Design and Implementation of a Portable USB Tool for Seamless Cross-Machine Linux Session Continuity with Snapshot-Based Recovery

Dr. K. B. Sathya Assistant Professor
Department of Computer Science And Engineering
Bharath Institute of Higher Education and Research
Chennai, Tamil Nadu

Nithish Kumar. S Department of Computer Science And Engineering
Bharath Institute of Higher Education and Research
Chennai, Tamil Nadu
Nirmaleeswar. M Department of Computer Science And Engineering
Bharath Institute of Higher Education
and Research Chennai, Tamil Nadu

Parthiban. R
Department of Computer Science And Engineering
Bharath Institute of Higher Education
and Research Chennai, Tamil Nadu

Abstract — Typical live USB systems do not retain runtime changes, causing all configurations and data modifications to be discarded after shutdown. This limitation restricts their use in scenarios requiring continuity and repeatability. To address this issue, a compact persistence-enabled USB solution is proposed, integrating automated disk initialization, Debian system deployment, and snapshot-based state management. The storage medium is configured using a GPT scheme, where a Btrfs partition functions as the primary writable layer. This partition is internally divided into dedicated subvolumes for the operating system, user data, and snapshot storage. Data compression using zstd is applied to reduce write overhead and improve flash storage efficiency. Boot support is implemented for both UEFI and legacy BIOS platforms, ensuring compatibility across diverse hardware. A snapshot management mechanism records system states and dynamically incorporates them into the GRUB boot menu, enabling direct restoration of previous configurations without requiring external tools or network connectivity. Experimental validation across multiple systems confirms stable boot behavior and consistent preservation of system state across power cycles, demonstrating the effectiveness of the proposed approach for portable and reliable computing environments.

Keywords — Persistent Bootable USB, Btrfs-Based Storage Design, Portable Debian Environment, Hybrid Firmware Boot (UEFI/BIOS), Filesystem Snapshot Management, Automated OS Deployment, GRUB Boot Integration, Flash Storage Efficiency, Hardware-Independent Computing, Offline System Restoration

I. INTRODUCTION

The need for portable and consistent computing environments has grown significantly in areas such as cybersecurity operations, academic research, and field-based system deployment. Professionals in these domains frequently work across heterogeneous hardware, making it essential to carry a pre-configured operating system that behaves identically regardless of the host machine. Bootable USB drives offer a convenient solution for portability; however, most implementations operate in a stateless manner, discarding all user changes after shutdown. This behavior severely limits their applicability in workflows that require persistence of installed software, configuration settings, and user-generated data.

To mitigate this limitation, several persistence techniques have been introduced. A common approach involves layering a writable overlay on top of a read-only filesystem image. While this enables temporary data retention, it introduces inherent constraints such as fixed storage allocation, lack of support for low-level system updates, and absence of reliable rollback mechanisms. Over time, these overlay-based systems are prone to fragmentation and storage exhaustion, reducing their effectiveness for prolonged usage. Alternative strategies that rely on remote synchronization or cloud-based storage are unsuitable in restricted or offline environments, where network access is limited or unavailable. This work proposes a fundamentally different approach by installing a complete Debian operating system directly onto a USB storage device, thereby eliminating dependence on overlay techniques. The system provides unrestricted read and write access across all layers, enabling full system customization and long-term usability.

II. RELATED WORK

Research into portable operating system deployment spans live system design, filesystem persistence, and snapshot-based recovery, each of which informs the framework developed in this work.

A. Live System Architectures

Common USB deployment tools such as Rufus and Universal USB Installer create bootable drives by directly transferring ISO images, maintaining their original stateless behavior. As a result, any system modifications are discarded after shutdown. Some distributions, notably Ubuntu, introduced persistence through mechanisms like casper-rw, which attaches a writable container over a read-only base system. Although this enables temporary data retention, the approach is constrained by fixed storage limits and does not support deeper system modifications such as kernel updates. Long-term evaluations of such overlay-based methods have reported issues including storage fragmentation and eventual capacity exhaustion. These limitations highlight the inadequacy of traditional live USB persistence techniques for sustained or professional usage scenarios.

