UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Ashish Kumar Sultania

Monitoring and Failure Recovery of Cloud-Managed Digital Signage

Master's Thesis (30 ECTS)

Supervisors: Assoc. Prof. Satish N. Srirama, University of Tartu, Estonia
Prof. Danilo Gligoroski, NTNU, Norway
Advisor: Prof. Tuomas Aura, Aalto University, Finland

Tartu 2017

Monitoring and Failure Recovery of Cloud-Managed Digital Signage

Abstract: DIGITAL SIGNAGE is widely used in various fields such as transport systems, trading outlets, entertainment, and others, to display information in the form of images, videos, and text. The reliability of these resources, availability of required services and security measures play a key role in the adoption of such systems. Efficient management of the digital signage system is a challenging task to the service providers. There could be many reasons that lead to the malfunctioning of this system such as faulty displays, network, hardware or software failures that are quite repetitive. The traditional process of recovering from such failures often involves tedious and cumbersome diagnosis. In many cases, technicians need to physically visit the site, thereby increasing the maintenance costs and the recovery time.

In this thesis, we propose a solution that monitors, diagnoses and recovers from known failures by connecting the displays to a cloud. A cloud-based remote and autonomous server configures the content of remote displays and updates them dynamically. Each display tracks the running process and sends the trace and system logs to the server periodically. These logs, stored at the server optimally using a customized log management module, are analysed for failures. In addition, the displays incorporate self-recovery procedures to deal with failures, when they are unable to create connection to the cloud. The proposed solution is implemented on a Linux system and evaluated by deploying the server on the Amazon Web Service (AWS) cloud. The main result of the thesis is a collection of techniques for resolving the display system failures remotely.

Keywords: Digital Signage, Monitor System, Recovery System, Self-healing, Self-Diagnose, Digital Displays, Autonomous Recovery

CERCS: P170 Computer science, numerical analysis, systems, control

Cloudi-hallatavate Digitaal Signage Seire ja ebaõnnestumine

Abstract: DIGITAAL SIGNAGE kasutatakse laialdaselt erinevates valdkondades, nagu näiteks transpordisüsteemid, turustusvõimalused, meelelahutus ja teised, et kuvada teavet piltide, videote ja teksti kujul. Nende ressursside usaldusväärsus, vajalike teenuste kättesaadavus ja turvameetmed on selliste süsteemide vastuvõtmisel võtmeroll. Digitaalse märgistussüsteemi tõhus haldamine on teenusepakkujatele keeruline ülesanne. Selle süsteemi rikkeid võib põhjustada mitmeid põhjuseid, nagu näiteks vigased kuvarid, võrgu-, riist- või tarkvaraprobleemid, mis on üsna korduvad. Traditsiooniline protsess sellistest ebaõnnestumistest taastumisel hõlmab sageli tüütuid ja tülikaid diagnoose. Paljudel juhtudel peavad tehnikud kohale füüsiliselt külastama, suurendades seeläbi hoolduskulusid ja taastumisaega.

Selles väites pakume lahendust, mis jälgib, diagnoosib ja taandub tuntud tõrgetest, ühendades kuvarid pilvega. Pilvepõhine kaug- ja autonoomne server konfigureerib kaugseadete sisu ja uuendab neid dünaamiliselt. Iga kuva jälgib jooksvat protsessi ja saadab trace'i, logib süstemisse perioodiliselt. Negatiivide puhul analüüsitakse neid serverisse salvestatud logisid, mis optimaalselt kasutavad kohandatud logijuhtimismoodulit. Lisaks näitavad ekraanid ebaõnnestumistega toimetulemiseks enesetäitmise protseduure, kui nad ei suuda pilvega ühendust luua. Kavandatud lahendus viiakse läbi Linuxi süsteemis ja seda hinnatakse serveri kasutuselevõtuga Amazon Web Service (AWS) pilves. Peamisteks tulemusteks on meetodite kogum, mis võimaldavad kaugjuhtimisega kuvariprobleemide lahendamist.

Keywords: Digitaal Signage, Monitori süsteem, Taastumine süsteem, Enesehooldus, Enesediagnostika, Digitaalsed kuvarid, Autonoomne Taastamine

CERCS:P170 Arvutiteadus, arvanalüüs, süsteemid, kontroll

Terms and Notions

Terms	Definition
Kernel Mode	In this mode, executed code has complete and unrestricted access to hardware. The crashes in kernel mode are catastrophic.
User Mode	In this mode, executed code has no direct access to hardware or memory. The crashes in user mode are always recoverable.
Firmware	The set of instructions programmed on a hardware device to communicate with other system hardware.
Runlevel	The operating state of an operating system. Each runlevel designates a different system configuration.
Symbolic link	Special file which points to another file and does not contain any data of the target file.
Unit file	It contains configuration directives that describe the unit and its behavior.
Hash	One-way cryptographic function that accepts a message of any length as input and returns a fixed-length digest value. This function is considered to be infeasible to invert.
MD5	Message-Digest Algorithm used to verify that a file has been unaltered.
tmpfs	RAM based temporary file system and generally mounted on /dev/shm.

List of Figures

List of Tables

Contents

1 Introduction

Cloud computing has emerged as the new computing platform to deliver various services over the Internet for both technical and economical reasons. These services are broadly classified into four categories: "Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), Software-as-a-Service (SaaS), and Data-as-a-Service (DaaS)" [1]. A cloud-based digital signage is a multimedia content player installed in public venues, connected to the Internet for streaming contents from a cloud server. The administrator can manage the content remotely. Some distinguishing characteristics of cloud computing such as elasticity, availability, broad network access, and scalability make it suitable for digital signage system.

In the recent years, digital signage solutions have significantly grown in size. They deliver information to many people and increasingly provide a means for interaction. For example, they are often found at airports or train stations to display arrival and departure schedules. Their use has been investigated in different environments such as urban public areas [2, 3], workplaces, third places [4] such as bars, cafes [5, 6], and rural villages [7, 8]. According to Grand View Research, the market size is expected to reach $20 billion by 2020 [9]. There are numerous factors promoting its deployment such as reduced cost, better visual coverage distance, widely available hardware, and dynamic content.

Different sizes of digital signage displays are available. LCD displays of size typically 40 inches are installed in many locations [10]. The small size displays are mostly used for user interaction via touchscreen, while medium and large sizes serve as an information delivering medium to viewers. There is a technical limitation on the LCD screen size and so to overcome this, multiple display screens may be installed, which is called “multi-screen digital signage”. In multi-screen digital signage, the original content is split into multiple partial views to the connected screens.

The display systems are used for various applications including advertising, information presentation, and entertainment. Generally, the main application is advertising, where the advertisements are combined with the content that has direct value to viewers. To provide the information digitally out of the display device, there exist multiple technologies such as QR codes and RFID tags. Displays can also show short art films; for example, the e-Campus display network at Lancaster University was used to show a number of videos commissioned by the campus's Nuffield Theatre [11]. In the UK, many cities have installed *BBC Big Screens*, a large public display with a sound system, for entertainment purposes [12].

The desirable characteristics of a digital signage system are the following [11]:

1. **Functional Requirements**: The system needs to provide functions such as

creating content, distributing content, scheduling content, running in real-time, controlling hardware, detecting hardware failure, remote management and user interaction.

2. **Stakeholders Requirements**: There are three types of stakeholders in the digital signage system: *display provider*, who has the display devices and space for them; *content producer*, who creates the content and provides its management; and *content viewer*, who views the content from the display devices. Each of them have different requirements which should be fulfilled by the system. It is important to engage the stakeholders in the designing of digital signage systems for long-term deployment [13].

3. **Reliability**: Reliability is the probability that a system performs the intended task for a specified time. The system can also have various issues or faults which make it unreliable. Regular failures can make the system unpopular.

4. **Remote Management**: The remote monitoring and management of the displays is needed to ensure reliability of the system.

5. **Cost Efficiency**: The most important factor to consider a technology is cost. The cost is generally compared with the profit after installing the system. The hardware cost generally decreases with growing acceptance of digital signage.

The focus of this thesis is on reliability and remote management. Our solution also indirectly improves the cost efficiency and helps the systems to meet stakeholder requirements.

The digital signage system can be viewed as different modules with their roles as shown in Figure 1. These modules are *content creation*, *scheduling and management*, *screen operations* and *user interaction*. Generally, the tools used to create the content are generic and not specific to any signage system. These tools provide simple interfaces for creating different types of content such as text, images, videos, audio, and animations. Our main research area in this system is the management functions of the scheduling and management module. Most of the modern commercial signage systems use cloud services for managing the content. The content can be uploaded by third parties. The scheduling component serves as back-end for the signage system and takes the responsibility of distributing the contents to appropriate displays. The content scheduling is challenging with the requirement of content rescheduling based on user interaction inputs, specific times, and limitations on displaying the same content. The display management component

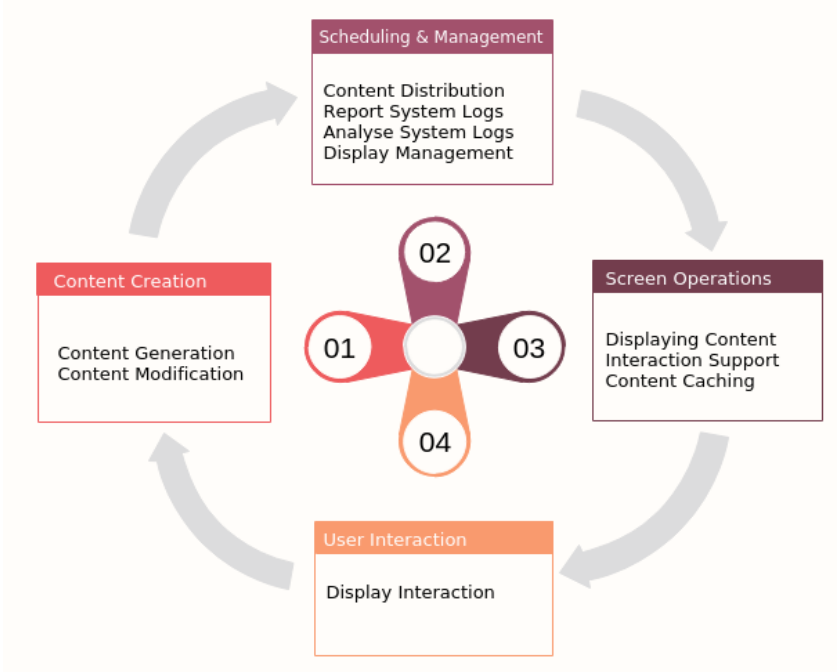

Figure 1: Digital signage system modules

handles various administrative functions such as remote monitoring of displays, controlling the power states of the devices and handling software updates [14]. From an architectural perspective, it is beneficial to separate the management of digital signage from content scheduling and distribution. As the system complexity increases, it becomes difficult to maintain and manage, therefore *autonomic computing* is required. Autonomic computing means adapting to any unpredictable changes by the system and hiding the intrinsic complexity from users. In this work, we present the implementation of remote monitoring and recovery of displays in a digital signage system.

1.1 Motivation

Software or hardware defects can cause errors which can impact the system operation and may even cause critical failures. It is necessary to recover from the failures promptly and to use the system logs for further analysis of the faults to prevent them cascading. This thesis is motivated by the management component as mentioned in Figure 1. This component includes functionality such as remote monitoring, controlling power and system log management. In case an error occurs at user end, it becomes a time consuming and costly task to solve it due to

the unavailability of any local technical experts. If the displays rely regularly on manual maintenance, it would have a negative impact on the business. Despite any failures, the stakeholders want the system to continue to operate without interruption. In 2004, an analysis [15] estimates that during the recovery process of application failures, roughly 75% of the time is spent in detecting and localizing the failure. The system should have self-managing features to configure, optimize, diagnose and recover. Furthermore, most of these issues should be solved from the cloud server by monitoring each display. Traditionally, most of the available system monitoring tools offer a post-mortem analysis of root-cause failure [16]. Therefore, a monitoring system can be introduced to collect information from the displays for detecting failures at runtime, predicting their causes, and rectifying them. The information used can be in the form of an event log (e.g. syslog [17]), or numeric data periodically read from sensors (e.g. /var/proc [18]). It is also important for a monitoring solution to be adaptive and flexible to allow changes in its configuration at runtime.

There are two main aspects of recovery for autonomous monitoring systems: physical layer of the system and logical layer [19]. Monitoring the physical layer is usually easy, but recovers manually. The logical level error recovery actions need a software recovery functionality to be injected into the system.

This thesis explores the possible solutions to build a robust and reliable digital signage system. An automated digital signage system diagnosis approach in a cloud environment is presented to ensure greater reliability and faster recovery from problems and, when possible, without any human intervention. The remote display's logs are transferred periodically to the cloud server, which analyses these logs to detect failures and sends appropriate instructions to the display for resolving any failures. This solution reduces not only the mean time to repair but also the cost of maintaining an operational system. The administrator or system developer can understand the system behavior and performance by accessing the old received logs of the system. Our application can be easily installed on any already developed Linux-based system to enhance the system reliability. Moreover, new features and support can be designed to enhance the application. It can help service providers to handle increasing IT management costs and also in negotiating the price with vendors.

This thesis mainly addresses the following questions:

- What type of display systems should be considered?
- What type of failures should be considered?
- How to detect the failures?
- What data is relevant to detect the failures?
- How to collect data and with what frequency?

- Which recovery techniques should be applied?
- How timely is the detection and recovery?
- How to notify administrator for non-recoverable failures?
- How efficient are the proposed approaches?

1.2 Structure

This thesis is structured as follows:

Chapter 2 presents related research works, including research on digital signage system and diagnosis system.

Chapter 3 provides a brief overview to understand the display system and its related components.

Chapter 4 introduces system analysis of different display failure modes and their possible solutions.

Chapter 5 describes the practical procedures for implementing the monitoring solution.

Chapter 6 provides details of the proposed solution and its evaluation.

Chapter 7 focuses on observations made during the design and deployment process, along with future work.

2 Related Work

2.1 Research on Digital Signage

Clinch et al. [20] summarize the challenges of deploying networked public displays. Similarly, Ojala et al. [21] describe the differences of development and evaluation in the real world scenario and experimental lab scenario. One of the recommendations is to provide web-based system to offer control to stakeholders, but providing too much flexibility may hamper the usability. Also in [22], Storz et al. mentioned thirteen challenges related to system development and maintenance from their experiences. They also mentioned the requirement of tools using which the stakeholders can monitor the internal state of the system. In 2005, researchers at Lancaster University designed and implemented an experimental digital signage system named *e-Campus system*. This e-Campus system offers APIs that enable developers for developing new applications to control the signage network [23].

The design of each display system is divided into five key management modules as subscription, sensor, playlist generation and scheduling, content rendering and lifecycle, and analytics. These modules are used to schedule, monitor, cache, and manage external inputs. Other research groups in Oulu, Finland deployed interactive displays within the city centre, which are modular web browser based architecture [24]. In 2013, Mikusz et al. [25] developed an application store for public displays for application developers and display owners. This application store provides various features including management tools. Sethi et al. [26] propose a user-assisted protocol for secure management of cloud connected digital wireless displays. Tatiraju [14] presents an efficient web-based solution for remote management of digital signage. He uses HTML5 and WebSocket to provide low-latency bidirectional communication. He focuses mainly on comparing different methods for displays to interact with cloud-based on different conditions.

Most of the current research on digital signage focuses on how and what data should be presented. In [27], a healthcare digital signage system is developed using gamification method which automatically detects the user and displays the feedback of health information. As the signage players or display systems require separate screen, audio, and input control, a process based virtualization scheme is proposed in [28]. Web browser process and streaming process are allocated for each signage player on a signage server. The server renders HTML images stored in a memory buffer and streams to the player. A new digital signage architecture based on *openstack* is proposed by Nguyen et al. [29] which leverages network function virtualization (NVF) technology to virtualize digital signage service operator. In 2016, similar to the architecture we have proposed for managing digital signage display devices, Mishima et al. demonstrate the prototype work in [30].

