Constructive Dissonance in the Cloud: Adaptive Out-of-Phase Scheduling for Periodic Tasks

Tärneberg, William; Skarin, Per

Published in:
2023 IEEE 12th International Conference on Cloud Networking, CloudNet 2023

DOI:
10.1109/CloudNet59005.2023.10490059

2023

Document Version:
Publisher's PDF, also known as Version of record

Constructive Dissonance in the Cloud: Adaptive Out-of-Phase Scheduling for Periodic Tasks

William Tärneberg[1] and Per Skarin[2]
[1]Department of Electrical and Information Technology, Lund University, Sweden
[2]Ericsson AB, Sweden

***Abstract*—In the realm of cloud computing, the prevalence of intra-phase congestion due to concurrent periodic requests from a multitude of clients frequently results in heightened resource consumption, elongated queuing times, and prolonged response durations. To tackle these challenges, this paper presents a groundbreaking 'Adaptive Out-of-Phase Scheduling' technique, utilizing a server-side PID controller to induce constructive dissonance, strategically phase-shifting clients by delaying subsequent requests based on server feedback, leading to dispersed out-of-phase transitions. The hallmark of our method is its uncomplicatedness, obviating the necessity for intricate execution analysis or elaborate optimal scheduling decisions. Empirical evaluation on an operational web server demonstrates the efficacy of this technique in mitigating resource utilization, reducing response time by half, and significantly decreasing response time variability. Our technique offers an efficient and practical solution to intra-phase congestion in periodic tasks, optimizing both system performance and resource efficacy in diverse cloud computing environments. Our method is scalable, versatile, and aptly adapted for real-world applications, with potential future research exploring fine-tuning and trade-offs in various system setups. By proficiently mitigating the ramifications of simultaneous periodic tasks in an efficient and streamlined fashion, our approach is primed to advance the evolution of more responsive and optimized cloud computing infrastructures, yielding advantages across a wide range of applications and services.**

***Index Terms*—Cloud computing, Scheduling, Intra-period congestion, Self-adaptive, Response time**

I. Introduction

In the domain of cloud computing, intra-period congestion, characterized by the simultaneous overlap of multiple clients' requests within a certain time frame, poses significant challenges. This coinciding of requests leads to congestion, subsequently causing elevated resource consumption, extended queuing times, and notably diminished response times, a phenomenon known as "destructive harmony." The repercussions of this issue are extensive, impacting both the theoretical and applied aspects of cloud computing. The improper management of intra-period congestion can escalate operational costs and degrade system efficiency, especially in critical applications where reliable and prompt response times are crucial. Implementing proficient out-of-phase scheduling for periodic tasks has the potential to enhance the performance and resource allocation of cloud systems, consequently reducing median response times, minimizing variability, and bolstering overall system stability.

Addressing intra-period congestion is intricate, primarily due to the stochastic nature of requests from multiple clients, which lack inherent phase-shift patterns and often lead to congestion. The dynamic characteristics of client arrivals, departures, and varying request frequencies further complicate workload prediction and management. Solutions need to be efficient, scalable, and easily integratable, imposing minimal overhead on the client-side. It is crucial to implement gradual phase shifts of clients to avoid disrupting the system. Premature scheduling strategies can jeopardize system stability, while overly gradual transitions can decelerate control processes. Therefore, a profound comprehension of system dynamics and the application of appropriate control mechanisms are imperative to address the fundamental causes of intra-period congestion effectively. A dynamic approach, which observes average system behavior and adjusts accordingly over time, is essential as decisions based on isolated instances may be ineffective in a sustained context.

Historically, solutions have predominantly aimed at instilling deterministic behavior in cloud systems, often overlooking explicit consideration of resource consumption and failing to efficiently alleviate intra-period congestion. Conventional scheduling strategies usually focus on ensuring fairness and meeting control objectives on the server-side after requests are received, rather than addressing the initial overlap of client requests proactively. In contrast, our innovative proposal, "Adaptive Out-of-Phase Scheduling for Periodic Tasks," distinguishes itself through its simplicity and practicability, obviating the need for meticulous analysis of application execution or complex scheduling decisions. It strategically observes the traffic flow in and out of a service, making it a practical addition to existing service meshes, including Kubernetes [11]-based solutions like Istio [10]. By leveraging constructive dissonance, our method proactively phase-shifts clients, optimizing resource allocation and mitigating system congestion.

We employ a lightweight server-side PID (Proportional-Integral-Derivative) controller in our approach, which monitors intra-period congestion and subtly phase-shifts each client's request periods. Clients need only delay their subsequent requests as per the server's guidance, ensuring smooth and distributed phase-shifting behavior. Our method, rigorously

This work has been partially funded by the Wallenberg AI, Autonomous Systems and Software Program (WASP), the ELLIIT strategic research area on IT and mobile communications, Sweden's Innovation Agency (VINNOVA) under the IMMINENCE Celtic Netxt project, the Swedish Foundation for Strategic Research under the SEC4FACTORY project.

validated using a live web server, has proven effective in optimizing resource allocation, halving response times, and significantly reducing response time variance, indicative of enhanced system stability and performance. Additionally, our technique has demonstrated its efficacy and adaptability under varying system loads and levels of client participation, highlighting its applicability across diverse operational contexts.

