The Impact of Resource Scheduling and Isolation on Cloud Gaming Performance

Amy Forde

COMP 314, Athabasca University

aforde1@learn.athabascau.ca

Abstract*— Cloud gaming services require efficient resource management to provide consistent, low-latency gaming experiences across shared infrastructure. This research investigates the impact of different CPU scheduling algorithms and resource isolation techniques on cloud gaming performance. We evaluate two Linux scheduling policies (Completely Fair Scheduler and SCHED_FIFO) in both isolated and non-isolated environments, with and without background load. Our testing framework measures key performance metrics including context switches, CPU usage, and scheduling delays using SuperTuxKart as a representative gaming workload. Results demonstrate that SCHED_FIFO with resource isolation provides optimal performance, reducing context switches by 28% under load compared to non-isolated configurations, while maintaining consistent CPU usage patterns. The combination of real-time scheduling and resource isolation showed particular benefits in multi-tenant scenarios, with only a 2.80% increase in context switches under load compared to 32.04% for CFS with isolation. These findings suggest that cloud gaming platforms could benefit from specialized scheduling and isolation strategies to maintain performance under varying load conditions. This research contributes to the understanding of operating system-level optimizations for cloud gaming infrastructure and provides insights for future cloud gaming platform design.

Keywords*— cloud gaming, resource isolation, scheduling, resource allocation

I. Introduction

Cloud gaming, once a laggy and clunky experience, has taken off in recent years as the advancements in OS technology are beginning to catch up with the lofty idea of high performance gaming on inexpensive hardware. Cloud gaming offers accessibility to top gaming performance to a much wider audience. As advancements in operating system technology catch up with the ambitious goal of delivering high-performance gaming on inexpensive hardware, the technical challenges of resource management become increasingly critical. Cloud gaming platforms must balance the demands of multiple users while maintaining consistent, low-latency performance across diverse hardware configurations.

This project will address this fundamental challenge in cloud gaming: the efficient management and allocation of cloud infrastructure resources to provide a fast and seamless gaming experience. This project will analyze how different resource isolation and CPU scheduling mechanisms can help or hinder cloud gaming performance. This project will consist of a literature review of recent studies, followed by a practical benchmarking evaluation in which 8 configurations will be tested using different scheduling algorithms, isolation environments and varying workloads. The main components of an operating system that will be addressed in this paper are CPU scheduling algorithms and resource isolation mechanisms and how these OS mechanisms affect performance on a cloud gaming workload. The significance of this research lies in its practical application to cloud gaming infrastructure optimization. As cloud gaming services continue to expand, understanding how both host and client operating system-level mechanisms such as scheduling and isolation can affect performance becomes crucial for service providers and platform developers.

II. Literature Review

Gaming is an interesting workload to test because it has unique demands over other computational tasks. Games require a combination of both GPU-intensive rendering and real-time processing demands that are not often required in equal performance demand in other computing tasks. On top of these demands, games require low latency between I/O requests and responses; most modern games rely on instantaneous responses. To make things more complicated, cloud servers utilize elasticity, load balancing, and redundancy to ensure a seamless service for the user. Even slight interruptions in these services can derail a gaming session, so cloud gaming servers need to be extremely reliable and efficient. Operating system resource management, such as scheduling and isolation, is vital to the success of cloud gaming.

While these requirements are demanding enough for a regular hardware-based gaming operating system, they put exceptional pressure on a cloud system. In [1], Wang et al. examine how cloud gaming workloads use improved scheduling mechanisms to run games efficiently on varying hardware configurations [1]. Resource use varies in intensity between games, and even within a single game. Some parts of gameplay are resource intensive, while others are not. The authors posit that a rebalancing of loading times and game play time, by shifting intensive computations to non-interactive moments such as loading screen time, may be a path to better performance. Using a real-time prediction scheduler, the authors were able to predict when a game was going to become very resource-intensive and dynamically reallocate resources, often using loading times to do so in order to prevent any performance issues during play-time [1]. This prediction algorithm, which uses machine learning, is an innovative concept that can help bridge the gap between high performance games and dynamic cloud server use. This approach aligns closely with the goals of this project, which will investigate, in the empirical evaluation section, how OS-level scheduling and resource isolation strategies can be optimized to support cloud gaming workloads.

To expand on the complexity of resource allocation in cloud gaming, one can analyze the traffic of a popular cloud gaming platform, *Google Stadia*. In a 2020 analysis of *Google Stadia* traffic [2], like in [1], it was found that there is a wide variance in traffic and resource requirements amongst games. This highlights the importance of dynamic and flexible resource allocation methods, since there will never be a one-size-fits-all solution for gaming. This analysis of traffic confirms the ideas discussed in [1], that games have varied and rapidly changing resource needs, and that loading screens can provide a ‘break’ in demand to allow for intensive resource allocation. While [2] primarily focuses on network resources, it indirectly underscores the importance of the client and server operating systems in ensuring low-latency performance. In an operating system context, latency is not only a function of the network but also of how efficiently the server OS schedules game processes and I/O operations. If the local OS scheduler introduces even minor delays in allocating CPU, GPU, or disk resources, the total time to render and transmit a frame increases, affecting network latency [3]. Resource allocation is not a simple task when it comes to gaming, particularly for games with high graphical demands.

In [4], Heo, Bardwaj, and Gavrilovska introduce a system called *Adrenaline* which dynamically adapts to changing video rendering requirements to provide a stable quality of service (QoS) on the user end [4]. *Adrenaline* achieves this by monitoring real-time performance and adjusting resource allocation accordingly. This helps to mitigate

latency that is noticeable by gamers or graphical issues during gameplay [4]. This article highlights the importance of flexible dynamic resource allocation required for fluctuating game needs and cloud resources. Like the previous article [2], the *Adrenaline* article focuses on optimizing network-layer optimizations, such as adaptive bitrate streaming and edge resource reallocation [4]. Additionally, as with the first article [1], this article also uses predictive algorithms and a scoring mechanism to predict spikes in computational demand and proactively adjust resource scheduling in anticipation of these spikes. *Adrenaline* is a prime example of how OS-level schedulers and isolation mechanisms can be tuned to meet the dynamic performance requirements of cloud gaming workloads.

