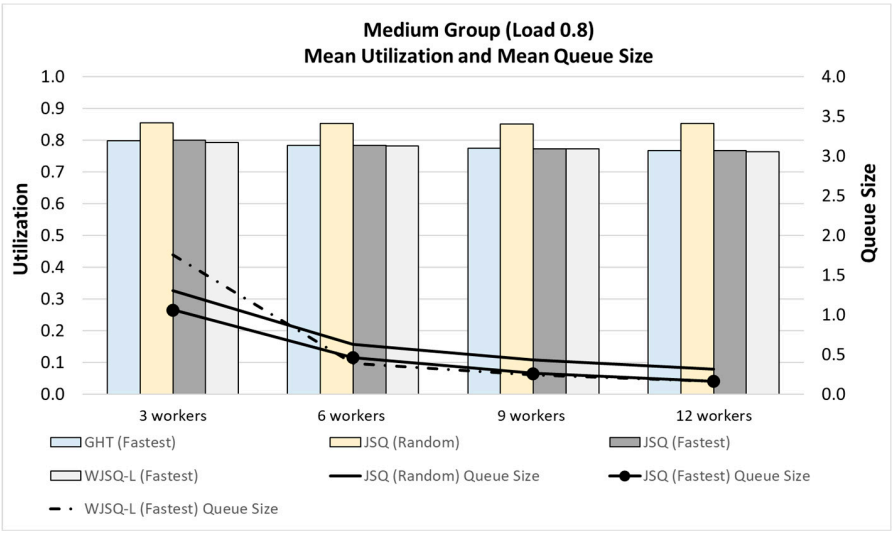
(**b**)

Figure 7. Average waiting time of all the tasks that arrived in the system. (**a**) Load = 0.8; (**b**) load = 0.93.

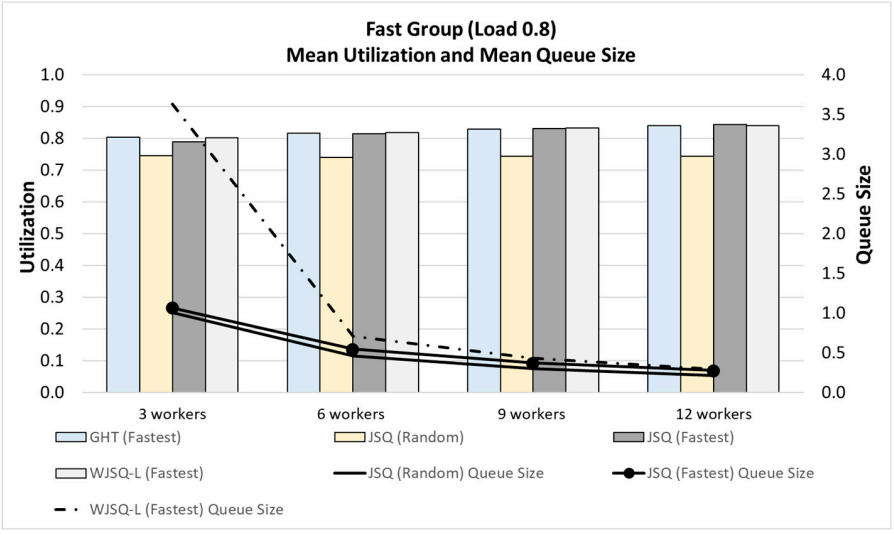
Simulation Results at Load 0.8: Figures 8a–c and 9a–c present the utilization of waiting times (vertical bars), and queue sizes (dashed lines) for slow, medium, and fast workers with 3, 6, 9, and 12 workers when the calculated load was $\rho = 0.8$. The task arrival rates increased proportionally with the number of workers, so the load remained the same for a larger number of workers.



(**a**)

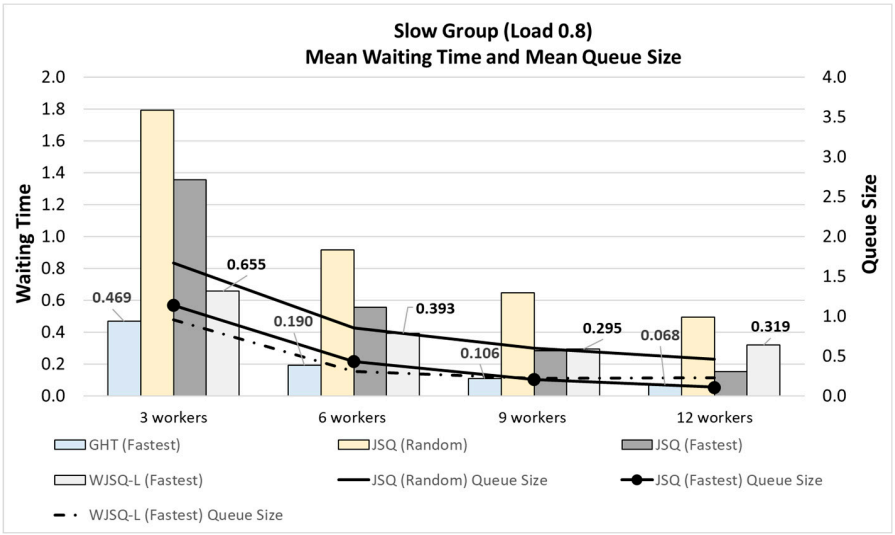


(**b**)