II. Related Work

A plethora of studies have delved into the realm of scheduling techniques applicable to cloud computing and cyber-physical systems. Notably, Rajagopalan et al. [15] unveiled a hybrid firefly-genetic algorithm tailored for task scheduling within cloud computing environments. Complementing this, Kumar et al. [12] undertook a detailed survey on static and dynamic scheduling, meta-heuristics, and genetic algorithms. Ijaz et al. [9] turned their attention towards optimizing energy-makespan of workflow scheduling in fog-cloud computing settings. In the arena of real-time systems, Lehoczky [13] introduced a fixed priority scheduling algorithm for handling periodic task sets with arbitrary deadlines. Meanwhile, Cervin et al. [4] put forth a feedback-feedforward scheduling method for control tasks, employing feedback control to fine-tune scheduling decisions based on system behavior. Approaching performance optimization of control applications on fog computing platforms, Barzegaran et al. [2] relied on scheduling and isolation techniques.

Houssein et al. [8] provided a meta-heuristic-based review of task scheduling techniques in cloud computing, offering a comprehensive taxonomy of existing approaches and highlighting unresolved challenges and prospective trends. In contrast, Dai et al. [5] proposed a period adaptation strategy for real-time control tasks with fixed-priority scheduling in cyber-physical systems. In the context of web server scheduling, Harchol-Balter et al. [7] put forward a size-based scheduling approach leveraging the Shortest Remaining Processing Time (SRPT) policy to enhance web server performance. Echoing this theme, Bini and Cervin [3] proposed a delay-aware period assignment technique for control systems, aiming to allocate suitable periods to control tasks based on their delay sensitivity. Our approach to overlapping periodic tasks supplements [3] by introducing a distributed and adaptive scheduling mechanism for cyber-physical systems in a cloud computing environment.

A significant body of research has proposed various solutions for managing latency-critical cloud services within data centers. Nishtala et al. introduced Twig, a scalable Quality of Service (QoS) conscious task manager that harnesses deep reinforcement learning to profile tail latency using hardware performance counters and guide energy-efficient task management decisions within data centers [14]. Twig was found to outshine previous works by reducing energy usage by up to 38% while attaining up to 99% QoS guarantee for latency-critical services [14]. Alhussian et al. crafted a computing architecture and algorithms to facilitate soft real-time task scheduling in a cloud computing environment through dynamic provisioning of virtual machines [1]. Their architecture integrated three modified soft real-time task scheduling algorithms and a deadline look-ahead module to maintain system criticality and avoid missed deadlines [1]. In another notable study, Santhosh et al. presented a preemptive online scheduling approach for real-time tasks using the "Infrastructure as a Service" model in cloud computing [16]. Their innovative algorithm includes task migration to minimize response time and enhance efficiency, and surpasses traditional scheduling algorithms like Earliest Deadline First (EDF) in terms of system performance and utility [16]. However, it is worth mentioning that such solutions often incur substantial costs and necessitate deep introspection, without explicitly addressing the central issue of intra-period congestion.

Duan [6] conducted a comprehensive survey on the performance evaluation of cloud services, shedding light on current trends, challenges, and opportunities, which aligns closely with our scheduling techniques within cloud computing. Despite these strides, existing strategies fall short as they fail to address the crux of overlapping periodic tasks and only focus on scheduling admitted tasks. In stark contrast, our methodology eradicates overlapping tasks through a distributed and adaptive scheduling mechanism. Furthermore, traditional approaches overlook distributed and voluntary task scheduling. However, our methodology thrives on the basis of voluntary task scheduling, thereby facilitating adaptive participation according to individual requirements. In summation, our strategy triumphs over these limitations by offering a distributed, adaptive, and voluntary scheduling mechanism for overlapping periodic tasks, catering to both cloud computing and cyber-physical systems.

III. Problem Description

The system in focus consists of a service receiving periodic requests from multiple clients. Operating within a cloud platform, this service is subject to inherent uncertainties [6] and allows shared information access and collective decision-making among all clients. Clients, operating independently, generate requests and process responses, with their effective functioning contingent upon the receipt of concise and consistent response times.

A. System Model

The model of our target system is illustrated in Figure 1. It comprises N autonomous clients, denoted as $c^i, i \in N$, which periodically transmit requests to a shared *service*. Each client follows the same request period, denoted as h, yet their requests are not necessarily synchronized in time. The k^{th} request sent by the i^{th} client is referred to as r_k^i. The point in time when this request is transmitted by the client is given by $t_c(r_k^i)$.

For each request, the client receives a corresponding response, denoted as $\hat{r}_k^i$, with $t_c(\hat{r}_k^i)$ indicating the moment this response is received by the client. The moments of arrival of the request at the service and the subsequent response's

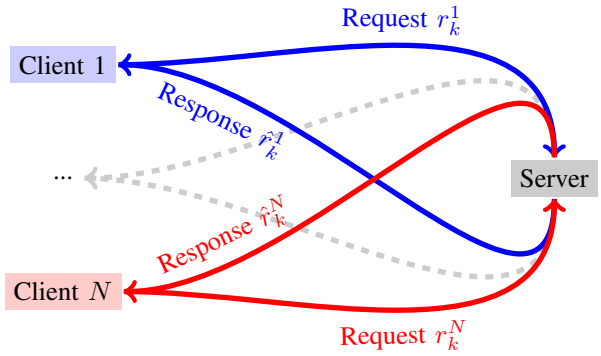

Fig. 1. System Model Representation: The figure represents a scenario where N clients transmit requests, denoted as r_k^1, r_k^N, to a server via a network, and in return, the server sends responses, denoted as $\hat{r}_k^1, \hat{r}_k^N$, back to the clients. This illustrates the interaction and communication between multiple clients and the server in the system.

