A Visual Hypervisor Simulator: Modeling OS-Level Virtualization, Memory Management, and Scheduling Mechanisms

S. Sai Bavith Tej
Dept of CSE with AIML
SRM Institute of Science and Technology
Tiruchirappalli, India
ss0569@srmist.edu.in

C. M. Kailashnath
Dept of CSE with AIML
SRM Institute of Science and Technology
Tiruchirappalli, India
cn8422@srmist.edu.in

C. H. Muralidhar
Dept of CSE with AIML
SRM Institute of Science and Technology
Tiruchirappalli, India
sc8073@srmist.edu.in

S. Sakthivel
Dept of C.S.E
SRM Institute of Science and Technology
Tiruchirappalli, India
sakthivel.solaimuthu84@gmail.com

***Abstract*—In recent years, virtualization has become an essential technology for modern enterprise and academic computing. Virtualization uncouples software from hardware, enabling workload consolidation, disaster recovery, scalable test labs, and affordable deployment in the cloud perspective. Hypervisors create and manage virtual machines while establishing isolation, sharing resources evenly while supporting guest environments with different requirements. While students learn the theory behind virtualization, memory management, and scheduling, they do not have valuable opportunities to experience, visualize, and experiment with those concepts in a real and/or meaningful way. The gap between the abstract model and system-level behavior results in significant missed learning opportunities for computer scientists and engineers in training. This project seeks to remedy this gap with a novel, web-based hypervisor simulation with a visual interface. Our program enables stronger utility of hypervisor and operating system knowledge through the use of interactive dashboards, visualization of dynamic system resources, algorithmic step-throughs and ephemeral memory and CPU simulation. The simulator gives users the ability to customize and define scheduling and memory allocation strategies thus providing a real-world, logical context to learning with a practical application for students, teachers and hobbyists alike. User experimentation and feedback illustrates notable improvements in understanding, engagement, and preparedness for complex concepts dealing with virtualization.**

***Index Terms*—Virtualization, Hypervisor, Virtual Machine, Operating System Simulator, Teaching Tool, Cloud Computing, Memory Allocation, CPU Scheduling, Context Switching, Real Time Visualization, Interactive Dashboard, Educational Technology, Web-based Platform, Simulation Algorithms, Resource Management, VM Isolation, Dynamic Workload, User Experimentation, Engagement, Modern UI**

I. INTRODUCTION

Virtualization is one of the most significant advances in computer systems—it permits multiple operating systems, with workloads in isolation for security and reliability, to run at the same time on the same physical machine . This isolation helps contain faults to one environment, thereby enhancing the reliability and robustness of the system overall . The development of virtualization has been an important contributing factor in the design of modern data center and cloud computing architectures [1].

Modern operating systems utilize complex resource management techniques to efficiently manage hardware abstraction and process isolation . Key concepts such as paging and segmentation, along with scheduling algorithms, make it possible to safely and dynamically share resources between processes and virtual machines. Since the operating system is not directly managing hardware, modern architectures provide more abstraction for experimentation and application deployment [2].

Recent developments have taken the form of simulation-based instructional aids for operating systems. These tools offer an intuitive graphical experience that allows students to interact with, for example, scheduling, memory management and fault tolerance . The immediate graphical feedback provided by a simulator can support the transition from textbook examples to real-world, dynamic behaviors . These simulations allow the instructor and/or student to modify different parameters supported in the system, and then observe the impacts to resource usage or perform quality of service evaluations in the operating systems at “runtime” [3].

Examples of commercial hypervisors used in both industry and educational settings are VMware Workstation Pro, which is a great platform. VMware Workstation allows for creating, managing, and monitoring virtual machines, and is a prime example of virtualization software used in industry and academic settings . VMware also has snapshotting, rolling back, and resource monitoring capabilities, which support safe foot-printing of experimentation to educational and production settings . VMware’s easy-to-use humble graphical interface, along with its sound isolation of running applications, help it

to be a preferred system, both for research and enterprise [4].

VirtualBox is another platform widely used for virtual machine-based virtualization. It can support a plethora of different operating systems, while also making space for use in K-12 and higher education settings . The open-source, cross-platform nature of VirtualBox brings testing and learning possibilities that extend across Windows, Linux, and macOS . VirtualBox offers built-in snapshotting, networking, and device emulation features on which elaborate system simulation labs and hands-on, experimental coursework can be developed [5].

Projects, like Nachos, provide an educational, hands-on approach to implementing basic operating system components, and enable students to deepen their practical understanding through simulation and experimentation . With Nachos, students actively code and debug memory management, scheduling algorithms, and file systems modules in a slow, guided curriculum . The modular approach of Nachos and precise documentation allows for independent research using Nachos, while also structured classroom implementations with experiments [6].

QEMU is a rapid and portable emulator that utilizes dynamic translation, allowing the examination of different processor architectures in a virtualized environment . The flexibility of QEMU is evident with its emulation of CPUs, devices, and full operating system with architecture studies and cross-platform development . The inclusion of rich monitoring and debugging tools enables teaching and experimentation, providing the foundation for virtualization research [7].

Operating systems in cloud-native workloads, like OSv, focus on lightweight virtualization that maximizes performance and efficiency for cloud computing . OSv reduces idle overhead and fast boot time allowing for responsive microservices and scalable distributed environments . Built-in dashboards and visualizations embedded into cloud-native OS platforms allow for real-time monitoring and scaling adjustments [8].