Resource management and scheduling are not areas of optimization in cloud gaming alone; cloud computing as a whole relies heavily on efficient optimization of resource allocation and scheduling. In [5], Gu et al. analyze the use of deep reinforcement learning to optimize job scheduling and resource management for cloud computing, not restricted to cloud gaming. The article categorizes Deep Reinforcement Learning (DRL) methods for cloud resource management into four main algorithm types: value-based, policy-based, actor-critic methods, and multi-agent DRL. Each of these categories is suited for different scheduling and resource allocation contexts [5], which as outlined so far, change rapidly due to varying resource demands and task types. These algorithms model job scheduling as a Markov Decision Process (MDP), allowing agents to learn optimal policies through environment interaction and reward-based feedback, enabling adaptive and scalable resource allocation decision-making in a cloud environment [5]. This article reinforces the necessity of dynamic scheduling strategies in cloud gaming environments, where fluctuating resource demands and user interactions [1],[2],[5] require operating system–level schedulers to make real-time, adaptive decisions to maintain performance and QoS.

In [6], Pavlidakis et al. introduce a technique called *Guardian* which allows for intensive GPU sharing mechanisms as required by cloud gaming, through memory isolation. *Guardian* utilizes PTX-level bounds checking and dynamic GPU partitioning to not only allow multiple clients to use the same GPU, but to enable more complete and secure GPU memory access than a system without *Guardian* [6]. This reduces the load on cloud CPUs and allows for more clients to use less resources. Resource isolation is also important in cloud gaming for security and fairness. This article shows how resource isolation techniques such as GPU isolation can improve performance in a cloud computing system by ensuring that there is fair use of resources by users and by game processes [6]. By analyzing a method to ensure safe and efficient GPU sharing, this article on *Guardian* complements the empirical evaluations of this paper which will examine how OS-level resource isolation impacts game performance under varying loads in cloud gaming environments.

In a similar vein, [7] examines how isolation impacts cloud gaming workloads, particularly concerning GPUs. This article primarily looks at NVIDIA's Multi-Instance GPU (MIG) technology and how it is implemented to allow efficient shared use of GPUs in a cloud gaming environment [7]. The article proposes that a multi-objective Integer Linear Programming (ILP) model be used to optimize the relationship between GPU-enabled virtual machines to reduce migration overhead and reduce hardware utilization while still meeting the resource-intensive demands of cloud computing [7]. Some of the main issues with MIG include fragmentation and underutilization of GPU resources. The authors of [7] address the problem of

MIG's limitations by proposing a GPU Resource Management Unit (GRMU), which, in testing, outperformed current VM placement policies such as First-Fit (FF), Max Configuration Capacity (MCC) etc. by increasing request acceptance by 22% and reducing hardware usage by 17% [7]. The GRMU framework's approach to optimizing GPU resource allocation is highly relevant to this project's focus on operating system–level scheduling and containerization in cloud gaming environments by outlining the importance of understanding and integrating multi-objective optimization methods to ensure better resource utilization and user experience under varying load conditions.

In [8], Khallouli and Huang provide an analysis of existing cloud cluster resource scheduling algorithms which examines the need for new and dynamic scheduling methodologies to address issues unique to cloud computing such as multi-users, scalability, dynamic resource demands and varying hardware configurations. The article categorizes scheduling architectures into three different categories: centralized (ex. YARN), two-level (ex. Mesos), and distributed designs (ex. Omega) and examines how machine learning can be integrated into these schedulers to ensure that the schedulers can adapt to ever-changing cloud environments [8]. Cluster schedulers can provide workload isolation and scalability by dynamically allocating resources across multiple cloud servers, utilizing resources more efficiently than regular schedulers [8]. This article on cluster resource scheduling informs the evaluations, in the empirical evaluation section of this project, of how different scheduling strategies impact game performance under varying loads in cloud gaming environments.

It is clear from a review of relevant literature that resource allocation scheduling [1],[3],[5],[8] and resource isolation techniques [6],[7] are vital for meeting the resource-intensive demands of cloud gaming. Additional mechanisms such as GPU partitioning (*Adrenaline*), memory isolation (Guardian), and resource management units (*GRMU*) [4], [6], [7] can further enhance the ability of cloud servers to meet the high demands of cloud gaming by providing a way in which to fine-tune resource control. Predictive scheduling techniques have emerged as particularly effective in adapting to the dynamic nature of gaming workloads [1], [4], [5]. This collection of literature supports the premise that operating-system techniques such as scheduling and isolation strategies can be used to optimize the performance of cloud gaming. With these conclusions in mind, the following empirical evaluation section of this project will outline a simple methodology to test different Linux scheduling policies under a controlled gaming workload, as well as benchmark a lightweight container-based environment against a native setup to assess performance and interference.

III. Empirical Evaluation

The empirical evaluation component of this project involves running controlled cloud gaming workloads under different operating system scheduling policies (CFS and SCHED_FIFO) and isolation environments (bare-metal and Docker containers). Tools like vmstat and pidstat, will be used to collect performance metrics such as CPU usage, number of context switches and system load impact. Workload will be simulated by introducing background interference with stress-ng in some tests to evaluate how well each configuration maintains performance under load.

This experiment will simulate a cloud gaming environment by evaluating how the operating system's scheduling and resource isolation mechanisms affect the performance of a game that would typically be run remotely on a cloud server. In order to be able to tinker with the schedulers and isolation tools, the games tested are not commercial cloud games, but rather we will emulate a cloud

environment by adding background load to simulate multiple users, mimicking cloud resource allocation by providing an isolation environment with cgroups, and limiting CPU resources to mimic a shared cloud environment. By using these emulated resource constraints, a simplified cloud server simulation is created. Additionally, the metrics used such as context switching, CPU usage, scheduling efficiency and load impact are directly related to cloud gaming performance.

