Building Computing Clusters of Web Browsers

Yao Li (202420700)

Advisor: Yasushi Shinjo

1 Introduction

In laboratories and offices, we see many unused PCs and idle PCs. Mobile devices such as smartphones and tablets also often have computing power comparable to old PCs.

It is widely practiced to connect PCs over a LAN to form small HPC clusters. Conventional cluster management software such as Apache Mesos and OpenHPC requires a dedicated OS installation, making the setup process complex. However, setting up and maintaining such clusters can be challenging due to the need for specialized hardware, software, and network configurations. While dual-booting allows regular PCs to serve as cluster nodes, this prevents their use for normal tasks like web browsing.

Volunteer computing projects like SETI@home and Folding@home have shown the potential of utilizing idle computational resources. However, these systems typically require specific software installation and face cross-platform compatibility challenges. Participants must download and install client software, which can be a barrier to entry for many users.

We propose a web browser-based approach to building computing clusters with the following goals:

- Developing a scalable architecture for dynamic resource pooling without complex infrastructure, targeting both small HPC clusters and volunteer computing environments
- Ensuring zero-installation deployment through standard web browsers
- Enabling efficient utilization of idle computational resources across the network
- Providing secure execution through browser-based isolation

2 Proposed Method

The system architecture of our browser-based computing cluster is as shown in Figure 1. The system consists of three main components:

- **Management Web Server**: A central server that coordinates the cluster operations, including task distribution, resource monitoring, and node management. It is responsible for maintaining the overall health of the cluster, ensuring that tasks are efficiently distributed based on the current load and availability of resources. The server also handles fault tolerance by redistributing the tasks that were running on failed nodes to active ones.
- **Web Browser Nodes Running on PCs**: Standard computers connected to the network, each running a web browser. These nodes can be any device with a web browser, including desktops, laptops, and even mobile devices. Each node contributes its idle computational resources to the cluster, allowing for a highly flexible and scalable system. Nodes can join and leave the cluster dynamically, providing a robust and adaptable computing environment.
- **Browser Applications**: Each browser runs an application in an isolated runtime environment for executing tasks. This application is designed to be lightweight and secure, leveraging the sandboxing capabilities of modern web browsers to ensure that tasks are executed in isolation from the host system.

This system faces several challenges that need to be addressed:

- Heterogeneity of browser performance.
- Unstable network connections.
- Efficient and safe code execution in an isolated environment.

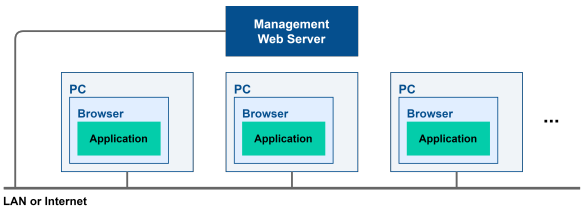

Figure 1: Cluster of Web Browsers.

- User-friendly interface for interactive use.

Our system architecture is built on the following key technologies to address to these challenges:

- **WebAssembly**: Provides high-performance task execution while maintaining platform compatibility. WebAssembly allows code to run at near-native speed in the browser, making it ideal for computationally intensive tasks. It also ensures that the same code can run on any device with a web browser, regardless of the underlying hardware or operating system. We run WebAssembly tasks in the WebAssembly Sandbox.
- **Web Real-Time Communication**[4]: Enables direct peer-to-peer communication between nodes for efficient data transfer. WebRTC supports low-latency, high-bandwidth communication among web browsers, which is essential for the performance of distributed computing tasks. It also includes built-in mechanisms for NAT traversal, allowing nodes behind different types of network configurations to communicate directly.
- **Using V86-Based Virtual Machine**[1]: Provides x86 emulation capabilities directly in the browser. This enables task execution in a manner similar to conventional cluster management software. The V86-based virtual machine allows us to run legacy applications and software that require a full x86 environment, providing a flexible and powerful execution platform within the browser.

We designe the system to be scalable, allowing for seamless joining and leaving of nodes to the cluster across different platforms and environments.

3 Implementation

Our implementation consists of three main components as shown in Figure 2: the management web server, the browser-based virtual machine, and the communication module.

[1] http://copy.sh/v86/

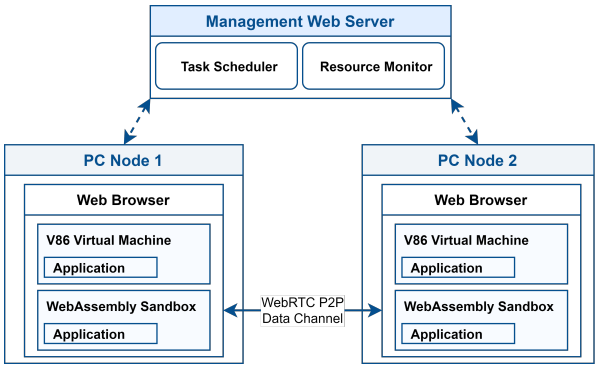

Figure 2: System Components.