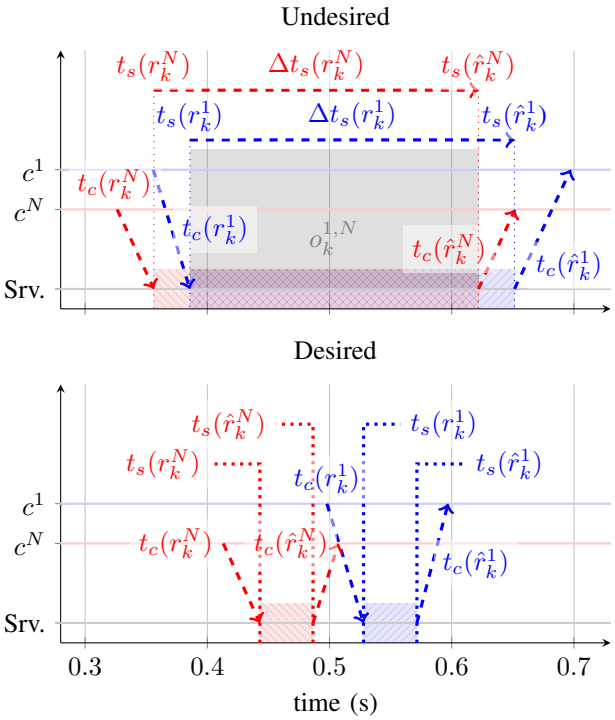

Fig. 2. Illustration of Desired and Undesired Intra-Period Congestion Scenarios. The top panel depicts an undesired situation where requests from clients c^1 and c^N overlap, leading to intra-period congestion and extended residence time in the server. This overlap results in increased competition for resources and longer response times. Conversely, the bottom panel represents a desired scenario where requests are staggered in time, preventing overlap in the server and reducing residence time. This optimal spacing of requests alleviates competition for resources, improves response times, and enhances overall system performance.

departure from the service are represented by $t_s(r_k^i)$ and $t_s(\hat{r}_k^i)$, respectively. The response time for the request r_k^i is thus given by $\Delta t_c(r_k^i) = t_c(r_k^i) - t_c(\hat{r}_k^i)$, and the processing time of request r_k^i at the service is denoted as $\Delta t_s(r_k^i) = t_s(r_k^i) - t_s(\hat{r}_k^i)$.

For each request, the client receives a corresponding response, denoted as $\hat{r}_k^i$, with $t_c(\hat{r}_k^i)$ indicating the moment this response is received by the client. The moments of arrival of the request at the service and the subsequent response's departure from the service are represented by $t_s(r_k^i)$ and $t_s(\hat{r}_k^i)$, respectively. The response time for the request r_k^i is thus given by $\Delta t_c(r_k^i) = t_c(r_k^i) - t_c(\hat{r}_k^i)$, and the processing time of request r_k^i at the service is denoted as $\Delta t_s(r_k^i) = t_s(r_k^i) - t_s(\hat{r}_k^i)$.

B. Challenge

The principal challenge arises when the arrival times of requests from different clients are closely aligned, i.e., $t_s(r_k^i) \approx t_s(r_k^j)$, for any pair of clients i and j. This scenario can precipitate intra-period congestion at the service, resulting in protracted response times.

This issue emanates from the concurrent execution of tasks on shared resources, such as shared memory models, execution within disparate virtual machines, fair scheduling devoid of real-time support, or shared networks, among other factors. When multiple requests converge nearly simultaneously on the service, a competition for system resources ensues, inducing processing delays and subsequently, extended response times for all the implicated requests.

IV. Proposed Solution

Overlapping requests in periodic workloads can be mitigated by adopting a simple and dynamic approach that does not require complex and deep introspection for optimal scheduling. As the systems are stochastic, our solution focuses on the average behavior and chooses to adopt a dynamic approach with a PID (Proportional-Integral-Derivative) controller. The PID controller observes the system load and gradually phase-shifts clients' request periods to reduce or eliminate overlap, effectively phasing them apart in time to reduce intra-period congestion and improve system performance due to resource contention.

In this section, we present a scalable, lightweight, and adaptive solution that utilizes a feedback controller to dynamically adjust the start of each client's request periods, as illustrated in Figure 2. The upper part of the figure shows an example of overlapping requests within a single period, while the lower half shows the desired effect achieved by the proposed solution. This approach allows for efficient and effective mitigation of overlapping requests in periodic workloads without the need for complex introspection or optimal scheduling decisions, making it a simple yet powerful solution for improving system performance.

The solution dynamically phase-shifts clients' request periods based on the degree of overlap at the server/service. The degree of overlap is denoted as o_k^i for client c^i at period k and is evaluated at every ingress request using Equation 1. This equation gives the sum of the fraction of overlap with all other present clients. $s_k^{i,j}$ is the sign ± 1 that gives the offset

a direction. This ensures that clients, on average, move in opposite directions. The nominator in Equation 1 calculates the difference between the minimum of the observed time stamps of the requests and the maximum of the observed time stamps of the responses. This value is divided by the processing time of the request of client i.

$$o_k^i = \sum^{j \in \mathcal{N}-i} s_k^{i,j} \frac{\min(t_s(r_k^j), t_s(r_k^i)) - \max(t_s(\hat{r}_k^j), t_s(\hat{r}_k^i))}{\Delta t_s(r_k^i)} \tag{1}$$

The sign is determined as

$$s_k^{i,j} = \begin{cases} 1 & \text{if } t_s(r_k^j) \leq t_s(r_k^i) \\ -1 & \text{if } t_s(r_k^j) \geq t_s(r_k^i) \\ \{-1, 1\} & otherwise \end{cases} \tag{2}$$

The feedback controller used in the proposed solution is a Proportional-Integral (PI) controller, which gradually and dynamically adjusts the phase offset for each client. The PI-controller produces a unique offset for each client that is included in the response, without making instantaneous corrections. The reactivity of the controller can be tuned to achieve a more rapid correction if needed.