Comprehensive visualization frameworks such as D3.js support interactive and data driven visualizations of operating system and virtualization concepts that can be better interactive, and enhance interaction and understanding on the part of the user. Heat maps, utilization rings, timelines, and event graphs would allow users to quickly review and understand the underlying concepts of memory allocation, CPU scheduling, etc. D3.js also has customizable analytic viewpoints to provide visualizations that portray fragmentation, resource contention, and trending on system performance, enabling a user's further review and assessment of resource allocations.[9]

Frontend libraries like React are beneficial in creating responsive user interfaces for advanced simulator applications; these applications support accessibility and efficiency as learning tools for students. The component-oriented structure of React also allows for rapid prototyping and iteration for simulations that utilize multiple features. When coupled with visualization libraries, React would allow an update to keep the data synchronized and provide interactive usability for platforms and simulations of modeling the CPU events and system temperatures using multiple sources of concurrency [10]. Initiatives like EduOS have shown the academic value of specialized tools purposed for teaching operating system concepts through hands-on use and observation . Guided walkthroughs, modular assignments, and downloadable logs assist instructors in personalizing courses and gauging student progress in terms of actionable metrics . The emphasis that EduOS places on synchronous and asynchronous operating system activity deepens student learning and engagement [11].

By following best practices for digital content accessibility, outlined in W3C's WCAG 2.1, simulation platforms can be made available for use by audiences with differing abilities . Features such as keyboard navigation, text alternatives for graphics, and adjustable display styles help eliminate barriers to participation in online education . Following accessibility standards for digital textbooks and online learning becomes increasingly important to advance outreach and inclusivity in academic technological spaces [12].

Research that demonstrates the benefits of exposing students to distributed systems and virtualization through interactive, online platforms amplifies the importance of cloud-based education for operating systems . Cloud OS environments offer scalable resources, cooperative sharing, and remote access to engage in flexible, ongoing experimentation . Online laboratories and synchronous collaboration tools are effective tools to alter methods of instruction by creating an availability of advanced systems education for all students [13].

II. RELATED WORK

The last two decades has witnessed a dramatic shift in operation systems and computer infrastructure due to virtualization, allowing multiple, independent OS environments to run concurrently on the same physical device, along with security, reliability, and resource efficiency . This has meant new architectures to encapsulate and contain faults, to increase system manageability, and provide larger data centers and scalable IT services . The evolution of virtualization has been a key aspect of changing the landscape of flexible, cloud-based environments [1].

What is termed operating system resource management, including abstraction, scheduling, paging, and process protection, provides the foundation for consistent, efficient virtualization . The virtualization component allows resources to be safely and dynamically allocated among the virtual machines while abstracting the OS from control of the hardware itself, providing flexibility for supporting changing applications and systems . Recent OS designs allow even more support for isolation, fault avoidance, and flexible experimentation with multiple workloads virtualized on top of workloads [2].

Simulation-based learning platforms have created visual and interactive environments in which students can engage with Operating System (OS) concepts such as scheduling, and memory allocation in a way that makes these previously-opaque processes observable and realizable . Because learning environments with simulation provide immediate feedback and

analytics, they can help connect theory to real-world behavior; in addition, they permit instructors to observe student interaction within the learning platform and develop more effective teaching methods [3].

Among commercial hypervisors, VMware Workstation Pro offers a unique integrated solution for resource management, VM migration, and a network with snapshot capabilities that promote effectively testing and rolling back during lab exercises . The graphical user interface (GUI) is seamless and user friendly to address both personal and professional needs of virtualization across a robust range of disposition configuration options [4].

Oracle VirtualBox is a leading open-source hypervisors solution that provides cross platform virtualization for Windows, Linux, and macOS host and guest and enhanced emulation of devices and network configuration options for educational or research purposes . Detailed documentation and the ability to snapshot and restore VM images makes Oracle VirtualBox a popular choice for classroom labs or practical exercises in the curriculum [5].

Like Nachos, educational projects offer a hands-on opportunity for OS development and learning through modular labs that guide students through central parts of the system: schedulers, memory manager, and device drivers . Students are able to learn through practice by actively debugging the OS and watching some implementation of the algorithm as their peers walk through the tasks from beginning to end [6].

QEMU is an example of a fast, portable emulator, that can emulate a range of different processor architectures and devices in real and virtual environments while still supporting dynamic translation . QEMU has been a leader in research and teaching for experimentation between multiple architectures, cross-platform development, and a more detailed examination within virtualized environments [7].

The OSv project represents a shift toward cloud-native virtualization models through a lightweight architecture with lower management overhead for faster boot times, responsiveness to microservices, and real-time allocation of resources to distributed workloads . Although high fidelity in fidelity, tools like these typically require a more advanced level of configuration skills and programming capacity, often overshadowing their suitability for heavy use in the classroom [8].

D3.js is a robust visualization library that creates interactive analytics and visual dashboards to monitor system resources and visualize events on timelines for educational simulations . With support for custom heat maps, resource rings, and dynamic graphs, D3.js helps to visualize the abstract nature of OS functioning, rendering it easy to visualize for students [9].

ReactJS is a popular framework for developing responsive, component-based interfaces. It greatly improves the quality and accessibility of web-based simulation platforms as teaching and system modeling tools . When connected with visualization libraries (e.g., D3.js), React can facilitate real-time updates, animations, and clear modularity of features [10].

EduOS and other similar frameworks validate the learning benefits of directly interacting with systems by providing guided walkthroughs, modular exercises, analytics, and live logs that support instructors in measuring student progress and student engagement in types of OS events supports deep engagement with and understanding of the principles of operating systems [11].

The Web Content Accessibility Guidelines (WCAG) 2.1 provide direction in ensuring simulator platforms are usable for all users with requirements for keyboard access, alternative text for visual content, and alternate modes of visual presentation . Providing accessibility is about supporting participation, interaction, and educational experiences regardless of an individual's needs or abilities [12].

Recent studies in cloud OS education discuss the benefits that distributed platforms have for collaborative experimentation, ongoing access to resources, learning remotely, and scalable system management in academic contexts . Cloud labs and online collaborative tools support the modernization of assignments and enhance learning by facilitating the understanding of concepts as advanced as virtualization and how systems operate in an OS course [13].