3.1 Management Web Server

The management web server implements two core modules:

- **Task Scheduler**: Handles task distribution and load balancing across nodes. It fragments computational tasks based on node capabilities and manages task allocation. The scheduler ensures that tasks are distributed in a way that maximizes the utilization of available resources while minimizing the overall execution time. It also takes into account the heterogeneity of the nodes, assigning tasks based on their specific capabilities and current load.

- **Resource Monitor**: Maintains performance profiles and monitors node status continuously. It tracks CPU usage, memory availability, and network conditions to optimize resource utilization. The resource monitor collects detailed metrics from each node and uses this information to adjust task assignments dynamically. This helps in identifying and mitigating potential bottlenecks, ensuring that the cluster operates efficiently even under varying workloads.

The management web server is designed to be lightweight and efficient, capable of handling a large number of nodes with minimal overhead. It uses modern web technologies to ensure compatibility and performance, and it includes mechanisms for fault tolerance and recovery to maintain the stability of the cluster.

3.2 Network Communication

The system implements two communication channels with distinct responsibilities:

- **RPC (Remote Procedure Call)**: Handles control communication between the management server and nodes, including:
  - Task distribution and status updates
  - Resource monitoring and node management
  - Node registration and heartbeat signals
- **WebRTC P2P**: Enables direct peer-to-peer communication between nodes through:
  - Data channels for efficient resource sharing
  - Direct node-to-node task coordination

This dual-channel approach optimizes communication efficiency by:

- Using RPC for reliable server-controlled operations, ensuring that critical control messages are delivered accurately and in a timely manner.
- Leveraging WebRTC for direct peer communication to reduce server load, enabling efficient data transfer and task coordination between nodes without overloading the central server.
- Maintaining centralized control while enabling decentralized data exchange, providing a balance between control and performance.

3.3 Applications

In our browser-based clusters, we plan to run the following applications using self-scheduling.

- The server has a task bag.
- Each worker running in a browser retrieves a task from the task bag in the server.
- The worker executes the task and returns the result to the server.

This type of applications are common in volunteer computing. The advantage of this method is that tasks are automatically assigned according to the node's computing power. A faster node finishes a task soon and can start on the next task.

We are also considering interactive applications by taking advantage of the fact that they run run in a browser. For example, an ray tracing application can output results to the canvas of a web browser. Unlike traditional volunteer computing, such interactive applications allow the user to see the results displayed in the browser and change the parameters. For example, in the ray tracing application, the user can change the position of a camera.

3.4 Current Progress

Our initial implementation has established core functionalities including:

- Basic virtual machine operations through V86
- WebRTC peer discovery and connection establishment
- Task distribution through the management server
- Resource monitoring and basic load balancing

Ongoing work focuses on enhancing system stability, optimizing task scheduling efficiency for the management application, and implementing flexible resource management APIs. We are also exploring ways to further optimize the performance of WebAssembly execution and improve the scalability of the system. Future developments will include robust fault tolerance mechanisms, enhanced security features, and expanded support for diverse applications and use cases.

4 Related Work

BOINC established a framework for volunteer computing projects [1]. Despite its robust infrastructure, BOINC requires client software installation. Our browser-based solution removes this barrier by running directly in web browsers, making it more accessible to a wider audience.

Golem provides a decentralized computing power marketplace for renting computational resources [2]. While effective for dedicated computing tasks, our approach simplifies participation by eliminating the need for specialized software installation. Additionally, our system leverages the ubiquity of web browsers to create a more inclusive and flexible computing environment.

Apache Mesos [3] demonstrates effective resource management. Building upon these concepts, our system combines WebAssembly's efficiency with browser accessibility to create a lightweight, platform-independent computing environment. Unlike Mesos, which requires a dedicated

OS installation and complex setup, our solution operates entirely within the web browser, providing a more user-friendly and easily deployable alternative.

5 Conclusion

In this paper, we proposed building web browser-based computing clusters that provide zero-installation deployment through standard web browsers, cross-platform compatibility, and secure execution through WebAssembly sandboxing. This approach leverages the ubiquity and cross-platform nature of web browsers, allowing any device with a browser to participate in the cluster, significantly lowering the barrier to entry and simplifying the setup process compared to traditional methods.

Our initial implementation has established core functionalities including WebRTC peer discovery and basic virtual machine operations. We have demonstrated that it is possible to create a distributed computing environment that operates entirely within the web browser, eliminating the need for specialized software installations and making it accessible to a broader audience.

While challenges remain in areas such as browser performance heterogeneity and connection stability, this approach demonstrates the feasibility of leveraging web browsers for distributed computing applications. Future work will focus on optimizing performance, enhancing fault tolerance, and expanding the range of supported applications.

References

[1] David P Anderson. Boinc: A system for public-resource computing and storage. In *Fifth IEEE/ACM international workshop on grid computing*, pages 4–10, 2004.

[2] Golem Network. Official website. https://www.golem.network/. Accessed: 2024-12-18.

[3] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for Fine-Grained resource sharing in the data center. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, pages 1–14, March 2011.

[4] Cullen Jennings, Florent Castelli, Henrik Boström, and Jan-Ivar Bruaroey. Webrtc: Real-time communication in browsers. https://www.w3.org/TR/2024/REC-webrtc-20241008/, 2024. Accessed: 2024-12-18.