STM JOURNALS

Recent Trends in
Parallel Computing

ISSN: 2393-8749
Volume 13, Issue 1, 2026
January–April
DOI (Journal): 10.37591/RTPC

https://journals.stmjournals.com/rtpc

Review RTPC

# Application of Proportional-Share with Punishment Principle for Resource-Sharing in Parallel Computing Applications

Manas Kumar Yogi[1]*

***Abstract***

*Efficient resource-sharing is a cornerstone of high-performance parallel computing. While proportional-share scheduling has long been a foundational approach for distributing resources according to predefined weights, its effectiveness can be compromised by tasks that over-consume their allocated share, leading to system-wide performance degradation and unfairness. This review article investigates the application of the "proportional-share with punishment" (PSWP) principle, a hybrid scheduling paradigm designed to address this challenge. PSWP integrates the flexibility of proportional sharing with a robust penalty mechanism that dynamically corrects resource over-consumption, ensuring long-term fairness and stability. This article thoroughly synthesizes various recent scholarly works from 2020 to 2026 to provide a comprehensive overview of the PSWP principle, its theoretical underpinnings, and its practical implementations in parallel computing platforms, cloud platforms, and distributed systems. We explore the various forms of punishment mechanisms, their triggers, and their impact on system performance metrics like throughput, latency, and fairness. Furthermore, we examine the challenges associated with implementing PSWP, including the overhead of monitoring and the potential for unintended consequences.*

**Keywords:** Big data, cluster, distributed, high-performance computing, resource

## INTRODUCTION

Parallel computing has become an indispensable tool for addressing complex computational problems in science, engineering, and data analytics. The power of parallel systems lies in their ability to harness the collective resources of multiple processing units to solve problems that would be intractable for a single processing unit. However, the efficient management of these shared resources, including CPU cycles, memory, and I/O bandwidth, is a critical challenge that directly impacts the performance and fairness of the entire system [1]. A key aspect of this challenge is the development of scheduling algorithms that can effectively and equitably distribute resources among competing tasks.

Proportional-share scheduling has emerged as a popular and effective approach for resource allocation in parallel and distributed systems [2]. The core idea behind proportional-share is to assign a weight to each task, representing its relative importance or resource entitlement, and then to allocate resources in proportion to these weights. This approach offers a flexible and intuitive way to manage resource-sharing, allowing system administrators to prioritize tasks and ensure that each task receives its fair share of resources. Seminal algorithms, such as lottery scheduling [3] and Stride Scheduling [4], have laid the groundwork for a wide range of proportional-share schedulers that are now widely used in modern operating systems and cluster management frameworks.

***Author For Correspondence**
Manas Kumar Yogi
E-mail: Manas.yogi@gmail.com

[1]Assistant, Professor, Department of CSE, Pragati Engineering College (A), Surampalem, Andhra Pradesh, India

Received Date: January 16, 2026
Accepted Date: January 21, 2026
Published Date: February 26, 2026

**Citation:** Manas Kumar Yogi. Application of Proportional-Share with Punishment Principle for Resource-Sharing in Parallel Computing Applications. Recent Trends in Parallel Computing. 2026; 13(1): 1–8p.

 

Despite its advantages, traditional proportional-share scheduling has limitations. One of the most significant challenges is the potential for tasks to "cheat" or over-consume their allocated shares, either intentionally or unintentionally. This can occur because of programming errors, unexpected workload fluctuations, or malicious behavior. When a task consumes more than its fair share of resources, it can lead to performance degradation for other tasks, creating a situation of unfairness and potentially jeopardizing the stability of the entire system. This led to the development of a more robust scheduling principle: proportional-share with punishment (PSWP).

The PSWP principle extends the traditional proportional-share model by incorporating a penalty mechanism to discourage and correct resource over-consumption [5]. Under this principle, tasks that exceed their allocated share are "punished" by having their future resource allocation reduced, their priority lowered, or their execution is throttled. This not only corrects for immediate unfairness but also creates a disincentive for future misbehavior. Thus, the PSWP principle offers a more resilient and equitable approach to resource-sharing, ensuring that the system can maintain fairness and stability even in the presence of misbehaving tasks.

This review article provides a comprehensive examination of the application of the PSWP principle in parallel computing applications. We will explore the theoretical foundations of this principle, analyze its various implementations, and evaluate its performance and challenges in this paper. By synthesizing the latest research in this area from 2020 to 2026, this article aims to provide a clear and in-depth understanding of the state of the art in PSWP and to identify promising directions for future research. The subsequent sections delve into the theoretical underpinnings of PSWP, its application in different parallel computing environments, various forms of punishment mechanisms, and challenges associated with its implementation [6, 7].