As display technology collects audience data to show effective advertisements, privacy becomes an open issue. Privacy should be supported by incorporating it into digital signage business models and data management practices. The digital signage service providers should develop and publish privacy policies and also notify audiences about what kind of data they are collecting. In 2011, the Digital Signage Federation adopted the privacy standards [31] for its members.

2.2 Monitoring and Fault Diagnosis System

There are different commercial monitoring tools available in the market such as Nagios [32], Ganglia [33], and OVIS [34] to track system health and status during operations. Traditionally, these monitoring tools collect sets of data from the system such as memory usage and response time. The effectiveness of these solutions is limited as these tools detect failures but can't predict the cause. Many companies focus on diagnosing their products remotely. Oracle provides Remote Diagnostics Agent (RDA) [35] to collect configuration and diagnostic data related to the operating system and targeted oracle products. Heikkinen et al. [36] provided a description on the remote monitoring and management tools developed for the maintenance of UBI-hotspots. They use the monitoring tool "Nagios" to detect component and system level issues, software "Happy page" to monitor the user interface issues that fetches the periodic screenshots from the hotspots, and Remote Desktop Connection (RDC) and Virtual Network Computing (VNC) to diagnose the faults. A web-based system for monitoring and assessing construction safety and health performance is developed in [37] to reduce occupational accidents. In this system, the data is observed directly and instantaneously in order to take fast educated preventive and corrective measures. The monitoring and fault diagnosis system based on the embedded Linux operating system is introduced in [38] to insure the reliability of the traction power supply system. There are many reasons to build the diagnosis system such as [35]:

- Minimized downtime
- Known failures resolution
- Collection of complete system technical information

Many digital signage solution providers claim that their solutions provide system diagnosis service. But, generally the main problem is that this service is supported only by customized media players. Also, most of these systems collect system logs to examine it manually and does not have a remote self-healing feature. Self-diagnosis and self-healing solutions for an embedded system are implemented by L. Sun et al. [39], which are based on several selected critical kernel data structures. When any system inconsistency caused by security attacks has been detected, some

predefined recovery functions are invoked. But their solutions require few modifications to the standard Linux kernel source code. Another technique known as Rollback-recovery [40] is widely used for many systems where the system copy at many checkpoints are saved to perform recovery from any fatal errors. When a failure occurs, the saved system copy can be used by failed processes to restart the computation from an intermediate state. Rollback-recovery technique is used to enhance the availability of the system, and is not aimed to resolve the errors. Some specific application related issues can be checked by developing a hardware framework which can reside on the same die as the processor. Iyer et al. [41] developed a reliability and security engine (RSE) as an integral part of the processor to provide reliability and security to the applications execute in parallel with the main processor's pipeline.

2.3 System Reliability

Computer system reliability remains as a crucial but unsolved problem [42]. IEEE Standard 1413.1 develops a framework of reliability prediction method for electronic equipment based on stress and damage models. This framework determines when a specific failure mechanism could occur in a given environment [43]. Operating system reliability improvement requires the systems to become tolerant to the failures in drivers and other kernel extensions. Operating systems, for example, Tandem Guardian and Unicos, have optimized for targeted environment with high reliability requirements. The reliability of the Unix-like operating system has been discussed by Herder et al. [44]. They suggested designs that can transparently survive crashes of critical components such as device drivers. They proposed to move operating system components and their functionality out of the kernel into unprivileged user-mode processes and to introduce protection barriers between all modules. Also, Swift [45] built a new subsystem called *Nooks* that allows existing device drivers and other extensions to execute safely in commodity kernels. When a driver failure occurs, Nooks detects the failure with the combination of hardware and software checks and triggers automatic recovery. The crash dump feature can be used to collect information about the CPU registers and memory for identifying the cause of any failure occurred. The system is unable to output any information when it hangs up, and hence Fujitsu developed the *diskdump* which collects the dump reliably even when a hang-up occurs [46]. In [47], a predictive self-healing mechanism is presented, which was implemented in the Solaris 10 operating system. It includes a diagnostic system that diagnoses the messages provided by the system logging facilities and triggers an automated recovery action when any vulnerability is observed. In [48], the authors propose a multi-agent system for diagnosing the behavior of different resources in a cluster environment. A set of agents, named as 'intelliagents' is responsible for monitoring

the system, detect performance, capacity and other problems and execute actions to repair the affected resources.

The Internet of Things (IoT) refers to the general idea of things that are managed via the Internet. IoT empowers billions of objects to communicate, recognize and respond without human intervention. Because of this scale, the reliability of the IoT devices becomes a concern. Several approaches of communication, transmission, sensing and processing in IoT system are discussed to address reliability in [49]. Angarita [50] proposes the concept of responsible objects to support self-healing IoT applications.

2.4 Other Cloud-based Management Systems

Similar to the digital signage monitoring system, researchers have developed cloud-based frameworks for other wireless sensor networks (WSN). Considering user management, access control, storage, and data retrieval Ahmed et al. [51] discussed an integrated framework of WSN and cloud. In [52], Chenaru et al. discusses an integrated architecture, considering RESTful services. They also present the advantages and challenges for a cloud-based system architecture for process monitoring and statistical analysis of network performance. Recently, Huo et al. [53] propose a cloud-based framework for on-line equipment fault diagnosis system that facilitates the integration and processing of mass sensor data generated from Industrial Sensing Ecosystem. Luo [54] designed remote monitoring cloud platform for healthcare information.

Cloud server is responsible for real-time monitoring, resource monitoring, emergency data processing, performance measurement, and other functions such as creating virtual machines, managing virtual resources, improving computation time, and reducing transmission time. The monitoring of the environment parameters, for example air quality can be done remotely. Partha [55] has presented a technique to monitor the particulate matter level based on cloud services to analyse and store the data. Cloud computing can also be helpful for monitoring transport system in real-time [56]. An effective cloud-based hierarchical architecture for effective resource management of vehicular network is proposed in [57] that facilitates sharing of computational resources, storage resources, and bandwidth resources among vehicles.

3 Technology Background

The technologies such as touch overlays, sensor, and QR code enable digital signages with interactive displaying digital contents for increasing customer engagement. A number of technologies such as cameras, RFID technologies, sensors, applications and network connectivity are used to monitor the ambient environment. The input information is processed in real-time and relevant contents are delivered on interpretation of viewer inputs [58]. Digital signage is cheaper than the printed papers for big audiences and also economical as the contents can be updated dynamically in real-time. There are many commercial cloud-based solutions available in the market provided by the companies such as Signagelive, Visix, ScreenHub, and Yodeck. Also, there are many open source platforms available such as Rise Vision, Mvix, Screenly, Concerto, and DisplayOp. Depending on the subscription type these solutions have different features and usages.

A standard digital signage system comprises display system, cloud server, IP network, and multimedia content [59]. Traditional signage systems have dedicated software installed on local computer connected to displays. At present, with the emergence of web technologies, it is possible to show high-quality media contents in a browser despite many available dynamic-content displaying applications [60]. The benefits such as platform independence, device independence, and easy application development encourage a number of researchers to use web technologies for digital signage [61]. Therefore, display system being used can be a thin-client connected to display devices. The thin-client only needs support of a web browser and network connection to access the system functionality.

A remote system administrator can use cloud server to define and schedule the content to specific players for a specific duration. This cloud server pushes the media content to the player through IP network. The display system can manage screens and send various logs and alerts to the server, which can be helpful for the administrators. This system can work on any platform such as Windows, MacOS, Linux, and Android. Many commercial solutions are supporting Linux kernel based display platform so, we are also considering Linux kernel for evaluating our work. Ubuntu team has released a version *Ubuntu Core* [62] to support a range of digital signage players, generally installed on Raspberry Pi or Intel NUC. Figure 2 depicts the overview of the system.

It can be said that the system has two terminals, one at server-side for management and another on the display side. The content files that are displayed and the scheduling their information can be edited at the management terminal which interacts with distribution server and data server. Each signage display shows the

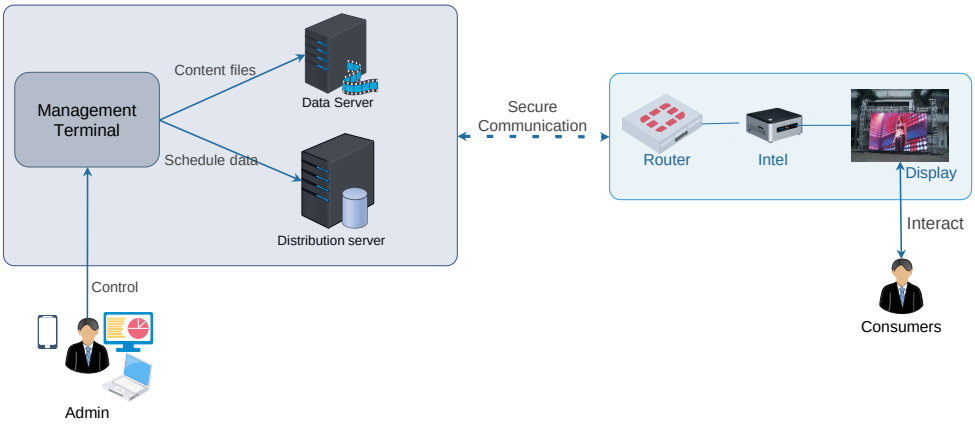

Figure 2: System overview

media contents according to its configured schedule. It also sends the consumer inputs and display results to the distribution server. The media contents can be distributed using two methods: push-type distribution or pull type distribution. The push-type system distributes the contents to the signage terminals as configured by the distribution server, whereas in the pull-type system display terminals have to inquire the server for any available media contents to download. If the contents to download are present, the system downloads it to the terminals. These existing digital signage communication architectures generate high traffic at the server when frequent content changes are required. Hence, the content transfer speed can become slow. The solution to this problem is to decentralize the network rather than having a centralized network with only one distribution server. Kim [63] proposes a Wi-Fi Direct based pipelining content distribution scheme for digital signage network to resolve the high traffic on the server. In the proposed scheme, all of display terminals can transmit the contents to each other with Wi-Fi Direct. A pre-ACK scheme is also proposed to support reliable and fast transmission.

Any minor errors can affect the reliability of the whole system and therefore the ability to diagnose bugs is important. These errors can occur at all levels of the digital signage system or components. As discussed below, in [64], Anu et al. mentioned various failures detected in cloud environment can also be realized for the cloud managed digital signage system:

- *Process Failure*: Any failure to process running in the display application can cause the application to stop.
- *Application Failure*: Any unusual circumstances within the thread of the

monitored application can cause failure.

- *Network Failure*: As digital signage system is based on cloud computing, any errors related to the network can stop the entire system.
- *Node or Screen player Failure*: Any hardware related problems, for example, hard disk errors can cause the node to fail.

Therefore, in the following sections, various concepts required to understand failure occurrences in each subsystem will be discussed. Few steps are required for managing digital signage system such as:

- *Monitor*: The process of collecting useful information to observe any unexpected behaviors. The behavior could focus on individual components, functionality of the system, or business requirements.
- *Analyze*: The process of detecting failures using diagnostic reports.
- *Plan*: The plan for taking actions to repair the failures. These actions depend on the type of faults.
- *Repair*: The repair actions needed to be executed.

Also, it is important to smartly manage the system. The following key objectives of the management module for a digital signage system are helpful.

- *Auto-aware*: The system should be aware of its current status and connections with other systems.
- *Auto-configurable*: In case of any changes, the system should be able to re-configure itself.
- *Auto-recoverable*: The system should be able to detect, diagnose and recover from any anomalies to maximize the reliability, availability, and survivability.
- *Auto-protecting*: The system should detect and protect itself from any unauthorized accesses.

In [65], the authors refer that the main objective of self-healing in the operating system is to avoid faults that require a system restart. In the next section we will start with the booting process description of a Linux operating system to understand the system for possible faults.

3.1 Linux Booting Process

The booting of the system starts when the processor executes instruction in kernel mode at a specific address. This address points to the code of the firmware. The firmware provides the required instructions to the device so that it can communicate with the other system hardware. There are different types of firmware interfaces, widely known are Basic Input/Output System (BIOS) and Unified Extensible Firmware Interface (UEFI). The BIOS checks all the devices (screen, keyboard, monitor, etc.) connected to the system and searches Master Boot Record (MBR). Usually, MBR is stored in the first sector on the hard disk to execute the bootloader program.

Grand Unified Bootloader (GRUB) is the widely used Linux bootloader program. In general, the main functionality of bootloader includes selecting kernel, allowing the user to edit kernel parameters, and providing support to boot operating systems. GRUB 2 (descendant of GRUB) code requires more space than the MBR space and so a multi-stage design is needed. It executes in two stages. In the first stage, the pointer jumps to the address mentioned at MBR, which is the location of GRUB 2's core image. The core image contains the GRUB kernel and drivers for handling the filesystem into memory. In the next stage, console is initialized and the configuration file containing all the kernel information loads. If this configuration file is missing, the bootloader would cease the boot process and provides users an opportunity for manual configuration. After that, the boot menu is presented to select a kernel version. When a kernel is selected, bootloader loads its image. This kernel image is then mounted to root file system and initial RAM-based file system (*initramfs*) into memory. Initramfs contains a small executable called *init* which handles the mounting of the real root file system [66]. Initramfs acts as a storage hardware driver support for the kernel. After the root file system has been found, it checks for errors and mounts. Once successful, the initramfs is cleaned and the systemd daemon on the root filesystem is executed.

Systemd starts as a first process and runs with process id 1. It is an init system used in Linux distributions to bootstrap the user space. It runs scripts to start userlevel services and is responsible for managing all other processes. Systemd is also the last daemon to terminate during the shutdown. Appendix I.II describes more about systemd.

A kernel is the core of an operating system. When the system boots up, it is the first part of operating system loads into the main memory. On Linux systems, the kernel is normally located at */boot/vmlinuz*. There are also many modules called loadable kernel modules that kernel can load on demand. In general, the kernel is responsible for the following tasks:

(a) Determining the allowed processes that can use the CPU.

(b) Memory management, i.e. keeping track of all allocated memory to the running processes and memory shared between processes.

(c) Acting as an interface between hardware and process.

We can check the kernel parameters from the file */proc/cmdline* as follows:

```
BOOT_IMAGE=/boot/vmlinuz-4.4.0-64-generic
root=UUID=5026baf7-71b3-4378-a585-2eb80f5e14b7 ro console=tty0 console=ttyS0,115200
```

The *ro* parameter instructs the kernel to mount the root filesystem in read-only mode upon starting user space. This is done to ensure that *fsck* can check the root filesystem safely. After this check, filesystem remounts to read-write mode. If the kernel receives parameters that it does not understand, these parameters are saved to pass to *init*. The parameter passing happens while starting the user space. For example, to indicate init program to start in single-user mode, *-s* can be passed as kernel parameter.