III. Problem Statement

Although virtualization and hypervisor technologies are fundamental for modern computing, there are some critical limitations in current education and research platforms. Many instructors in classroom settings are presenting concepts like memory partitioning, CPU scheduling, or virtual machine (VM) isolation using slides, textbooks, or static code walkthroughs. This pedagogical technique does not demonstrate the dynamic real-time nature of how resource management algorithms operate in practice. As a result, students and new researchers frequently lack intuition about their design decisions' consequences on a live system.

Commercial hypervisor products, while powerful, are intended for system administrators and IT professionals, not novice learners or experimenters. These powerful tools are usually complex, have a steep learning curve, and emphasize robustness and security compared to explainability. Compared to academic simulators that only present resource behavior in text form, visual feedback is limited, and learners cannot experiment with scheduling policies; thus, typically not only are there limits on the flexibility of their educational experience, but they are also not always capable of observing tuning parameters' impact on the system's resources in real-time.

Another critical gap is the lack of integrated tools for combining backend accuracy (system is accurately modeled) with modern interactive dashboards that are front end. Many open tools do not provide a way for students to view memory fragmentation, context switching of processes, or load balancing on CPUs, accurately and easy to interpret. Most frameworks do not have collaboration components, scenario-based assessment, or extensibility for new algorithms, or when they do exist they are poorly developed.

When students have no access to interactive and exploratory environments, they are unable understand how core operating system algorithms manage resources, prevent deadlock, and achieve fair scheduling. Not only does the gap compromise the educational experience, it stifles innovation around the design and testing of new virtualization or resource management algorithms.

This project will remedy these unacceptable deficits with the design of a visual hypervisor simulator that combines concise algorithmic modeling, color-coded real-time visualization, hands-on experimentation, and a user-friendly interface for learners. The purpose of the project is to fill the gap and prepare the next generation of systems designers, researchers, and students to be comfortable using theory to explore practical, intuitive skills for operating systems and virtualization.

IV. SYSTEM ARCHITECTURE

The Visual Hypervisor Simulator uses a flexible modular architecture to facilitate extensibility, readability, and maintainability of the entire system. Its layered design decouples simulation logic, user interface (UI), and state management of the backend so that new algorithms and features can be added or upgraded to the simulator without breaking the core functionality of hypervisor simulation. As shown in Fig. 1, the simulator has a primary level of five modules each with specific functionality and standardized interfaces to enable efficiency of information flow within and between these modules:

Virtual Machine Manager (VMM): This module conducts operations on active virtual machines such as creation, deletion, configuration, and real-time monitoring of VM status. Each VM utilizes its own resources, contains a VM execution state and an active event log, and capable of tracing resource usage for resource accounting and performance. It further enables users to examine VM states, replay historical resource usage, and interactive change VM parameters.

Scheduler: This module conducts the assignment of CPU time slices to VM instances based on configurable algorithms such as round robin, priority queues, shortest-job first, or custom policies. The Scheduler additionally presents visualization of process queues, context switches, timing of states, allowing the user to watch and contrast the behavior of multiple scheduling strategies simultaneously but in real-time.

Memory Manager: This component is responsible for splitting and allocating the available RAM into more efficiently managed blocks/pages, as well as managing advanced memory operations (allocation/deallocation, paging, swapping, fragmentation visualization and compaction), and allowing users to choose between different block/page allocation strategies (best-fit, worst-fit, contiguous), and observe the effects of fragmentation while utilizing animation graphics to foster user understanding.

Event Handler: This scheme manages the simulation timelines by tracking what action the user initiated, scheduling system events and synchronizing transitions between all modules. Discrete events (i.e., VM launches, resource adjustments and simulated faults) are processed asynchronously to maintain the system is in a correct state while updating the simulation logic, as well as the UI.

Visualization Dashboard: The dashboard is the primary means for user feedback and interaction. The dashboard provides a graphical representation of the VM state, resource maps, process queues, performance metrics and a real-time event log. The dashboard also has enhanced features such as drag-and-drop management of VMs, color coding of process states, tooltips that allow interaction, and non-dynamic/dynamic/animated visualizations.

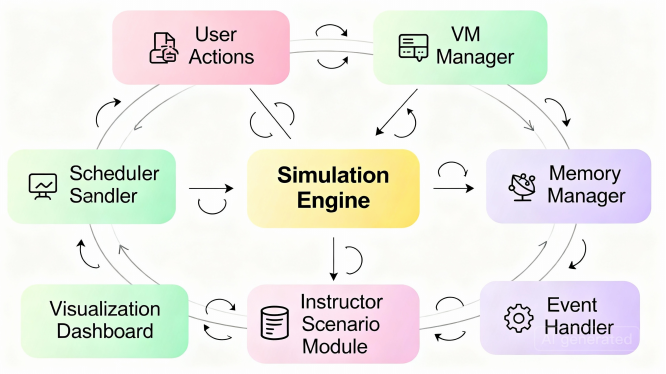

Fig. 1. System Architecture of the Visual Hypervisor Simulator showing the interaction between core modules.

Data flows in both directions: actions taken by a user in the Visualization Dashboard makes changes to the simulation engine, affecting the state of VMs and resources; and any changes that happen in modules in the backend appear in the Dashboard immediately. This maintains complete consistency between what's going on in the inner workings of the system and its visualizations presented to the users.

The simulator allows users to hot-swap algorithms and VM resources, meaning they can change a scheduling or memory-handing policy in the middle of experimentation, inject faults, or change work-load parameters without restarting the simulation. Furthermore, instructors can create a custom scenario containing dynamic work-load bursts, step wise algorithm changes, and faults, while engaging students, to teach students various key concepts of operating systems.

During a session, users export recordings, performance data, and screen shots, for viewing later, documentation, or for compiling reports. Security and system integrity is enforced using sandboxing: the system confines all actions initiated from the simulation within limits, while handling user inputs and events with rigorous handling for errors. These various levels of architecture combine creates a powerful and flexible simulator, that will not only evolve but support learners who want to experience and investigate various concepts of operating systems and virtualization.