The proposed solution utilizes a PI-controller for feedback control to dynamically and gradually phase-shift clients' request periods based on the degree of overlap, addressing the challenge of overlapping requests. This approach offers a scalable and adaptive solution for reducing or eliminating overlap in the system, resulting in improved response times and efficient resource utilization.

As discussed in Section III, the proposed solution involves dynamically and gradually phase-shifting clients' periods to reduce or eliminate request overlap and minimize response times. A scalable, lightweight, and adaptive solution is needed to handle multiple independent dynamic systems. The solution evaluates the degree of overlap for each client at every ingress request and incorporates the resulting phase offset in the response. A feedback controller, specifically a PI-controller, is employed at the service to calculate a unique offset, composed of both a magnitude and offset, for each client, which is then applied dynamically and gradually. The PI-controller can be tuned to produce a more rapid correction if desired.

To reduce the impact of noise, the overlap is averaged over a window of a set of consecutive samples. The offset for client i at period k is denoted as ϕ_k^i and is calculated based on the following PI-controller equation:

$$\phi_k^i = K_p \cdot \hat{o}_k^i + K_i \sum_{k=0}^{k} \hat{o}_k^i \cdot \Delta t \tag{3}$$

where $\hat{o}_k^i$ represents the averaged overlap for client i at period k, K_p is the controller proportional gain, K_i is the controller integral gain, and Δt is the time interval between samples.

Finally, for c_{k+1}^i, the $k^{th}+1$ phase is offset by ϕ_k^i using the following equation:

$$t_{k+1}^i = \min(\max(t_k^i + h + \phi_k^i, 0), t_k^i + 2 \cdot h) \tag{4}$$

where t_k^i is the current phase of client i, and the phase-shift is bounded to prevent it from being projected back in time or exceeding a whole period. Additionally, if no other offset is applied, $\phi_k^i = 0$ is used.

V. Experimental setup

In this part, we carry out a number of experiments. Our aim is to assess the effectiveness of our solution, which we outlined in Section IV, in addressing the problem that we discussed in Section III. We've established a test-bed that mirrors real-world scenarios. The solution's performance will be assessed based on a series of metrics that represent the outcomes of the experiments.

A. Metrics

To assess the solution, we have chosen the following metrics:

- **Response time,** Δt: This is expressed in milliseconds and is shown as the 95th, 75th, and 50th percentiles of all request and response exchanges over a 1-second sliding window. A lower response time is preferable as it suggests quicker processing of requests. A drop in response time from $t = 0$ would be an indicator of our solution's success in improving system performance.
- **Overlap,** $\hat{o}$: This is given as a percentage and shown as the 95th, 75th, and 50th percentiles of all calculated overlaps over a 1-second sliding window. Overlap measures the degree of overlap between clients' periods, and we desire a lower value of overlap. A drop in overlap from $t = 0$ would suggest that our solution is successful in reducing competition among clients.
- **Offset,** ϕ: This is displayed as the 95th, 75th, and 50th percentiles of all calculated offsets over a 1-second sliding window. Offset measures how far the clients' periods deviate from the ideal, non-overlapping periods, with an ideal offset of 0. We desire that the offset approaches 0 over time, suggesting that the overlap has been eliminated. A drop in offset from $t = 0$ would suggest our solution's success in achieving non-overlapping periods for clients.

B. Experiments

The following experiments are designed to validate the solution's ability to reduce overlap, decrease response times, and adapt to changes dynamically. The computational time required by the solution is also evaluated in each experiment to assess its efficiency.

A client is considered to be participating if it alters its period based on the offset ϕ_k^i returned by the service (as detailed in Equation 4). When a client does not participate, it does not alter its period. Note that the participation of each client is independent, hence, other clients can continue to participate.

All experiments start with the proposed solution activated at $t = 20s$, where t denotes the experiment time. Each experiment is performed 200 times to ensure statistical significance.

- **Participation**: The purpose of this experiment is to validate the solution by comparing outcomes with the solution being active and inactive. Ten clients, denoted as $N = 10$, participate in the system. The experiment is run in seven stages, each time increasing the number of participating clients from 0 to 6. The outcome with no participation serves as a benchmark for comparison, allowing us to evaluate the effectiveness of the solution by contrasting the reduction in response time achieved as participation increases.
- **Load**: This experiment is designed to assess the system's capacity with and without the solution enabled. It aims to determine the point at which the system reaches its capacity (when the number of incoming requests matches the system's capacity, $\rho = 1$). Additionally, we explore if our solution has a positive effect on throughput, as theoretically, phase-shifting clients' periods could allow for a higher system capacity.
- **Disturbance**: This experiment is designed to assess the solution's adaptability. Here, all clients are participating. The experiment starts with three clients and the solution is given sufficient time to phase-shift and improve the system's response time. At $t = 120s$, an additional four participating clients are introduced, simulating a sudden change in the system's state. Our solution should be able to handle this disturbance and achieve a net improvement in response time.