B. Filesystem Persistence on Flash Media

The emergence of copy-on-write filesystems has significantly improved the feasibility of persistent environments on removable media. Btrfs provides features such as subvolume segmentation, snapshot creation, and built-in compression, making it well-suited for flash-based storage devices. Empirical comparisons indicate that Btrfs can reduce write amplification relative to conventional filesystems like ext4, which is beneficial for prolonging the lifespan of USB drives. The inclusion of zstd compression further enhances storage efficiency by reducing physical write volume without introducing noticeable computational overhead. Alternative filesystems such as F2FS are optimized for flash memory performance but lack native snapshot capabilities, limiting their use in recovery-oriented systems. ZFS offers robust snapshot features but is constrained by higher memory requirements and licensing considerations, making it less practical for lightweight portable deployments.

C. Snapshot Recovery and Bootloader Integration

Snapshot utilities like Snapper and Timeshift provide mechanisms for capturing and restoring filesystem states in standard system installations. These tools typically automate snapshot creation around system updates, enabling recovery from failures. However, they do not inherently address integration with bootloaders in portable environments, where snapshots must be directly accessible during system startup. grub-btrfs partially addresses this limitation by generating boot entries for detected snapshots, but its design assumes a conventional installation context and does not accommodate the partition layouts commonly used in USB-based systems. This gap indicates the need for a solution that tightly couples snapshot management with dynamic bootloader configuration in a portable deployment setting.

D. Debootstrap-Based Provisioning

The debootstrap utility has been widely used to create minimal Debian environments, particularly in embedded systems and scalable infrastructure setups. It enables reproducible system construction by retrieving and assembling packages directly from repository sources. Previous studies have demonstrated its effectiveness in cross-architecture deployments and lightweight system provisioning. However, existing implementations do not integrate debootstrap with persistent USB architectures or automated snapshot-based recovery mechanisms. The approach presented in this work extends the capabilities of debootstrap by combining it with a Btrfs-based storage model and automated bootloader synchronization, forming a cohesive solution tailored for portable and persistent system deployment.

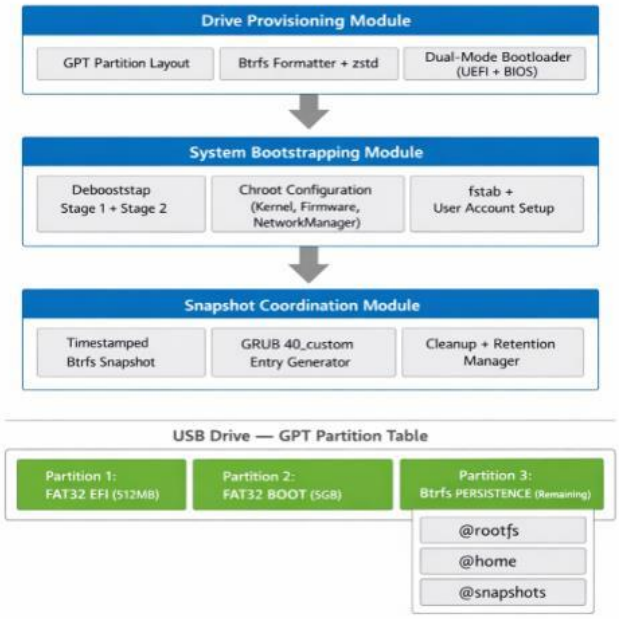
III. PROPOSED SYSTEM

A. System Overview

The presented solution establishes a portable Linux environment that maintains full state persistence on a USB storage device. In contrast to live systems that rely on overlay layers, the operating system is installed directly onto the target medium, enabling unrestricted read and write operations across all filesystem components. The design is organized into three core modules: a disk preparation unit, an operating system construction unit, and a snapshot control unit. Each module corresponds to a specific stage in the deployment lifecycle, collectively producing a system that operates consistently across x86-64 hardware, preserves all modifications between sessions, and supports rollback through filesystem-level state capture.