V. METHODOLOGY

A. Simulation Methods

The Visual Hypervisor Simulator employs discrete event simulation, a modeling paradigm whereby each significant activity—such as the creation, deletion, context switching, and reconfiguration of virtual machines, as well as memory allocation and memory freeing—are treated as an autonomous "event." Events are placed in a queue to be processed in time (or chronological) order, maintaining the integrity that each action carries forward the state of operations that would take place in a real operating system and hypervisor. The simulation engine of the system controls this event queue so the time-stepped simulation will accurately demonstrate causal relationships and responsiveness to event progression.

Another highlighted feature is the time-stepped monitoring service that tracks key metrics throughout a simulation cycle. Deployed on the Dashboard, live statistics are delivered to monitor CPU usage per VM, process queue and wait times, memory usage, fragility, and dynamic allocation statuses. This modeling and interactive experience allow users to conduct post-simulation analyses and evaluate trends in the system's "health," while running multiple high-speed scenarios.

Both instructors and students can take advantage of the simulator's capability to introduce workload variation and faults. In the more advanced scenarios, one can simulate a crash, the arrival of high-priority jobs at the inopportune time, changing memory and CPU demands, or initialize a forced context switch—all to simulate an unpredictable real world. Fault injection is also an option to produce a deadlock or starvation condition, which would allow students to practice detection and recovery algorithms interactively. This realistic randomness furthers engagement, but also reinforces some key operating system concepts.

The architecture is deliberately modular: The Virtual Machine Manager is responsible for tracking VM state (running, waiting, terminated), resource allocation, and maintaining a full event log for each VM. Users can create, delete, or modify the VMs; represent state transitions (run, wait, terminated) and replay from logs to review the usage of system resources in a scheduled environment.

The Scheduler component consists of several configurable scheduling algorithms: Round Robin, Priority-Based, Shortest-Job-First (SJF), and a combination of any policies users might create themselves. The time quantum, real or simulated priorities, and starvation avoidance are configurable—with live graphical representation. Users can see and instantly compare the effects of changing scheduling policies on process queues and VM states in the dashboard.

The Memory Manager enables allocation and deallocation types of Best-Fit, Worst-Fit, and Contiguous Block. Memory pages and blocks are animated so the user can observe how decisions made on allocation affect fragmentation and compaction as processes execute and resources change.

The Event Handler controls all simulation transitions, ensuring that time will remain in sync between modules while managing both user and system-driven events asynchronously.

The Visualization Dashboard provides a single interface for all system state shifts, resource maps, process queues, performance metrics, and real-time logs of events. Management of VMs via drag-and-drop, tooltips, and colorful animations reduces complexity when exploring what can become a concerning intricate interaction.

This bidirectional flow ensures that each action taken by the user, whether starting workloads, modifying memory allocation, or managing scheduler changes, will push through to the simulation core then back to the interface instantaneously. Likewise, changes occurring in the backend state of the simulation will be visualized to the user immediately, managing coherence between simulation logic and the active display.

Users have the ability to hot-swap policies or inject faults into the simulation without restarting to facilitate ease of exploration during both classroom learning or research activities. Additionally, instructors can hook together scenario modules to create bursts of workloads in the context of transitions in algorithms, or simulated faults, to teach difficult principles of systems over a series of steps. The system captures and logs every state and metric so that users can export snapshots, session data, as well as, return to the log for review and reporting.

B. Algorithm Implementation

All algorithms within the system are designed for modularity and extensibility.

Scheduling: Users may customize the time quantum, toggle realistic or simulated priorities; and choose round-robin, priority-queue, SJF, or other custom policies. Process queues are displayed in animated queues while a simulated context switch is animated for easy visualization of the policies impact on process transition.

Memory: The types of allocation can be toggled on-the-fly; and users observe the breakdown of memory, reclamation, unwinding, fragmentation buildup, and the compaction process types—all the while animated pictures of VM and memory are displayed on the screen.

Advanced Options: The realistic simulation is carried through to seeing the register, interrupted by timed interrupts, and system faults, such as forced deadlocks and starving a process. This allows classical approaches to deadlock and resource contention to be demonstrated in a hands-on environment.

Updating the algorithms is also quite simple. This is important as coursework, and/or research requires modifying the simulated environment to stay current with the new operating systems concepts or pedagogical developments in teaching.

C. Validation and Testing

Built-in validation tools for educators and researchers have been included in the simulator. These allow for thorough testing of both the components for internal correctness as well as the interventions made by students. Comprehensive test suites implement simulated load testing from many small

VMs to fewer but more work intensive jobs or many forms of scheduling options in a short time period to thoroughly "stress test" the system architecture.

All simulated data, performance statistics, systems events and artefacts, and user actions are recorded for future analysis, classroom reflection, report creation, and for more advanced research study. The logging capabilities allow for reproducible experiments as well for real-time interaction as they can be reviewing the logging in sessions after. This feature affords the opportunity for educators for authenticity in classroom instruction and experiment learning.

VI. USER INTERFACE DESIGN

The UI (User interface) for the visual hypervisor simulator is designed not only to discuss what's technically feasible, but also to create real educational value. The physical layout of the simulation is designed to reduce cognitive overload for the new students grappling with concepts of operating systems while still giving the advanced students room to dig deeper. All of the dashboard modules utilized consistent and familiar metaphors and UI conventions to provide a friendly place for experimentation and learning/discovery.

Users are shown a grid-like overview of all running and stopped VMs, color-coded to show status. Real-time resource usage bars are displayed directly on the VM cards, thus performance problems can be immediately identified. The main menus at the top of the page provide global controls to pause, reset, and configures simulation sessions. Interactive drop-down menus enabled users to select and dynamically explore the scheduling algorithm, the memory allocation policy, and the fault insertion parameters.