The summary of the boot process can be seen from the Flowchart 3:

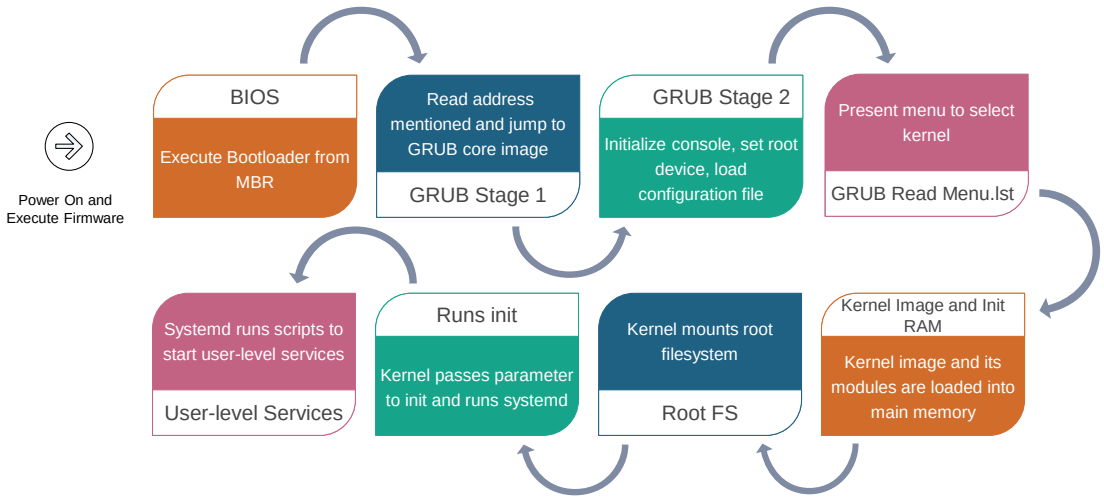

Figure 3: Booting process overview

3.2 Linux Kernel Logging

The kernel logs are useful for detecting and diagnosing problems with the system. Logging within the kernel is performed using the printk function which is embedded

with log level as:

```
printk( KERN_CRIT "Error code %08x.\n", val );
```

It copies a string into the kernel log buffer using a special function that manages the bound of ring. The size of the kernel ring buffer ranges from 16kb up to 1MB. The logging overview can be seen from Figures 4.

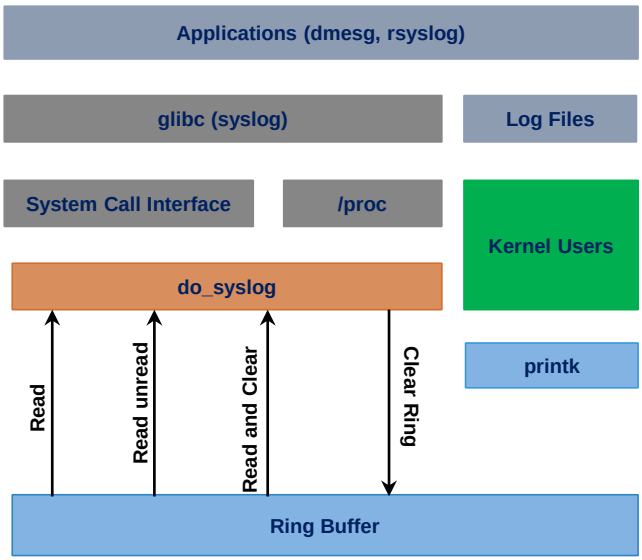

Figure 4: Kernel logging APIs [67]

The *syslog* serves as an I/O and control interface to the kernel log message ring buffer. The *kmsg proc* file system is an I/O path that provides a binary interface for reading log messages from the kernel buffer. This is commonly read by the daemon (*klogd* or *rsyslogd*) that consumes the messages and route them to the appropriate log files. The *rsyslog* daemon has its configuration file as */etc/rsyslog.conf* to understand the kmsg file system interface. The *rsyslog* daemon is also responsible to filter the required log levels.

All system messages are stored in the directory */run* can be consumed by rsyslog and redirected to directory */var/log*. User space provides various access points to read and manage kernel logging. The command *journalctl* reads logs across a variety of its attributes. Also, the command *dmesg* prints and controls the kernel ring buffer. It uses the system call *klogctl* to read the kernel ring buffer and emits it to standard output (stdout) [67]. The kernel ring buffer can be emptied using the -c option of dmesg. The log level can be changed using -n option, and the size of the kernel log messages buffer using -s option. By default dmesg identifies the buffer size using SYSLOG_ACTION_SIZE_BUFFER operation to *klogctl*.

The Linux kernel messages provide information to help debug critical bugs. The messages seem as [68]:

```
kernel BUG at kernel/signal.c:1599!
Unable to handle kernel NULL pointer
dereference at virtual address 00000000
pc = 84427f6a
Oops: 0001 [#1]
```

The four digit value after the parameter Oops, shows the error code in hexadecimal. The error code is encoded as shown in Table 1 [68]:

Bit number	**Value = 0**	**Value =1**
0	No Page Found	Protection Fault
1	Read access	Write access
2	Kernel-mode access	User-mode access
3	N/A	Use of Reserved bit detected
4	N/A	Instruction fetch fault

Table 1: Kernel oops error codes

So, the Oops error code, 0x0001 means it was a page protection fault, read access in kernel mode. The error codes 0x0000, means a page was not found by a read access in kernel mode.

3.3 Backup and Deduplication

This section covers some techniques for file backups and storage that we use for log management. In the cloud-based digital signage monitoring system, the server receives data periodically from the displays. It is necessary to manage these data optimally so that backup is available, recoverable, and consume minimal storage. There are a few key parameters considered for backup such as [69]:

- ***Backup schedule***: The timing and frequency for requesting displays data.
- ***Data retention period***: The period for which data should be kept before deletion.
- ***Backup type selection***: The data backup criteria, i.e. incremental or full. Incremental backups help in decreasing disk space consumption.
- ***Backup restoration tests***: The periodic tests of backup restoration process are required to make sure that the restoration works properly.

There are mainly three common types of data backup method [70]:

- ***Full backup***: Full copy of the entire data set is saved. It requires more storage space than other types of backups, but a process of restoring lost data from backup is fast.
- ***Partial backup***: In this type, only changed data after the last backup are saved. It begins with a full backup. This is further categorized into two types:
  - ***Incremental backup***, stores only data changed after the last incremental backup.
  - ***Differential backup***, stores only data changed after the last full backup.
- ***Mixed backup***: A sequence of full backup and some differential backups rotated repeatedly.

The differences between these types are summarized in the Table 2

Backup Type	**BackUp Data**	**Backup Time**	**Restore Time**	**Storage Space**
Full Backup	All data	Slow	Fast	High
Incremental Backup	New or Modified	Fast	Moderate	Low
Differential Backup	All data since last full	Moderate	Fast	Moderate

Table 2: Difference between types of backup

The full backup often makes many redundant data copies, as datasets of the display system don't have many changes between the backups. Incremental backups are quicker to perform and less storage intensive than full backups.
As from the displays, many copies of the same file can be received time to time. Therefore, to eliminate redundancy a space saving mechanism called *Data deduplication* can be used. It replaces multiple copies of data with references to a single compressed copy, thereby reducing the amount of needed space capacity. There are various tools available for data deduplication such as *FSlint* and *fdupes*. These tools scan directories to search duplicate files and provide actions such as to show, delete or replace the files with hardlinks. They work by comparing file sizes, MD5 signatures of partial file, MD5 signatures of the entire file, and then a cryptographic hash function of the entire file for verification. The comparisons of both these tools can be checked at [71] resulting that the fslint is better in detecting duplicates.

3.4 Reliability

Reliability is defined as the ability of a system to function over a specified period of time. Therefore, the reliability of a system can be characterized in terms of the system's individual modules. The widely used reliability metrics are availability, mean time to repair (MTTR), mean time to failure (MTTF), failure rate, and repair rate. The expected time of correct execution before a failure occurs is measured as MTTF [72]. MTTR is the average time required to troubleshoot and repair failed equipment of the system. Mean time between failures (MTBF) refers to the amount of time that elapses between one failure and the next. Mathematically, MTBF is the sum of MTTF and MTTR, i.e. the total time required for a device to fail and that failure to be repaired. Whereas, availability is the proportion of time that the system is available to do useful work. Availability is calculated as MTTF/(MTTF+MTTR).

Our goal is to improve reliability of a digital signage system by reducing the frequency of operating failures, or by reducing time to repair and recover from a failure. There could be different categories of failures as mentioned in [73]:

1. *Timing failures* occur when a module or subsystem violates timing constraints.

2. *Response failures* occur when a module outputs incorrectly, such as incorrect values or the state transitions.

3. *Crash failures* occur when the component stops producing any outputs.

4. *Omission failures* occur when a module omits to respond to any inputs.

Each failure category requires its own detection and containment mechanisms. For simplicity, reliable systems commonly assume that only a subset of these behaviors occurs and remove from considering the other failure modes.

3.5 Summary

This chapter described various modules of digital signage system such as display system, content delivery cloud server, IP network and multimedia content in terms of its functionality. A method to reduce traffic on the centralized server is also discussed. Further, it elucidated about various failures detected in a typical cloud-based digital signage environment.

Besides, the booting process of a Linux-based system is discussed that explains about the BIOS, bootloader, kernel, and init system. An overview of Linux-based kernel logging is also provided. In addition, it described the features of a log

management module that optimally backs up the logs eliminating redundancy. Finally, this chapter mentioned about reliability and different categories of failures that can affect it.

4 Analysis of Display Failure Modes

In this chapter, we discuss various issues which can occur in the digital signage system and solutions to each of the issues. The operating systems taken into consideration are Linux-based which provides many advantages over other propriety operating system e.g. Windows. Few benefits of using Linux-based systems are mentioned as follows [74]:

(a) The usage price is low, as it is open source.
(b) The Linux kernel is highly efficient, as it can work on low RAM.
(c) It supports x86 and many cross platform system.
(d) The Linux kernel structure of the network is complete, provides various Ethernet types support.

In the following sections, various key components of a digital signage system are analysed.

4.1 Display Devices

A display device is an output device for showing data in visual form. There are different varieties of display devices available, for example, LCD, CRT, LED, Projector, and OLED. In general, a display device turns off its screen after a configured time to save the power consumption and remains active for a signal on the video cable. There exists an internal circuit in it which actively watches the signal. The CPU doesn't communicate directly with the display device. It sends commands to the graphics card which in turn generates an electric signal and this signal is transformed into a picture on the screen. The communications with graphics card happen through graphics card driver that translate CPU requests into graphics card commands. In the case of in-built graphics card, a portion of main memory is used for video data, however, a dedicated graphics card has separate video RAM. For the system having multiple graphics cards, NVIDIA develops Scalable Link Interface (SLI) to link them together for producing combined output.

A separate electronic circuit, graphics processor unit (GPU) is designed for efficient 3D rendering and video decoding. The GPU presents on a video card, or embedded on the motherboard. The GPU can access the video RAM using the specialized direct memory access (DMA) component of video card. The GPU maintains a command ring buffer to communicate with the CPU. Any command that needs to be executed by the GPU is submitted to the queue in this buffer. This can be visualized from the Figure 5.
The signal output to display monitor is either an analog signal (VGA) or a digital signal (HDMI or DVI). The standard protocol known as *Display Data Channel/-Command Interface (DDC/CI)* was introduced in 1998 for digital communication

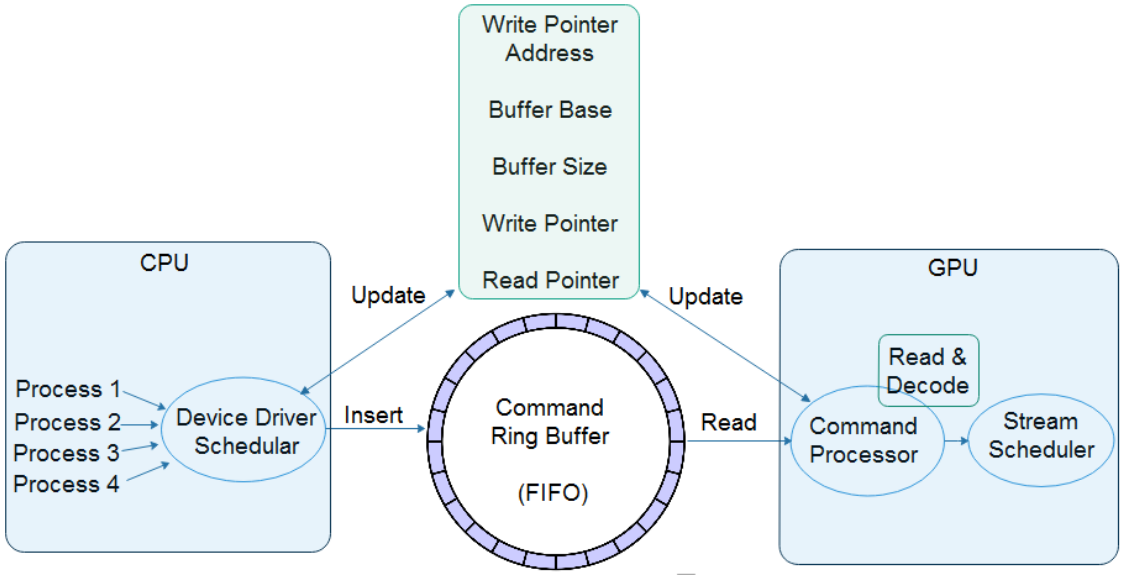

Figure 5: Ring buffer between the CPU and GPU

between a display device and a graphics card adaptor. It enables the host to adjust parameters of a display device and to receive sensor data from it.
There can be many problems that can affect contents to be displayed on screen, as explained below:

(a) *Connected computer or thin client is powered off*: This scenario may be identified by the cloud, in case no data is received for a certain period of time. Cloud can also try to send external instructions to check if the display system responds. But remotely it is difficult to predict with confidence that the connected computer is powered off, because there could other failures leads non-responding displays such as network access problem.

(b) *The cable connecting the computer and display device is disconnected*: The display screens are managed by the windowing system. There are various windowing systems available for different operating system such as X11, Mir, Desktop Window Manager, and Quartz Compositor. The configuration utility of these windowing system can be used to check which display devices are enabled on the display ports. This failure can only be recovered manually, so the responsible stakeholders can be alerted using external communication methods such as SMS or email.

(c) *The cable connecting the computer and display device is faulty*: This failure is a kind of hardware failure which can be detected as the previously described

problem using the configuration utility of the windowing system and needs manual intervention to replace the faulty cable.

(d) *Multiple display devices attached and the wrong one is selected*: There are many tools available that interact with the windowing system manager to get the list of desktops, switch desktops, and manage running windows. These tools can be helpful in getting the information of applications running in particular display and can be moved to correct one.

(e) *The operating system has crashed having a malfunctioning display driver*: The driver failure can be detected by the operating system itself and shown as pop-up messages. There could be an entry in the system logs about this failure. Therefore, by monitoring the system logs, it is easy to detect. To resolve the driver issue, it requires re-installation of driver or rebooting the system.

(f) *The display device itself got faulty or powered off*: This failure is also a hardware issue and can be detected by checking the communication channel between the display device and graphics card. In the case of failure, display device would not respond to the graphics card adaptor. The faulty display device requires replacement, so the stakeholders would be notified.

(g) *The application is running with incorrect orientation*: Now a days, web-enabled display devices are capable of determining their orientation. But still the displays available at economical prices don't come with a sensor for detecting its orientation. Therefore, the identification of wrong orientation is mostly based on user reports. To resolve it, cloud server can provide remote functionality to stakeholders for changing the orientation.

Other problems related to the display are:

1. ***Blank Screen***: Generally, blank or black screens indicate a failure in feeding data to the GPU. As explained before the kernel uses a driver for doing graphics display which can misbehave sometimes. So, disabling this driver can recover the system.
   Another reason of black screen during the system booting is the movement of video mode setting into the kernel. When the windowing system server starts, the programming of the hardware specific clock rates and registers on the video card happen in the kernel rather than in its driver. Some cards don't work properly with this setting and ends up with a black screen. The solution is changing the bootloader configuration so that to instruct the kernel for not loading the video drivers and use BIOS modes until windowing system is loaded [75].

2. ***Screen Freeze***: The screen freeze means the screen is being displayed, but doesn't update the contents. This happens if the video driver sends corrupted data to the GPU, that hangs up the GPU. In most of the cases, GPU freezes due to bugs in the kernel code, or due to the bugs caused by other components in the graphics framework. These bugs can be solved by rebuilding those components. System logs records freezing failure messages that can be identified with the starting line text as "GPU lockup". Also, screen freezes when screensaver or video player changes DPMS settings, so it is better to turn off the screensaver or disable DPMS permanently.

It is also useful to take an image of screen dump to understand the information being displayed on the display device. Appendix I.V proposes some methods for obtaining screen dump using python. To save the network bandwidth, displays should be configured to send images in different scales.