## THEORETICAL FOUNDATIONS OF PROPORTIONAL-SHARE WITH PUNISHMENT

The PSWP principle fundamentally builds upon the well-established theoretical foundations of proportional-share scheduling, augmenting them with mechanisms to enforce compliance and fairness. Understanding these foundational concepts is crucial for appreciating the nuances and effectiveness of the PSWP in complex parallel computing environments.

### Proportional-Share Scheduling

Proportional-share scheduling aims to allocate resources among competing tasks such that each task receives a share proportional to its assigned weight. This concept is rooted in the idea of *fairness*, which is defined as the ratio of allocated resources to requested resources. Key algorithms that embody this principle include the following:

- *Lottery scheduling*: Introduced by Waldspurger and Weihl, lottery scheduling assigns a number of "tickets" to each task, proportional to its share. At each scheduling interval, a lottery is held, and the task holding the winning ticket is executed [8]. This probabilistic approach provides a simple yet effective method for achieving proportional sharing over time.
- *Stride scheduling*: Also developed by Waldspurger and Weihl, Stride Scheduling is a deterministic proportional-share algorithm. Each task is assigned a "stride" value that is inversely proportional to its weight. Tasks with smaller strides (higher weights) advance more quickly in virtual time, ensuring that they receive their proportional-share [9]. This method offers better predictability and lower variance in resource allocation compared to lottery scheduling.
- *Weighted fair queuing (WFQ):* Originally designed for network packet scheduling, WFQ was adapted for CPU scheduling. It assigns a weight to each flow (or task) and services them in a manner that approximates fluid flow, ensuring that each flow receives its proportional-share of bandwidth [10]. The WFQ is known for its ability to provide both fairness and isolation among competing flows.

These algorithms provide the basic machinery for the proportional distribution of resources. However, they typically assume that tasks are well-behaved and do not actively attempt to exceed their allocated shares. The introduction of the punishment principle addresses this assumption.

 

### Integrating The Punishment Principle

The punishment principle is newly introduced to deter and correct deviations from the agreed-upon proportional shares. It transforms a purely descriptive allocation mechanism into a prescriptive one. The theoretical basis for punishment often draws from economic game theory and behavioral economics, where penalties are used to incentivize cooperative behavior and maintain equilibrium in shared resource systems [11].

In the context of PSWP, punishment mechanisms are designed to:

- *Detect over-consumption*: Identify tasks that consume resources beyond their proportional-share, often by monitoring their actual resource usage against their allocated virtual time or credits.
- *Apply penalties*: Implement a mechanism to reduce the future resource allocation of an offending task. This can involve decreasing weight, increasing its virtual time, or temporarily suspending its execution.
- *Restore fairness*: Bring the system back to a state where all tasks receive their proportional shares and misbehaving tasks are appropriately disciplined. The goal is not merely to punish but to restore overall system fairness and stability.

The effectiveness of the punishment principle hinges on several factors, including the accuracy of detection, severity and timeliness of the penalty, and transparency of the mechanism to the tasks. An overly aggressive punishment can lead to under-utilization, while a too lenient one may fail to deter misbehavior. Therefore, the design of effective penalty functions is a critical theoretical and practical consideration [12].

## PUNISHMENT MECHANISMS AND THEIR TRIGGERS IN PARALLEL COMPUTING

The implementation of the punishment principle in parallel computing environments involves various mechanisms, each tailored to specific resource types and application behaviors. The choice of punishment mechanism and its trigger conditions significantly impact the overall fairness, efficiency, and responsiveness of the system. This section categorizes common punishment mechanisms and their triggers.

### Types of Punishment Mechanisms

Punishment in PSWP can manifest in several forms, ranging from subtle adjustments to drastic actions.