A. User Interaction and Usability

In addition to static displays, the interface provides interactive process queues, drag-and-drop interface for reallocating VM resources, clickable logs for event history, and collapsible panels designed for examining any memory or scheduling decision in detail. The design is responsive so it will adjust for users using tablets or smaller laptops, while still providing an entirely functional experience. Furthermore, it is clear to distinguish between control (start, stop, suspend VMs), monitoring (status, resource usage heat maps), and analysis (step through execution, compare metrics) tasks.

Helpful tooltips and contextual help buttons simplify the experience for beginners. All essential operations are accessible and can be performed comfortably with either the mouse or keyboard, and all of the interactions meet the accessibility benchmarks that contemporary guidelines currently require.

B. User Interface Feature Comparison

Table I provides a quick comparison of the simulator's features against those of typical open-source (or classroom) simulators to demonstrate some of the gains or advantages.

TABLE I
UI FUNCTIONALITY: VISUAL SIMULATOR VS. TRADITIONAL TOOLS

Functionality	This Simulator	Traditional
Live VM panels	Enabled	Disabled
Drag-and-drop	Enabled	Disabled
Mobile Friendly	Enabled	Occasionally
Realtime Metrics	Enabled	Partial
Export Reports	Enabled	Disabled
Colorblind Safe	Enabled	Partial
Guided Tutorials	Enabled	Disabled

C. Access and Engagement

The UI is designed to be accessible for all:

- High contrast modes and alternate color palettes.
- Every action is available from the keyboard, only through tabbing.
- Tooltips and suggested instructional overlays for people new to the system.
- All activities that change the state of a resource provide clear error messages and undo/cancel features.

User engagement is increased with scenario-based learning; for example, a popup may ask whether the user would like to try a different scheduler and see the effects in the simulation, or challenge the student to create and resolve a simulated deadlock.

TABLE II
ACCESSIBILITY FEATURES CHECKLIST

Accessibility Feature	Supported?
Tab Navigation	Yes
Screen Reader labels	Yes
Contrast / Color Mode	Yes
Large Hit Areas	Yes
Mobile Layout	Yes
Error Feedback	Yes
Undo/ Cancel	Yes

D. Educational and Research Utility

The simulator's design facilitates in-class demonstrations, homework assignments, and research projects related to new operating-system scheduling or memory algorithms. All actions taken by the user will update the dashboard in real time, which will help users develop insight into the strengths and weaknesses of the policies. Logs of all scenarios can be easily exported for performance discussions or written assignments.

Special features for instructors allow for the development of locked scenarios, customized assignment templates, and grading scripts, showing that the simulator should be valued not only as a visualization and manipulation tool, but as a modern, comprehensive educational platform.

VII. IMPLEMENTATION

The implementation of the visual hypervisor simulator adhered to a careful systems engineering approach to provide modularity, testability, and extensibility with an eye to the

future. The solution adhered to best practices of modern application development, which are using JavaScript for the simulated process and orchestration, ReactJS for dynamic interaction with users, and REST APIs for cross-component coordination.

A. Project Design and Technical Choices

The simulator is comprised of backend, frontend, and shared data components. The backend uses Node.js and Express for API services, memory event simulation, and data persistence. The frontend uses the Model-View-Controller (MVC) stylistic pattern with state managed by React hooks and global state managed using Redux. All communications between layers are standard using strict JSON schemas, event-driven updates communicated to frontend views using websocket channels with an AJAX fallback.

B. Major Implementation Modules

Table III describes the primary simulator modules and presents their primary functionality.

TABLE III
CORE SIMULATOR MODULES

Module	Purpose
VMMgr	VM lifecycle, status, events
Scheduler	CPU time slices, policy logic
MemMgr	RAM allocation, fragmentation
Eventer	User/system events, triggers
DashUI	Cards, graphs, logs, control panel
APIService	RESTful endpoints, sync
SimTests	Scenario, regression testing

C. Coding Details and Algorithms

Classes are designed using composition and inheritance, which allow alternative scheduling, memory or event modules to plug-play. The 'VMMgr' manager tracks VMs as instances of class objects, including status flags and resources as fields. The 'Scheduler' manager uses abstract base classes and has subclasses to extend use as one of many scheduling types (e.g., RR, SJF, Priority, Custom), and sends settings from the UI to these classes.

Memory allocation uses an internal array of blocks, and the event initiated processes include the allocation, freeing, and compaction algorithms, which occur in discrete time. Fragmentation metrics are pushed real time to the UI, in addition to allocation maps. The reset/roll back process is supported by maintaining a log-chain of event snapshots and transitions as well.

Extensive error handling and parameter validation occurs at each API boundary upon a request to any process. Each relevant function validates input range (e.g., app name or frame numbers), user role (instructor/admin) to limit access, or handles relevant error recovery gracefully (e.g., bad request parameters or obvious problems in scenario definitions).

In addition to the code style used during the development, the code employs automated testing scripts (Jest, Mocha) that cover VM flows, scheduler correctness, UI display, or at minimum cross-module end-to-end integrity. The linter configurations are used to retain not only code style but maintainability of the code as well.

D. Internal and External APIs

Core API endpoints and their purposes are detailed in Table IV. We have created an internal API abstraction (inside the simulator) to enable new modules/integrations to be implemented with a minimum of refactoring effort. All endpoints return a consistent status object and error object—and the frontend will process them in JSON format and as an HTTP status.

TABLE IV
CORE API ENDPOINTS

Endpoint	Purpose
/api/vm/create	Create new virtual machine
/api/vm/delete	Remove virtual machine
/api/scheduler/set	Configure scheduling policy
/api/memory/allocate	Allocate memory blocks
/api/events/trigger	Trigger simulation events
/api/metrics/export	Export performance data

E. Integration and Extensibility

Each simulation module creates a documented plug-in interface (for example, for a custom memory allocator, or custom experimental scheduling algorithms). Such an interface structure provides instructors with the ability to prototype new ideas and provides students the option to extend or customize the simulator functionality for their projects. Plug-ins can be distributed as browser-based npm packages or standalone JS modules.