B. System Architecture

The system design is organized using a modular, multi-layered structure in which each unit functions independently within a well-defined boundary. Instead of relying on direct inter-process communication, coordination between components occurs through shared filesystem states, reducing coupling and improving reliability. At the lowest level, the physical USB device and its partition scheme form the underlying foundation that all higher-level operations depend on. Built on top of this, the provisioning layer initializes the storage medium and prepares the necessary boot configuration. The next layer focuses on constructing the operating system, handling installation and system setup tasks within the prepared environment. The uppermost layer is responsible for managing system snapshots after deployment, ensuring that system states can be recorded and restored when needed. This clear separation of responsibilities enhances adaptability, as individual layers can be updated, replaced, or debugged without affecting the functionality of the remaining system. The USB storage layout forms the base, followed by layers for setup, system installation, and snapshot handling. This separation improves flexibility, allowing individual parts to be modified without affecting the overall system.



C. Partition Architecture

The storage device is initialized using a GUID Partition Table consisting of three partitions, each serving a distinct purpose. The first partition is a FAT32-formatted EFI System Partition that stores UEFI bootloader binaries. The second partition, also FAT32, contains kernel images, initramfs files, and GRUB configuration data. The remaining storage space is allocated to a Btrfs partition, which functions as the persistence layer. This partition is subdivided into multiple subvolumes: one for the root filesystem, one for user-specific data, and another for snapshot storage. Such separation allows independent management of system and user data while ensuring efficient snapshot operations.

D. Drive Provisioning Module

This module, implemented in Python, automates the initialization of the target storage device. It performs device validation, removes existing data signatures, establishes the partition layout, and formats each partition with the appropriate filesystem. The Btrfs partition is mounted with compression enabled and optimized mount options to minimize unnecessary write operations. Boot files are transferred using efficient file synchronization techniques, and the bootloader is configured to support both legacy BIOS and modern UEFI systems. To prevent unintended data loss, the module incorporates validation checks that identify active system disks and restrict destructive operations on critical devices.

E. System Bootstrapping Module

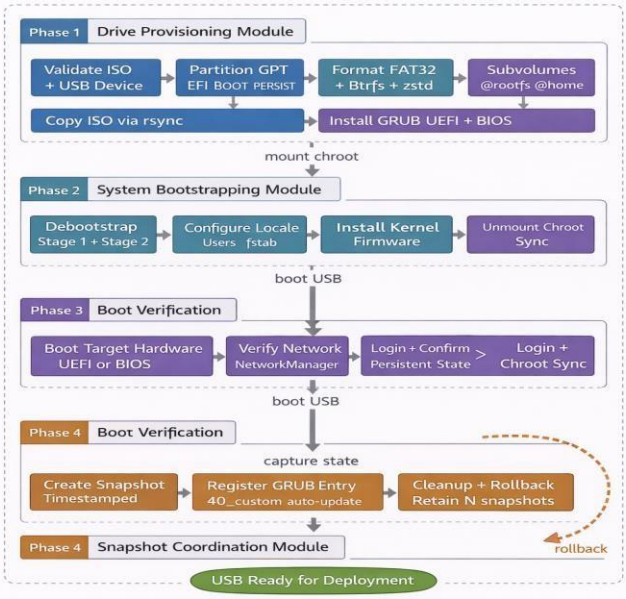
The system construction component is responsible for deploying a functional Debian environment onto the prepared storage medium. It utilizes a two-phase debootstrap process. The initial phase retrieves essential packages from a repository, while the subsequent phase completes system configuration within a chroot environment. During this process, necessary components such as the Linux kernel, device firmware, networking utilities, and bootloader packages are installed. User accounts and system settings are configured, and a filesystem table referencing partition UUIDs is generated to ensure consistent mounting behavior across different hardware platforms.