The commonly used windowing system for display in Linux is X Window System (X11). It provides the basic framework for displaying windows and its interaction with input devices. For X11, the program *xwd* (X Window dump) can be used to capture the screen dump and the tools e.g. *xwininfo* can be used to get the screen dump in text mode. The tool *wmctl* can be used to interact with the windowing system manager and other displaying windows information can be obtained from *xdotool, xprop, xset* and *xrandr*. There exist a tool *DDCcontrol* which relies on DDC/CI control channels and so can be used to control display device without touching the hardware controls. The usage of this tool is explained in Appendix I.I. The official list of display devices and graphics cards supported by the tools can be checked from [76].

4.2 Storage Device

The files are stored persistently in storage devices such as Hard disk drive (HDD), Solid-state drive (SSD) or flash drive. HDD consists of a metal platter with a magnetic coating that stores data, and a read-write head on an arm accesses the data while the platters are spinning. On the other hand, SDD stores data on interconnected flash memory chips. SSDs are more expensive than hard drives in terms of storage, but inherently faster. In HDD, the read-write head stays over the drive platter hanging over a distance which can damage the disk on vibration. Therefore, in the non-stationary locations SSD storage should be used as it doesn't have any moving components. The HDD has advantages in price, capacity, and availability over the SSD. Whereas the SSD had advantages over HDD on speed, ruggedness, noise, and fragmentation. These devices are connected with CPU using controllers. The storage device controller is responsible for device operations

and data transfers. The common types of controllers are IDE, SATA, SCSI, SAS, USB, etc.

In recent times, SSD is being widely used in display systems for storing data considering better performance and reliability. Failures in SSD are unpredictable. Latest storage drivers alert when the life of SSD approaches its end.

The problems related to storage could be:

(a) *Develops bad memory blocks*: SSD can develop bad memory blocks with time and so corruption of stored data. It handles block management by remapping bad blocks with the set of spare blocks. Unfortunately, the data written on the bad blocks can't be corrected.

(b) *Develops read-only partition*: Sometime when the file system encounters errors, it decides automatically to remount to read-only for protecting itself from further damage. This incident can be checked from the system logs. The solution to solve this issue depends on the file system type. In case the file system is not the root partition, complete unmounting the partition and then remounting can be helpful. Whereas, if it is the root partition or remounting doesn't work, rebooting the system is required so that it can check and remount the file system cleanly.

In the case of read-only SSD, it may happen that the SSD enters into its final life stage because it can no longer able to replace write cycle depleted cells with the spare ones. Therefore, the only solution left is to backup the files and replace the storage disk.

Also, if there is no space left, SSD can become read only, in that case unwanted data needs to be deleted.

(c) *Fails booting*: If the drive could not boot, it may have the following problems [77]:

- Damaged boot storage area
- Corrupted system configuration setting
- Bug in the operating system kernel
- Incompatible interface or outdated drivers
- Operating system unable to understand the drive format

(d) *Corrupts filesystem*: File systems provide a means of organizing data on a storage medium. The basic layout of Linux file storage is described in Appendix I.III. There are a number of scenarios in which a file system might get corrupted such as via a hard reboot. To repair the corrupt filesystem, during the booting process Linux automatically runs a file system check

command *fsck*.

In Linux, the health of the hard drive can be checked using *SMART*. The command *df* can be used to find the space left on the disk. df lists all the mounted partitions with their size status. Linux has set aside a number of blocks in the file system, known as reserved blocks which can be accessed by root user to fix some problem. The largest directory can be found using the command *du*. It can be used to save the output in sorted manner in a file as:

```
sudo du -ckx | sort -n > /tmp/space-root
```

The remounting of the disk to read-only in Linux is reported to system logs with the lines begin with *EXT4-fs error* as:

```
EXT4-fs error (device dm-0): ext3_find_entry: reading directory
#23922966 offset 0 0:0:0:0: rejecting I/O to dead device
```

The command to remount partition to read-write is

```
sudo mount -o remount,rw /partition_path
```

The physical disk error can be tracked from system logs logged as [78]:

```
hde: dma_intr: status=0x51 { DriveReady SeekComplete Error }
hde: dma_intr: error=0x40 { UncorrectableError }, LBAsect=2399063,
sector=598768 end_request: I/O error, dev 21:05 (hde), sector 598768
raid1: hde5: unrecoverable I/O read error for block 598768
```

The command *badblocks* can be used to check the hard disk for bad sectors as:

```
sudo badblocks -v /dev/device
```

fsck can be used to mark badblocks unusable using

```
sudo fsck -l badblocks_output /dev/device
```

4.3 Network

Network is the communication path for the purpose of transmitting, receiving and exchanging data between cloud and displays. There could be many problems related to network connection such as:

(a) *Fault in network interface card*: A network interface card (NIC) is a circuit installed in a system so that system can be connected to a network. The boot time messages from kernel provide idea if the operating system detects the presence of network interfaces or not. It can also be checked using PCI configuration space with the list of devices attached to PCI buses. The failure in detecting it using boot time messages or PCI configuration can be

because of faulty or unsupported hardware. Network card failures can also be troubleshot by checking TX (transmit) or RX (receive) errors.

(b) *Unstable connection*: Older modems and routers can't deal properly with IPv6. This might cause unstable connection or even a complete failure to establish any connection. Disabling IPv6 feature can resolve it.

Also, if the system has a combined Bluetooth and wireless chipset, enabling Bluetooth can block wireless connections (WiFi or LTE). The solution is to disable Bluetooth and reboot the machine. In the case of bad wireless connection, disabling power management for the wireless card can help.

(c) *Unplugged Ethernet cable*: The situation of an unplugged Ethernet cable can be checked from network file system entry. For Linux, file `/sys/class/net/enp0s25/carrier` tells if eth[n] is up or not.

Linux includes network manager software which aims to make the network connections "just work". If it doesn't work properly, it may require to reconfigure the network manager.

The list of devices on PCI buses can be checked on Linux using:

```
lspci | grep Ethernet
```

which outputs as:

```
01:01.0 Ethernet controller:  Intel Corporation 82547EI
02:01.0 Ethernet controller:  Intel Corporation 82540EM
```

The command *iwconfig* can be used to check whether power management is on or not. To disable the power management, in the file `/etc/NetworkManager/conf.d/default-wifi-powersave-on.conf` change the value of parameter *wifi.powersave* to 2 and reboot the system.

4.4 Sound

Sound is stored in a computer as digital form i.e. sampled and quantized into a finite number of bits. The stored digital waveform is clocked out through a digital to analog converter, then passed through an analog amplifier to a loudspeaker. A sound card attached to the motherboard is required for this processing. This processing can also be done on modern video cards with HDMI, to output sound along with video. The sound system architecture of Linux can be briefly found at [79]. The error in detecting sound card can be checked from the system logs. The troubleshooting sound remotely is quite difficult with the system.

In Linux, the sound devices connected to the system can be checked from:

```
cat /proc/asound/cards
```

If the display device has integrated audio support, it can be configured using the tool *DDCcontrol*.

4.5 Slow System

There could be many reasons to slow down a system's performance such as:

(a) *Background programs*: The most common reason for a slow machine is unwanted programs running in the background. A regular check of all the running processes and services is required. To resolve with this kind of issue, all unwanted services should be disabled or removed.

(b) *Out-of-Memory*: Another reason for slowing a system could be out-of-memory issue. It is an undesired state of system when no additional memory can be allocated to any process. The information about the physical RAM status and swap usage can be used to analyse this.

(c) *Low disk space*: The system requires disk space for saving temporary files and unavailability of this space can make the system slow. The space of the primary disk should be regularly monitored and the unused files should be deleted.

(d) *Outdated Software*: The outdated software or drivers can hamper the system performance. So, regular updating them is recommended.

(e) *Hardware issues*: There could be many hardware issues that could slow down the system, such as damaged components in the system. It can be detected by performing hardware diagnostic tests using various available utilities and fixed by replacing the hardware.

Considering the above mentioned issues, troubleshooting of a slow system can be started by analysing the metrics such as average system load, system uptime, total number of running processes, total used memory, total free memory, and the number of resources being used. The other metric to consider could be I/O wait. If I/O wait is low and idle percentage is high, it can inferred the problem is not because of CPU resources. There might be network problems, or slow queries to databases [80]. If both I/O wait and idle percentage are low, then it is likely that user time percentage would be high. So, diagnosis of high user time is required. That requires identification of processes consuming the most CPU. In the case of high I/O wait, further detailed diagnosis required.

In Linux, the command to check the load is *uptime* which provides output as:

```
13:35:03 up 103 days, 8 min, 5 users, load average:  1.03, 10.12, 17.01
```

The three numbers after load average- 1.03, 10.12, and 17.01, represent the 1st, 5th, and 15th minute average load on the machine, respectively. Also, the command *TOP* is helpful for getting system information. Linux caches the file into RAM and doesn't necessarily remove it after program execution. To find out how much RAM is really being used by the processes, it requires to subtract the file cache from the used RAM. If the mathematical difference of actual used memory and file cache is high, and the swap usage is also high, then there could be a memory leak problem.

The Linux kernel also has an out-of-memory (OOM) killer that can kick in if the system runs dangerously low on RAM. However, it is not necessary that the OOM killer kills the real culprit. If the OOM killer kicks in and take any actions, it is always written to the syslog as:

```
4626417.32051_1317.hostname:2,
S:Out of Memory:  Killed process <PID> (<PROCESS_NAME>)
```

The kernel maintains an oom_score for each process which can be checked from */proc* filesystem under the pid directory. The higher the value of oom_score of any process, the higher is its likelihood of getting killed by the OOM Killer in an out-of-memory situation.

4.6 Device Drivers

A device driver is a program that controls a device attached to the system. These devices can be a storage device, display device, network card, etc. A device driver converts the operating system I/O instructions to the instructions that a device can understand [81]. The device driver failures can cause system failure such as crash, hang, or exhibit arbitrary incorrect behavior. The robust system should be able to detect faulty driver behavior and recovers by restoring with a properly functioning driver. Generally, in current systems recovery requires system reboot and recovery of the affected application state [82].

4.7 Summary

In this chapter, we discussed about the issues that can occur in various components of a digital signage system and their possible solutions. The key components discussed includes display devices, storage device, network, sound, and device drivers. The working of each of the components is also briefly explained in the respective sections.

The mechanism of communication between CPU and display device is described. GPU and CPU communicates using a ring buffer. A standard protocol

is used for digital communication between a display device and a graphics card adaptor. Storage devices such as HDD and SDD are compared and concluded that SDD should be preferred for digital signage system. The problems related to storage such as bad memory blocks, read-only partition, corruption of filesystem and booting failure and network-related problems such as fault in NIC, unstable connection, and unplugged Ethernet cable are also explained. Additionally, several reasons that can affect system's performance are mentioned such as out-of-memory and low disk-space.

5 Cloud-Based Automatic Failure Analysis and Recovery

This chapter describes the design of the proposed application used to monitor and recover from failures in the digital signage displays. First, we explain the architecture of our system. Second, we describe the implementation of our solution. Third, we present the evaluation scenarios and explained how each scenario is tested. This chapter describes the ways of using and further enhancing our application.

5.1 System Architecture

The digital signage system consists of a Centralized Maintenance and Monitoring (CMM) server that controls multiple display systems from cloud. The end-user can interact with CMM using a Web interface. The communication between the CMM server and the displays is done via a secure WebSocket. WebSocket [83] is a protocol used to establish a full duplex communication over a single TCP connection. After the connection is established, WebSocket supports exchange of message-oriented text and binary data frames between the displays and the cloud server.

A display system consists of a lightweight processor machine and a display device. The display device is connected to HDMI or VGA port of this machine. The advertising contents of the subscriber can be displayed using any web browser, preferably Mozilla or Chrome, running on the machine in Kiosk mode.

The displays periodically send debug logs at regular intervals. These debug logs are pre-configured for each display system. The logs received at the CMM server would then be analysed and stored in a database. These logs are accessible to system administrators. The CMM server invokes its log manager component to reduce the storage space required to store these periodic logs. These set of logs are analysed by the CMM and corresponding actions are generated. These actions are sent to the display system.

When the display system boots up, the proposed application starts automatically as a system default service. It is initiated just after the *systemd init daemon* starts. Any of the unwanted default activated services should be disabled from the display system. Also, the system is configured to start a web-browser with a specified web URL in kiosk mode. The administrators can remotely instruct displays using CMM server, to run commands or scripts. As the display systems are installed in remote locations, there is no possibility for them to get user inputs during its booting process. Therefore, it is difficult for any user level application to diagnose problem during the machine boot. The end-to-end digital signage system integrated structure is depicted in Figure 6. The main advantages with such

system architecture are:

- It supports both detection and recovery of system failures.
- It enables self-healing of a malfunctioning system without manual interaction.
- It is a lightweight solution.
- It automatically handles all the known failures.

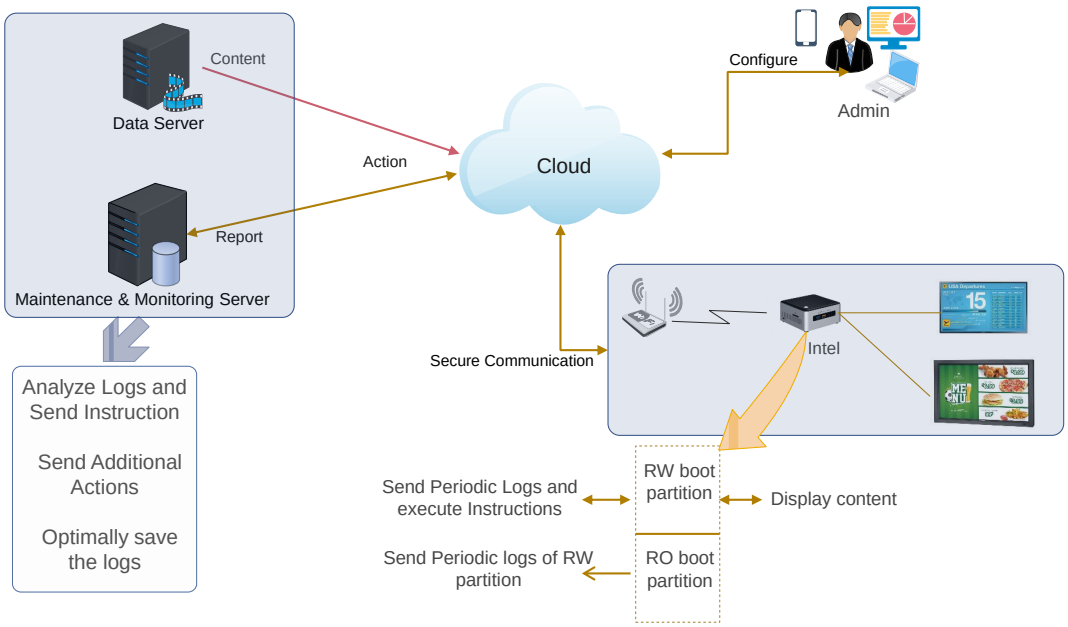

Figure 6: Proposed solution

The functionality of CMM server is categorized into following modules:

- **Log Collector**- This module is used to collect periodic logs from the monitoring display.
- **Log Analyser**- This module is used to parse the logs, transform the monitoring results, and decide appropriate actions for each display. There can be different actions such as closing the browser, rebooting the display system, notifying administrator about any physical layer failures and others.
- **Storage Manager**- This is used to store the periodic logs in an optimized non-redundant way to a persistent database.

- **Administrator Controller**- This is used by the system administrators to send external instructions to the displays such as changing configuration or the closing the display device.
- **Instruction Generator**- This module generates all the instructions required to recover the display from any failures.
- **Instruction Notifier**- This module is used to send all the automatically generated and external administrator actions to the monitored display.

Figure 7 illustrates the overview of different modules connected together for the monitoring system.