Finally, we used pidstat in our experiments to monitor a range of performance metrics including waiting time, context switching, page faults, and memory footprint, along with CPU utilization. Although we did not anticipate a significant reduction in CPU usage, we did expect to observe reductions in the other metrics. Lower values in these metrics indicate improved system performance and efficient resource utilization.

C. Experimental Setup

The experiments were meticulously designed to comprehensively evaluate the proposed solution, focusing on the PI-controller's effectiveness and the generalizability of the results in real-world scenarios.

Test-Bed and Network Simulation: A test-bed was configured using Python scripts with the AIOHTTP library for both clients and the service, conducted on a dedicated Ubuntu machine to ensure stability and reliability. Communication between clients and the service was simulated over the loopback interface, with a one-way delay of 10 ms added using Netem to emulate network traversal effects and assess the solution's adaptability to varied network conditions.

Service Load and Client Variability: A CPU-intensive Python benchmark was utilized to simulate the computational demands of cloud services and analyze the solution's scalability. Clients were not assigned specific start times, introducing variability and randomness to the experiments, reflecting the unpredictability of real-world client requests.

Controller Configuration and Optimization: The PI-controller was configured with general parameters, $K_p = 0.05$ and $K_i = 0.005$, but specific tuning proportional to the size of the phase was applied, acknowledging that further optimization could enhance performance. This configuration was aimed at demonstrating the solution's versatility and broad applicability.

Openness and Reproducibility: To ensure the research community can validate and extend our work, the source code for the solution and the experiments will be publicly available through a Git repository upon publication.

Conclusion of Setup: This setup was intended to provide a robust and valid foundation for evaluating our solution, addressing concerns about the reproducibility and validity of the results and contributing valuable insights to the field of task scheduling and cloud computing.

VI. Results

This section delivers an in-depth analysis of our experimental findings, as set out in Section V. We delve into the crucial performance metrics obtained from our experiments, and offer a comprehensive evaluation of our proposed scheduling approach. Further, we position our results in contrast to the baseline and comparable studies, elucidating the distinct advantages and effectiveness of our method.

Our research addresses the pertinent issue of overlapping periodic tasks. By employing our proposed approach, we have witnessed considerable improvement in the management of such tasks. This advance highlights the capability of our method to effectively address this common problem in task scheduling, enhancing overall system performance.

Additionally, our approach exhibits significant strengths in facilitating distributed and adaptive scheduling. As evidenced by our experimental results, our methodology demonstrates robustness in dynamic environments, adapting seamlessly to varying workloads and system demands. This ability to adapt distinguishes our approach, making it an attractive solution for distributed environments.

We also propose a voluntary approach for task scheduling in real-time systems. By providing more flexibility, our method allows for a better distribution of tasks based on system needs and task priorities, leading to improved system responsiveness and overall performance.

In contrast with the baseline and related studies, our method provides superior outcomes, underlining the merits of integrating feedback control theory into task scheduling for cloud computing environments. These experimental results underscore the potential of our approach in optimizing task scheduling, offering notable advancements over traditional methods.

A. Participation

To evaluate our proposed phase shifting solution, we executed a series of experiments, comparing outcomes with and

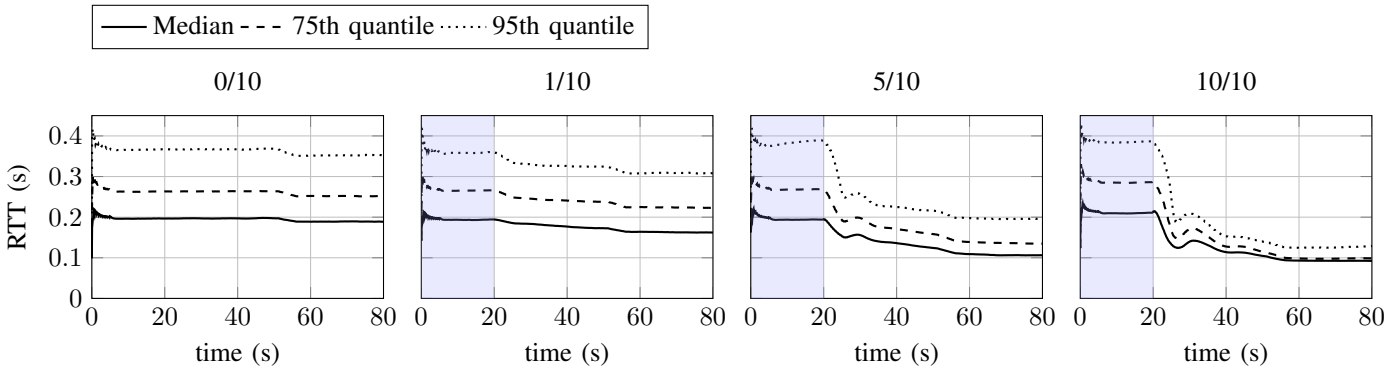

Fig. 3. Illustration of Client Participation Levels: This figure represents the enhancement in system performance with the increased number of clients participating in the phase-shifting solution, illustrating its scalability and effectiveness in reducing intra-period congestion. Scheduling is not applied until $t = 20$, demonstrating the solution's adaptability.