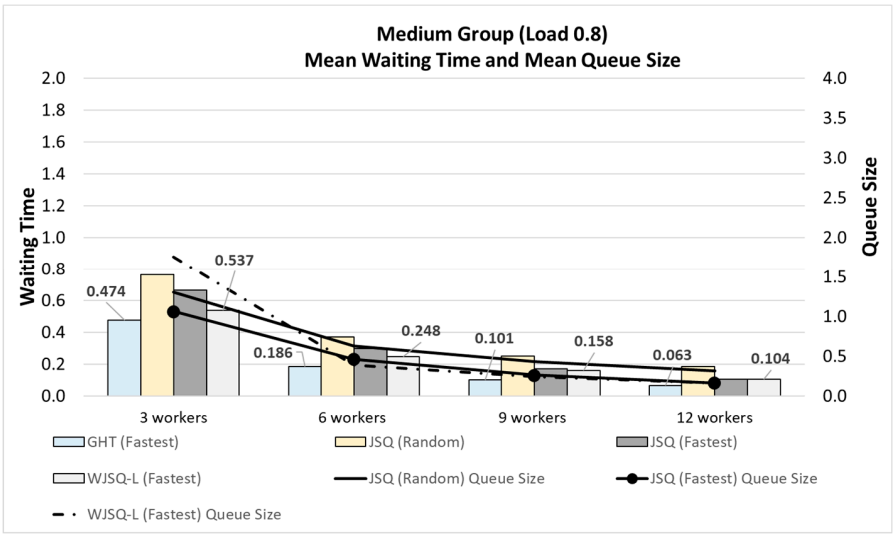


(**c**)

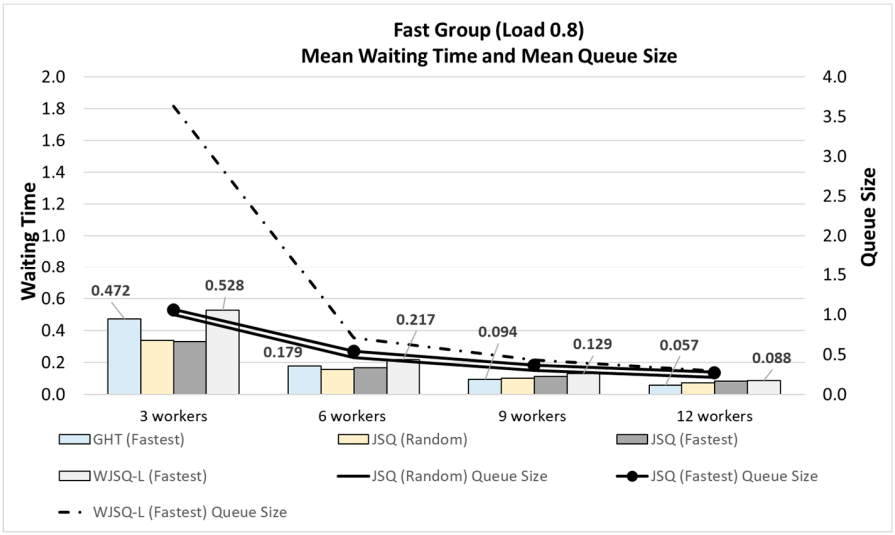
Figure 8. The utilization of workers in the system with different sized groups at load 0.8. (**a**) Slow group; (**b**) medium group; and (**c**) fast group.



(**a**)



(**b**)



(**c**)

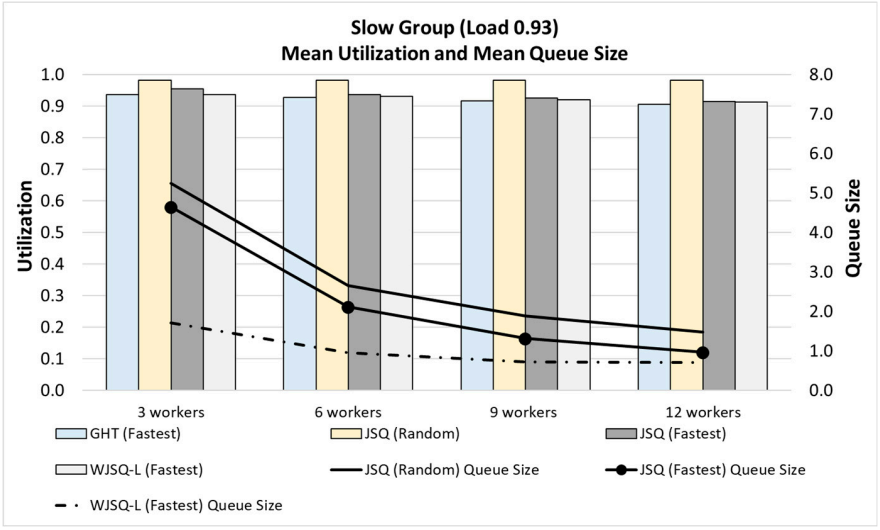
Figure 9. The mean waiting time in the system with different sized groups at load 0.8. (**a**) Slow group; (**b**) medium group; and (**c**) fast group.