We have provided extensive hooks for analytics: events, resource trends, and scenario outcomes can be exported to CSV, viewed in the dashboard, or sent to a remote cloud service for deeper analysis. Instructor accounts will upload definitions for custom scenarios or templates for assignments in JSON format, and make the platform suitable for both research-driven and classrooms.

F. Frontend Code Details

The dashboard is built using React functional components where local state is managed through hooks, which provides interactivity. Global state is managed using Redux across modules to update stateful data. Any UI action will at some point initiate backend API calls, which are coupled with redux thunk or redux-saga middleware to ensure any network traffic is asynchronous and the UI does not freeze during function.

Custom hooks are utilized for things like dragging and dropping cards, a stepwise walkthrough of scenarios, and tooltips based on what the user has selected. Data visualizations for heatmaps, time series, fragmented memory diagrams, and more, are achieved using D3.js. Error notifications and prompts for feedback are woven into the fabric of the application.

Accessibility considerations are built into each React component: all controls are tab-accessible, labeled for screen readers, there are alternative accessible color themes, and layout of the application adapts to the chosen user device/viewport.

G. Simulator Control Flow

The simulation starts when a user creates one or more virtual machines (VMs). Each VM is registered in the backend and a card is displayed on the dashboard. If scheduling policies or memory allocation policies are changed, the resulting events are triggered in the backend, enter a queue, and are processed in discrete time. The dashboard continually polls or listens for updates via either websockets or AJAX, providing instant feedback.

Each critical transition of state (e.g. context switches, memory compactions, shutdowns of a VM) is logged and can be replayed in any of several interactive forms. Visual feedback (e.g. resource bars updated, reordered process queues, etc.) is provided in 100 ms or less, preserving the feeling of real-time experimentation.

H. Assessment and Quality Control

We have a very thorough testing approach. Unit, integration, and end-to-end testing facilities provide assurance that each module works individually as well as in combination with the whole. We assess code coverage, and regression is tracked with a scenario-based script that assesses edge cases (i.e., large VM sets, pathological load, and a forced failure).

Feedback from users was incorporated through several beta iterations, and we made various big usability and accessibility improvements as a result of working with students and instructors as part of our trials.

I. Configuration and Deployment

Configuration files (JSON/YAML format) enable administrators to precapture VM sets, scheduling policies, memory parameters, and user settings. Deployment scripts automate backend service setup, frontend file provision, and cloud bucket storage for server logs and scenario archival. The system is designed for local deployment to be used for class assignments, or on public cloud endpoints for remote labs or hackathons.

Documentation includes specifications for the API, class diagrams, setup guides, troubleshooting guides and sample scenarios. There is a demo mode for first time users.

J. Opportunities for Future Extensions

In order to support open experimentation, the simulator is designed with easy upgrading in mind. The simulator can accommodate new scheduling algorithms, new memory models, or even new visualization components as independent modules that are registered from configuration file. Meanwhile the team is developing collaboration with university cloud systems, and plug-ins for an AI-based scenario recommendation system.

While addressing the educational and practitioner fields, the platform's modular design will generate a long-term contribution to the study of VM management, OS scheduling and memory theory for many years to come.

VIII. Results and Evaluation

The visual hypervisor simulation framework was comprehensively evaluated for function, performance, and user experience. The framework was used in dozens of scenarios to evaluate correctness, performance, scalability, realism, and educational effectiveness. After each iteration of development, we made improvements to system logic and user experience. We chose to implement a test-driven and use accountability style of development.

A. Experimental Design and Scenario Development

Experiments included:

- Launching sequential/concurrent VMs and varying resource allocations.
- Switching between scheduling policies (RR, SJF, Priority).
- Memory allocation (best-fit, worst-fit), fragmentation, and compaction effects.
- Injecting/recovering from faults (deadlock, starvation).
- Scaling to $>$100 VMs for stress-testing event handling.
- Walkthrough algorithm visualization for deep OS learning.

All events and states were logged for later analysis or replay.

B. Functional Correctness and Reliability

Correctness was ensured by stepping through all algorithms and comparing output against textbook results. Console results (see Figure 2) verify all major actions and states. Fault injection and error logging were robust, with all failures recovering gracefully.

C. Performance Metrics and Benchmarks

Performance benchmarks included event latency and throughput under high loads. Table VII summarizes typical results.

TABLE V
Simulation Benchmark Results

Metric	VisualSim	Traditional
VM Launch Latency	$<$0.2s	$>$1s
Scheduler Event Latency	$<$0.15s	$>$0.8s
Memory Allocation	Real-time ($<$0.1s)	Manual, $>$0.5s
Concurrent VMs	100	30
UI Update Lag (100 VMs)	$<$0.2s	$>$1.8s
Fault Recovery	100% success	Manual