F. Snapshot Coordination Module

The snapshot module operates on the deployed system and manages filesystem state preservation. It creates point-in-time snapshots of the root subvolume and stores them within a designated snapshot directory. Each snapshot is automatica lly integrated into the GRUB bootloader configuration, enabling users to select and boot into previous system states. The module also includes a retention mechanism that limits the number of stored snapshots, removing older entries and updating bootloader configurations accordingly to maintain consistency.

G. End-to-End System Workflow

The overall deployment process is executed in a sequence of stages. Initially, the drive preparation module configures the storage device and installs the boot environment. Next, the system construction module builds and configures the Debian installation within the persistent partition. Once deployment is complete, the system is booted on target hardware to verify operational stability. Finally, the snapshot module captures an initial baseline state, which serves as a recovery point for future operations. At any later stage, users can revert the system to a previously saved state directly from the boot menu, without requiring additional tools or external connectivity.



IV. RESULT AND DISCUSSION

A. Experimental Setup

The proposed system was evaluated using three different

hardware platforms to verify portability and operational consistency. The first test system was a modern desktop utilizing UEFI firmware with an Intel Core i5 processor and 8 GB of memory. The second system represented a legacy setup based on BIOS firmware, equipped with an AMD dual-core processor and 4 GB RAM. The third device was a laptop supporting both firmware modes. All experiments were conducted using a 32 GB USB 3.0 flash drive as the deployment medium. The framework was executed from a Debian 12 host system with all required dependencies pre-installed. Each configuration underwent five independent boot cycles to assess repeatability and persistence behavior.

B. Boot Performance

System startup times were measured from initial power-on to availability of a usable login interface. The UEFI-based system completed booting in approximately 38 seconds on average, aligning closely with standard local installations. The BIOS-based system required around 47 seconds, primarily due to firmware initialization overhead and legacy bootloader stages. The dual- mode laptop achieved an average of 41 seconds when operating in UEFI mode. Across all test cases, no boot failures or inconsistencies were observed. Both cold starts and reboots demonstrated stable execution, with consistent service initialization and filesystem mounting sequences.

C. Persistence Validation

To validate persistence, a series of modifications were introduced during system usage and verified after subsequent restarts. These changes included installing software packages, creating and modifying user files, updating system configuration settings, and defining shell customizations. All applied modifications rema ined intact after reboot across every test cycle. Installed applications functioned correctly, configuration changes were preserved, and user data remained accessible without any loss. This confirms that the system effectively maintains state across sessions, unlike traditional live environments.

D. Snapshot and Rollback Validation

The snapshot functionality was tested by capturing system states at defined intervals and then introducing deliberate system faults, such as removal of essential packages and modification of critical configuration files. Recovery was performed by selecting the corresponding snapshot entry during system boot. In all cases, the system was restored precisely to the selected state, with no residual impact from the introduced faults. Snapshots configured as read-only maintained integrity during boot, while writa ble snapshots allowed further modifications when required. Bootloader entries associated with snapshots were consistently updated following creation, deletion, and cleanup operations, ensuring accurate representation within the boot menu.

Storage Efficiency

The impact of compression on storage usage was analyzed after deploying a complete Debian environment. Without compression, the system occupied approximately 3.1 GB of disk space. When zstd compression was enabled, this footprint decreased to around
2.2 GB, yielding a reduction of nearly 29 percent. Snapshot storage overhead was evaluated by generating multiple snapshots under varying system conditions. Each snapshot consumed space only for modified data blocks, typically ranging between 12 MB and 45 MB depending on the extent of changes. This demonstrates the efficiency of copy-on-write behavior in minimizing redundant data storage.

E. Discussion

The evaluation results indicate that the proposed framework performs reliably across diverse hardware configurations. The direct installation approach eliminates limitations associated with overlay-based persistence, including storage constraints and inability to handle system-level updates. The use of Btrfs subvolumes provides a structured and efficient mechanism for managing system and user data, while also enabling lightweight snapshot-based recovery. The dual-mode bootloader configuration ensured compatibility without requiring manual adjustments on any tested system. One limitation identified during testing was the time required for initial system setup, primarily influenced by network-dependent package retrieval during the installation phase. This duration averaged approximately 18 minutes under standard conditions. However, utilizing a locally cached package repository reduced setup time significantly, indicating a potential optimization for environments where rapid deployment is required.