- *Weight reduction*: One of the most common methods is to dynamically reduce the weight assigned to a task. A lower weight directly translates to a smaller proportional-share of resources in subsequent scheduling cycles. This is a flexible mechanism that can be adjusted incrementally [13].
- *Virtual time acceleration*: In algorithms such as Stride Scheduling, tasks maintain a virtual time counter. A punishment can involve artificially advancing a task's virtual time, making it appear as though it consumes more resources than it actually does. This effectively delays the next turn to execute until other tasks catch up [14].
- *Priority demotion*: For systems that use priority-based scheduling, a misbehaving task can have its priority lowered temporarily or permanently. This reduces the chances of it being scheduled, thereby limiting resource access.
- *Resource throttling/limiting*: Direct throttling involves imposing hard limits on the amount of a specific resource (e.g., CPU cycles, I/O operations, and network bandwidth) that a task can consume within a given period. This is often used for tasks that exhibit bursty or excessive resource demand.
- *Credit/quota system adjustment*: In systems where tasks earn credits for under-utilization and spend them for execution, punishment can involve deducting credits or reducing the rate at which new credits are earned. Conversely, tasks that behave well might earn bonus credits [15].
- *Task suspension/termination*: In extreme cases of persistence or severe resource abuse, a task may be temporarily suspended or even terminated. This is typically a last resort, reserved for critical violations that threaten system stability.

 

### Triggers For Punishment

The decision to apply a punishment is based on detecting specific behaviors or conditions that violate the established resource-sharing policies (Table 1). Common triggers include:

- *Exceeding proportional-share*: The most direct trigger is when a task consistently consumes more resources than its allocated proportional-share over a defined measurement window. This is often detected by comparing actual usage with the theoretical fair share [16].
- *Service level agreement (SLA) violations*: In cloud and distributed computing, tasks often operate under SLAs that specify performance guarantees (e.g., minimum throughput and maximum latency). Failure to meet these SLAs, especially if caused by the task's excessive resource demands, can trigger penalties [17].
- *Deadline misses*: For real-time or time-sensitive parallel applications, missing a deadline can be a critical issue. If a task's resource consumption contributes to its own or other tasks' deadline misses, punishment mechanisms might be invoked to reallocate resources more effectively.
- *Resource monopolization*: Detecting patterns where a single task or a small group of tasks monopolizes a shared resource, starving other tasks, can trigger punishment to redistribute resources more equitably.
- *Misreporting resource demands*: In systems where tasks declare their resource requirements, intentionally or unintentionally misreporting these demands to gain an unfair advantage can trigger punishment [18].

## APPLICATIONS OF PROPORTIONAL-SHARE WITH PUNISHMENT IN PARALLEL COMPUTING

The PSWP principle currently finds diverse applications across various parallel computing paradigms, where efficient and fair resource allocation is paramount. Its ability to enforce fairness and prevent resource monopolization makes it particularly valuable in environments with competing workloads and shared infrastructure. This section explores its applications in high-performance computing (HPC), cloud computing, and distributed systems.

### High-Performance Computing

In HPC environments, large-scale scientific simulations, data analyses, and machine learning tasks are often run concurrently on shared supercomputing resources. Ensuring fair access to CPU cores, memory, and high-speed interconnects is critical for maximizing the throughput and meeting job completion deadlines. Traditional proportional-share schedulers are used to allocate resources based on the project priorities or user quotas. However, individual jobs or users might inadvertently (due to inefficient code) or intentionally (to gain an advantage) consume more resources than their fair share, thereby impacting other critical jobs.

**Table 1.** Common punishment mechanisms and their triggers in PSWP.

| Punishment mechanism | Description | Typical triggers |
|---|---|---|
| Weight reduction | Dynamically decreases the task's assigned weight, reducing its future proportional-share. | Consistent over-consumption of resources beyond its allocated share. |
| Virtual time acceleration | Artificially advances a task's virtual time, delaying its next execution turn. | Exceeding fair share in algorithms like Stride Scheduling. |
| Priority demotion | Lowers the task's scheduling priority, making it less likely to be scheduled. | Repeated resource abuse or minor SLA violations. |
| Resource throttling | Imposes hard limits on specific resource consumption (CPU, I/O, network). | Bursting resource demands, exceeding predefined usage caps. |
| Credit/quota adjustment | Deducts earned credits or reduces future credit-earning rate. | Spending more credits than earned, violating credit-based resource policies. |
| Task suspension/ termination | Temporarily halts or permanently stops the execution of a misbehaving task. | Persistent and severe resource abuse, critical system instability. |

 

PSWP can be applied in HPC to:

- *Enforce user/project quotas*: If a user's jobs collectively exceed their allocated proportional-share of computer time or storage, subsequent jobs from that user could be subjected to a penalty, such as reduced priority or delayed start times [19].
- *Manage resource-intensive kernels*: Certain computational kernels within parallel applications may be disproportionately resource-hungry. The PSWP can identify such kernels and apply temporary throttling to ensure that other parts of the application or other applications receive their due share.
- *Prevent "greedy" jobs*: Jobs that exhibit aggressive resource acquisition behavior, potentially starving others, can be identified, and their resource requests or allocations can be adjusted downwards until fairness is restored.