As discussed in [1], [2], [5], [8], different real-time scheduling and isolation strategies will influence game performance according to the chosen metrics. Fig. 1 below shows the configuration for the test. The scheduling algorithms chosen to test are Completely Fair Scheduler (CFS) and Real-time FIFO (SCHED_FIFO). They will be tested in both a native environment and in an isolation environment CFS represents a general-purpose fair scheduler for a baseline, and SCHED_FIFO enables strict priority-based execution which should provide better performance.

IV. Methodology

A. Operating System

Linux Ubuntu 24.04

B. Scheduling Algorithms

CFS - Completely Fair Scheduler

Real-time FIFO

C. Resource Isolation

Bare Metal (no isolation)

cgroups (CPU and memory limits)

D. Metrics

CPU Usage

Context Switches

System-load Impact

E. Tools

vmstat: system wide statistics

pidstat: process-specific statistics

stress-ng: load generator

cgroups: resource isolation

Bash scripting for automation

F. Test Configurations

TABLE I
A list of the different test configurations.

Test Configurations			
	Scheduler	Environment	Background Load
1	CFS	Bare-metal	No
2	CFS	Bare-metal	Yes
3	FIFO	Bare-metal	No
4	FIFO	Bare-metal	Yes
5	FIFO	Isolated	No
6	FIFO	Isolated	Yes
7	CFS	Isolated	No
8	CFS	Isolated	Yes

G. Game to Be Tested

SuperTuxKart version 1.4

The duration of each configuration will be 300 seconds or five minutes. Each configuration will utilize the same game, played in the same way, for the same amount of time. Supertuxkart is a 3D, open-source racing game that has audio, low-range graphics, keyboard or game-pad input, and various states such as loading screens, game time, etc [9]. The game was chosen for its compatibility with Linux benchmarking tools.

V. Results

The results, as charted in Fig. 1 and Tables II and II, reveal several key ways in which scheduling algorithms, and environmental isolation affect cloud gaming performance particularly in regards to context switches, CPU usage, impact of resource

isolation, and impact of scheduler choice. Isolation reduced context switching by 28% under load. SCHED_FIFO combined with isolation under load had the best performance in terms of context switches at 4734.01 switches per second. In contrast, CFS, without isolation, had the highest rate of switches at 6579.72 switches per second. Again, SCHED_FIFO with isolation under load had the best performance in terms of CPU efficiency at 16.91% usage. Overall, isolation provided reduced overhead, more consistent performance, better load handling and decreased the amount of CPU impact compared to non-isolated tests. While the two scheduling algorithms had similar results when there was no load present, SCHED_FIFO had consistently better performance under load showing lower overhead overall and had the best CPU performance with isolation.

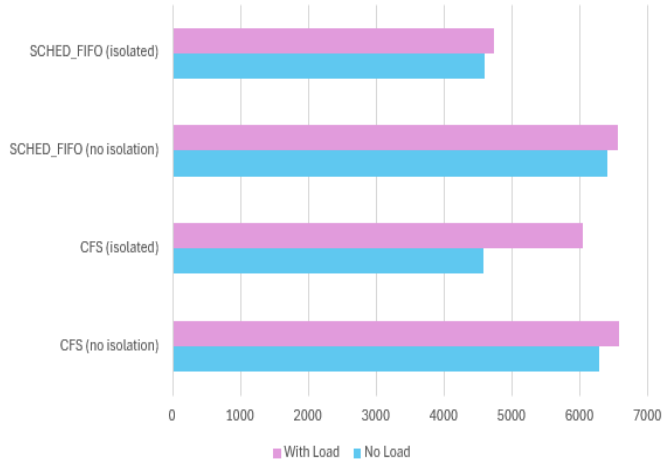

Fig. 1 Context Switches comparison.

In fig 1., the blue bars represent the amount of context switches with no load, and the purple bars represent the amount of context switches with load. The number of context switches for both CFS and SCHED_FIFO remains similar when there is no background load. However, when background load is introduced, the numbers vary more widely. When in an isolated environment with load, SCHED_FIFO outperforms CFS with a considerably smaller amount of context switches.

TABLE II

CPU Usage				
Configuration	User CPU (%)	System CPU (%)	Total (%)	Game CPU (%)
CFS (no isolation)(no load)	1.33	0.1	1.43	12.5
CFS (no isolation)(load)	16.52	0.14	16.67	13.5
CFS (isolated)(no load)	1.88	0.19	2.06	12.0
CFS (isolated)(load)	17.83	0.17	18.00	13.85
SCHED_FIFO (no isolation)(no load)	1.39	0.07	1.47	12.5
SCHED_FIFO (no isolation)(load)	16.48	0.09	16.57	13.5
SCHED_FIFO (isolated)(no load)	1.58	0.12	1.70	12.96
SCHED_FIFO (isolated)(load)	16.79	0.12	16.91	15.16

In table II, CPU usage is explored in terms of user CPU usage, system CPU usage, the total system/user CPU usage and the game CPU usage for each test configuration. In this metric, a higher CPU utilization for the game (Game CPU) is a sign of good performance because it indicates that the application, in this case SuperTuxKart, is getting the CPU time that it needs to run well. SCHED_FIFO, isolated with load, has the highest game CPU usage, indicating that this configuration is making the most efficient use of CPU resources. In the second column, the system CPU usage should be low, as a high percentage here would indicate that there is a lot of system overhead which takes CPU resources away from user applications such as SuperTuxKart. Again, as in Fig 1., there is not much difference between the CFS scheduler and the SCHED_FIFO scheduler when there is no load on the system. The impact becomes clearer in the test configurations with load; the percentage of total system and user CPU usage is much higher when there is background load. Additionally, the test configurations in which the game was run in isolation show small increases in CPU usage by the game, indicating that the isolation environments maintain higher game CPU usage. When comparing CFS to SCHED_FIFO, SCHED_FIFO has consistently lower system CPU usage, paired with similar or higher game CPU usage, indicating higher performance overall for SCHED_FIFO.