We expect the utilization of any worker to be 0.8 as the system load is balanced among the categories. Hence, when the system load is 0.8, there are not enough tasks to keep all the workers busy, the queue size changes between 0 and 1, and some workers are idle part of the time. To observe the utilization of all three categories of one simulation, the reader must combine the views of the three graphs in Figure 8a–c. For example, the utilization, when simulating local random TB with three workers, is depicted for the slow worker in the left yellow column in Figure 8a, the medium worker in the left yellow column in Figure 8b, and the fast worker in the left yellow column in Figure 8c. The JSQ random TB routing policy, the second from the left vertical bar in each processing category and for any number of workers, overloads the slow workers at the expense of the medium and fast workers and does not provide the expected performance. For all the policies, the global queue, the local queue JSQ fastest TB, and WJSQL, worker utilization is changed with the increase in the number of workers. For the simulations with 3 and 6 workers, each worker utilizes 80% of the simulation time, equivalent to the system's load (0.8). The two left bar groups are shown in Figure 8a–c. For the simulations with 9 and 12 workers, the slower workers (Figure 8a) are not fully utilized, while the faster workers (Figure 8c) are overutilized (Figure 8a–c two right bar group). As was shown for the GH and LH systems (Section 4.1 last paragraph), the waiting times in a system with homogenous workers are reduced with an increase in the number of workers, regardless of the architecture and policy used (refer to Figure 9a–c). For instance, in the case of the medium group (Figure 9b), the waiting times in a system with 12 workers are less than in a system with 9 workers, which is less than in a system with 6 and 3 workers. This occurs even when the load is the same while the arrival of tasks is at a higher rate and proportional to the number of workers. The impact is more significant in the heterogeneous case than in the homogeneous case, where the statistical multiplexing is higher. It explains why the utilization for 9 and 12 workers was reduced.

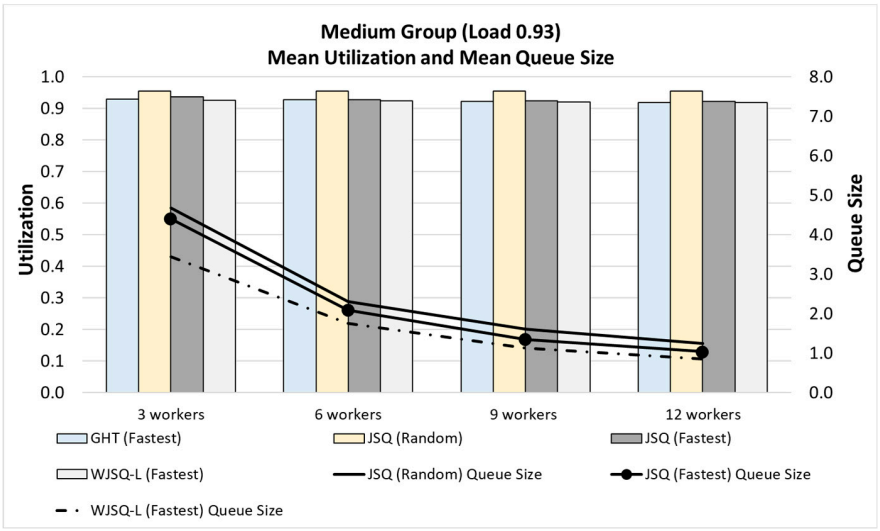
Figure 9a–c presents the waiting times. When running the GHT fastest (most left bar in each group and category), the waiting time for any task, no matter to whom it was assigned, is the same. For example, when there are six workers in the slow (Figure 9a), medium (Figure 9b), and fast (Figure 9c) categories, the average waiting times for tasks are 0.19, 0.186, and 0.179, respectively. This follows the explanation provided in Section 4.1, which states that there is no distinction between workers from different categories in the global queue system. This is because all tasks are in the same queue until the workers retrieve them. The waiting times for GHT fastest are also lower than those for all other policies.