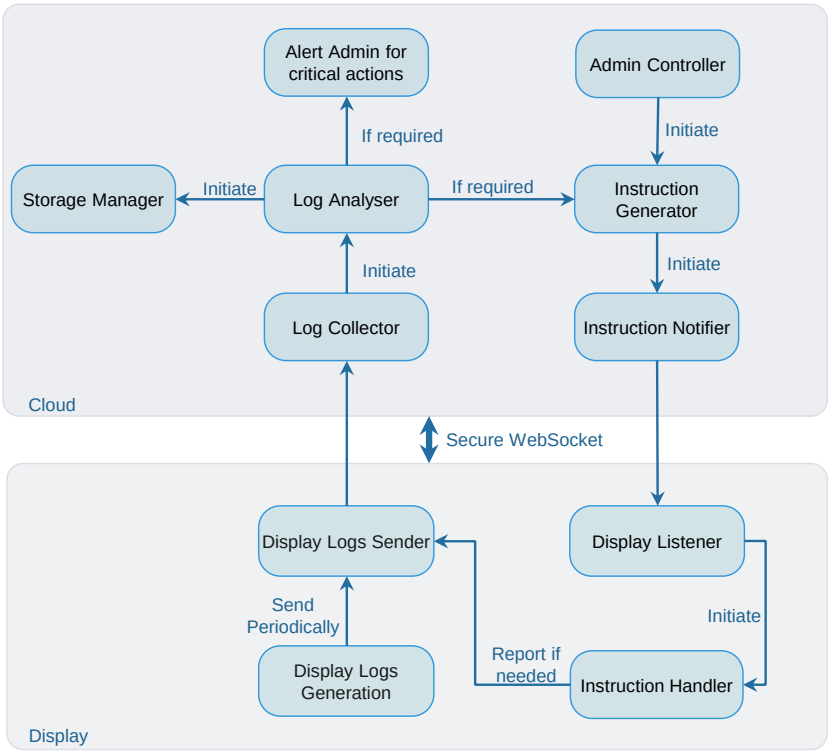

Figure 7: Monitoring system

5.2 Implementation

The proposed solution is implemented in Python and Node.js. The software effort used are shown in Table 3.

Programming language	Lines of code	Lines of comment	Number of files
Python	1001	88	14
JavaScript	588	150	5
Shell	30	0	8
Total	**1619**	**238**	**27**

Table 3: Software effort of the implemented application

The source code is uploaded to the GitHub repository and open to public. The application is divided into two modules, the client and the CMM server. The code is modular to allow easy extension of any new features in future. The display side python files as shown in Figure 8. The working of different files on display side is explained below:

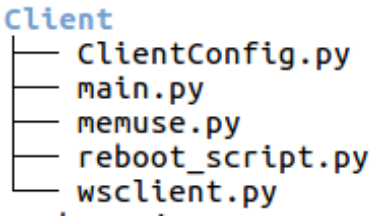

Figure 8: Display side files

- **main.py**: The display starts its execution with this file. The functions implemented here are to check system settings, generate periodic logs, create a thread for establishing network with cloud, and send periodic logs. It also runs a network analyser in a thread.
- **wsclient.py**: It opens the channel to communicate with the cloud using a secure websocket. Upon receiving instructions from server, it processes the instructions and if required, responds feedback to cloud server.
- **reboot_script.py**: The network analyser thread uses this file to reboot itself, either normally or using a recovery partition. The recovery partition method is explained in Chapter 6.
- **memuse.py**: This is used to compute CPU usage using the */proc* files.
- **ClientConfig.py**: This file is configurable by the administrators and contains various setup configuration parameters.

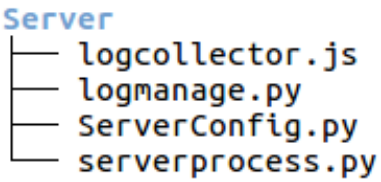

Figure 9: Server side files

The files used at CMM server are shown in Figure 9.
The working of different files on the server side is explained below:

- **logcollector.js**: It runs websocket server. When it receives any data from display, it invokes *serverprocess.py*. It also create https server for providing an interactive administrative console.

- **serverprocess.py**: This file has various functions which analyse the received logs and send instructions to display if required.

- **logmanage.py**: This file contains code responsible for the management of periodic logs so that it would take less space. The periodic logs are saved by default under *./uploadserver*. The first function of the log manager is to find the duplicates. This is done by firstly comparing the size of files and if it matches then comparing their hashes. The following algorithm is used to compare the hashes:

  Step 1: Create hash table of files where the key is the 'file size' .

  Step 2: Calculate hash table with hash of first N bytes (assume 1024) only for the files having the same file size. This is done to make algorithm faster.

  Step 3: For files with the same hash, calculate the hash of the full contents and files with matching ones are unique.

  This algorithm is similar to the *fdupes* algorithm. We have compared duplicated counts of our algorithm with other widely used stable tools by executing them on 50 periodic logs, the result is shown in Table 4. According to this comparison, we can conclude that our algorithm is better for the display system logs.

- **ServerConfig.py**: This file is configurable by the administrator and contains various configuration parameters.

Algorithm	Number of Duplicates
FSlint	1171
fdupes	1333
dupeGuru	345
Rdfind	1232
rmlint	730
Our algorithm	1337

Table 4: Comparison of duplicates detection algorithms

An interactive web application is developed to provide various external control to administrators. The benefit of a web application is that it is accessible by various devices such as smartphone, notepad. The web user interface is developed in html, the profile page is shown in Figure 10.

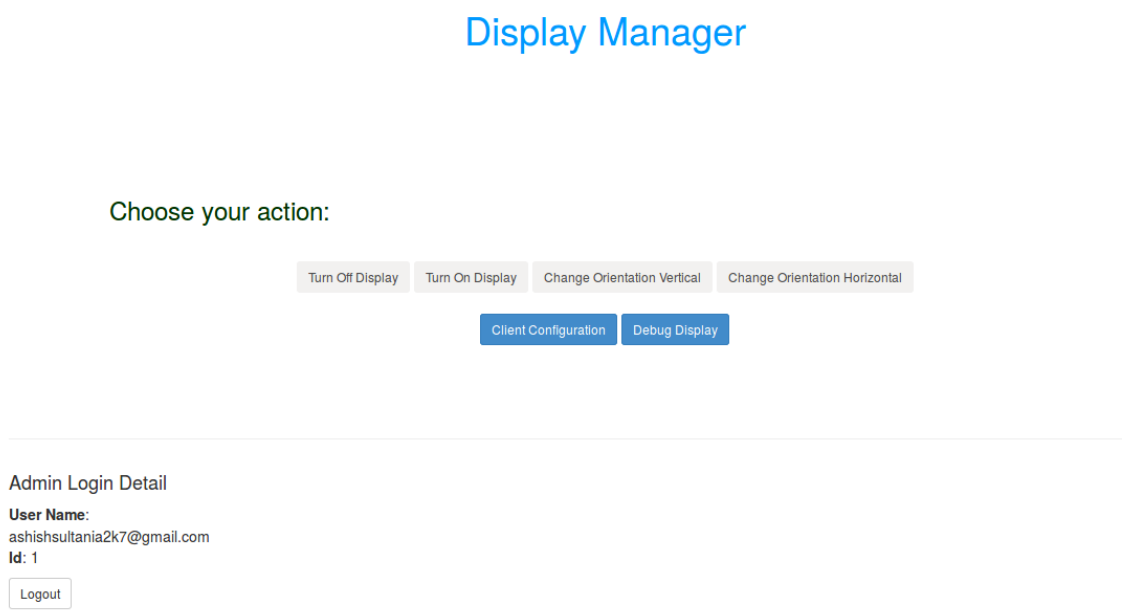

Figure 10: Web user interface profile page

Figure 11 shows the important files used to develop the web application and explained below:

- **database.js**: It is a configuration file for setting paths for database file and others.

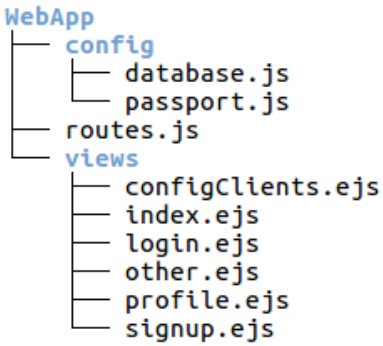

Figure 11: Web UI files

- **passport.js**: It is a configuration file for database table.
- **route.js**: This file has back-end code for all the website locations.
- **views**: This folder consists files used for developing front-end graphical interface.

5.3 Evaluation Scenarios

There are many scenarios that can happen to the display system running used to display content.

5.3.1 Display Application Not Running

This scenario can be observed when the default display application is not started automatically after booting the display. The cloud can easily detect it by analysing the periodic logs looking particularly the window display information. The cloud would then instruct the display system to run the application in Kiosk Mode.

For Linux, the cloud checks the *wmctrl.log*. If the default application's name is not logged in the log, it will send its starting command to the display.

5.3.2 Display Application Crashed

It can happen that the display application crashes after sometime as observed from Figure 12. This case also requires the window display information from periodic logs i.e. for Linux *wmctrl.log*. In addition to this, the cloud can also analyse the application crash log and find the reason for its crash. This further analysis can help in fixing any reoccurring external issues.

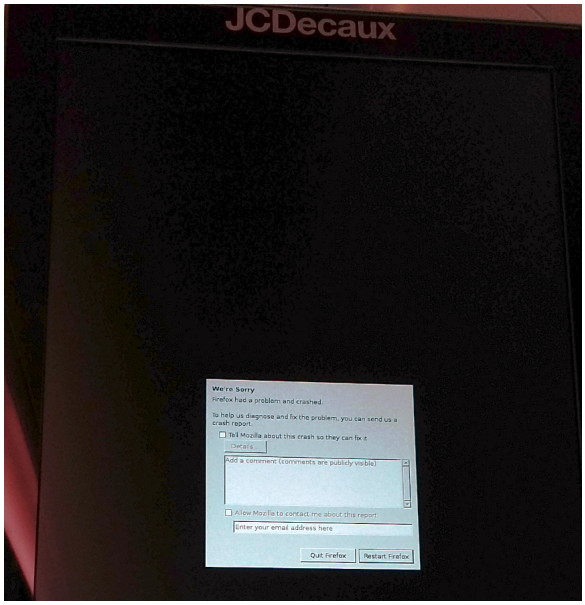

Figure 12: Firefox crash notification on Helsinki metro train display

5.3.3 Display Application Not in Kiosk Mode

The default mode of running display application is kiosk so that it covers the whole screen area. But sometimes it may be disabled, which is analysed by cloud server from the window display information. Cloud application compares the geometry of the display device and display application for their equality. The details on running Firefox and Google Chrome in kiosk mode on Linux-based operating system is explained in Appendix I.IV.

5.3.4 Display Application Not Active

This is the most common scenario experienced by the users. In this case a different application such as notification messages pops on top of the display application as seen in Figure 13 [84].
It can be detected by comparing the required application window-id with the currently running application window-id. These logs are also obtained from the windows displaying information.

For Linux, *xdotool* and *wmctrl* can be used to get the required windows displaying information. Appendix I.VI illustrates python code using module *wnck* explaining the method to get active window-id and controlling the window.

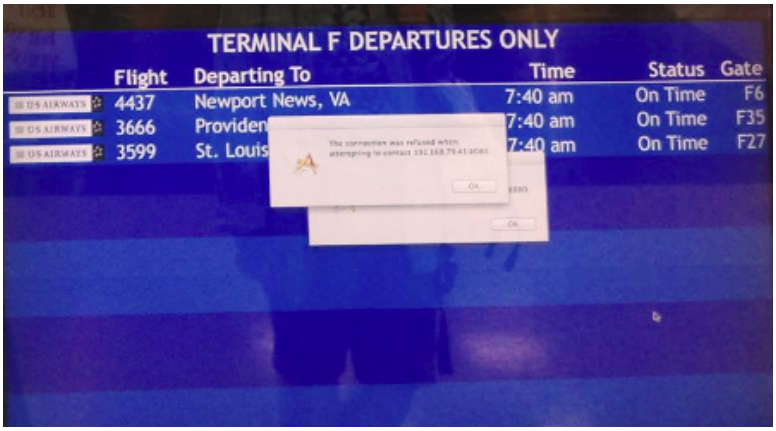

Figure 13: Screen fail notification message at DC-area airport

5.3.5 Display Application Not on Default URL

The digital signage solution works with a browser on a particular defined URL. If the display is not running at the same URL, the cloud server can detect it using the browser's session log files. For the browser Firefox, these log files are taken from the session store backup files. The server can then instruct display to restart the browser with appropriate URL.

5.3.6 Unnecessary Tab Opened on Display Application

It is also observed that in the browsers, unwanted tabs are opened and the required default-URL running-tab goes to the background. Even in this case the application is not working on the configured URL so it can be solved as the previous case.

5.3.7 Display Device Disconnected

This can happen when the HDMI or VGA cable connecting to the display device gets disconnected. This disconnection can happen physically or because of any other cable problems. The cloud server detects it from display controller communication information. As the resolution requires replacement or connecting the cable physically, cloud notifies the administrators about the problem using external channel such as email.

For Linux, *xrandr.log* can help in detecting the communication with the cable and so if the device connected.

5.3.8 Display Device Powered Off

In the case of any power fluctuation, sometime it happens that the display device doesn't get powered on automatically. If the display device is powered off and the power supply is connected, it can be powered on remotely by issuing command on display communication channel. This can be achieved by using various standard tools such as *DDCcontrol*.

5.3.9 Turning Off Display Device

The power socket of the display devices are physically protected. Our application provides support to turn off the display device instantly by using the cloud feature that sends a script to turn off the display device. The mechanism to turn off is similar to powering it on, i.e. setting the power parameter address of display device with a required value (on, off, standby).
Web user interface (UI) provide functionality to administrator to turn off the display device.

5.3.10 Changing Orientation

It is observed that the orientation of the application running can be wrong as seen from the Figure 14. It can be changed by the cloud by issuing appropriate instructions to display controller.
For testing on our setup, we have implemented scripts such as *configrotateleft.sh*, *configrotatenormal.sh*. Also, Web UI provides functionality to change orientation.

5.3.11 Changing Brightness

The cloud also provides support to change the brightness of the display devices. The administrator can select this functionality with defined value as its parameter.

For testing our setup, we have implemented a script *configbrightness.sh*.

5.3.12 Changing Audio for HDMI/VGA connected player

The cloud also provides support to change the audio of the display devices. The administrator can select this functionality with defined value as its parameter.

For testing our setup, we have implemented a script *configaudio.sh*.

Figure 14: Wrong orientation observed at Helsinki airport

Also, the architecture supports display devices debug mechanism. An administrator can send commands or shell scripts to execute them on display and get results. The debugging feature is also implemented on Web UI.

5.4 Testing on Linux

We have implemented the above mentioned evaluations using Python. The default display application used for testing is Mozilla Firefox. The display device is Fujitsu P27T-7 UHD connected with Intel x86 machine. This x86 machine is configured to run the proposed application as a start-up service and the cloud server application runs on Amazon AWS. This section describes the procedures required to setup a x86 or thin client machine running Linux.

5.4.1 Configurations for Linux

In this section, various procedures required to configure a Linux machine so as to enable and implement our proposed solution are mentioned.

5.4.1.1 Grub Configuration

The grub default configuration file located at */etc/default/grub* can be changed for the required actions as mentioned below:

- The parameter *GRUB_DEFAULT* with value zero means that the boot-loader would select the first menu entry to boot. To configure the default boot kernel from our proposed application, change the value of this parameter to *saved*. Then the command prompt can be used to change the default boot entry as following command:
  `sudo grub-set-default XX`
- The parameter *GRUB_DISABLE_RECOVERY="true"* can be uncommented to disable the generation of recovery mode menu entries.
- The parameter *GRUB_DISABLE_SUBMENU=y* can be added to disable generation of any sub-menu with advance options.
- The parameter *GRUB_DISTRIBUTOR="Main OS"* is used to rename the operating system with a desired one.
- To disable the memtest menu from the boot console use command:
  `chmod -x /etc/grub.d/20_memtest86+`
- To disable upstart menu, change the parameter *SUPPORTED_INITS* to *sysvinit*: */lib/sysvinit/init systemd*: */lib/systemd/system* by removing the upstart part in file */etc/grub.d/10_linux* .

