Improving the Startup Time of Digital TV

Heeseung Jo[*] Hwanju Kim[†] Hyun-Gul Roh[‡]
KAIST KAIST KAIST

Joonwon Lee[§]
Sungkyunkwan University

Seungryoul Maeng[**]
KAIST

CS/TR-2009-310

June 29, 2009

K A I S T
Department of Computer Science

[*] heesn@camars.kaist.ac.kr
[†] hjukim@camars.kaist.ac.kr
[‡] hgroh@camars.kaist.ac.kr
[§] joonwon@skku.edu
[**] maeng@camars.kaist.ac.kr

Improving the Startup Time of Digital TV

Heeseung Jo, Hwanju Kim, Hyun-Gul Roh, Joonwon Lee, and Seungryoul Maeng

Abstract — *Digital TV was introduced with promises for higher quality displays and converging internet services which require more computing power and complex software systems. Such complexity entails a prolonged startup time that is similar to booting a computer system.*

In this paper, we propose two approaches to reduce the startup time of digital TV. Based on the suspend-resume technique employed in many commodity operating systems, our mechanisms replace the cold startup process of digital TV with the resuming from a suspend image. We also analyze the bottlenecks in the startup process and suggest hardware supports to improve the startup time. The proposed approaches reduce the startup time of digital TV by about 50%.[1]

Index Terms — Digital television (DTV), booting time, startup time, suspend-resume

I. INTRODUCTION

Digital television (DTV), introduced in the late 1990s, has inaugurated a new era of services rendered by televisions. It boosts several business sectors including consumer electronics industries by appealing end users with high definitions of displays. According to 2006-2007 Report on Chinese Digital TV Industry, China, one of the largest consumer market, has 14.5 million DTV users in 2007, and the rate of growth climbed to 308.2% from 2005 to 2007 [10]. According to a DTV manufacturer, the compound annual growth rate of DTV market is 18%, and this rate will increase [11]. Moreover, analog TV broadcasting will be terminated in the US by February 17th, 2009 [12].

DTV boasts higher-quality images, sound, and more programming choices. It also allows services such as multiplexing (more than one program on the same channel), electronic program guides (EPG), and additional languages, spoken or subtitled [1]. Also many internet services are expected to be supported on a DTV, which eventually generates a new market.

For such high quality media and new services, DTV requires much more hardware and software support than analog TV. For example, the parsing and rendering of EPG information, which is sent from a broadcasting station, is under the responsibility of DTV software. The DTV software system usually requires a boot loader and an operating system (OS) for booting and management of hardware as a traditional embedded system does. For the simplicity of design, each service of DTV is usually implemented as a thread resulting hundreds of threads running concurrently.

Among many issues introduced by the complexity of DTV software, long startup time and channel switching delay are considered the most serious ones for user acceptance. The delay in changing channels, which is about one second, makes channel surfing almost intolerable. Also the long startup, which takes about 6-15 seconds, will alienate impatient users.

This paper describes mechanisms to improve the startup time of a commercial DTV system. The startup procedure includes booting OS, initializing hardware devices, and launching DTV application until initial menu shows up. To enhance the startup time of the DTV system, we adopt the software based approaches by using the *snapshot boot* technique [3]. Based on the *suspend-resume* technique, it snapshots a running system image including CPU registers and memory state, and resumes the system with the suspend image instead of booting. The snapshot boot technique is ideal for DTV since the state after system booting is hardly changed once it is deployed.

Applying the snapshot boot technique, however, is not always easy for all embedded systems. In the case of DTV, high complexity of hardware makes it very difficult to adopt the snapshot boot technique. Also reading a snapshot from non-volatile storage and initializing hundreds registers takes almost the same time as normal startup.

Our work focuses on how to optimize the suspend-resume technique to a complex DTV system. We also propose two approaches to improve the startup time of DTV. Our work provides the following contributions.

1. DTV experience: We adopt the suspend-resume and the snapshot boot techniques to a commercial DTV system. To our knowledge, this is the first attempt on a DTV system, and we believe that our experiences will be useful for the large and complex embedded consumer electronics such as DTV, PVR, mobile handset, and so forth.

2. Suspend-resume optimization: Applying the native suspend-resume technique to DTV is not much beneficial for the startup time due to the specific device drivers and the long loading time of the DTV application. Especially, we analyze which points are crucial for resuming time. We propose the techniques to reduce these overheads.