As was expected, the WJSQL is more effective than the JSQ random TB. It is better than the JSQ's fastest TB algorithm, which is only for slow workers and when the number of workers is 3, 6, and 9. As the number of workers increases, JSQ's fastest TB for medium and fast workers outperforms WJSQL and is like the global case. This finding is significant because a simple JSQ is more practical to deploy than WJSQL. The probability of this scenario increases when there are more workers. Additionally, when there are many workers, waiting times and queue sizes decrease, resulting in more idle workers. In such cases, the global or WJSQL has no advantage over the JSQ Fastest TB. WJSQL is more effective than the JSQ random TB and JSQ fastest TB algorithms for managing slow workers and reducing waiting times for assigned tasks. To achieve this, the system must monitor worker performance and apply appropriate policies when workers work slower.

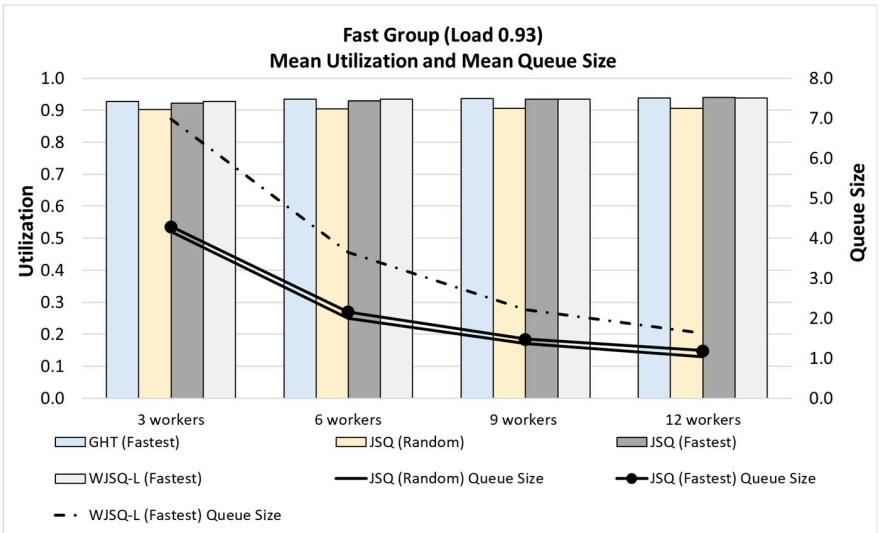
Simulation Results at Load 0.93: Figure 10a–c for utilization and Figure 11a–c for waiting times present the waiting times (vertical bars) and queue sizes (dashed lines) for heterogeneous systems with 3, 6, 9 and 12 workers and mean inter-processing times of (1, 0.5, 0.25) for the slow, medium, and fast groups. The calculated load is $\rho = 0.93$. The system is loaded, and the queue sizes are more extended—there are no idle workers most of the time.



(**a**)

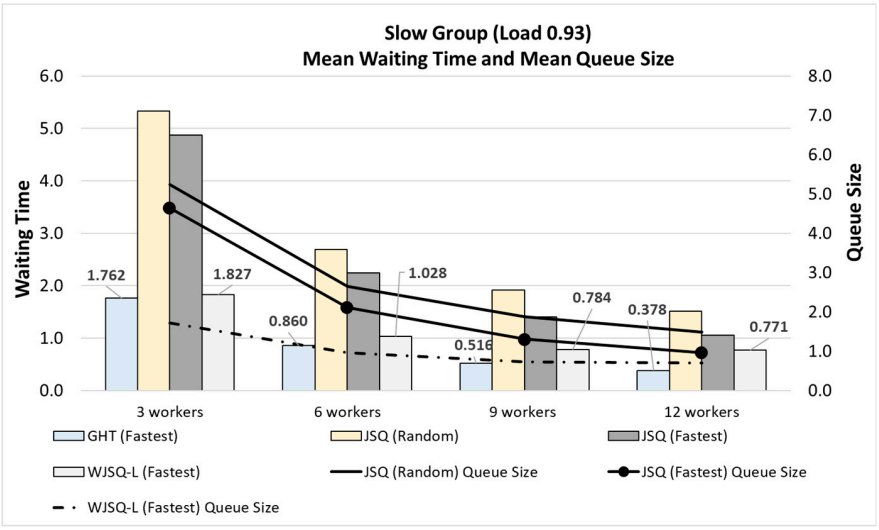


(**b**)