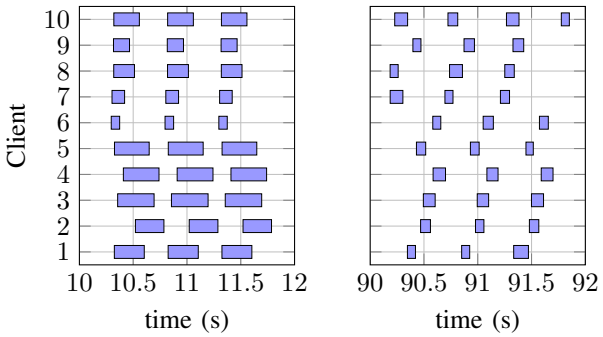

Fig. 4. Comparison of Client Request Duration and Overlap: This figure offers a snapshot of the system before and after the activation and convergence of phase shift, depicting the substantial reduction in request duration and overlap, illustrating the effectiveness of the phase-shifting solution in mitigating intra-period congestion and optimizing system performance.

without our method's implementation. Within these experiments, we had a total of ten clients ($|N| = 10$) continually dispatching requests to the server. These clients participated in the phase shifting algorithm to varying degrees.

Figure 3 displays the results, wherein the level of client participation ranged from 0 to 10 out of 10 clients. The left chart depicts a baseline system where no clients were engaged in the phase shifting solution, while the right chart illustrates the scenario where all clients were involved. It is pertinent to note that the phase shifting solution was not activated until 20 seconds into the experiment, which is denoted by the vertical dashed line in the charts.

As anticipated, the baseline system without phase shifting saw a fairly constant response time throughout the experiment, suggesting overlapping requests and consequent elevation in response times. Conversely, with the phase shifting solution's activation (at t=20), the response time started to decrease rapidly and consistently, as demonstrated in the right chart. Remarkably, the median response time was reduced by 60ms, and the 95th percentile response time was halved, signaling substantial system performance enhancement.

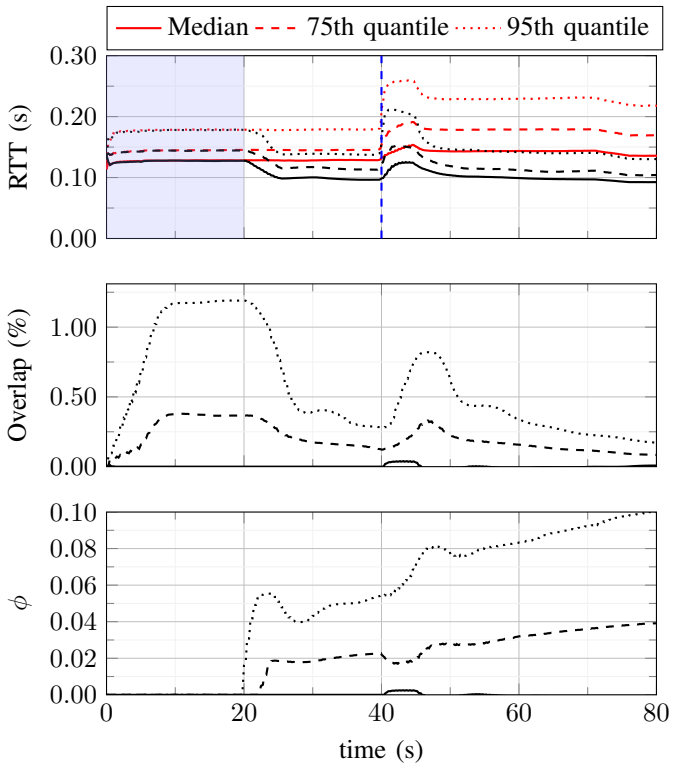

Fig. 5. Response to Disturbance: This figure demonstrates the system's resilience and adaptability to disturbances, with phase shifting initiated at $t = 20$. It contrasts system performance under varied loads, underscoring the improved stability and resilience provided by phase shifting in dynamic environments.

Even with partial participation in the phase shifting algorithm, such as one or five out of ten clients, all clients noticed an improvement in response times, as depicted in Figure 3. This underscores the robustness of our proposed solution and its capability to alleviate request overlaps, even when not all clients are participating.

To further highlight the phase shifting solution's impact,

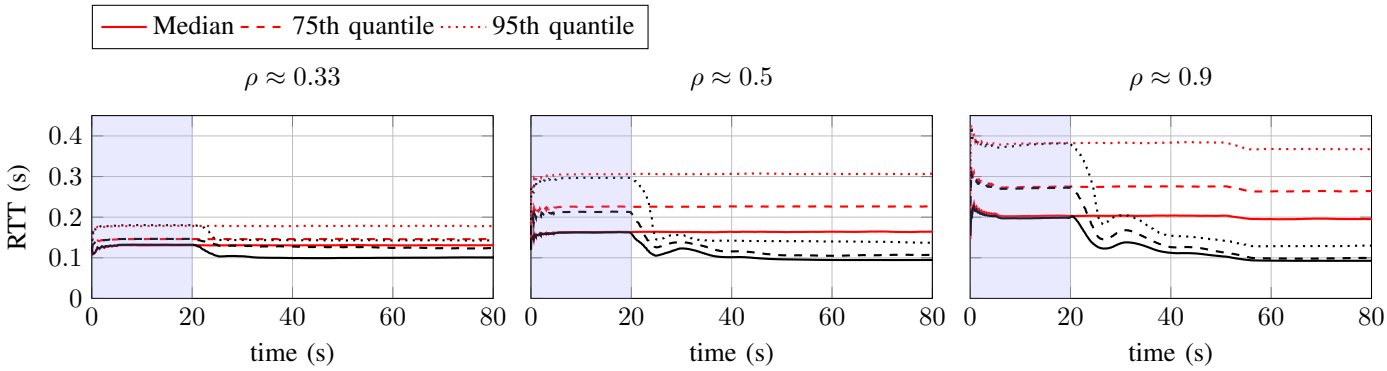

Fig. 6. Performance under Varying Load: This figure delineates the system's response to different loads by varying the number of clients. Scheduling is not applied until $t = 20$, highlighting the considerable improvement in system stability and response times achieved through phase shifting under increased loads.