And to make any changes to the grub active, it is necessary to update the configuration using:

```
sudo update-grub
```

5.4.1.2 Autorun Script on Boot

This section explains the settings that are necessary for auto starting an application on the system having systemd version of init. Let's assume, it is needed to execute python script named client.py, stored in the directory */home/Documents*. The procedure to make this script autorun as startup service is as follows:

- **Step 1**: Create a unit file at directory */etc/systemd/system/* named as *client.service* to instruct systemd about this new service. The following text should be written to the file:

```
[Unit]
Description= Python Script Service
After=syslog.target network.target

[Service]
Type=idle
ExecStart=/usr/bin/python /home/Documents/client.py
Restart=always

[Install]
WantedBy=multi-user.target
```

The structure of unit files is organized with sections denoted by a pair of square brackets. When systemd parses the file, section order does not matter to it. The first section found in the example unit file is the [Unit] section. This is used for defining metadata for the unit and configuring the relationship of the unit to other units. The directive *Description* describes the name and basic functionality of the unit. The directive *After* lists the units, required to be started before starting the current unit.
The [Service] section is used to provide configuration that is only applicable for the current service. The directive *Type* describes how to categorize the service by its process and daemonizing behavior. The Type is set to *idle* ensures that the command *ExecStart* only runs when everything else has loaded. The directive *ExecStart* specifies the full path and the arguments of the command to be executed to start the process. Finally, the directive *Restart* specify the circumstances under which systemd attempts to automatically restart the service. Systemd will trigger restart accordingly when the service was stopped.
The section *Install* is optional and used to define the behavior of the unit if it is enabled or disabled. The directive *WantedBy* allows to specify a dependency relationship. When the mentioned unit is enabled, a directory will be created within */etc/systemd/system* named after the specified unit with .wants appended to the end. This file creates a symbolic link to the current unit, stating dependency.
More details about other directives can be checked from [85]. In case, we need to store the output of this script, ExecStart should change as:

```
ExecStart=/usr/bin/python /home/Documents/client.py > /home/client.log 2>&1
```

- **Step 2**: Change the permission of service file to 644
- **Step 3**: Enable this service from systemd controller using commands:

```
sudo systemctl daemon-reload
sudo systemctl enable client.service
sudo systemctl start client.service
```

The status of the service can be checked using:

```
sudo systemctl status myscript.service
```

The list of all the active units can be checked using:

```
systemctl list-units --all --state=active
```

5.4.1.3 Autorun Application on Boot

We describe the procedure to start an application automatically by default in a Linux system. This can be done by creating unity launchers, that are files stored with a '.desktop' extension. The configuration file: */etc/xdg/autostart/firefox.desktop* can be created and written with the following lines:

```
[Desktop Entry]
Type=Application
Name=Mozilla
Exec=firefox http://example.com
OnlyShowIn=Unity;GNOME;
NoDisplay=true
StartupNotify=false
Terminal=false
```

The parameters used in the file are self explainable such as *Name* is the name of the application, *Type* specifies the type of the launcher file, and *Exec* is the path to the executable file. The type can be Application, Link or Directory. *Terminal* specifies whether the application can run in a terminal window or not. More details on various parameters can be checked at [86].

5.4.1.4 Enabling Automatic Login After Suspend

A greeter is a GUI that prompts the user for credentials. It is possible to use LightDM without a greeter, but only if an automatic login is configured [87]. The command used to disable the login prompt is

```
gsettings set org.gnome.desktop.lockdown disable-lock-screen 'true'
```

5.4.1.5 Enabling Automatic Login After Power On

The auto-login automates the client login process, this is done by creating a file named as *50-myconfig.conf* in directory */etc/lightdm/lightdm.conf.d/*. This file content should be:

```
[SeatDefaults]
autologin-user=<USERNAME>
```

5.4.1.6 Disabling DPMS

This is required to prevent screen from blanking after a certain time to save power for a digital signage solution. This can be achieved using

`gsettings set org.gnome.settings-daemon.plugins.power idle-dim false`

and

`gsettings set org.gnome.desktop.session idle-delay <seconds>` (0 to disable)

The screensaver can be disabled using:

`gsettings set org.gnome.desktop.screensaver idle-activation-enabled false`

5.4.2 System Setup

The following procedure is required to setup our application on the display system.

- **Step 1**: Install a Linux operating system in the display system and follow Section 5.4.1.1 to change their grub configurations.
- **Step 2**: Configure the operating system installed at a partition (say */dev/sda1*) to autorun the browser. Section 5.4.1.3 describes the procedure for configuring it.
- **Step 3**: Run the script *offservices.sh* in both the operating system to disable unnecessary active services. To disable the active services, the procedure is explained in Appendix I.II.
- **Step 4**: Configure the operating system with the actions mentioned in Sections 5.4.1.4, 5.4.1.5, and 5.4.1.6.
- **Step 5**: Download our application from [88] on the display machine and cloud.
- **Step 6**: Configure the application by editing *ClientConfig.py* at the display and *ServerConfig.py* at cloud server.
- **Step 7**: Deploy our application following the steps mentioned in Section 5.4.1.2.

5.5 Summary

In this chapter, we proposed the design of the application used to monitor and recover from the signage display failures and described the advantages of using such an approach. The application starts automatically as a system default service. This CMM server is categorized into various modules such as log collector, log analyser, storage manager, administrator controller, instruction generator and instruction notifier. The working of the implemented application is also described. The application also provides an interactive web application to control the display externally. The comparison of various log management tools for duplicates detection is also performed and the results are discussed.

The developed application is evaluated for different scenarios and tested on a Linux-based player. The resulting analysis and recovery from the failure for each evaluation is also discussed. Further, the procedures to configure a Linux-based player for enabling the proposed solution is explained.

6 Autonomous Recovery of Displays

This chapter describes the design of the proposed architecture to make displays autonomously recoverable. First, we will present the architecture overview. Secondly, the evaluation scenarios are specified. Finally, the testing procedures and setup done for a Linux system is described.

6.1 System Architecture

The display is enabled with a capability to recover itself from issues related to maintaining a connection with cloud. In case of such failures, the display tries to reboot itself as a first step to solve them. The proposed architecture is depicted in Figure 6. The display system is configured to have two partitions. The first partition is read-write partition, where our application is able to show contents on display and interact with the cloud server. In case the application is unable to maintain a connection with the cloud, it tries to recover autonomously. It would also try to reboot the display system to recover. The system would reboot only for a defined number of times. If rebooting doesn't recover from the problem, display system reboots to the second partition. The second partition is read-only recovery partition. Its action is to send debug logs of the first partition to cloud CMM server. This debug logs need to be analysed manually by the system administrator to diagnose the problem.
There are many benefits of using read-only root filesystem as discussed below:

- **Stability**: By stopping the installation and removal of any packages, the behavior of the system automatically becomes reproducible. This can in turn improve stability.
- **Privacy**: Any changes to data remain in volatile memory. The creation of a separation between the system and the user data with read-only configuration improves privacy.
- **Security**: An additional security measure can be used to check for malicious software.
- **User-control**: Any unintended changes can't be performed.

6.2 Evaluation Scenarios

6.2.1 Network Problems

The application installed on the display system periodically checks its network connectivity such as the IP address or detects for Ethernet cable disconnection.

If these cases occur, it tries to solve the failure internally. If the system fails to recover, it reboots itself with the same partition. The application maintains the reboot count. After few configured attempts of rebooting, the system turns off the display device and reboots to recovery partition. After booting in recovery partition, all the necessary logs of first partition are sent to the maintenance server.

6.2.2 Reboot Loops

Sometimes the operating system is stuck in an infinite reboot loop. This happens often due to a bad system component or hardware. With our application, this scenario can be detected only if the loop starts after the complete booting of the system. This constraint is because the monitoring application needs to maintain the complete reboot counter. After few configured attempts of reboot loop, system action is same as explained in the previous scenario.

6.3 Testing on Linux

6.3.1 Configurations for Read-Only Linux Root File System

The read-only root file system means no modification to the underlying file system is allowed. A *union* filesystem can be used to implement read-only file system, which takes an existing filesystem and transparently overlays it on a newer one. It allows files and directories of a separate filesystem to co-exist under a single roof. The implementation of the union file system is widely based on *aufs* and *overlayfs*. Therefore, in the following section, we will discuss the procedure for implementing read-only file system using *aufs* and *overlayfs*.

6.3.1.1 Using aufs

The procedure of using advanced multi-layered unification filesystem (aufs) is explored in this section. The following mentioned steps should be used:

- **Step 1**: The amount of available disk space for file systems and its layout before any changes can be seen from Figure 15:
- **Step 2**: Install aufs tool and configure it, run following commands:

```
sudo apt-get install aufs-tools
echo aufs >> sudo /etc/initramfs-tools/modules
```

- **Step 3**: Copy the script from [89] section "Script" to a file naming as __rootaufs at directory */etc/initramfs-tools/scripts/init-bottom/* and change its permission to 0755.

```
sultana1@ubuntu:~$ df
Filesystem     1K-blocks    Used Available Use% Mounted on
udev              476996       0    476996   0% /dev
tmpfs              99640    6372     93268   7% /run
/dev/sda1       19478204 3929212  14536512  22% /
tmpfs             498188     212    497976   1% /dev/shm
tmpfs               5120       4      5116   1% /run/lock
tmpfs             498188       0    498188   0% /sys/fs/cgroup
tmpfs              99640      92     99548   1% /run/user/1000
```

Figure 15: Disk space usage before changes

- **Step 4**: Update the parameter GRUB_CMDLINE_LINUX of the grub configuration file located at */etc/default/grub* as *GRUB_CMDLINE_LINUX="aufs=tmpfs"* and load the changes as:

```
update-grub
update-initramfs -u
```

- **Step 5**: Check the partition, */dev/sda1* is now read-only as seen from Figure 16

```
sultana1@ubuntu:~$ df
Filesystem     1K-blocks    Used Available Use% Mounted on
udev              476928       0    476928   0% /dev
tmpfs              99640    6412     93228   7% /run
/dev/sda1       19478204 3988352  14477372  22% /ro
aufs-tmpfs        498188   28788    469400   6% /rw
aufs              498188   28788    469400   6% /
tmpfs             498188     212    497976   1% /dev/shm
tmpfs               5120       4      5116   1% /run/lock
tmpfs             498188       0    498188   0% /sys/fs/cgroup
tmpfs              99640       4     99636   1% /run/user/108
tmpfs              99640      56     99584   1% /run/user/1000
```

Figure 16: Disk space usage after changes

6.3.1.2 Using Overlayfs

An overlay filesystem combines two filesystems: *upper* filesystem and *lower* filesystem. The lower filesystem is not writable and the upper filesystem is writable. Any modification to a file from the lower filesystem folder is done by creating its copy in the upper directory, and then modified in the upper one. This ensures no modification in the files of lower filesystem. Similarly, the removal of a file simulate itself

at lower filesystem by creating a "whiteout" file. When the overlayfs is dismounted, this state information will be lost [90]. The steps followed for using overlayfs to make root filesystem as read-only are as below:

- **Step 1**: Install overlay filesystem using the commands mentioned below:

```
sudo apt-get update && sudo apt-get dist-upgrade
sudo apt-add-repository ppa:cloud-initramfs-tools/ppa
sudo apt-get install -y overlayroot
```

- **Step 2**: Add a line *export overlayfs='yes'* to file */$HOME/.bashrc*
- **Step 3**: Configure overlayfs by assigning *overlayroot="tmpfs"* in the file */etc/overlayroot.conf* and reboot.
- **Step 4**: The system is now booted in text mode as read only filesystem and *etc/fstab* is updated by adding following line:

```
root@ubuntu:~# mount | grep overlay
overlayroot on / type overlay (rw,relatime,lowerdir=/media/root-ro,upperdir=/media/root-rw/overlay,w
orkdir=/media/root-rw/overlay-workdir/_)
```

Figure 17: fstab update

Sometimes errors can appear during installing overlayroot, following is one of the errors showed during the experiment:

```
E: /usr/share/initramfs-tools/hooks/fsprotect failed with return 1.
update-initramfs:  failed for /boot/initrd.img-4.4.0-66-generic with 1
```

To resolve this, use the following command:

```
sudo chmod -x /usr/share/initramfs-tools/hooks/fsprotect
```

6.3.2 System Setup

The procedure to enable this feature requires some additional work. It is explained using the steps mentioned below:

- **Step 1**: Install an another Linux operating system in the display system by creating a second partition.
- **Step 2**: Change grub configurations as mentioned in Section 5.4.1.1.

- **Step 3**: Boot the operating system on the second partition (say *dev/sdb1*) and add the following line to */etc/fstab* to mount the another operating system partition:
  /dev/sda1 /mnt ext4 defaults,errors=remount-ro 0 1
- **Step 4**: Run the script *offservices.sh* in both the operating system to disable unnecessary active services.
- **Step 5**: Configure the operating system of the second partition with the actions mentioned in Sections 5.4.1.4, 5.4.1.5, and 5.4.1.6.
- **Step 6**: Download our application from [88] on the display in the second partition and configure the application by editing *ClientConfig.py*.
- **Step 7**: Deploy our application following the steps mentioned in Section 5.4.1.2
- **Step 8**: Now make the operating system installed on second partition, a read-only root partition using the procedure mentioned in Section 6.3.1

6.4 Summary

In this chapter, we proposed the design of the application to make displays autonomously recoverable. The display system is configured with two partitions, read-write and read-only. The system is evaluated for network failure scenario and also verified to check if it is stuck in an infinite reboot loop. Further, the procedure to create a read-only partition in Linux-based system using aufs and overlayfs is described.

7 Conclusion and Future Work

Digital signage is gaining popularity due to its capability of displaying multimedia content. A digital display system encompasses many operations that include content creation, content delivery, display management and user interaction. In this thesis, our main topic of focus is display management operations. A typical management operation involves various steps such as monitoring the displays, analysing diagnostic reports, determining the actions to take in case of failures, and scheduling and performing recovery. A cloud-based solution architecture for remote monitoring and maintenance of a digital signage system has been proposed. The idea emerges from the failure behaviour of current digital signage systems. These failures could be related to process, application, network, or any system hardware. Traditionally, the recovery from such failures requires physical diagnosis by technicians making it a time-consuming and costly. A cloud-based digital signage can configure, manage and resolve the issues occurring in the displays remotely in real-time and increases the reliability of the system.

The proposed architecture is designed to be efficient and scalable and it can be used without changing any kernel modules of the system running digital signage application. The server receives diagnostic logs periodically from the displays. These logs are analysed for failure. If any failures are detected, appropriate instructions are sent to the display to resolve it via a bi-directional and secure WebSocket communication channel. This bi-directional channel helps administrator to send any external instruction to displays without waiting for its execution. The server also manages and save the logs optimally by removing the duplicate logs. The main advantages of the proposed architecture are that it supports real-time detection of any failures, remotely and autonomously resolve it with minimal manual interaction and handles all the known failures. The solution also includes autonomous recovery capability to resolve issues related to maintaining connection with the cloud. We have designed, implemented and evaluated the proposed architecture for a Linux-based display by connecting it to cloud with server application running on Amazon Web Service. The implementation can be found at the git repository [88].

Future work concerns deeper analysis of a particular operating system. One possible direction of future work could be to extend the proposed solution for supporting displays running on other operating systems such as Microsoft Windows, and Mac-OS. It is also interesting to consider different display managers used by various Linux-based operating systems e.g. *Mir*.

References

[1] Karthika S.; Lavanya T.; Gokila G.; Arunraja1 A.; Sarumathi S.; Saravana S.; Gokilavani A. Load balancing and maintaining the QoS on cloud partitioning for the public cloud. *International Journal of Innovative Research in Computer and Communication Engineering*, 2014, vol. 2, no. 2, pp 2872-2877.