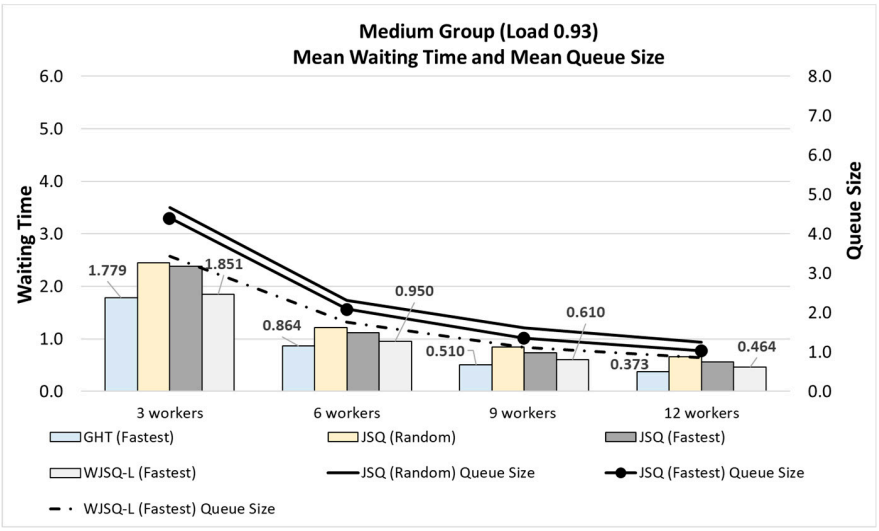


(**c**)

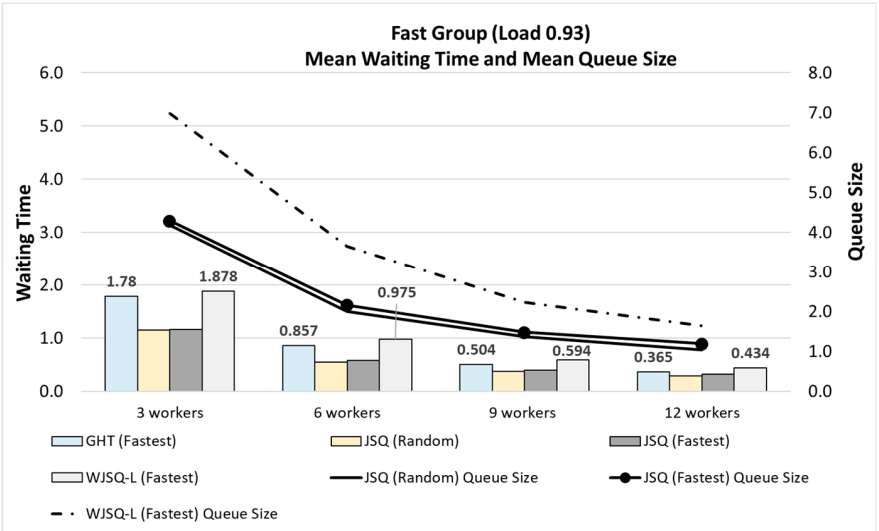
Figure 10. The utilization of the workers in the system with different sized groups at load 0.93. (**a**) Slow group; (**b**) medium group; and (**c**) fast group.



(**a**)



(**b**)

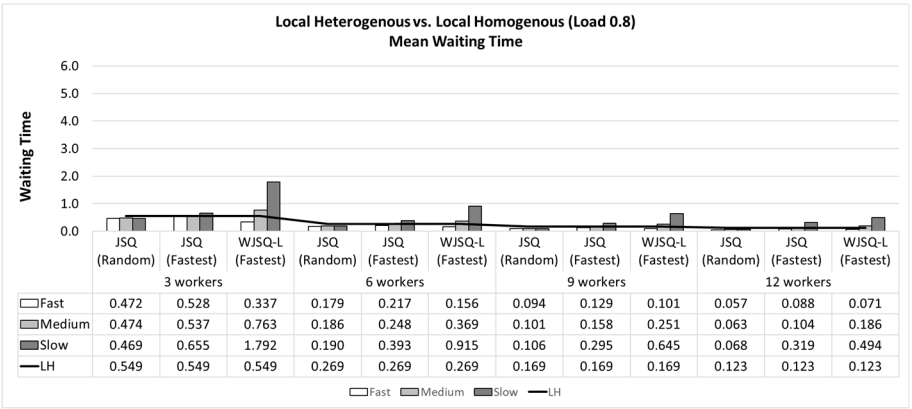


(**c**)

Figure 11. The mean waiting time in the system with different sized groups at load 0.93. (**a**) Slow group; (**b**) medium group; and (**c**) fast group.