TABLE III

Load Impact Analysis				
Configuration	No Load (CS)	Load (CS)	CS Increase	CPU Increase
CFS (no isolation)	6286.39	6579.72	4.67%	15.24%
CFS (isolation)	4584.85	6053.81	32.04%	15.94%
SCHED_FIFO (no isolation)	6407.72	6567.01	2.49%	15.10%
SCHED_FIFO (isolation)	4605.28	4734.01	2.80%	15.21%

CS = Context Switches per second

In Table III, the impact of background load on the various test configurations is outlined: the baseline of no load configurations had less context switches than the configurations with load, as expected. The test configurations in isolation show less context switches with and without load than the ones ran in a bare metal environment. As with previous data sets, isolation shows increased performance. When comparing scheduling algorithms, SCHED_FIFO emerges, again, as the most efficient, with SCHED_FIFO in isolation performing the best with 4734.01 context switches, only a 2.49% increase when load is added. This data set shows the benefits of a more efficient scheduling algorithm, particularly when resource isolation is utilized. This disparity between context switches between load/no load and isolation/no isolation is seen in Fig. 1 as well and could indicate that the CFS scheduler algorithm is a sub-optimal choice for this particular configuration. In comparison, the isolated SCHED_FIFO configuration performs much more efficiently, particularly when comparing the load configurations.

VI. Discussion

The results are consistent with findings in [1], [4] - [8]: resource isolation, combined with more efficient scheduling can significantly contribute to improved cloud gaming performance. The results could have some practical implications for cloud gaming platforms. The increased performance of SCHED_FIFO with isolation (4734.01 context switches versus 6579.72 for CFS non-isolated) suggests that cloud gaming providers could benefit from implementing real-time scheduling policies with resource isolation. A slight increase in overhead due to isolation methods seems to be worth the cost for increased performance, lower amounts of context switches and better handling of load. Isolation, in particular, suggests benefits for multi-user environments and for scalability as more and more users are added. This conclusion aligns with the research outlined in the literature review, though there are some limitations to consider.

There are several limitations to this experiment that need to be considered: a local environment was used to simulate a cloud environment, results may vary on an actual cloud infrastructure, the game used to test was a single game that is not overly complicated compared to triple A titles and other more advanced games. This scope of this project required a minimal amount of test configurations and algorithms. Ideally, a larger scale project would include many more configurations. Despite these limitations, the literature and the results indicate that cloud gaming performance benefits from dynamic scheduling algorithms and isolation environments. Because every game is different, and as gaming technology advances, no single configuration will emerge as best, rather predictive algorithms paired with intelligent learning mechanisms [1], [4], and [5] that can adapt to dynamic workloads will continue to provide the best performance.

VII. Conclusions

Although this project explored a very simplified version of resource allocation in cloud gaming, the results provide an insight into the complicated relationship between scheduling and resource allocation, and highlights the amount of precision needed by both the host and server operating systems in order to provide high performance cloud gaming services. Future studies should include network simulations, as many cloud games are online, multi-user simulations, as many cloud games are multiplayer, and testing should be done

on an actual cloud server. Additionally, as discussed in [5], artificial intelligence has the potential to create improved outcomes for some of the mechanisms discussed in this project.

VIII. Related Work

There is a notable absence of academic literature on gaming, particularly game tests as has been presented in this project. There is no shortage of periodical articles, blog posts or social media posts regarding gaming. Considering the many types of advanced technology used in gaming, this disparity is a bit alarming. Because the gaming industry is so popular, and monetized, it can be posited that many game developers prefer to do research and testing internally to protect proprietary technologies and methodologies. There are a few areas in which gaming research is popular, other than the field of cloud gaming as cited in this project; notably psychology and the field of artificial intelligence. While psychology research tends to lean towards the effect of gaming on the human brain, artificial intelligence research has potential for advancing the computational technologies that drive video games. In [10], a survey of academic studies from 1971-2022 found 2604 academic articles on AI in gaming, and while this is a low number for such a broad timeframe, it is a considerable amount of research when compared to the general lack of gaming research. This disparity could be, at least partly, due to the benefits of using a gaming environment to test AI methodologies and models, as the gaming industry has been using AI for NPCs, computer opponents since the 70s.

IX. Future Implications

Artificial intelligence-enhanced resource management seems to be the direction of the future of cloud gaming. While this project focuses on traditional scheduling approaches (CFS and SCHED_FIFO), the future of cloud gaming resource management likely lies in AI-driven solutions. AI can contribute to predictive methods of resource allocation, intelligent resource isolation methods, categorizing workloads for real-time optimization, and providing intelligent algorithms to manage loads and multiusers on a cloud gaming server [5]. While there are many technical challenges in the field of AI, AI's use in prediction and automation would prove useful to further research in the field of cloud gaming, and in the field gaming in general.

References

[1] H. Wang, M. Yu, Z. Lin, Y. Zhou, and X. Liu, "Fine-Grained Cloud Game Co-location on Heterogeneous Platforms," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, 2024, pp. 115–124.

[2] C. Carrascosa and B. Bellalta, "Cloud gaming: Analysis of Google Stadia traffic," *arXiv preprint arXiv:2004.02499*, 2020.

[3] A. Silberschatz, P. B. Galvin, and G. Gagne, Operating System Concepts, 10th ed. Hoboken, NJ, USA: Wiley, 2018.

[4] H. Fang, Y. Zhuang, M. Li, Y. Huang, and Y. Zhang, "Adrenaline: Adaptive Rendering for Scalable Cloud Gaming," *arXiv preprint arXiv:2403.01390*, 2024.