**Cloud Computing and Virtualized Environments**

Cloud computing platforms rely heavily on virtualization to share the underlying physical resources among multiple virtual machines (VMs) or containers. Tenants provide resources based on service agreements, and the cloud provider must ensure that each tenant receives their contracted share while efficiently utilizing the infrastructure. The PSWP is highly relevant for maintaining quality of service (QoS) and enforcing SLAs.

- *SLA enforcement*: If a VM or container consistently exceeds its allocated CPU, memory, or I/O limits, it can be penalized. This might involve dynamically reducing the allocated virtual resources, migrating them to a less contended host, or even imposing financial penalties on the tenant [20].
- *Fairness among tenants*: In multi-tenant cloud environments, the PSWP ensures that no single tenant can monopolize resources at the expense of others, even if their applications exhibit burst or unpredictable demands. This is crucial for maintaining a stable and predictable service for all users.
- *Resource over-subscription management*: Cloud providers often oversubscribe resources to improve utilization. The PSWP can help manage the contention that arises during peak loads by penalizing VMs that contribute disproportionately to resource scarcity, thereby protecting the performance of well-behaved VMs.

**Distributed Systems and Big Data Processing**

Distributed systems, including big data processing frameworks such as Apache Hadoop and Spark, involve the coordination of resources across a cluster of machines. Resource managers in these systems (e.g., YARN) often employ proportional-share schedulers (such as Fair Scheduler or Capacity Scheduler) to allocate resources among different applications or users. The dynamic and heterogeneous nature of big data workloads makes the PSWP particularly useful.

- *Task-level fairness*: Within a single Spark job, different stages or tasks may have varying resource requirements. If a task consistently runs over its allocated slot, it can delay the entire job or affect other jobs. PSWP can apply penalties to such tasks, forcing them to relinquish resources or slow down.
- *Handling stragglers*: In distributed computing, "stragglers" (tasks that run significantly slower than others) can bottleneck the overall job completion. While not always due to malicious over-consumption, if a straggler is consuming resources inefficiently, a punishment mechanism could reallocate its resources to other, more productive tasks or even restart it on a different node.
- *Cross-application isolation*: PSWP can ensure that a resource-intensive big data application does not negatively impact other potentially higher-priority applications running on the same cluster by penalizing its excessive resource demands.

## PERFORMANCE EVALUATION AND CHALLENGES IN IMPLEMENTING PSWP

The successful implementation of PSWP in parallel computing applications requires careful consideration of its performance implications and the inherent challenges in its design and deployment. Evaluating the PSWP involves assessing its impact on fairness, throughput, latency, and overhead introduced by the punishment mechanisms.

 

**Table 2.** Key challenges in implementing proportional-share with punishment (PSWP).

| Challenge category | Description | Impact on PSWP effectiveness |
|---|---|---|
| Monitoring accuracy | Difficulty in precisely measuring fine-grained resource consumption across distributed systems. | Leads to unfair or ineffective punishments if misbehavior is not accurately detected. |
| Behavior definition | Establishing clear criteria for what constitutes punishable resource over-consumption. | Ambiguity can result in inconsistent application of penalties or user dissatisfaction. |
| Penalty design | Crafting effective mathematical functions for reducing resource shares. | Poor design can cause instability, under-utilization, or failure to deter misbehavior. |
| Computational overhead | The cost of monitoring, detection, and enforcement of punishment mechanisms. | Can negate performance benefits if overhead is too high, reducing overall efficiency. |
| Dynamic environments | Adapting PSWP to changing workloads and heterogeneous hardware resources. | Reduces responsiveness and fairness if the system cannot dynamically adjust punishment policies. |
| User acceptance | Ensuring users perceive punishment mechanisms as fair, transparent, and justified. | Lack of acceptance can lead to user frustration, attempts to circumvent policies, or reduced adoption. |

## Performance Metrics

When evaluating PSWP, several key performance metrics are typically considered:

- *Fairness*: This is the primary goal of the PSWP. Fairness can be quantified using metrics such as Jain's Fairness Index, which measures how equal resources are distributed among tasks relative to their respective weights. A higher index indicates better fairness of the model. The PSWP aims to improve fairness by actively correcting deviations from proportional shares.
- *Throughput*: The total amount of work completed by the system per unit of time. Although PSWP prioritizes fairness, it must do so without significantly degrading the overall system throughput. An effective punishment mechanism should ideally improve the throughput by preventing resource monopolization and ensuring efficient resource utilization across all tasks.
- *Latency*: The time taken for a task to be completed or for a resource request to be fulfilled. Punishment mechanisms can potentially increase the latency of misbehaving tasks, which is an intended effect. However, it should not unduly increase the latency of well-behaved tasks.
- *Resource utilization*: Percentage of time resources are actively used. The PSWP aims to maintain high resource utilization by preventing idle resources while ensuring fair distribution. Overly aggressive punishment could lead to under-utilization if tasks are throttled too severely.
- *Stability*: The ability of the system to maintain consistent performance and fairness over time, particularly under varying workloads and in the presence of misbehaving tasks. The PSWP should contribute to system stability by quickly addressing resource imbalances.

## Challenges in Implementation

Implementing an effective PSWP system presents several highly significant challenges Table 2:

- *Accurate resource monitoring*: Precisely measuring the resource consumption of individual tasks in a fine-grained manner across a distributed parallel system is complex and challenging. Inaccurate monitoring can lead to unfair punishments or undetected misbehavior.
- *Defining "punishable" behavior*: Establishing clear and unambiguous criteria for what constitutes a punishable offense is crucial. This involves defining thresholds for over-consumption, identifying patterns of resource abuse, and distinguishing between legitimate bursty demands and malicious behavior.
- *Designing effective penalty functions*: The design of the penalty function (e.g., how much to reduce a task's share, for how long, and under which conditions) is critical. An ill-designed function can lead to oscillations in resource allocation, under-utilization, or insufficient deterrence.

 

- *Overhead of punishment mechanism*: The monitoring, detection, and enforcement of punishment mechanisms introduce computational overhead. This overhead must be carefully managed to ensure that the benefits of improved fairness and stability outweigh the costs of implementation.
- *Dynamic workloads and heterogeneity*: Parallel computing environments often feature dynamic workloads with varying resource demands and heterogeneous hardware. Adapting PSWP to these changing conditions and diverse resource types (CPU, GPU, memory, and network) adds complexity.
- *User acceptance and transparency*: Users of parallel computing systems may react negatively to punishment mechanisms if they are perceived as unfair, opaque, or arbitrary. Transparency in how punishment is applied and clear communication of policies are essential for user acceptance.
- *Cascading effects*: A punishment applied to one task might have cascading effects on dependent tasks or the overall application's performance. A careful design is needed to avoid unintended negative consequences on the system's critical path.

## CONCLUSION

The PSWP principle represents a significant advancement in resource-sharing for parallel computing applications. By integrating the foundational fairness of proportional-share scheduling with robust mechanisms to deter and correct resource over-consumption, PSWP offers a more resilient and equitable approach to managing shared computational resources. This review highlighted the theoretical underpinnings of proportional-share scheduling, including algorithms such as Lottery and Stride Scheduling, and detailed how the punishment principle extends these by introducing detection and penalty mechanisms to enforce compliance. We explored various punishment types, such as weight reduction and virtual time acceleration, and their triggers, including exceeding proportional shares and SLA violations. The broad applicability of the PSWP across HPC, cloud environments, and distributed systems underscores its importance in maintaining system stability and fairness in diverse parallel workloads. While challenges remain in accurate monitoring, effective penalty design, and managing computational overhead, ongoing research and technological advancements continuously refine PSWP implementations. Future directions will likely focus on more adaptive and intelligent punishment systems, leveraging machine learning to predict and prevent resource abuse, and developing standardized frameworks for transparent and user-acceptable penalty policies. Ultimately, the PSWP is crucial for optimizing resource utilization and ensuring equitable access in the increasingly complex landscape of parallel computing.