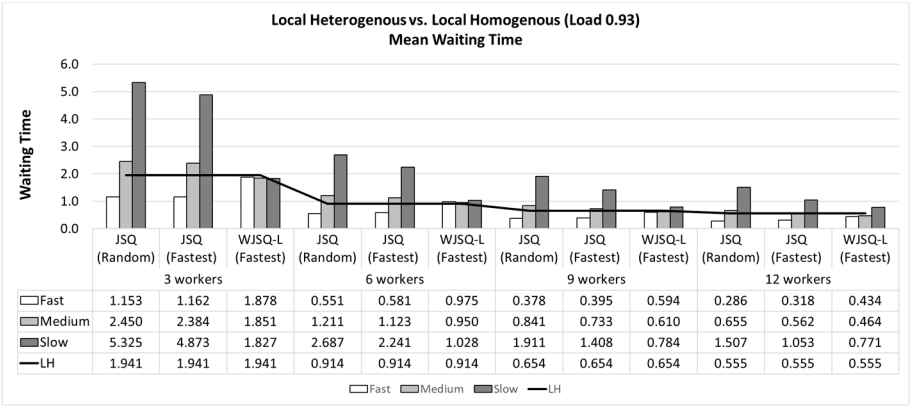
The performance of the global queue for fast workers is not as good as before. This is because the waiting times are longer than all other local policies, indicating that fast workers cannot maximize their potential when the queue is global. In the local queues, the queues represented by the dashed lines are larger than usual but not as large as expected, given the high load of 0.93. The queue of the fast workers when the routing policy is WJSQL is longer than the others, which indicates the good classification of the processing rate and a finer, more accurate assignment to the fast worker. The WJSQL, when the load is 0.93, performs better than the JSQ fastest for the slow and medium workers and also when the number of workers is high. As shown for load 0.8, the waiting times of the tasks assigned to fast workers in a local queue JSQ fastest TB system are lower than those for the WJSQL, highlighting the advantage of JSQ fastest TB. When the load is 0.93, and there are more workers in a system, the waiting times are reduced but not so sharply as for load 0.8. This is demonstrated for all systems except JSQ random TB, especially when the systems are LH and LHT.

Figure 12a,b is focused on the local queue architecture and compares the average waiting times in each group of workers when the architecture is local with JSQ fastest TB, WJSQL, and JSQ random TB. The LH line shows the waiting time for the local homogeneous case. Figure 12b shows that when the load is 0.93, the waiting times shown by the LH line for all routing policies are slower (higher waiting time) than those of the fast worker, like those of the medium worker, and faster by far than the slow workers. The waiting times of the slow workers are far higher and highlight the slow server problem discussed in [10,11]. The WJSQL is lower than the other. WJSQL assigns the tasks well; the queue sizes for the slow and the fast groups are short and long, but the waiting times are similar. When the number of workers is 12, the waiting times for tasks assigned to the slow group are higher than the others.



	JSQ (Random)	JSQ (Fastest)	WJSQ-L (Fastest)	JSQ (Random)	JSQ (Fastest)	WJSQ-L (Fastest)	JSQ (Random)	JSQ (Fastest)	WJSQ-L (Fastest)	JSQ (Random)	JSQ (Fastest)	WJSQ-L (Fastest)
	3 workers			6 workers			9 workers			12 workers		
Fast	0.472	0.528	0.337	0.179	0.217	0.156	0.094	0.129	0.101	0.057	0.088	0.071
Medium	0.474	0.537	0.763	0.186	0.248	0.369	0.101	0.158	0.251	0.063	0.104	0.186
Slow	0.469	0.655	1.792	0.190	0.393	0.915	0.106	0.295	0.645	0.068	0.319	0.494
LH	0.549	0.549	0.549	0.269	0.269	0.269	0.169	0.169	0.169	0.123	0.123	0.123

(**a**)



	JSQ (Random)	JSQ (Fastest)	WJSQ-L (Fastest)	JSQ (Random)	JSQ (Fastest)	WJSQ-L (Fastest)	JSQ (Random)	JSQ (Fastest)	WJSQ-L (Fastest)	JSQ (Random)	JSQ (Fastest)	WJSQ-L (Fastest)
	3 workers			6 workers			9 workers			12 workers		
Fast	1.153	1.162	1.878	0.551	0.581	0.975	0.378	0.395	0.594	0.286	0.318	0.434
Medium	2.450	2.384	1.851	1.211	1.123	0.950	0.841	0.733	0.610	0.655	0.562	0.464
Slow	5.325	4.873	1.827	2.687	2.241	1.028	1.911	1.408	0.784	1.507	1.053	0.771
LH	1.941	1.941	1.941	0.914	0.914	0.914	0.654	0.654	0.654	0.555	0.555	0.555

(**b**)

Figure 12. The waiting times for LH and LHT. (**a**) Load 0.8; (**b**) load 0.93.

5. Discussion and Concluding Remarks

Our simulation results show that GH performs slightly better than LH, but the results are controversial for GHT and LHT. We demonstrate that push mechanisms that manage the assignment procedures and heterogeneity-aware algorithms enhance the waiting time in local queue systems and balance the workload, especially when multiple processing rate categories classify the workers. We compared different architectures and routing policies for groups of 3 to 12 workers categorized by their processing rates and for two load levels: regular (0.8) and high (0.93). Setting the number of processing rates to three categories is important since it reflects the real-world case, and previous work is usually used to simulate only two categories. We discovered that JSQ's fastest TB routing policy for local queues is more effective when assigning tasks to medium and fast workers than global queues. In a local queue environment, the control mechanism makes the assignment of slow workers easier to fix. Our research suggests that WJSQL can reduce the system's average waiting times, particularly the tasks assigned to the slow workers. For all heterogeneity-aware policies, it is crucial to use the actual processing rates of workers by monitoring and applying online learning of the processing rate. The JSQ Fastest TB is simple to implement, knowing the rates.