[1] This paper is supported by KOSEF grant funded by the Korea government (MOST) (No. R01-2006-000-10724-0) and by the MKE (Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Advancement) IITA-2008-C1090-0801-0020.

Heeseung Jo, Hwanju Kim, Hyun-Gul Roh, and Seungryoul Maeng are with the Korea Advanced Institute of Science and Technology (KAIST) (e-mail: heesn@camars.kaist.ac.kr, hjukim@camars.kaist.ac.kr, hgroh@camars.kaist. ac.kr, maeng@kaist.ac.kr).

Joonwon Lee (Corresponding) is with the SungKyunKwan University (e-mail: joonwon@skku.edu).

3. Application level suspend-resume: Besides suspending the state of a booted system in the previous work, our approach tries to suspend the DTV system when the DTV application is running, and we analyze the gains and the losses.

The rest of this paper is organized as follows. The next section summarizes the mechanism of the suspend-resume technique and the related work. Section 3 describes the DTV system and environments used in this research. Section 4 and 5 illustrate the design and detailed mechanism of our approaches; Raw Restoration and Warm Restoration, respectively. Section 6 presents and analyzes the evaluation results. Finally, we conclude in Section 7.

II. Background and Related Work

This section describes the suspend-resume mechanism and related work, which has been proposed to reduce the booting time of embedded systems.

A. Software Suspend-Resume

The suspend-resume technique is invented to save power when a system is in an idle state [4]. There are three power suspend states: standby, suspend-to-RAM, and suspend-to-disk. Our basic approach is to utilize the snapshot boot technique by using suspend-to-disk [3]. Although these power states are for power saving, they are also useful to reduce the booting time in an embedded system by resuming from a suspend image. It is a software based technique that is independent of hardware devices such as APM or ACPI. Note that we refer to the suspend-to-disk as *suspend* and the restoration from a suspend image as *resume* in this paper for brevity.

The suspend procedure comprises the following steps.

1. Freeze all processes - All existing processes should stop running so that the system state is immutable.

2. Suspend and turn off devices - To prevent the change of device states, all devices except storage for suspending are turned off in this stage. At least, one storage device is left active to save a suspend image.

3. Save the processor state - To store processor states, this stage saves processor registers such as generic registers, segment registers, co-processor registers, and so forth.

4. Save the memory state - Memory pages are divided into two groups: one containing memory pages that are used by the suspend codes, and the other for the rest pages. The latter can be saved directly to storage. The former, on the other hand, is copied to safe memory location to avoid modification by the suspend codes and then saved to the storage. Some meta-data that contains the original memory location of the copied pages are saved in the storage. In the process of saving memory pages, we compress the pages to reduce the size of the suspend image.

5. Halt system - If all above stages are completed, it turns off system.

On the other hand, the resume procedure is initiated in the late *init* call of the kernel. After loading the kernel core and device drivers, the kernel decides whether it has to start the resume procedure or not. The decision can be defined by a kernel parameter. If the kernel decides to resume, the rest of the resume procedure is the reverse order of the suspend procedure.

B. Related Work

There is several previous research to improve the startup time of an embedded system. Bird proposed several techniques to enhance the booting time in a commodity OS [2]. He analyzed each step of the boot sequence and suggested various methods such as kernel execution-in-place (XIP), probe delay elimination, RTC read synchronization, and so forth. Wool proposed methods to optimize the boot time for an embedded OS [7]. His approach not only concerns the time reduction needed to initialize the core kernel and drivers but also addresses application-level considerations and other areas of the kernel such as linear flash memory access and time-optimized module insertion. Park et al. applied the well-known boot time reduction techniques to a commercial product [5]. They targeted a digital camera for their research. Through their experiments, they analyzed the detailed boot time consumption for each boot step.

Kaminaga proposed a snapshot boot mechanism to enhance the startup time of a commodity OS by using the suspend-resume technique [3]. The snapshot boot technique suspends the running kernel, and resumes with the suspend image instead of booting. To reduce resume time, it resumes the suspend image from a boot loader, not the kernel. With the snapshot boot technique, he also applied other techniques such as XIP, pre-linking.

The snapshot boot technique is an approach to enhance the startup time of an embedded system. It, however, is largely dependent on hardware devices. In the real world, embedded systems are more complicated than those used in the previous research. For example, our DTV system has various dedicated devices to support the functionalities of DTV, and runs a large DTV application communicating with these devices. These devices are hard to be adopted in the snapshot boot technique.