[5] Y. Gu, Z. Liu, S. Dai, C. Liu, Y. Wang, S. Wang, G. Theodoropoulos, and L. Cheng, "Deep Reinforcement Learning for Job Scheduling and Resource Management in Cloud Computing: An Algorithm-Level Review," *arXiv preprint arXiv:2501.01007*, 2025.

[6] M. Pavlidakis, G. Vasiliadis, S. Mavridis, A. Argyros, A. Chazapis, and A. Bilas, "Guardian: Safe GPU Sharing in Multi-Tenant Environments," *in Proc. 25th Int. Middleware Conf. (Middleware '24)*, 2024, pp. 1–13.

[7] A. Siavashi and M. Momtazpour, "A Multi-Objective Framework for Optimizing GPU-Enabled VM Placement in Cloud Data Centers with Multi-Instance GPU Technology," *arXiv preprint arXiv:2502.01909*, 2025.

[8] W. Khallouli and J. Huang, "Cluster resource scheduling in cloud computing: literature review and research challenges," in *The Journal of Supercomputing*, vol. 78, no. 4, 2022, pp. 6901–6942.

[9] SuperTuxKart Developers, SuperTuxKart Documentation. [Online]. Available: https://doxygen.supertuxkart.net/. [Accessed: May 24, 2025].

[10] L. F. S. Colares, J. C. S. Santos, R. M. C. Andrade and G. A. L. Paillard, "Review and analysis of research on Video Games and Artificial Intelligence: a look back and a step forward," Procedia Computer Science, vol. 204, pp. 315-323, 2022, doi: 10.1016/j.procs.2022.08.038.

[11] "Bash Reference Manual," DevDocs, 2024. [Online]. Available: https://devdocs.io/bash/. [Accessed: Jun. 5, 2024]

Appendix

I. Raw Data

Raw data is shown below, Bash output from running scripts. Scripts outlined in section II of this appendix.

Test One

```
==================== TEST RESULTS ====================
Test Configuration: CFS, Native, No Load
------------------------------------------------

1. System CPU Usage:
User CPU: 1.33%
System CPU: 0.10%
Total CPU: 1.43%

2. SuperTuxKart Process Metrics:
Last 5 entries of process data:
10:00:51 AM  1000    37155  13.00   0.00   0.00   0.00  13.00    7  supertuxkart
10:00:52 AM  1000    37155  12.00   1.00   0.00   0.00  13.00    3  supertuxkart
10:00:53 AM  1000    37155  12.00   0.00   0.00   0.00  12.00    4  supertuxkart
10:00:54 AM  1000    37155  13.00   0.00   0.00   0.00  13.00    4  supertuxkart
Average:     1000    37155  13.48   0.62   0.00   0.02  14.10    -  supertuxkart

3. Complete Summary:

=== Test Summary ===
Configuration: cfs - native - no_load
Date: Thu Jun  5 10:03:14 AM MDT 2025

System Metrics:
CPU Usage (from vmstat):
User CPU: 1.33%
System CPU: 0.10%
Total CPU: 1.43%

Process Metrics (SuperTuxKart):
Average CPU Usage: 0.05%
Average Memory Usage: 0.00%

Context Switches (per second):
Average: 6286.39
```

Test Two

```
==================== TEST RESULTS ====================
Test Configuration: CFS, Native, With Load
------------------------------------------------
```

1. System CPU Usage:
User CPU: 16.52%
System CPU: 0.14%
Total CPU: 16.67%

2. SuperTuxKart Process Metrics:
Last 5 entries of process data:

10:41:41 AM	1000	39494	11.00	0.00	0.00	0.00	11.00	2	supertuxkart
10:41:42 AM	1000	39494	11.00	0.00	0.00	0.00	11.00	5	supertuxkart
10:41:43 AM	1000	39494	13.00	0.00	0.00	0.00	13.00	3	supertuxkart
10:41:44 AM	1000	39494	13.00	1.00	0.00	0.00	14.00	2	supertuxkart
Average:	1000	39494	12.54	0.54	0.00	0.00	13.08	-	supertuxkart

3. Complete Summary:

=== Test Summary ===
Configuration: cfs - native - with_load
Date: Thu Jun 5 10:41:47 AM MDT 2025

System Metrics:
CPU Usage (from vmstat):
User CPU: 16.52%
System CPU: 0.14%
Total CPU: 16.67%

Process Metrics (SuperTuxKart):
Average CPU Usage: 0.05%
Average Memory Usage: 0.00%

Context Switches (per second):
Average: 6579.72

========== COMPARISON WITH PREVIOUS TEST ==========
Previous test (CFS, no load):
- Total CPU: 1.43%
- Context Switches: 6286.39 per second
- SuperTuxKart CPU: ~12-13%

Test Three

========== TEST RESULTS ==========
Test Configuration: SCHED_FIFO, Native, No Load
---

1. System CPU Usage:
User CPU: 1.39%
System CPU: 0.07%
Total CPU: 1.47%

2. SuperTuxKart Process Metrics:
Last 5 entries of process data:

11:03:09 AM 1000 43680 12.00 1.00 0.00 0.00 13.00 4 supertuxkart
11:03:10 AM 1000 43680 12.00 0.00 0.00 0.00 12.00 4 supertuxkart
11:03:11 AM 1000 43680 12.00 1.00 0.00 0.00 13.00 4 supertuxkart
11:03:12 AM 1000 43680 13.00 0.00 0.00 0.00 13.00 5 supertuxkart
Average: 1000 43680 12.86 0.57 0.00 0.01 13.43 - supertuxkart

3. Complete Summary:

=== Test Summary ===
Configuration: sched_fifo - native - no_load
Date: Thu Jun 5 11:03:14 AM MDT 2025

System Metrics:
CPU Usage (from vmstat):
User CPU: 1.39%
System CPU: 0.07%
Total CPU: 1.47%