Given the same load, the simulation results show that all the systems demonstrated decreased waiting times when the number of workers increased. This statistical fact was discussed in the literature but not in the context of global and local queues. We expected the waiting time to remain the same because the number of servers was scaled up with the arrival rate, so the average server utilization stayed the same. In this study, the larger number of workers was 12, but we have already seen that the decrease tends to converge, especially when the load is 0.93. The following research on this subject should include more workers in a real marketplace and find the link between the decrease in wait time and the JSQ policies.

Local queue architecture is a common feature of many systems that use digital servers or human workers as part of their workforce. Applying appropriate routing policies to optimize performance and resource utilization is essential. In environments with fluctuating workloads, exploring various routing strategies is crucial. For example, call centers may need to adapt scheduling based on agents' availability and skills, especially during peak times like lunch hours or evenings. In cloud computing, strategies may focus on the dynamic allocation of computational resources to manage demand spikes efficiently. These variable workloads require adaptive algorithms that adjust scheduling policies based on real-time conditions, highlighting the importance of robust system design to accommodate diverse and dynamic workload scenarios. Our study examined the WJSQL routing strategy and learned workers' processing rates. We plan to conduct further studies to analyze the sensitivity of the WJSQL strategy to process rate changes, enhancing our understanding of its adaptability.

Today, organizations that provide services that use local queues implement local queue infrastructure. For example, a virtual marketplace management system. Message queue (MQ) systems, such as Apache Kafka or RabbitMQ, implement an asynchronous communication pattern between applications operated over different computer systems. The system paradigm follows the publish/subscribe model, where messages are placed in the queue until the recipient pulls them. MQ systems do not provide a service for a local queueing system. After highlighting the advantages and importance of heterogeneity-aware routine policies, we plan to add push control and dedicated architecture into message queues. This will make them suitable for systems with human workers and others.