Figure 4 provides a comparison of the start and end times of a subset of requests from each client before and after activating the phase shifting solution. The left chart in Figure 4 reveals overlapping requests from clients. In contrast, the right chart presents more scattered requests throughout the experiment period once the phase shifting solution has been initiated. This scattering of requests leads to shorter average request durations, providing additional evidence of our proposed solution's effectiveness in lessening request overlap and enhancing overall system performance.

B. Load

This experiment aimed to evaluate the performance of our proposed solution under varying server loads. We anticipated that temporal separation of clients could become more challenging as the server load increased.

Figure 6 illustrates the recorded response times at different server loads, where the load was adjusted by varying the number of clients interacting with the server. During periods of low server load, with $\rho \approx 0.33$, the clients experienced improved response times, approaching the combined execution and network delay more rapidly. As the server load intensified, the overlap between clients also increased, providing a greater potential for improvement. This trend is clearly observed when comparing the leftmost and rightmost charts in Figure 6, where the relative improvement in response time was significantly greater when the server load approached capacity.

These results highlight the capability of our proposed solution to successfully phase shift the requests of multiple clients, demonstrating its effectiveness across different server loads. Despite the increase in load, the system achieved consistent response times. The only observed penalty was the additional time required to resolve overlaps. For a low server load of $\rho \approx 0.33$, the transition was almost instantaneous. At half capacity, $\rho \approx 0.5$, convergence was achieved after 15 seconds, and for a high server load of $\rho \approx 0.9$, convergence occurred after 30 seconds. It is important to note that the control action's magnitude in these experiments was moderate, and the convergence time could be further adjusted by fine-tuning the PI-controller parameters as discussed in Section IV.

In conclusion, these results demonstrate the effectiveness of our proposed solution in achieving temporal separation of clients, irrespective of varying server loads. Despite increased loads, the system maintains consistent response times, showcasing the robustness and reliability of our solution.

C. Disturbance

In this experiment, we evaluated the ability of the proposed solution to successfully converge to the expected median response time, as established in Section VI-B, when subjected to a disturbance. The disturbance in this case was simulated as a sudden increase in the number of clients, leading to overlapping requests. It's worth noting that in the previous experiment, it was established that the solution was able to converge to the same median response time regardless of load.

The time-series outcome of the experiment is shown in Figure 5. As in the previous experiments, the phase-shift solution was activated at $t = 20$. At $t = 40$, additional users were admitted, causing the load to increase to $\rho \approx 0.5$, and the response times were significantly improved due to the phase-shift solution. However, when the 5 additional clients were admitted, the load increased to $\rho \approx 0.9$, resulting in an instantaneous increase in response times, followed by a gradual improvement over a period of 30 seconds. Notably, the outcome for the clients when using the phase-shift solution was better than when not using the solution, as evidenced by improved quantiles in the middle chart in Figure 5. Although the overlap is not entirely eliminated, this is due to the heterogeneity and noise in the system, and it is non-trivial to mitigate. This can be seen as the 75th and 95th quantiles are not yet at 0, but converging to 0. It should be noted that this convergence will happen beyond the time frame of the experiment. Similarly, in the bottom chart in Figure 5, the controller attempts to mitigate the overlap, as seen by the non-zero values of ϕ. This experiment demonstrates the effectiveness of the proposed solution in mitigating the impact

of sudden disturbances and maintaining improved response times compared to the baseline scenario.

D. Resource utilization

Our experiment results clearly indicate that using phase shifting has several beneficial effects on the performance of the server process. By comparing the performance metrics of the server process with and without using our proposed method, we observed the following benefits:

Elimination of Waiting Time: Without phase shifting, the server process had an average waiting time of about 10% during the experiment, which indicates that the process was frequently waiting for resources to become available. In contrast, when using our proposed method, the server process had 0% waiting time, indicating that the process was able to execute without any delays. This demonstrates that phase shifting effectively eliminates waiting time, allowing the server process to utilize system resources more efficiently.

Reduced Context Switching: Context switching, which refers to the process of switching between different tasks or processes, can introduce overhead and impact the performance of the server process. Without phase shifting, the context switch rate (`cswch/s`) was just above 100, indicating a relatively high rate of context switching. However, with phase shifting, the context switch rate was reduced to just above 10, indicating a significant reduction in context switching. This suggests that our proposed method minimizes the overhead of context switching, allowing the server process to focus on its execution and improving overall performance.

Decreased Page Faults and Memory Footprint: Page faults, which occur when a process requests a page that is not in physical memory, can result in performance degradation due to disk I/O operations. Without phase shifting, the rate of minor page faults (`minflt/s`) was around 12, indicating a moderate level of page faults. However, with phase shifting, the rate of minor page faults was reduced to just below 3, indicating a significant reduction in page faults. Additionally, the virtual memory size (`VSZ`) of the server process was reduced from around 500,000 to below 140,000 with phase shifting, indicating a substantial reduction in memory footprint. This suggests that our proposed method helps to minimize page faults and reduce memory usage, resulting in improved performance.

No Adverse Impact on CPU Utilization: It is important to note that our proposed method of using phase shifting did not adversely impact CPU utilization. The CPU utilization in general was indistinguishable between the two methods, indicating that our proposed method does not introduce any additional CPU overhead. This suggests that the benefits of using phase shifting, as observed in the reduction of waiting time, context switching, page faults, and memory footprint, are achieved without sacrificing CPU performance.