Process Metrics (SuperTuxKart):
Average CPU Usage: 0.05%
Average Memory Usage: 0.00%

Context Switches (per second):
Average: 6407.72

========== COMPARISON WITH PREVIOUS TESTS ==========
1. CFS, no load:
- Total CPU: 1.43%
- Context Switches: 6286.39 per second
- SuperTuxKart CPU: ~12-13%

2. CFS, with load:
- Total CPU: 16.67%
- Context Switches: 6579.72 per second
- SuperTuxKart CPU: ~11-14%

Test Four

========== TEST RESULTS ==========
Test Configuration: SCHED_FIFO, Native, With Load
---

1. System CPU Usage:
User CPU: 16.48%
System CPU: 0.09%
Total CPU: 16.57%

2. SuperTuxKart Process Metrics:
Last 5 entries of process data:
11:12:24 AM 1000 44427 13.00 0.00 0.00 0.00 13.00 2 supertuxkart
11:12:25 AM 1000 44427 14.00 0.00 0.00 0.00 14.00 3 supertuxkart
11:12:26 AM 1000 44427 13.00 0.00 0.00 0.00 13.00 5 supertuxkart
11:12:27 AM 1000 44427 13.00 1.00 0.00 0.00 14.00 0 supertuxkart

Average: 1000 44427 12.66 0.56 0.00 0.00 13.22 - supertuxkart

3. Complete Summary:

=== Test Summary ===
Configuration: sched_fifo - native - with_load
Date: Thu Jun 5 11:12:29 AM MDT 2025

System Metrics:
CPU Usage (from vmstat):
User CPU: 16.48%
System CPU: 0.09%
Total CPU: 16.57%

Process Metrics (SuperTuxKart):
Average CPU Usage: 0.04%
Average Memory Usage: 0.00%

Context Switches (per second):
Average: 6567.01

========== COMPARISON WITH ALL PREVIOUS TESTS ==========
1. CFS, no load:
- Total CPU: 1.43%
- Context Switches: 6286.39 per second
- SuperTuxKart CPU: ~12-13%

2. CFS, with load:
- Total CPU: 16.67%
- Context Switches: 6579.72 per second
- SuperTuxKart CPU: ~11-14%

3. SCHED_FIFO, no load:
- Total CPU: 1.47%
- Context Switches: 6407.72 per second
- SuperTuxKart CPU: ~12-13%

Test Five

========== SCHED_FIFO ISOLATED TEST RESULTS ==========
Test Configuration: SCHED_FIFO, Isolated, No Load
---

1. System CPU Usage:
User CPU: 1.58%
System CPU: 0.12%
Total CPU: 1.70%

2. SuperTuxKart Process Metrics:
Last 5 entries of process data:
12:50:32 PM 0 88071 12.00 0.00 0.00 0.00 12.00 2 supertuxkart
12:50:33 PM 0 88071 11.00 0.00 0.00 0.00 11.00 2 supertuxkart

12:50:34 PM 0 88071 12.00 1.00 0.00 0.00 13.00 2 supertuxkart
12:50:35 PM 0 88071 11.00 0.00 0.00 0.00 11.00 2 supertuxkart
Average: 0 88071 12.48 0.48 0.00 0.00 12.96 - supertuxkart

3. Complete Summary:

=== Test Summary ===
Configuration: sched_fifo - isolated - no_load
Date: Thu Jun 5 12:50:35 PM MDT 2025

System Metrics:
CPU Usage (from vmstat):
User CPU: 1.58%
System CPU: 0.12%
Total CPU: 1.70%

Process Metrics (SuperTuxKart):
Average CPU Usage: 0.04%
Average Memory Usage: 0.00%

Context Switches (per second):
Average: 4605.28

Test Six

========== FINAL TEST RESULTS ==========
Test Configuration: SCHED_FIFO, Isolated, With Load
---

1. System CPU Usage:
User CPU: 16.79%
System CPU: 0.12%
Total CPU: 16.91%

2. SuperTuxKart Process Metrics:
Last 5 entries of process data:
12:58:59 PM 0 88712 15.00 0.00 0.00 0.00 15.00 10 supertuxkart
12:59:00 PM 0 88712 16.00 0.00 0.00 0.00 16.00 10 supertuxkart
12:59:01 PM 0 88712 15.00 1.00 0.00 0.00 16.00 10 supertuxkart
12:59:02 PM 0 88712 15.00 0.00 0.00 0.00 15.00 10 supertuxkart
Average: 0 88712 14.73 0.43 0.00 0.00 15.16 - supertuxkart

3. Complete Summary:

=== Test Summary ===
Configuration: sched_fifo - isolated - with_load
Date: Thu Jun 5 12:59:03 PM MDT 2025

System Metrics:
CPU Usage (from vmstat):
User CPU: 16.79%
System CPU: 0.12%

Total CPU: 16.91%

Process Metrics (SuperTuxKart):
Average CPU Usage: 0.05%
Average Memory Usage: 0.00%

Context Switches (per second):
Average: 4734.01

Test Seven

========== ISOLATED TEST RESULTS ==========
Test Configuration: CFS, Isolated, No Load
---

1. System CPU Usage:
User CPU: 1.88%
System CPU: 0.19%
Total CPU: 2.06%

2. SuperTuxKart Process Metrics:
Last 5 entries of process data:

10:09:59 AM	1000	37706	13.00	0.00	0.00	0.00	13.00	1	supertuxkart
10:10:00 AM	1000	37706	13.00	0.00	0.00	0.00	13.00	1	supertuxkart
10:10:01 AM	1000	37706	12.00	2.00	0.00	0.00	14.00	1	supertuxkart
10:10:02 AM	1000	37706	14.00	0.00	0.00	0.00	14.00	1	supertuxkart
Average:	1000	37706	12.32	0.64	0.00	0.01	12.96	-	supertuxkart

3. Complete Summary:

=== Test Summary ===
Configuration: cfs - isolated - no_load
Date: Thu Jun 5 10:10:02 AM MDT 2025