Author Contributions: Conceptualization, I.A. and M.A.; methodology, I.A.; software, E.T.; validation, E.T.; formal analysis, I.A. and M.A.; writing—original draft preparation, M.A.; writing—review and editing, I.A. and M.A.; supervision, I.A. and M.A.; project administration, M.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Acknowledgments: We are grateful to Vera Tilson for providing valuable feedback and insightful criticism that enhanced the quality and direction of our work. Her contributions were essential to the success of this research.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Angelopoulos, C.M.; Nikoletseas, S.; Raptis, T.P.; Rolim, J. Design and evaluation of characteristic incentive mechanisms in mobile crowdsensing systems. *Simul. Model. Pract. Theory* **2015**, *55*, 95–106. [CrossRef]
2. Pu, L.; Chen, X.; Xu, J.; Fu, X. Crowdlet: Optimal Worker Recruitment for Self-Organized Mobile Crowdsourcing. In Proceedings of the IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, USA, 10–14 April 2016.
3. Jain, A.; Akash, D.S.; Parameswaran, A.; Widom, J. Understanding Workers, Developing Effective Tasks, and Enhancing Marketplace Dynamics: A Study of a Large Crowdsourcing Marketplace. *VLDB Endow.* **2017**, *10*, 829–840. [CrossRef]
4. Ipeirotis, P.G. Analyzing the Amazon Mechanical Turk marketplace. *XRDS Crossroads ACM Mag. Stud.* **2010**, *17*, 16–21. [CrossRef]
5. Do, H.T.; Shunko, M.; Lucas, M.T.; Novak, D.C. Impact of behavioral factors on performance of multi-server queueing systems. *Prod. Oper. Manag.* **2018**, *27*, 1553–1573. [CrossRef]
6. Gumbel, H. Waiting lines with heterogeneous servers. *Oper. Res.* **1960**, *8*, 504–511. [CrossRef]
7. Smith, D.R.; Whitt, W. Resource sharing for efficiency in traffic systems. *Bell Syst. Tech. J.* **1981**, *60*, 39–55. [CrossRef]
8. Grassmann, W.K.; Zhao, Y.Q. Heterogeneous multiserver queues with general input. *INFOR Inf. Syst. Oper. Res.* **1997**, *35*, 208–224. [CrossRef]
9. Alves, F.S.Q.; Yehia, H.C.; Pedrosa, L.A.C.; Cruz, F.R.B.; Kerbache, L. Upper bounds on performance measures of heterogeneous $M/M/c/$ queues. *Math. Probl. Eng.* **2011**, *2011*, 702834. [CrossRef]
10. Rubinovitch, M. The Slow Server Problem. *J. Appl. Probab.* **1985**, *22*, 205–213. [CrossRef]
11. Rykov, V.V.; Efrosinin, D.V. On the slow server problem. *Autom. Remote Control* **2009**, *70*, 2013–2023. [CrossRef]
12. Kleinrock, L. A delay dependent queue discipline. *Nav. Res. Logist. Q.* **1964**, *11*, 329–341. [CrossRef]
13. Sharif, A.; Stanford, D.; Taylor, P.; Ziedins, I. A multi-class multi-server accumulating priority queue with application to health care. *Oper. Res. Health Care* **2014**, *3*, 73–79. [CrossRef]
14. Li, N.; Stanford, D.A. Multi-server accumulating priority queues with heterogeneous servers. *Eur. J. Oper. Res.* **2016**, *252*, 866–878. [CrossRef]
15. Winston, W. Optimality of the shortest line discipline. *J. Appl. Probab.* **1977**, *14*, 181–189. [CrossRef]
16. Weber, R.R. On the optimal assignment of customers to parallel servers. *J. Appl. Probab.* **1978**, *15*, 406–413. [CrossRef]
17. Johri, P.K. Optimality of the shortest line discipline with state-dependent service rates. *Eur. J. Oper. Res.* **1989**, *41*, 157–161. [CrossRef]
18. Gupta, V.; Harchol-Balter, M.; Sigman, K.; Whitt, W. Analysis of join-the-shortest-queue routing for web server farms. *Perform. Eval.* **2007**, *64*, 1062–1081. [CrossRef]
19. Ainbinder, I.; Allalouf, M.; Braslavski, N.; Mahdizada, H. Reducing Marketplace Response Time by Scoring Workers. In Proceedings of the 2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA), Paris, France, 20–22 November 2018.
20. Banawan, S.; Zeidat, N. A comparative study of load sharing in heterogeneous multicomputer. In Proceedings of the 25th Annual Simulation Symposium, Orlando, FL, USA, 6–9 April 1992.
21. Bhambay, S.; Mukhopadhyay, A. Asymptotic optimality of speed-aware JSQ for heterogeneous service systems. *Perform. Eval.* **2022**, *157*, 102320. [CrossRef]
22. Ellens, W.; Kovács, P.; Núñez-Queija, R.; van den Berg, H. Routing policies for a partially observable two-server queueing system. In Proceedings of the 9th EAI International Conference on Performance Evaluation Methodologies and Tools, Berlin, Germany, 14–16 December 2016; pp. 111–118.
23. Vargaftik, S.; Keslassy, I.; Orda, A. LSQ: Load balancing in large-scale heterogeneous systems with multiple dispatchers. *IEEE/ACM Trans. Netw.* **2020**, *28*, 1186–1198. [CrossRef]
24. Gardner, K.; Jaleel, J.A.; Wickeham, A.; Doroudi, S. Scalable load balancing in the presence of heterogeneous servers. *Perform. Eval.* **2021**, *145*, 102151. [CrossRef]
25. Rudolph, L.; Slivkin-Allalouf, M.; Upfal, E. A simple load balancing scheme for task allocation in parallel machines. In Proceedings of the Third Annual ACM Symposium on Parallel Algorithms and Architectures, Hilton Head, SC, USA, 21–24 July 1991.
26. Gupta, V.; Walton, N. Load Balancing in the Nondegenerate Slowdown Regime. *Oper. Res.* **2019**, *67*, 281–294. [CrossRef]
27. Shunko, M.; Niederhoff, J.; Rosokha, Y. Humans are not machines: The behavioral impact of queueing design on service time. Management Science. *Manag. Sci.* **2018**, *64*, 453–473. [CrossRef]
28. Delasay, M.; Ingolfsson, A.; Kolfal, B.; Schultz, K. Load effect on service times. *Eur. J. Oper. Res.* **2019**, *279*, 673–686. [CrossRef]

29. Sunar, N.; Tu, Y.; Ziya, S. Pooled vs. dedicated queues when customers are delay-sensitive. *Manag. Sci.* **2021**, *67*, 3785–3802. [CrossRef]
30. Zhou, W.; Wang, D.; Huang, W.; Guo, P. To pool or not to pool? The effect of loss aversion on queue configurations. *Prod. Oper. Manag.* **2021**, *30*, 4258–4272. [CrossRef]
31. Green, L. Queuing Theory and Modeling. In *Handbook of Healthcare Delivery Systems*; CRC Press: New York, NY, USA, 2011; p. 10027.
32. Anylogic. Available online: https://www.anylogic.com/ (accessed on 15 March 2024).
33. Sakasegawa, H. An approximation formulaL q $\simeq \alpha \cdot \rho \ \beta/(1-\rho)$. *Ann. Inst. Stat. Math.* **1977**, *29*, 67–75. [CrossRef]