[2] Alt F.; Kubitza T.; Bial D.; Zaidan F.; Ortel M.; Zurmaar B.; Lewen T.; Shirazi S.; Schmidt A. Digifieds: Insights into deploying digital public notice areas in the wild. *10th International Conference on Mobile and Ubiquitous Multimedia*, 2011, pp 165-174.

[3] Memarovic N.; Schieck A.; Schnadelbach H.; Kostopoulou E.; North S.; Ye L. Capture the Moment: "In the Wild" Longitudinal Case Study of Situated Snapshots Captured Through an Urban screen in a community setting. *18th ACM Conference on Computer Supported Cooperative Work and Social Computing*, 2015, pp 242-253.

[4] Memarovic N.; Fels S.; Anacleto J.; Calderon R.; Gobbo F.; Carroll J. Rethinking Third Places: Contemporary design with technology. *The Journal of Community Informatics*, 2014, vol. 10, no. 3, pp 1-13.

[5] Jose R.; Pinto H.; Silva B.; Melro A. Pins and posters: Paradigms for content publication on situated displays. *IEEE Computer Graphics and Applications*, 2013, vol. 33, no. 2, pp 64-72.

[6] Mccarthy J.; Farnham D.; Patel Y.; Ahuja S.; Norman D.; Hazlewood R.; Lind. J. Supporting community in third places with situated social software. *Fourth International Conference on Communities and technologies*, 2009, pp 225-234.

[7] Jones M.; Harwood W.; Bainbridge D.; Buchanan G.; Frohlich D.; Rachovides D.; Frank M.; Lalmas M. "Narrowcast yourself": designing for community storytelling in a rural Indian context. *7th ACM Conference on Designing interactive systems*, 2008, pp 369-378.

[8] Taylor N.; Cheverst K. Social interaction around a rural community photo display. *International Journal of Human-Computer Studies*, 2009, vol. 67, no. 12, pp 1037-1047.

[9] Digital signage market analysis by technology. `http://www.grandviewresearch.com/industry-analysis/digital-signage-market`. (checked 08.05.2017)

[10] Yackey B. High-brightness displays What it takes to take the screen outside. Digital Signage Today. 2008.

[11] Satyanarayanan M. Pervasive displays understanding the future of digital signage. Morgan and Claypool Publishers. 2014.

[12] BBC Big Screen -Wikipedia. `https://en.wikipedia.org/wiki/BBC_Big_Screen` (checked 18.05.2017)

[13] Memarovic N.; Elhart I.; Rubegni E. Developing a networked public display system. *IEEE Pervasive Computing*, 2016, vol. 15, no. 3, pp 32-39.

[14] Tatiraju V. Management of pervasive displays. School of Electrical Engineering. Aalto University. 2015. `https://aaltodoc.aalto.fi/handle/123456789/19180`

[15] Chen M.; Accardi A.; Kiciman E.; Lioyd J.; Patterson D.; Fox A.; Brewer E. Path-based failure and evolution management. *1st conference on Symposium on Networked Systems Design and Implementation*, 2004, pp 23-37.

[16] Alonso J.; Torres J.; Griffith R.; Kaiser G.; Silva L. Towards self-adaptable monitoring framework for self-healing. *Grid and Services Evolution*, 2008, pp 1-9.

[17] Gerhards R. The syslog protocol. Internet RFC 5424, 2009.

[18] Linux programmer's manual PROC(5). `http://man7.org/linux/man-pages/man5/proc.5.html` (checked 24.05.2017)

[19] Funika W.; Pegiel P.; Bubak M.; Kitowski J. Towards role-based self-healing in autonomous monitoring systems. *International Conference on Complex, Intelligent and Software Intensive Systems*, 2010, pp 1063-1068.

[20] Clinch S.; Davies N.; Friday A.; Efstratiou C. Reflections on the long-term use of an experimental digital signage system. *13th international conference on Ubiquitous computing*, 2011, pp 133-142.

[21] Ojala T.; Kostakos V.; Kukka H.; Heikkinen T.; Linden T.; Jurmu M.; Hosio S.; Kruger F.; Zanni D. Multipurpose interactive public displays in the wild: Three years later. *IEEE Computer*, 2012, vol. 45, no. 5, pp 42-48.

[22] Storz O.; Friday A.; Davies N.; Finney J.; Sas C.; Sheridan J. Public ubiquitous computing systems: Lessons from the eCampus display deployments. *IEEE Pervasive Computing*, 2006, vol. 5, no. 3, pp 40-47.

[23] Storz O.; Friday A.; Davies N. Supporting content scheduling on situated public displays. *Computers and Graphics*, 2006, vol. 30, no. 5, pp 681- 691.

[24] Ojala T.; Kukka H.; Linden T.; Heikkinen T.; Jurmu M.; Hosio S.; Kruger F. Ubi-hotspot 1.0: Large-scale long-term deployment of interactive public displays in a city center. *Fifth International Conference on Internet and Web Applications and Services*, 2010, pp 285-294.

[25] Mikusz M. Building and using an application store to support public display users. University of Stuttgart. 2014. `https://elib.uni-stuttgart.de/handle/11682/3274`

[26] Sethi M.; Oat E.; Francesco M.; Aura T. Secure bootstrapping of cloud-managed ubiquitous displays. *ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2014, pp 739-750.

[27] Ogi T.; Ito K.; Nakada G. Healthcare digital signage using Gamification method. *18th International Conference on Network-Based Information Systems*, 2015, pp 511-516.

[28] Lee W.; Lee H.; Lee Y.; Yu D.; Lee H; Choi M. Digital signage system using virtualization technology. *International Symposium on Consumer Electronics*, 2014, pp 1-2.

[29] Nguyen V.; Park Y.; Kim Y. A design of digital signage system using OpenStack. *Annual Conference of the Korean Institute of Communication Sciences*, 2016, pp 1112-1113.

[30] Mishima K.; Sakurada T.; Hagiwara Y. Cost effective digital signage system using low cost information device. *13th IEEE Annual Consumer Communications and Networking Conference*, 2016.

[31] Digital signage federation-Digital signage privacy standards. `http://www.digitalsignagefederation.org/wp-content/uploads/2017/02/DSF-Digital-Signage-Privacy-Standards-02-2011-3.pdf` (checked 15.06.2017)

[32] Nagios The Industry Standard In IT Infrastructure Monitoring. `https://www.nagios.org/`. (checked 24.05.2017)

[33] Ganglia monitoring system. `http://ganglia.info/`. (checked 24.05.2017)

[34] Brandt J.; Gentile A.; Hale D.;Pebay P. OVIS: A tool for intelligent, real-time monitoring of computational clusters. *20th International Parallel and Distributed Processing Symposium*, 2006.

[35] Oracle remote diagnostics agent. http://www.oracle.com/us/support/software/premier/remote-diagnostics-agent-068525.html. (checked 10.05.2017)

[36] Heikkinen T.; Linden T.; Ojala T.; Kukka H.; Jurmu M.; Hosio S. Lessons learned from the deployment and maintenance of UBI-hotspots. *4th International Conference on Multimedia and Ubiquitous Engineering*, 2010, pp 1-6.

[37] Cheung S.; Cheung K.; Suen H. CSHM: Web-based safety and health monitoring system for construction management. *Journal of Safety Research*, 2004, vol. 35, no. 2, pp 159-170.

[38] Zhang G.; Tong X.; Zou S.; Jiang X.; Wu G. A novel insulation on-line monitoring and fault diagnosis system used for traction substation *IEEE International Symposium on Electrical Insulation*, 2002, pp 199-202.

[39] Sun L.; Nilsson D. Online self-healing support for embedded systems. *IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, 2009, pp 283-287.

[40] Elnozahy E.; Alvisi L.; Wang Y.; Johnson D. A survey of rollback-recovery protocols in message-passing systems. *ACM Computer Surveys*, 2002, vol. 34, no. 3, pp 375-408.

[41] Iyer R.; Kalbarczyk Z.; Pattabiraman K.; Healey W.; Hwu W.; Klemperer P.; Farivar R. Toward application-aware security and reliability. *IEEE Security and Privacy*, 2007, vol. 5, no. 1, pp 57-62.

[42] Gillen A.; Kusnetzky D.; McLaron S. The role of Linux in reducing the cost of enterprise computing. IDC white paper, Red Hat Inc. 2002.

[43] IEEE Standard 1413.1-2002. IEEE guide for selecting and using reliability predictions based on IEEE 1413. IEEE Standard, 2003

[44] Herder J.; Bos H.; Gras B.; Homburg P.; Tanenbaum A. Reorganizing UNIX for reliability. *Asia-Pacific Conference on Advances in Computer Systems Architecture*, 2006, pp 81-94.

[45] Swift M. Improving the reliability of commodity operating systems. University of Washington, 2005. http://pages.cs.wisc.edu/~swift/papers/thesis.pdf

[46] Kurobane N. Rapidly growing Linux OS: Features and reliability *Fujitsu Sci. Tech. J.*, 2005, vol. 41, no. 3, pp 318-322.

[47] Shapiro M. Self-Healing in modern operating systems. *ACM Magazine Queue - Programming Languages*, 2004, vol. 2, no. 9, pp 66-75.

[48] Corsava S.; Getov V. Intelligent architecture for automatic resource allocation in computer clusters. *17th International Symposium on Parallel and Distributed Processing*, 2003.

[49] Prasad S.; Kumar C. A green and reliable Internet of Things. *Communications and Network*, 2013, vol. 5, no. 1, pp 44-48.

[50] Angarita R. Responsible Objects: Towards self-healing Internet of Things applications. *Autonomic Computing (ICAC)*, 2015.

[51] Ahmed K.; Gregory M. Integrating wireless sensor networks with cloud computing. *Seventh International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, 2011, pp 364-366.

[52] Chenaru O.; Stamatescu G.; Stamatescu I.; Popescu D. Towards cloud integration for industrial wireless sensor network systems. *9th International Symposium on Advanced Topics in Electrical Engineering*, 2015, pp 917-922.

[53] Huo Z.; Mukherjee M.; Shu L.; Chen Y.; Zhou Z. Cloud-based data-intensive framework towards fault diagnosis in large-scale petrochemical plants. *Wireless Communications and Mobile Computing Conference*, 2016, pp 1080-1086.

[54] Luo S.; Reb B. The monitoring and managing application of cloud computing based on Internet of Things. *Computer Methods and Programs in Biomedicine*, 2016, vol. 130, pp 154-161.

[55] Ray P. Internet of Things cloud-based smart monitoring of air borne PM2.5 density level. *International conference on Signal Processing, Communication, Power and Embedded System*, 2016

[56] Lei H.; Xing T.; Zhou X. Monitoring travel time reliability from the cloud: Cloud computing based architecture for advanced traffic information dissemination. *90th Annual Meeting of the Transportation Research Board*, 2010, pp. 35-43.

[57] Yu R.; Zhang Y.; Gjessing S.; Xia W.; Yang K. Toward cloud-based vehicular networks with efficient resource management. *IEEE Network*, 2013, vol. 27, no. 5, pp 48-55.

[58] Digital signage. `https://en.wikipedia.org/wiki/Digital_signage`. (checked 08.05.2017)

[59] Digital signage: the right information in all the right places. 2011. https://www.w3.org/2012/06/signage/statements/Technology_Watch_Report_November_2011-Digital_signage.pdf. (checked 26.05.2017)

[60] Sethi M.; Lijding M.; Francesco M.; Aura T. Flexible management of cloud-connected digital signage. *IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops*. 2015, pp 205-212.

[61] Linden T.; Heikkinen T.; Kostakos V.; Ferreira D.; Ojala T. Towards multi-application public interactive displays. *Intl. Symposium on Pervasive Displays*, 2012, pp 1-9.

[62] Digital signage on Ubuntu. https://www.ubuntu.com/internet-of-things/digital-signage. (checked 26.05.2017)

[63] Kim T. Wi-Fi Direct based pipelining contents distribution scheme for digital signage network. Chung-Ang University. 2014. http://uc.cse.cau.ac.kr/publications/TJ_thesis.pdf

[64] Anu; Bala A. Scrutinize : Fault monitoring for preventing system failure in cloud computing. *International Journal of Innovations and Advancement in Computer Science*, 2015, pp 211-219.

[65] Tanenbaum A.; Herder J.; Can H. We make operating systems reliable and secure?. *Journal Computer*, 2006, vol. 39, no. 5, pp 44-51.

[66] Administration guide | SUSE Linux enterprise server 12 SP2. 2017. https://www.suse.com/documentation/sles-12/singlehtml/book_sle_admin/book_sle_admin.html. (checked 26.05.2017)

[67] Jones M. Kernel logging: APIs and implementation. 2010. http://www.ibm.com/developerworks/library/l-kernel-logging-apis. (checked 08.05.2017)

[68] Melamut J. Ubuntu Debugging. http://odm.ubuntu.com/uhs/2011/Ubuntu-Debugging_A5-booklet.pdf. (checked 08.05.2017)

[69] Anicas M. 5 Ways to improve your production Web application server setup. https://www.digitalocean.com/community/tutorials/5-ways-to-improve-your-production-web-application-server-setup . (checked 08.05.2017)

[70] Types of Data Backup. https://www.handybackup.net/backup_articles/backup-type.shtml. (checked 08.05.2017)

[71] Drad. Find Duplicate Files - A Comparison of fdupes and fslint, 2012. http://www.adercon.com/ac/node/56 . (checked 08.05.2017)

[72] Gray J.; Siewiorek D. High-availability computer systems. *IEEE Computer*, 1991, pp 39-48.

[73] Cristian F. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 1991, vol. 34, no. 2, pp 56-78.

[74] Liu C.; Zhang Y.; Wang H. Linux-based remote monitoring and control of CNC machine tool of SQLite. *International Conference on Electronic and Mechanical Engineering and Information Technology*, 2011, pp 09-12.

[75] How to set NOMODESET and other kernel boot options in grub2. https://ubuntuforums.org/showthread.php?t=1613132. (checked 08.05.2017)

[76] Boichat N. DDCcontrol documentation, 2006. http://ddccontrol.sourceforge.net/doc/ddccontrol-0.4.pdf. (checked 08.05.2017)

[77] Hard Disk and Solid State Drive Problems and Their Symptoms. http://scsc-online.com/DriveProblems.html. (checked 08.05.2017)

[78] Using dmesg for disk troubleshooting. http://www-01.ibm.com/support/docview.wss?uid=swg21387697. (checked 08.05.2017)

[79] Litt S. Troubleshooting Linux Sound. 2013. http://www.troubleshooters.com/linux/sound/sound_troubleshooting.htm. (checked 08.05.2017)

[80] Rankin K. DevOps Troubleshooting: Linux Server Best Practices. Addison-Wesley. 2012.

[81] Rouse M. device driver, 2010. http://searchenterprisedesktop.techtargetcom/definition/device-driver. (checked 08.05.2017)

[82] Le M.; Gallagher A.; Tamir Y.; Turner Y. Maintaining network QoS across NIC device driver failures using virtualization. *Network Computing and Applications*, 2009, pp 195-202.

[83] Fette I.; Melnikov A. Rfc 6455: The websocket protocol. In: IETF, December 2011.