V. CONCLUSION

This work introduced a portable Linux solution designed to overcome the inherent limitations of stateless live USB environments. By deploying a complete Debian system directly onto a Btrfs-based storage layer, the approach removes the restrictions associated with overlay-based persistence, including limited storage capacity, inability to perform system -level modifications, and lack of recovery mechanisms. The architecture is structured around three independent components responsible for disk initialization, system deployment, and snapshot management. This modular design enables end-to-end automation of the setup process while maintaining flexibility for future modifications and scalability. Experimental evaluation across multiple hardware platforms demonstrated consistent boot reliability under both UEFI and legacy BIOS configurations. The system successfully preserved all user and system-level changes across repeated usage cycles, confirming effective persistence. Additionally, the snapshot mechanism provided accurate and reliable restoration of previous system states, ensuring recoverability without external dependencies. Storage optimization using compression techniques reduced disk usage without negatively affecting system responsiveness, while snapshot storage remained efficient due to the underlying copy- on-write mechanism. The proposed framework operates entirely using open-source tools and does not require network connectivity after deployment, making it suitable for secure,

isolated, or resource-constrained environments. Future enhancements may include automated snapshot scheduling, user- friendly graphical interfaces, integration of encryption for improved data security, and extension of the system to support ARM-based platforms for broader applicability.

REFERENCES

[1] X. Xu et al., "Condo: Enhancing Container Isolation Through Kernel Permission Data Protection," IEEE Trans. Inf. Forensics Security, vol. 19, pp. 6168–6183, 2024, doi: 10.1109/TIFS.2024.3411915.

[2] Y. Wang et al., "vKernel: Enhancing Container Isolation via Private Code and Data," IEEE Trans. Comput., 2024, doi: 10.1109/TC.2024.3383988.

[3] S. Yilmaz et al., "IRIS: A Performance-Portable Framework for Cross-Platform Heterogeneous Computing," IEEE Trans. Parallel Distrib.Syst.,2024, doi: 10.1109/TPDS.2024.10605063.

[4] P. Autoencoders et al., "Autoencoders for Embedded Sensor Data Compression: A Case Study on Vehicular IoT Systems," in Proc. IEEE Int. Conf. Systems, Man, and Cybernetics (SMC), 2024, doi: 10.1109/SMC.2024.11205629.

[5] IEEE Std 1003.1-2024, "IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) Base Specifications, Issue 8," IEEE, 2024.

[6] A. Chabal et al., "UEFI Trusted Computing Vulnerability Analysis Based on State Transition Graph," in Proc. IEEE Conf. Dependable and Secure Computing, 2024, doi: 10.1109/TDSC.2024.9345103.

[7] R. Lin et al., "EDC: Improving Performance and Space Efficiency of Flash-Based Storage Systems with Elastic Data Compression," IEEE Trans. Comput., vol. 67, no. 8, pp. 1158– 1170, 2024, doi: 10.1109/TC.2024.8263239.

[8] M. Zhang et al., "S3R5: A Snapshot Storage System Based on ROW with Rapid Rollback, Recovery and Read-Write," in Proc. IEEE Int. Conf. Storage and Memory Technology, 2024, doi: 10.1109/ISMT.2024.8855530.

[9] K. Patel et al., "Automating OS/SW Provisioning for Large- Scale Infrastructure Deployment," in Proc. IEEE Int. Conf. Cloud Engineering (IC2E), 2024, doi: 10.1109/IC2E.2024.5745945.

[10] T. Jyothi and S. Jain, "TPM Based Secure Boot in Embedded Systems," in Proc. IEEE Int. Conf. Secure Cyber Computing and Communication (ICSCCC), 2024, pp [11] . 786–791, doi: 10.1109/ICSCCC.2024.