System Metrics:
CPU Usage (from vmstat):
User CPU: 1.88%
System CPU: 0.19%
Total CPU: 2.06%

Process Metrics (SuperTuxKart):
Average CPU Usage: 0.05%
Average Memory Usage: 0.00%

Context Switches (per second):
Average: 4584.85

Test Eight

========== ISOLATED WITH LOAD TEST RESULTS ==========
Test Configuration: CFS, Isolated, With Load
---

1. System CPU Usage:
User CPU: 17.83%
System CPU: 0.17%
Total CPU: 18.00%

2. SuperTuxKart Process Metrics:
Last 5 entries of process data:
12:42:17 PM 0 87436 14.00 1.00 0.00 0.00 15.00 0 supertuxkart
12:42:18 PM 0 86705 14.00 0.00 0.00 0.00 14.00 5 supertuxkart
12:42:18 PM 0 87436 12.00 0.00 0.00 0.00 12.00 3 supertuxkart
Average: 0 86705 13.51 0.35 0.00 0.00 13.85 - supertuxkart
Average: 0 87436 13.38 0.47 0.00 0.00 13.85 - supertuxkart

3. Complete Summary:

=== Test Summary ===
Configuration: cfs - isolated - with_load
Date: Thu Jun 5 12:42:19 PM MDT 2025

System Metrics:
CPU Usage (from vmstat):
User CPU: 17.83%
System CPU: 0.17%
Total CPU: 18.00%

Process Metrics (SuperTuxKart):
Average CPU Usage: 0.05%
Average Memory Usage: 0.00%

Context Switches (per second):
Average: 6053.81

II. Bash Scripts

Non-Isolated Test Configuration Script

```
cat << 'EOF' > ~/cloud_gaming_experiment/run_game_test_v6.sh
#!/bin/bash

scheduler=$1
environment=$2
load_condition=$3
DURATION=300

timestamp=$(date +%Y%m%d_%H%M%S)
output_dir=~/cloud_gaming_experiment/raw_data/$scheduler/$environment/$load_condition/$timestamp
mkdir -p $output_dir

echo "Starting test: $scheduler - $environment - $load_condition"
echo "Test will run for 5 minutes"
echo "Results will be saved in: $output_dir"

# Start monitoring
echo "Starting performance monitoring..."

# System statistics monitoring
vmstat 1 $DURATION > "$output_dir/system_metrics.log" &
vmstat_pid=$!

# Detailed process statistics
pidstat 1 $DURATION > "$output_dir/process_metrics.log" &
pidstat_pid=$!

# Start game based on configuration
case "$environment" in
```

```
    "native")
        if [ "$scheduler" = "sched_fifo" ]; then
            echo "Starting SuperTuxKart with SCHED_FIFO on bare metal..."
            # Launch game without sudo for SCHED_FIFO
            supertuxkart --no-graphics-cache --profile-tracks=1 &
        else
            echo "Starting SuperTuxKart with CFS on bare metal..."
            supertuxkart --no-graphics-cache --profile-tracks=1 &
        fi
        game_pid=$!
        ;;
    "docker")
        xhost +local:docker
        if [ "$scheduler" = "sched_fifo" ]; then
            echo "Starting SuperTuxKart with SCHED_FIFO in Docker..."
            docker run --rm \
                -e DISPLAY=$DISPLAY \
                -v /tmp/.X11-unix:/tmp/.X11-unix \
                supertuxkart-test \
                supertuxkart --no-graphics-cache --profile-tracks=1 &
        else
            echo "Starting SuperTuxKart with CFS in Docker..."
            docker run --rm \
                -e DISPLAY=$DISPLAY \
                -v /tmp/.X11-unix:/tmp/.X11-unix \
                supertuxkart-test \
                supertuxkart --no-graphics-cache --profile-tracks=1 &
        fi
        ;;
esac

# If load condition is with_load, start stress-ng
if [ "$load_condition" = "with_load" ]; then
    stress-ng --cpu 2 --timeout ${DURATION}s &
    stress_pid=$!
fi

# Countdown timer
for ((i=DURATION; i>0; i--)); do
    printf "\rTest running: %d seconds remaining..." $i
    sleep 1
done
echo -e "\nTest duration completed."

echo "Cleaning up processes..."
kill $vmstat_pid 2>/dev/null
kill $pidstat_pid 2>/dev/null
if [ "$load_condition" = "with_load" ]; then
    kill $stress_pid 2>/dev/null
fi
pkill supertuxkart
docker stop $(docker ps -q) 2>/dev/null

sleep 2

# Generate summary
echo -e "\n=== Test Summary ===" > "$output_dir/summary.txt"
echo "Configuration: $scheduler - $environment - $load_condition" >> "$output_dir/summary.txt"
echo "Date: $(date)" >> "$output_dir/summary.txt"

echo -e "\nSystem Metrics:" >> "$output_dir/summary.txt"
```

```
echo "CPU Usage (from vmstat):" >> "$output_dir/summary.txt"
awk 'NR>2 {total_usr += $13; total_sys += $14; count++}
    END {
      printf "User CPU: %.2f%%\n", total_usr/count;
      printf "System CPU: %.2f%%\n", total_sys/count;
      printf "Total CPU: %.2f%%\n", (total_usr+total_sys)/count
    }' "$output_dir/system_metrics.log" >> "$output_dir/summary.txt"

echo -e "\nProcess Metrics (SuperTuxKart):" >> "$output_dir/summary.txt"
grep "supertuxkart" "$output_dir/process_metrics.log" | \
    awk '{cpu+=$8; mem+=$12; count++}
        END {
          if(count>0) {
              printf "Average CPU Usage: %.2f%%\n", cpu/count;
              printf "Average Memory Usage: %.2f%%\n", mem/count
          }
        }' >> "$output_dir/summary.txt"

echo -e "\nContext Switches (per second):" >> "$output_dir/summary.txt"
awk 'NR>2 {total += $12; count++}
    END {
      if(count>0) printf "Average: %.2f\n", total/count
    }' "$output_dir/system_metrics.log" >> "$output_dir/summary.txt"

echo -e "\nTest completed. Results saved in: $output_dir"
EOF

chmod +x ~/cloud_gaming_experiment/run_game_test_v6.sh
```