[84] https://s-media-cache-ak0.pinimg.com/originals/92/96/bb/9296bbd00c5cbd46df14aca42a6e6f14.jpg. (checked 08.05.2017)

[85] Ellingwood J. Understanding systemd units and unit files, 2017. https://www.digitalocean.com/community/tutorials/understanding-systemd-units-and-unit-files. (checked 08.05.2017)

[86] Brown P.; Blandford J.; Taylor O.; Untz V.; Bastian W.; Lortie A.; Faure D. Desktop Entry Specification. 2015. https://standards.freedesktop.org/desktop-entry-spec/desktop-entry-spec-latest.html. (checked 08.05.2017)

[87] LightDM. https://wiki.archlinux.org/index.php/LightDM. (checked 08.05.2017)

[88] https://github.com/ashishsultania/sample.git. (checked 08.05.2017)

[89] rootaufs Script. https://help.ubuntu.com/community/aufsRootFileSystemOnUsbFlash#rootaufs_Script. (checked 08.05.2017)

[90] Denholm T. Explaining OverlayFS - What it Does and How it Works. https://www.datalight.com/blog/2016/01/27/explaining-overlayfs-%E2%80%93-what-it-does-and-how-it-works/. (checked 08.05.2017)

[91] LinuxLogFiles - Community Help Wiki. https://help.ubuntu.com/community/LinuxLogFiles. (checked 08.05.2017)

[92] http://stackoverflow.com/questions/18167299/how-can-i-import-gtk-gdk-from-gi-repository. (checked 08.05.2017)

[93] http://www.pbx-brasil.com/Pesquisa/Ferramentas/programacao_python/aula00/scripts/filme/NaoFuncionam/screengrab.py. (checked 08.05.2017)

Appendix

I. Glossary

I.I DDCcontrol Usage

The list of available controls can be checked using *ddccontrol -p*. This command probes all the available monitors and list controls supported by the first probed one. The output of the command provides monitor name, color settings, audio setting, power control parameters, etc. This can be seen as follow:

ddccontrol -p

```
ddccontrol version 0.4.2
Copyright 2004-2005 Oleg I. Vdovikin oleg@cs.msu.su
Copyright 2004-2006 Nicolas Boichat nicolas@boichat.ch
This program comes with ABSOLUTELY NO WARRANTY.
You may redistribute copies of this program under the terms
of the GNU General Public License.

Probing for available monitors..........
Detected monitors :
 - Device: dev:/dev/i2c-0
   DDC/CI supported: Yes
   Monitor Name: VESA standard monitor
   Input type: Digital
  Automatically selected
Reading EDID and initializing DDC/CI at bus dev:/dev/i2c-0...

EDID readings:
        Plug and Play ID: FUS080F [VESA standard monitor]
        Input type: Digital

= VESA standard monitor
> Color settings
        > Brightness and Contrast
                > id=brightness, name=Brightness, address=0x10, delay=-1ms,
          type=0 supported, value=100, maximum=100
                > id=contrast, name=Contrast, address=0x12, delay=-1ms,
          type=0 supported, value=50, maximum=100
        > Color maximum level
                > id=red, name=Red maximum level, address=0x16, delay=-1ms,
          type=0 supported, value=100, maximum=100
                > id=green, name=Green maximum level, address=0x18, delay=-1ms,
          type=0 supported, value=100, maximum=100
                > id=blue, name=Blue maximum level, address=0x1a, delay=-1ms,
          type=0 supported, value=100, maximum=100
```

```
> Others
        > Restore defaults
              > id=defaults, name=Restore Factory Defaults, address=0x4,
          delay=2000ms, type=1
                Possible values:
                      > id=default - name=Restore Factory Defaults, value=1
                supported, value=0, maximum=1
              > id=defaultluma, name=Restore Brightness and Contrast,
          address=0x5, delay=2000ms, type=1
                Possible values:
                      > id=default - name=Restore Brightness and Contrast,
              value=1 supported, value=0, maximum=1
              > id=defaultcolor, name=Restore Factory Default Color,
          address=0x8, delay=2000ms, type=1
                Possible values:
                      > id=default - name=Restore Factory Default Color,
              value=1 supported, value=0, maximum=1
        > Audio
              > id=audiospeakervolume, name=Audio Speaker Volume Adjust,
          address=0x62, delay=-1ms, type=0
                supported, value=39, maximum=100
        > Degauss
              > id=degauss, name=Degauss, address=0, delay=-1ms, type=1
                Possible values:
                      > id=default - name=Degauss, value=1
                supported, value=0, maximum=255
        > OSD
              > id=osdorientation, name=OSD Orientation, address=0xaa,
          delay=-1ms, type=2
                Possible values:
                      > id=landscape - name=Landscape, value=1
                      > id=portrait - name=Portrait, value=2
                supported, value=1, maximum=2
        > Input settings
              > id=inputsource, name=Input Source Select, address=0x60,
          delay=-1ms, type=2
                Possible values:
                      > id=analog - name=Analog, value=1
                      > id=digital - name=Digital, value=3
                supported, value=3, maximum=15
        > Power control
              > id=dpms, name=DPMS Control, address=0xd6, delay=-1ms,
          type=2
                Possible values:
                      > id=on - name=On, value=1
                      > id=standby - name=Standby, value=4
                supported, value=1, maximum=5
```

To change any parameters, we have to check the address and range of the parameters. For example, to change audio parameter of the device, check address which is 0x62 and maximum supported value is 100. Its value can be changed to 50 using command:

```
ddccontrol -p -r 0x62 -w 50
```

which will provide output as

```
Writing 0x62, 0x3250...
Control 0x62: +/50/100 C [Audio Speaker Volume Adjust]
```

I.II Systemd

Systemd starts up and supervises entire system based on units. Units are categorized by the type of resources they represent. Units are defined with files known as unit files. Unit files primarily serve as a collection of meta information, which have name and type. The various types of unit are as follows [85]:

1. ***service***: It describes how to manage a service or application on the server.
2. ***socket***: It describes a network or IPC socket to allow communication paths to processes. It always has an associated .service file, that starts when an activity is seen on the socket with defined unit.
3. ***device***: It describes a device that has been designated as needing systemd management by udev or sysfs filesystem.
4. ***mount***: It defines a mount point on the system to be managed by systemd. Entries within */etc/fstab* can have units created automatically.
5. ***automount***: It configures a mount point that requires automount a file system based on a request for a file or directory within that file system.
6. ***target***: A target unit is used to provide synchronization points for other units when booting up or changing states. It represents the state of the system at any one time. Each target has a name instead of a number. For example, multi-user.target instead of runlevel 3 or reboot.target instead of runlevel 6.
7. ***snapshot***: It allows to take snapshots of the current state of the system. It is created automatically by the command *systemctl snapshot*.
8. ***path***: It checks the existence of files or directories, and create them as needed.
9. ***swaps***: It describes the swap space on the system.
10. ***timers***: It defines a timer managed by systemd, similar to a cron job for delayed or scheduled activation.
11. ***slices***: It is associated with control group nodes, allowing resources (such as CPU and memory) to be restricted or assigned to any processes associated

with the slice.

During boot time, systemd renames itself to init. This can be verified by checking */sbin/init* file, which has a symbolic link to */lib/systemd/systemd*. The custom created unit files and modified unit files reside at */etc/systemd/system*. If a unit file with the same name exists in the locations (/lib/ and /etc/), systemd uses the one under */etc*. If a service or target is enabled to start at boot time, a symbolic link is created for that service unit file under appropriate directories in */etc/systemd/system*. Unit files under */etc/systemd/system* are actually symbolic links to the files with same name under */lib/systemd/system*.
Systemd creates the listening sockets before the actually starting the daemons, and then just pass the sockets during exec() to run them in parallel. If in case a service needs another service, which is not fully up at that time, then the connection is queued in providing services and the client will potentially block on that single request.
Control groups (cgroup) provide a means of tracking the processes belong to a given service. If a process belonging to a specific cgroup forks, its children will also become a member of the same group. All the components of the service have a tag to make sure that all of these components start and stop properly. The restriction on these cgroups can be imposed in various ways, I/O bandwidth, memory, CPU usage, etc.
Systemd also provides the concept of journal which collects, stores, and processes logging data. It creates and maintains binary files based on information received from the kernel, user processes, standard error output, etc. By default, it stores log files in memory or in a volatile directory */run/log/journal/* as a small ring buffer. Logs can be retrieved using APIs (journalctl, systemdctl).

The list of active services in an Linux-based systems can be obtained from the command

```
systemctl list-units | grep .service | grep running
```

Some of the services can be disabled using the command:

```
sudo systemctl disable service_name.service
```

A disabled service can be started by another service. So, to forcibly disable a service without un-installation, it needs to be masked. The command to mask it is

```
sudo systemctl mask service_name.service
```

This will create a symlink to */dev/null*. The boot time taken by the kernel and userspace can be checked using the commant *systemd-analyze*

I.III Linux-based File System

There are two categories of files, namely user files and system files. In this section, we are focusing on system files that control the system operation. This includes system startup scripts, different configuration files, system log files, etc. All the file systems, e.g. ext3, ext4, tmpfs, and zfs are implemented in the kernel. When a program wants to read from a file, it issues various system library calls. These calls ultimately end up in the kernel having form of open(), read(), close() sequence. There are many directories in the Linux system summarizes as:

(a) ***/proc***: It contains runtime system information such as system memory, mounted devices, and hardware configuration. The filesystem mounted on /proc is called *procfs*. The files in this directory are of zero sizes, as they act as pointer to the actual process where information reside. The length calculation of each file on every directory read would potentially be expensive. Therefore, the zero byte size is an optimization on the kernel side as many of the files are dynamic and can easily vary in length. Since, this is a virtual file system which resides in memory, new /proc file system is created on every reboots.

(b) ***/bin***: It contains useful command binaries.

(c) ***/boot***: It has the static files required for booting i.e. kernel, bootloader, and other data used before the kernel begins executing user-mode programs.

(d) ***/etc***: It holds all the system configuration files.

(e) ***/dev***: It contains all the files that represent peripheral devices such as HDD, CD-ROM, printers to provide low-level access to these devices.

(f) ***/tmp***: It contains temporary files often created by user programs and got flushed out when system get restarted.

(g) ***/opt***: It is reserved for add-on packages.

(h) ***/usr***: It contains all the user binaries, their documentation, libraries, header files, etc.

(i) ***/var***: It has variable data such as system logging files, and temporary files.

The directories such as */home* and */tmp*, are generally set aside for user files.

Linux system logs its activities, that can be used for troubleshooting the system and application issues. These log files are typically saved as an ASCII text mode in directory */var/log*. Some of the logs are discussed as below [91]:

1. ***System Logs***: It contains Linux system functions related logs which include:

   (a) ***Authorization Log***: It tracks the usage of authorization systems such as sudo command, PAM system, and sshd. The log file can be accessed from */var/log/auth.log*.

   (b) ***Debug log***: It provides debug messages of the system and applications that write to syslogd at the DEBUG level. It can accessed from */var/log/debug*.

   (c) ***Daemon Log***: It contains information about the running system and application daemons e.g. Gnome display manager daemon. It can accessed from */var/log/daemon.log*.

   (d) ***Kernel Log***: It can be accessed from */var/log/kern.log* and provides messages from the kernel. These messages may be useful for troubleshooting a new or custom-built kernel.

   (e) ***Kernel Ring Buffer***: The system initialization script `/etc/init.d/bootmisc.sh` saves the bootup messages in ring buffer. The parameter CONFIG_LOG_BUF_SHIFT in the */boot/config-x.x.x-xx-generic* file can be changed to modify the buffer size.

2. ***Application Logs***: All the installed applications generate logs in */var/log* with a familiar name. For example, X11 server logs messages are stored in the file */var/log/Xorg.0.log*.

3. ***Non-Human-Readable Logs***: There are some log files in */var/log* which are readable by applications such as:

   (a) ***Login Failures Log***: Any failures in login are logged into files at */var/log/faillog*.

   (b) ***Last Logins Log***: The last logins log at */var/log/lastlog* used in conjunction with the lastlog command.

   (c) ***Login Records Log***: The file */var/log/wtmp* contains the list of recent login records and can be used by utilities e.g. command *who* to present the list of currently logged in users.

There are some logs which are automatically archived and renamed itself after a predefined time-frame. The purpose of this is to compress old logs to save disk space, and available for inspection as needed.

I.IV Kiosk Mode

Kiosk Mode is offered by browser applications to run in full screen, without any browser user interface such as toolbars and menus. Applications such as Firefox don't provide any command line parameter. Therefore, to enable it in Kiosk mode, one of the following commands can be:

- `xdotool key F11`
- `xdotool search --sync --onlyvisible --class "Firefox" windowactivate key F11`
- `wmctrl -r mozilla -b add,fullscreen`

The browser, *Google Chrome* provides parameter *kiosk* to enable kiosk mode. The following command can be executed to run it in kiosk mode:

```
google-chrome --kiosk http://example.com
```

I.V Screeenshots

The screenshots of the displayed contents can be obtained using various method by using different modules of python. The use of these modules are presented in following subsections.

Method 1

It use gtk module which provide object of root windows. Using this object user can obtain window size, pixel buffer, etc. The implementation can be checked as below [92]

```
import gtk.gdk

dr_win = gtk.gdk.get_default_root_window()
size = dr_win.get_size()
print "The size of the window is %d x %d" % size
pb = gtk.gdk.Pixbuf(gtk.gdk.COLORSPACE_RGB, False, 8, size[0], size[1])
pb = pb.get_from_drawable(dr_win, dr_win.get_colormap(),
                          0, 0, 0, 0, size[0], size[1])
if (pb != None):
    pb.save("screenshot.png","png")
```

Method 2

Here we use *wx* module to get the screenshot. The implementation can be checked as below [93]:

```
import sys
import os
import wx
from PIL import Image

# Create an App instance
wx.App()

screen  = wx.ScreenDC()
size    = screen.GetSize()
bmp     = wx.EmptyBitmap(size[0], size[1])
mem     = wx.MemoryDC(bmp)

custom_width  = 200
custom_height = 200

mem.Blit(0, 0, size[0], size[1], screen, 0, 0)

# Release bitmap
del mem

#Save the screenshot as png image
bmp.SaveFile('screenshot.png', wx.BITMAP_TYPE_PNG)

#Get custome size image
PilImage    = Image.new('RGB', custom_width, custom_height )
PilImage.fromstring( myWxImage.GetData() )
PilImage.show()
```

Method 3

The module *ImageGrab* is simple to use to get the screen dump.

```
import ImageGrab
img   = ImageGrab.grab()
return img.show()
```

Method 4

Here, modules Xlib and PIL are used.

```
from Xlib import display, X
from PIL import Image

custom_width  = 200
custom_height = 200

root    = display.Display().screen().root
raw     = root.get_image(0, 0,
            custom_width,
            custom_height,
            X.ZPixmap,
            0xffffffff)

image = Image.frombytes("RGB",
            (custom_width, custom_height),
            raw.data,
            "raw",
            "BGRX")

image.show()
```

Method 5

The module *autopy* used to get bitmap which can be saved as image.

```
import autopy
screen = autopy.bitmap.capture_screen()
screen.save('screenshot.png')
```

Method 6

This method uses PyQt5.

```
from PyQt5.QtWidgets import QApplication

app         = QApplication([])
screen      = app.primaryScreen()
screenshot  = screen.grabWindow(QApplication.desktop().winId())
screenshot.save('screenshot.png')
```

I.VI Control Windows

Python also provides modules to get the active window id, name and ability to operate on these windows. The code can be checked as below. The problem of this code to implement in our solution is that the obtained window id remains as local variable and changes on every call of function *wnck.screen_get_default()*. It is therefore difficult to instruct from server to operate remotely in different thread.

```
import logging
import sys
import wnck
import pygtk
import gtk


def get_active_window():
    active_window_name = None
    screen = wnck.screen_get_default()
    screen.force_update()
    window_list = screen.get_windows()

    #Get all the window name
    if len(window_list) == 0:
        print "No Windows Found"
    for win in window_list:
        print win,win.get_name()

    window = screen.get_active_window()
    if window is not None:
        pid = window.get_pid()
        print pid
        with open("/proc/{pid}/cmdline".format(pid=pid)) as f:
            active_window_name = f.read()
    return active_window_name

print("Active window: %s" % str(get_active_window()))

#To minimize any window:
for w in window_list:
    w.minimize()
    #w.close(0) #It will close window
```

Non-exclusive licence to reproduce thesis and make thesis public

I, **Ashish Kumar Sultania**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1 reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

   of my thesis **Monitoring and Failure Recovery of Cloud-Managed Digital Signage** supervised by Assoc. Prof. Satish Narayana Srirama and Prof. Danilo Gligoroski, and advised by Prof. Tuomas Aura.

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 01.08.2017