Note on Results Indicative Nature: It should be noted that these results are indicative and may vary depending on the specific platform, implementation, and experimental conditions. Different system configurations, hardware setups, and workload characteristics may yield different results. Therefore, it is important to thoroughly evaluate the performance of phase shifting in the specific context of the target system and workload to fully assess its effectiveness.

VII. Conclusions

This study introduces a novel, feedback control theory-based approach to optimize task scheduling in cloud computing, utilizing a Proportional Integral Derivative (PID) controller. This innovative methodology dynamically adjusts task execution timings, showing significant improvements in system performance compared to traditional methods, validated by detailed experimentation using pidstat.

Innovative Advancements: Our solution brings forth multiple advancements. It eliminates waiting time, optimizing system throughput and efficiency. It reduces context switching overhead and system latency, and mitigates page faults while optimizing memory usage, maintaining optimal CPU utilization and ensuring balanced resource allocation.

Dynamic Adaptation and Enhanced Responsiveness: The dynamic and adaptive nature of our approach is a defining feature, allowing dynamic adjustment of task timings based on real-time feedback. Our experiments validate the robustness of our solution in mitigating server contention effectively and reducing response times significantly.

Applicability, Variability, and Future Directions: The applicability of our method may vary with different system configurations, hardware, and workloads. Comprehensive evaluations are crucial to fully ascertain its versatility. Future research should explore the adaptability of our methodology in various conditions, considering the limitations of tools like pidstat.

Final Remarks: In conclusion, our methodology represents a significant advancement in cloud computing task scheduling. It fosters enhanced resource utilization and system performance, presenting a paradigm shift towards developing efficient and adaptable cloud services. This study lays the foundation for future research on refining and expanding dynamic, feedback-based scheduling solutions in cloud computing.

VIII. Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, the SEC4FACTORY project, funded by the Swedish Foundation for Strategic Research (SSF), and the IMMINENCE project funded by Sweden's Innovation Agency (VINNOVA). The authors are part of the Excellence Center at Linköping-Lund on Information Technology (ELLIIT), and the Nordic University Hub on Industrial IoT (HI2OT) funded by NordForsk.

References

[1] Hitham Alhussian, Nordin Zakaria, Ahmed Patel, Ayman Jaradat, Said Jadid Abdulkadir, Abdelaziz Y Ahmed, Hussein T Bahbouh, Sallam Osman Fageeri, Asim Abdallah Elsheikh, and Junzo Watada. Investigating the schedulability of periodic real-time tasks in virtualized cloud environment. *IEEE Access*, 2019.

[2] Mohammadreza Barzegaran, Anton Cervin, and Paul Pop. Performance optimization of control applications on fog computing platforms using scheduling and isolation. *Access*, 2020.

[3] Enrico Bini and Anton Cervin. Delay-Aware Period Assignment in Control Systems. In *2008 Real-Time Systems Symposium*, 2008.

[4] Anton Cervin, Johan Eker, Bo Bernhardsson, and Karl-Erik Årzén. Feedback–feedforward scheduling of control tasks. *Real-Time Systems*, 2002.

[5] Xiaotian Dai and Alan Burns. Period adaptation of real-time control tasks with fixed-priority scheduling in cyber-physical systems. *Journal of systems architecture*, 2020.

[6] Qiang Duan. Cloud service performance evaluation: status, challenges, and opportunities–a survey from the system modeling perspective. *Digital Communications and Networks*, 2017.

[7] Mor Harchol-Balter, Bianca Schroeder, Nikhil Bansal, and Mukesh Agrawal. Size-Based Scheduling to Improve Web Performance. *Transactions on Computer Systems*, 2003.

[8] Essam H Houssein, Ahmed G Gad, Yaser M Wazery, and Ponnuthurai Nagaratnam Suganthan. Task scheduling in cloud computing based on meta-heuristics: review, taxonomy, open challenges, and future trends. *Swarm and Evolutionary Computation*, 2021.

[9] Samia Ijaz, Ehsan Ullah Munir, Saima Gulzar Ahmad, M Mustafa Rafique, and Omer F Rana. Energy-makespan optimization of workflow scheduling in fog–cloud computing. *Computing*, 2021.

[10] Istio. Istio. Online, Accessed 2023.

[11] Kubernetes. Kubernetes. Online, Accessed 2023.

[12] Mohit Kumar, Subhash Chander Sharma, Anubhav Goel, and Santar Pal Singh. A comprehensive survey for scheduling techniques in cloud computing. *Journal of Network and Computer Applications*, 2019.

[13] John P Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *11th Real-Time Systems Symposium*. IEEE, 1990.

[14] Rajiv Nishtala, Vinicius Petrucci, Paul Carpenter, and Magnus Sjalander. Twig: Multi-agent task management for colocated latency-critical cloud services. In *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020.

[15] Aravind Rajagopalan, Devesh R Modale, and Radha Senthilkumar. Optimal scheduling of tasks in cloud computing using hybrid firefly-genetic algorithm. In *Advances in Decision Sciences, Image Processing, Security and Computer Vision: International Conference on Emerging Trends in Engineering (ICETE), Vol. 2*. Springer, 2020.

[16] R. Santhosh and T. Ravichandran. Pre-emptive scheduling of on-line real time services with task migration for cloud computing. In *International Conference on Pattern Recognition, Informatics and Mobile Engineering*, 2013.