Isolated Test Configuration Bash Script

```
cat << 'EOF' > ~/cloud_gaming_experiment/run_game_test_isolated.sh
#!/bin/bash

scheduler=$1
isolation=$2  # 'none' or 'isolated'
load_condition=$3
DURATION=300

timestamp=$(date +%Y%m%d_%H%M%S)
output_dir=~/cloud_gaming_experiment/raw_data/$scheduler/$isolation/$load_condition/$timestamp
mkdir -p $output_dir

echo "Starting test: $scheduler - $isolation - $load_condition"

# Create cgroup
if [ "$isolation" = "isolated" ]; then
    # Create cgroup and set limits
    sudo cgcreate -g cpu,memory:/game_group
    # Limit to 50% of CPU resources
    sudo cgset -r cpu.shares=512 game_group
    # Limit memory to 2GB
    sudo cgset -r memory.limit_in_bytes=2G game_group
    echo "Resource isolation enabled: CPU shares=512, Memory=2GB"
fi

vmstat 1 $DURATION > "$output_dir/system_metrics.log" &
vmstat_pid=$!
pidstat 1 $DURATION > "$output_dir/process_metrics.log" &
pidstat_pid=$!
```

```
# Start game with configurations applied
if [ "$scheduler" = "sched_fifo" ]; then
    if [ "$isolation" = "isolated" ]; then
        echo "Starting SuperTuxKart with SCHED_FIFO in isolated environment..."
        sudo cgexec -g cpu,memory:/game_group chrt -f 99 supertuxkart --no-graphics-cache --profile-tracks=1 &
    else
        echo "Starting SuperTuxKart with SCHED_FIFO..."
        sudo chrt -f 99 supertuxkart --no-graphics-cache --profile-tracks=1 &
    fi
else
    if [ "$isolation" = "isolated" ]; then
        echo "Starting SuperTuxKart with CFS in isolated environment..."
        sudo cgexec -g cpu,memory:/game_group supertuxkart --no-graphics-cache --profile-tracks=1 &
    else
        echo "Starting SuperTuxKart with CFS..."
        supertuxkart --no-graphics-cache --profile-tracks=1 &
    fi
fi

# If load condition is with_load, start stress-ng
if [ "$load_condition" = "with_load" ]; then
    if [ "$isolation" = "isolated" ]; then
        # Run stress outside the game's cgroup
        stress-ng --cpu 2 --timeout ${DURATION}s &
    else
        stress-ng --cpu 2 --timeout ${DURATION}s &
    fi
    stress_pid=$!
fi

# Countdown timer in shell
for ((i=DURATION; i>0; i--)); do
    printf "\rTest running: %d seconds remaining..." $i
    sleep 1
done
echo -e "\nTest duration completed."

# Cleanup
kill $vmstat_pid 2>/dev/null
kill $pidstat_pid 2>/dev/null
if [ "$load_condition" = "with_load" ]; then
    kill $stress_pid 2>/dev/null
fi
pkill supertuxkart

# Remove cgroup if it was created
if [ "$isolation" = "isolated" ]; then
    sudo cgdelete cpu,memory:/game_group
fi

# Generate results summary
echo -e "\n=== Test Summary ===" > "$output_dir/summary.txt"
echo "Configuration: $scheduler - $isolation - $load_condition" >> "$output_dir/summary.txt"
echo "Date: $(date)" >> "$output_dir/summary.txt"

echo -e "\nSystem Metrics:" >> "$output_dir/summary.txt"
echo "CPU Usage (from vmstat):" >> "$output_dir/summary.txt"
awk 'NR>2 {total_usr += $13; total_sys += $14; count++}
     END {
        printf "User CPU: %.2f%%\n", total_usr/count;
        printf "System CPU: %.2f%%\n", total_sys/count;
```

```
        printf "Total CPU: %.2f%%\n", (total_usr+total_sys)/count
    }' "$output_dir/system_metrics.log" >> "$output_dir/summary.txt"

echo -e "\nProcess Metrics (SuperTuxKart):" >> "$output_dir/summary.txt"
grep "supertuxkart" "$output_dir/process_metrics.log" | \
    awk '{cpu+=$8; mem+=$12; count++}
        END {
          if(count>0) {
              printf "Average CPU Usage: %.2f%%\n", cpu/count;
              printf "Average Memory Usage: %.2f%%\n", mem/count
          }
        }' >> "$output_dir/summary.txt"

echo -e "\nContext Switches (per second):" >> "$output_dir/summary.txt"
awk 'NR>2 {total += $12; count++}
    END {
      if(count>0) printf "Average: %.2f\n", total/count
    }' "$output_dir/system_metrics.log" >> "$output_dir/summary.txt"

echo -e "\nTest completed. Results saved in: $output_dir"
EOF

chmod +x ~/cloud_gaming_experiment/run_game_test_isolated.sh
```

III. Software

Operating System

Linux Ubuntu 24.04

Command Line Interface

GNU Bash
https://www.gnu.org/software/bash/

Bash Scripts help
https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html
https://tldp.org/LDP/abs/html/

Tools

Monitoring tools:
https://www.logicweb.com/knowledge-base/linux-tips/33-command-line-tools-to-monitor-linux-performance/
pidstat (replaced perf, as perf did not want to work for the configurations):
https://www.man7.org/linux/man-pages/man1/pidstat.1.html
cgroups:
https://en.wikipedia.org/wiki/Cgroups

Game Software

SuperTuxKart version 1.4

https://supertuxkart.net/Main_Page