International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

" Cross-Platform System Monitoring Using Flask and Socket.IO "

[1]Shahan Wish Paul,[2] Abha Choubey

[1]Department of Computer Science and Engineering
[2]Shri Shankaracharya Technical Campus, Bhilai, Chhattisgarh, India
shahanjagdalpur125@gmail.com abha.is.shukla@gmail.com

ABSTRACT :

This research paper presents the architecture, development, and evaluation of a cross-platform real-time system monitoring application built using Flask, Flask-SocketIO, psutil, and Chart.js. Designed for both Linux and Windows environments, this system provides live performance insights through a web interface, making it ideal for desktop diagnostics and remote server monitoring.

The core of this system lies in its efficient use of WebSockets to transmit metrics data in real time, thereby avoiding the latency and load of traditional polling techniques. By integrating the psutil library, the application gathers low-level system statistics with negligible performance impact, while the frontend uses responsive charts and a clean interface to display insights. The system supports multiple deployment strategies including Docker, local execution scripts, and manual environment setup.

This paper elaborates on the project's detailed methodology, its layered architecture, and its thorough testing procedures. Furthermore, it highlights current limitations and proposes several future enhancements that will further increase functionality, performance, and security, thereby making it a versatile tool for developers, sysadmins, and IT professionals alike.

1. Introduction

As computing systems grow more complex and distributed, the need for reliable, real-time system monitoring tools has become paramount. Traditional desktop-based tools, though effective in isolated environments, fall short when it comes to remote accessibility, cross-platform compatibility, and extensibility.
This project addresses these challenges by developing a browser-based monitoring tool using modern web technologies. It provides near-instantaneous feedback on system performance metrics including CPU usage, memory allocation, disk activity, network throughput, and process statistics. The primary motivation is to deliver a light-weight yet robust system that can serve both personal desktops and remote enterprise environments.

Key motivations for this tool include:

- The need for transparent resource usage on shared or public systems.
- The ability to monitor servers without GUI access
- The requirement for customizable, open-source alternatives to tools like Nagios or SolarWinds
- Providing educational value in understanding OS behavior and application performance.

2. Literature Review

2.1 Evolution of System Monitoring Tools:

Monitoring tools have evolved from basic terminal utilities to fully fledged dashboard platforms. Legacy tools such as `top`, `htop`, and `iotop` were effective for real-time data but offered no network access or data visualization. With increasing demand for remote management, tools like Zabbix, Grafana, and Prometheus gained prominence. However, these often come with steep learning curves and server-side setup complexity.

2.2 Flask for Backend Architecture:

Flask is well-documented and extensible, allowing developers to create REST APIs, serve dynamic content, and integrate WebSocket functionality via Flask-SocketIO. Research has shown Flask to be suitable for rapid prototyping and deployment of small to medium scale applications, particularly in real-time dashboard development.

2.3 Benefits of Socket.IO for Real-Time Updates:

Unlike HTTP polling, which consumes resources by repeatedly querying the server, WebSockets maintain an open connection. Socket.IO adds fallback support and room-based broadcasting which enables multiple clients to receive targeted updates. In this system, every metric update (once per second) is transmitted via sockets, ensuring seamless interactivity.

2.4 Using psutil for Metric Collection:

psutil abstracts system-level calls into high-level Python interfaces, enabling access to CPU stats, virtual memory, swap usage, disk read/write bytes, network I/O, and running process details. Its thread-safe implementation and cross-platform compatibility make it ideal for real-time applications.

2.5 Frontend Visualizations with Chart.js and Bootstrap:

Chart.js provides beautiful animated charts and supports customization through APIs. In this application, real-time line charts are used to plot CPU usage, pie charts for memory distribution, and tables for process views. Coupled with Bootstrap, the UI is mobile-responsive and clean.

3. Methodology

3.1 Step 1: Requirement Gathering

Requirements were gathered from system administrators, developers, and educators. Desired features included:

- Platform independence
- Live updates without page reload
- Minimal system overhead
- Ability to deploy without cloud dependence
- Responsive design for tablets and mobile

3.2 Step 2: Application Planning and Technology Selection

The team selected Flask due to its simplicity, Socket.IO for real-time updates, and psutil for metric access. Docker was chosen for containerized deployment. Chart.js and Bootstrap were chosen for frontend development due to their minimal setup and active support.