III. DTV System and Environment

A. DTV Architecture

The DTV system used in this research is a current commercial product. Its hardware is composed of an embedded CPU [9], 128 MB memory, and NAND flash memory as a storage device. Additionally, there are several special devices designed for DTV. Most of all, the tuner and the decoder devices are important for DTV functionality. The tuner device receives video stream from a broadcasting station, and the decoder device is responsible for decoding the stream.

Besides, there are graphic and sound devices for DTV functionality. Fig. 1 shows the software architecture of the DTV system used in this research.

The DTV software system consists of a bootloader, an OS, device drivers, and DTV application. The DTV system uses a small bootloader [8] and an embedded OS. They follow a general embedded system booting sequence. After loading the kernel, the system initiates a kernel module named s*ettop.ko* that is in charge of controlling DTV-specific devices. This module is a thin layer used as a medium to connect user-level applications with the kernel.

The DTV system in our project utilizes an integrated library, called *libsettop.so*, which controls overall DTV devices in the user-level mode. The library provides a group of APIs for application software. Using this library, the DTV application initializes hardware devices and displays decoded video stream, menu, and channel information onto screen. In our DTV system, the kernel uses 40 MB memory space of 128 MB, and the libsettop.so library resides on the remaining 88 MB memory space. The library controls the devices by reading and writing registers and uses this part of main memory as buffer space for video stream and data structures for devices.

The DTV application named *exeDTV* is a large multi threaded application to provide the DTV functionalities while the DTV is turned on. In addition to the basic DTV functionalities, it is in charge of menu navigation, the control of video and sound qualities, and other non-television services. The DTV application consists of 3.4 million lines of C++ codes, and its size is about 8 MB. At the beginning of exeDTV, it initializes all necessary devices and reads resources such as menu images, fonts, and so forth. These resources are stored in NAND flash storage separately from the exeDTV binary.

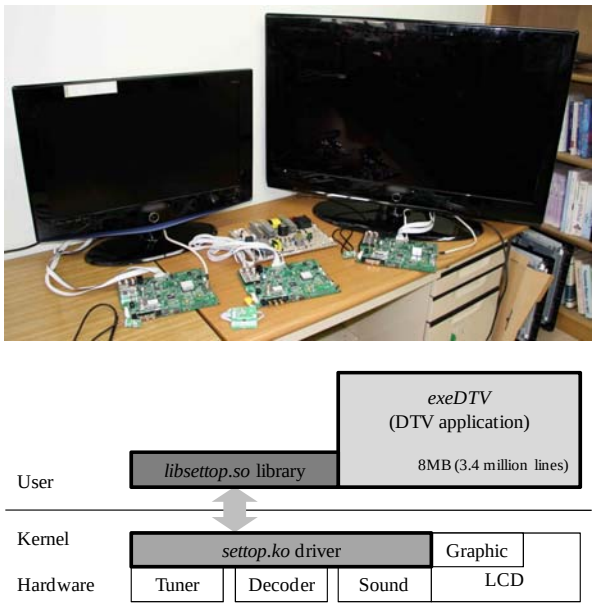

Fig. 1. The software architecture of the DTV system

B. Normal Startup Time

In this paper, we define the completeness of system booting is the state that screen is on, since common users perceive screen as a criterion for power-on of DTV. Note that the startup time in this paper denotes the elapsed time from cold power-down state to DTV screen-on state. Therefore, the booting sequence passes through the all software architecture including the bootloader, the kernel, device drivers, and the DTV application.

The normal startup time of DTV is shown in Fig. 2. The x-axis of the figure is the execution time in second after the DTV is turned on. Note that the read throughput of the flash memory storage is 5 MB/s. To the end of the kernel, about 3.6 seconds are consumed including the bootloader. The shell script is the *init* script, which is in charge of mounting file systems, loading modules, and starting the DTV application. Application loading notifies the elapsed time between the start of DTV application to the screen-on. It takes the largest part in the startup sequences of the DTV system. The application prepares to show screen including initialization of devices and heap area and resource loading. The overall time consumed for the startup of DTV system is 10.78 seconds in total.