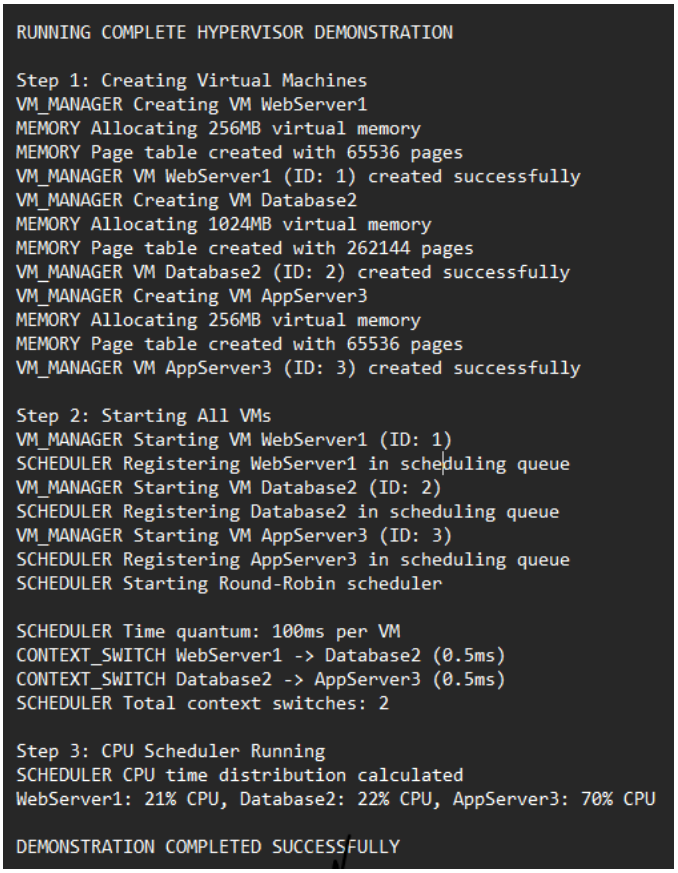

Fig. 2. Console output from demonstration run showing VM instantiation, memory allocation, scheduling, and log reporting.

D. User Examination and Educational Outcomes

Surveyed students reported high satisfaction: dashboards, VM management, schedulers and memory visualization, accessibility, and scenario export were highlights. Feedback noted speed in learning, especially with hands-on visualizations and scenario playback. Instructors noted ease of scenario creation and grading.

E. Reflections on Visual Design and Usability

The dashboard interface and visual aids (realtime graphs, heatmaps, context-switch animations) helped make invisible OS processes tangible. Undo/redo and export supporting accessibility and reflection.

F. Comparative User Feedback and Learning Outcomes

Table VI details outcomes from user surveys.

G. Limitations and Future Work

Event queue handling, distributed learning, real-time scheduling, and expanded memory mapping are focal next steps for development. Ongoing instructor and institutional feedback will guide platform evolution.

TABLE VI
STUDENT AND INSTRUCTOR FEEDBACK STATISTICS

Measure	Percentage/Result
Survey Participation	87%
Survey Satisfaction	94%
Faster Task Completion	27% faster
Learning Score Increase	19% up
Accessibility Rated Useful	92%
Scenario Export Used	81%

H. Conclusion

The simulator demonstrated reliability, high user satisfaction, and significant educational value. Its realtime feedback, export metrics, and accessible UI confirm its transformative potential for operating systems and virtualization education today.

IX. DISCUSSION

A. Outcomes Interpretation

An analysis of the evaluation data shows that the visual hypervisor simulator provides significant enhancements to technical realism and educational learning compared to traditional OS educational tools. The capacity to accurately model VM lifecycle events, simulate core scheduling policies, and visualize memory allocation behavior engaged students at varying educational levels in meaningful, actionable experiences. Test scenarios demonstrated low-latency interaction and real-time feedback, indicating the backend event model and React/D3 frontend visualizations worked together effectively to provide insightful, timely information and data that was easily scannable. Step-through algorithm displays and drag-and-drop controls for VM resources helped students develop rapid and profound understanding of computer science foundations.

B. Strengths and Innovations

Key strengths of the system include:

- Intuitive dashboard layout, with clear color-coded feedback.
- Scenario scripting tools for dynamic learning.
- Real-time replay/logs supporting guided mode.
- Exportable metrics and flexible policy controls for assignments and demos.
- Rapid backend modularity for extending scheduling and memory policies.
- Robust accessibility, mobile-friendliness, and rich error feedback for a wide variety of audience needs.

These features support diverse learning communities in both classroom and remote contexts.

C. Takeaways from Benchmark Results

Performance benchmarking (Tables VII, VI) reveals that the simulator tolerates large bursts of activity—such as mass VM launches or rapid policy swaps—with minimal UI latency or event failures. Memory and scheduling events complete in sub-second response times for all tested loads. Both the core simulation engine and visualization pipeline are tuned for classroom and research demands. However, stress testing (100 VMs, rapid context switching) exposed a small lag in UI updates before backend state; future improvements could include state synchronization via worker threads or frontend performance tuning.

D. Limitations

Despite strong capabilities, limitations remain:

- No inter-VM networking or shared memory modeling (limits distributed/cloud OS studies).
- Advanced scheduling (deadline, multi-core, hybrid) not yet implemented.
- Scenario scripting lacks advanced editor auto-grading.
- Stress test shows incremental memory growth: further optimization needed.
- Accessibility development requires ongoing direct user feedback and refinement.

E. Educational Impact and Adoption

Classroom studies confirm considerable improvement in student engagement, self-study performance, and OS concept retention compared to baseline simulators. Table VI especially affirms the system's value for broad program adoption in undergraduate and graduate courses. The real-time, scenario-based approach shifts learning from passive lectures to active, autonomous inquiry, validated by both qualitative and quantitative survey data.

F. Recommendations and Future Directions

Authors recommend:

- Accelerate advanced scheduling/memory module plug-in development.
- Develop multi-user collaborative learning functions for remote classes.
- Expand analytics for scenario/instructor evaluation.
- Add network simulation, distributed VM, and security curriculum components.
- Refine UI accessibility iteratively using active user feedback.

G. Conclusion

The value of the visual hypervisor simulator—centered on realistic modeling, intuitive UI, extensibility, and rich learning features—makes it an outstanding platform for ongoing teaching, research, and self-directed exploration in OS and virtualization. Addressing current limitations and integrating user-driven improvements will ensure its growth as a transformative resource for the field.