3.3 Step 3: Backend Implementation

- A Flask app with SocketIO namespaces was initialized.
- A scheduled task was created using Python's threading to emit psutil stats every second.
- Data is emitted in JSON format, categorized into CPU, memory, disk, network, and process stats.

3.4 Step 4: Frontend Development

- HTML layout divided into dashboard panels.
- JavaScript functions were created to connect via Socket.IO and parse received data.
- Charts update using Chart.js APIs, and data is dynamically inserted into DOM.

3.5 Step 5: Docker and Script-Based Deployment

- Dockerfile created with Python environment and dependencies.
- docker-compose.yml defined volume, port, and runtime settings
- Bash (.sh) and Batch (.bat) scripts provided for quick execution without Docker.

3.6 Step 6: Cross-Platform Testing

Application was tested on:

- Ubuntu 20.04, 22.04

- Windows 10, 11
- Edge, Firefox, Chrome browsers

Functional parity was confirmed across environments.

3.7 Step 7: Performance Profiling

- Flask-SocketIO events timed and logged
- psutil metrics sampled at various intervals
- UI latency and frame-rate profiled using browser developer tools

3.8 Step 8: Feedback and Iterative Refinement

A beta group of users evaluated the application. Suggestions like dark mode, font scaling, and detailed tooltips were implemented in version 2.

4. System Architecture

The architecture has three distinct layers:

- Data Layer:
- Uses psutil to fetch system metrics.
- Gathers CPU frequency, core utilization, memory buffers, cache, and swap stats.
- For network: packets sent/received, error rates, and bandwidth use.
- Top 10 resource-consuming processes collected with sorting options.

B. Communication Layer:

- Flask-SocketIO manages real-time connections.
- Uses rooms to support multi-client broadcasts.
- CORS enabled for cross-origin testing and deployment.
- JSON schema defined for each metric type.

C. Presentation Layer:

- Bootstrap grid system for layout.
- Chart.js used for all visualizations (line, pie, bar).
- User-configurable update intervals (1s, 3s, 5s)
- Themes supported using CSS variables
- Optimized for mobile via meta tags and CSS media queries.

5. Results and Evaluation

5.1 Quantitative Results:

CPU Overhead: 1.5% average on dual-core system

- Memory Usage: 80MB idle, 100MB under full load
- WebSocket latency: 100–250ms (local), 200–600ms (LAN)
- Page load time: 1.3s average on Firefox

5.2 Qualitative Feedback:

Users praised:

- Responsive and clean UI
- Ease of setup via scripts or Docker
- Compatibility across systems

5.3 Comparative Performance:

Compared with existing tools (like Netdata, Glances), this system showed:

- Lighter setup (no databases)
- Better UI responsiveness
- Easier codebase for modifications

5.4 Observed Limitations:

- No persistent logging (future DB integration planned)
- Manual process refresh required (planned automation)
- No alert or notification system yet implemented

6. Discussion

This system fills a unique niche between basic CLI tools and enterprise-level dashboards. It is ideal for developers, students, and system admins who require lightweight, real-time monitoring.

Advantages:

- No external agents required
- Modular components simplify debugging
- Runs identically across OSes

Challenges:

- Handling socket disconnections gracefully
- Avoiding race conditions in parallel data emission
- Designing mobile-compatible graphs for small screens

Scalability Potential:

- Redis-based pub/sub architecture for horizontal scaling
- Add user login and dashboards for multiple monitored systems
- Export data to CSV or remote APIs for further analytics

7. Conclusion and Future Work

This research project demonstrates the effectiveness of building a full-stack, real-time system monitor using open-source technologies. The combination of Flask, Socket.IO, psutil, and Chart.js creates a responsive, accessible, and efficient solution for cross-platform performance tracking.
In the future, the project can evolve into a more comprehensive tool with capabilities such as:

- Authentication and user session handling
- System log file parsing and visualization
- Integration with notification services like Slack or email alerts
- Cloud deployment for team-wide access

It has practical value not just for system administrators, but also for students learning operating system internals, developers optimizing application performance, and teams managing remote infrastructure.