## REFERENCES

1. Waldspurger CA, Weihl WE. Lottery scheduling: flexible proportional-share resource management. First Symposium on Operating Systems Design and Implementation (OSDI '94), Monterey, CA, USA. 1994. Monterey, CA. Berkeley (CA): USENIX Association; 1994. p. 1–11. Available from: https://www.usenix.org/conference/osdi-94/lottery-scheduling-flexible-proporti–onal-share-resource-management
2. Saewong S, Rajkumar RR, Lehoczky JP, Klein MH. Analysis of hierarchical fixed-priority scheduling. Proceedings of the 14th Euromicro Conference on Real-Time Systems (ECRTS 2002), Vienna, Austria. 2002. p. 152–160. doi:10.1109/EMRTS.2002.1019197.
3. Parekh AK, Gallager RG. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. IEEE/ACM Trans Netw. 1993;1(3):344–357. doi:10.1109/90.234856.
4. Fehr E, Gächter S. Altruistic punishment in humans. Nature. 2002;415(6868):137–140. doi:10.1038/415137a. PMID: 11805825.
5. Brown Coverdale H. Putting proportional punishment into perspective. Crim Law Philos. 2025;19(2):181–201. doi:10.1007/s11572-024-09736-5.
6. Sheng Y, Cao S, Li D, Zhu B, Li Z, Zhuo D, et al. Fairness in serving large language models. In: Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24); 2024. Santa Clara, CA. Berkeley (CA): USENIX Association; 2024. p. 965–988.

 

7. Chandra A, Adler M, Shenoy P. Deadline fair scheduling: bridging the theory and practice of proportionate fair scheduling in multiprocessor systems. Proceedings Seventh IEEE Real-Time Technology and Applications Symposium, Taipei, Taiwan. 2001. p. 3–14. doi:10.1109/RTTAS.2001.929861.
8. Vuppalapati M, Fikioris G, Agarwal R, Cidon A, Khandelwal A, Tardos É. Karma: resource allocation for dynamic demands. Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23); 2023. Boston, MA. Berkeley (CA): USENIX Association; 2023. p. 645–662.
9. Tang S, Chai Q, Yu C, Li Y, Sun C. Balancing fairness and efficiency for cache sharing in semi-external memory system. 49th International Conference on Parallel Processing (ICPP '20), Edmonton, AB, Canada. 2020 Aug 17–20. Article 13. doi:10.1145/3404397.3404450.
10. Windrich I, Kierspel S, Neumann T, Berger R, Vogt B. Enforcement of fairness norms by punishment: a comparison of gains and losses. Behav Sci (Basel). 2024;14(1):39. doi:10.3390/bs14010039. PMID: 38247691.
11. Ohdaira T. The probabilistic pool punishment proportional to the difference of payoff outperforms previous pool and peer punishment. Sci Rep. 2022;12(1):6604. doi:10.1038/s41598-022-10582-5. PMID: 35459880.
12. Shreedhar G, Tavoni A, Marchiori C. Monitoring and punishment networks in an experimental common pool resource dilemma. Environ Dev Econ. 2020;25(1):66–94. doi:10.1017/S1355770X19000457.
13. Blanco E, Struwe N, Walker JM. Experimental evidence on sharing rules and additionality in transfer payments. J Econ Behav Organ. 2021;188:1221–1247. doi:10.1016/j.jebo.2021.06.012.
14. Patel Y, Yang L, Arulraj L, Arpaci-Dusseau AC, Arpaci-Dusseau RH, Swift MM. Avoiding scheduler subversion using scheduler-cooperative locks. Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys); 2020 Apr 15–18; Heraklion, Greece. 2020. p. 1–17. doi:10.1145/3342195.3387521.
15. Li J, Liu Y, Wang Z, Xia H. Egoistic punishment outcompetes altruistic punishment in the spatial public goods game. Sci Rep. 2021;11(1):6584. doi:10.1038/s41598-021-85814-1. PMID: 33753774.
16. Obaidat MS, Boudriga NA. Fundamentals of performance evaluation of computer and telecom–unication systems. Hoboken (NJ): John Wiley & Sons; 2010. doi:10.1002/9780470567203.
17. Zahedi SM, Fan S, Lee BC. Managing heterogeneous datacenters with tokens. ACM Trans Archit Code Optim. 2018;15(2):1–23. doi:10.1145/3191821.
18. Cho H, Ravindran B, Jensen ED. An optimal real-time scheduling algorithm for multiprocessors. 27th IEEE International Real-Time Systems Symposium (RTSS 2006), Rio de Janeiro, Brazil. 2006. p. 101–110. doi:10.1109/RTSS.2006.10.
19. Badia RM, Pierson JM, Morin C, Kortas S, Parlavantzas N. Market-based autonomous resource and application management in the cloud [dissertation]. Argonne (IL): Argonne National Laboratory; 2014.
20. Ziv T, Whiteman JD, Sommerville JA. Toddlers' interventions toward fair and unfair individuals. Cognition. 2021;214:104781. doi:10.1016/j.cognition.2021.104781. PMID: 34051419.

 