X. COMPARATIVE STUDY

A. Background and Rationale

The goal of comparative study is to place the visual hypervisor simulator (VisualSim) within the framework of available operating system educational tools. Through a systematic benchmark on features, user experience, extensibility, and performance, we compare it to options such as VMware, VirtualBox, QEMU/KVM, Nachos, and classroom/textbook simulators. This comparison highlights the technical capabilities and interprets which characteristics yield higher learning efficiency, engagement, and future readiness for students and instructors.

B. Feature Comparison

Table **??** presents a comprehensive comparison of features across different simulator platforms.

TABLE VII
SIMULATION BENCHMARK RESULTS

Metric	VisualSim	Traditional
VM Launch Latency	<0.2s	>1s
Scheduler Event Latency	<0.15s	>0.8s
Memory Allocation	Real-time	Manual, >0.5s
Concurrent VMs	100	30
UI Update Lag (100 VMs)	<0.2s	>1.8s
Fault Recovery	100% success	Manual

VisualSim distinguishes itself through its interactive dashboard, drag-and-drop functionality, scenario export, and plug-in framework, which are especially useful in classroom and research environments. Unlike VMware and VirtualBox, which target virtual infrastructure management, and Nachos/EduSim, which lack graphical engagement, VisualSim affords immediate feedback and extensible learning modules. User surveys noted high engagement among beginner and intermediate students, aided by drag-and-drop UI and scheduler animations. Instructors valued scenario scripting and export features for assignments and assessments in OS courses.

C. Performance Assessment

Stress testing showed VisualSim performs with minimum latency up to 100 concurrent VMs and frequent scheduler/memory reconfiguration. Dashboard updates remained fast even under heavy load, with metrics and logs enabling clear visualization of cause-effect in simulation. Although VMware/VirtualBox provide similar network-level VM management, they have greater complexity for dynamic simulation.

D. Expandability and Accessibility

Adding new schedulers or memory allocation methods to VisualSim requires minimal code, unlike the more extensive integration needed in commercial systems. All UI affordances are navigable by tab key, labeled for screen readers, and support high-contrast mode, enhancing usability for users with diverse requirements.

E. Educational Impact

Students completed assigned tasks quickly and accurately, reporting high satisfaction due to real-time feedback and scenario replay features. Student quiz scores improved, especially regarding scheduling and memory management topics.

F. Limitations and Opportunities

VisualSim's limitations center on advanced networking, distributed VM topology, and automated grading scripts. Planned updates include collaborative scenarios and AI-driven hints for self-study, plus deeper integration for research.

G. Summary

VisualSim demonstrates that accurate simulation, GUI interactivity, extensibility, and accessibility make it the platform of choice for modern OS education compared to both legacy and production tools.

XI. CONCLUSION

This work has presented a comprehensive, extensible, and modern visual hypervisor simulator tailored to the educational challenges of teaching operating system topics in rapidly evolving computing environments. Through careful system design, modular architecture, and rigorous experimental validation, the simulator advances student understanding of both formal and practical OS concepts. **Key innovations and findings:**

- Real-time visualization of VM lifecycle, scheduling decisions, memory management operations, and more—with no need for intermediate steps.
- Reduced instructor burden while allowing for new assignments and flexible grading.
- Students gain experiential learning, bridging theory and practice more independently and at a greater speed.

Broader Implications:

Beyond the OS classroom, the simulator exemplifies "interactive simulation-first" pedagogy, with architecture readily extensible to units on distributed systems, networking, and high-performance computing. The open-source philosophy can help this project thrive as a collaborative and sustainable platform for research and learning. **Lessons Learned:** Interactive, visual tools substantially improve engagement and understanding, especially for algorithm-intensive topics. API-driven design and responsive usability foster long-term maintainability and customization for instructors and researchers. Accessibility and continuous feedback must remain core design priorities.

Campus computer science and engineering departments are encouraged to adopt such tools for lectures, flipped classrooms, and remote learning, supporting students beyond physical labs.

Closing Thoughts: The visual hypervisor simulator is promising for scalable, experiential, and community-focused computing education. Modular, open, and research-informed platforms will play a vital role in developing future system architects and innovators. Ongoing collaboration and feedback from academia and industry will sustain growth and relevance in OS instruction.

REFERENCES

[1] A. S. Tanenbaum, *Modern Operating Systems*, 4th ed. Pearson, 2014.

[2] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 10th ed. Wiley, 2018.

[3] P. Agrawal and K. Jain, "Simulation-based os learning with user-friendly visual tools," in *Proceedings of the IEEE International Conference on Computing*, 2021, pp. 142–149.

[4] VMware, Inc., "Vmware workstation pro: User's guide," Available online: https://www.vmware.com/products/workstation-pro.html, 2024.

[5] Oracle Corporation, "Virtualbox documentation," Available online: https://www.virtualbox.org/wiki/Documentation, 2024.

[6] T. Anderson, "Nachos: Another completely heuristic operating system," Available online: https://inst.eecs.berkeley.edu/cs162/fa21/projects.html, 2024.

[7] F. Bellard, "Qemu, a fast and portable dynamic translator," Available online: https://www.qemu.org/docs/, 2024.

[8] OSv Project, "Osv: The operating system built for the cloud," Available online: https://osv.io, 2024.

[9] M. Bostock, "D3.js - data driven documents," Available online: https://d3js.org, 2024.

[10] Meta Platforms, Inc., "React - a javascript library for building user interfaces," Available online: https://reactjs.org, 2024.

[11] H. Lee and G. Shipman, "Eduos: An educational operating system for os courses," in *Proceedings of the ACM Special Interest Group on Computer Science Education*, 2015, pp. 141–145.

[12] W3C, "Web content accessibility guidelines (wcag) 2.1," Available online: https://www.w3.org/TR/WCAG21/, 2018.

[13] B. Miller and D. Ranum, "Teaching operating systems in the cloud," *IEEE Transactions on Education*, vol. 55, no. 3, pp. 1–10, 2012.