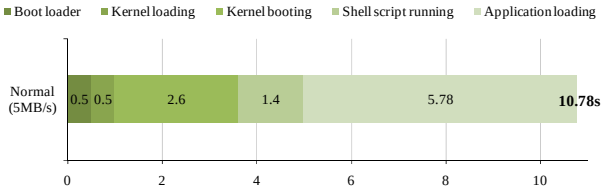

Fig. 2. The normal startup time (seconds) of the DTV system

IV. RAW RESTORATION

This section describes *Raw Restoration* (RawRes), a method to improve DTV startup time. It is an approach to apply the suspend-resume technique on our DTV system with the DTV application. Especially, this approach focuses on the restoring of the DTV application state and the special devices for DTV system.

A. Overview

The snapshot boot technique is hard to be applied for a DTV system, since it requires modification of a bootloader. Such modification needs a large amount of engineering cost. Moreover, the modification is dependent on hardware features such as CPU architecture and storage device type. Actually, DTVs are developed on many kinds of hardware platforms, and are continuously changed according to business requirements. Another possible approach is to apply only the suspend-resume technique to restore a kernel booted state without both resuming from a bootloader and restoring the DTV application. It, however, is not beneficial, since the starting time of the resume process is almost the end of the kernel initialization and the reading time of the suspend image sets off the benefit of the suspend-resume technique.

Therefore, RawRes is designed to suspend and resume the

DTV application state. Fig. 3 shows a scenario to utilize RawRes on a common DTV system. RawRes suspends the system with the running DTV application and resumes the entire system to application running state. Due to resuming to application running state, RawRes eliminates the long loading time of the DTV application. The suspend procedure is performed once in a factory to create a suspend image, and then it is reused whenever a user turns on the DTV. As described in the previous section, the DTV application uses libsettop.so library to control the devices. Since the registers of DTV-specific devices are much various and abundant, suspending entire system including device states is a challenging problem. This section mainly focuses on how to save and restore the device states and the memory space managed by libsettop.so library.

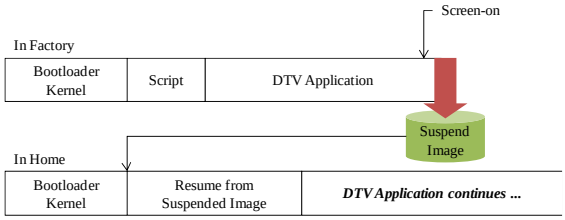

Fig. 3. The scenario of Raw Restoration

B. DTV Application and Device State

Although suspending the kernel and general devices is well known technique and is used in embedded systems, our challenges are the complete restoration of the kernel and the DTV application with reconstructing device states. The DTV system has designated devices such as tuner and decoder, and suspending and resuming these devices should be carefully handled to recover the state of DTV application, since they are tightly coupled with the DTV application.

The libsettop.so library controls the devices as shown in Fig. 4. Most of RawRes techniques are to guarantee the continuity of the states of the libsettop.so library between suspending and resuming. Our DTV board has about 22000 registers for controlling tuner, decoder, graphic, sound, and so forth [6]. In addition, the top 88 MB area memory, which is invisible to the kernel, is managed by the libsettop.so library in the form of heap. This memory area is dynamically allocated by only the APIs of libsettop.so library, which are similar to *malloc* and *free* of C libraries, whenever required.

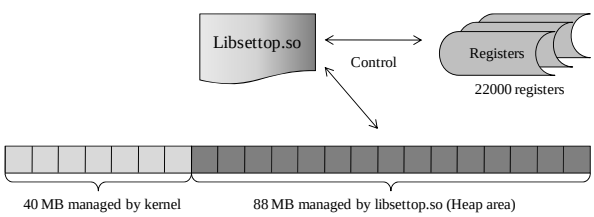

Fig. 4. The memory management by libsettop.so library

To guarantee the continuity of the DTV application and the libsettop.so library, RawRes should store over 22000 32-bits registers and the area of 88 MB memory at suspending time and restore them at resuming time. The integrity of the register values between suspending and resuming is critical because the DTV application expects the same register values when it is resumed. RawRes stores the device registers at the time just before CPU states are stored. To dump all the register states, we define a register structure, and the register values are copied to this structure. The challenge storing the registers is that we can dump only 98% of the register values into the suspend image. The rest of registers cannot be copied and restored due to their particular properties; an example of those registers is a *test-and-set* register type of which value is changed whenever it is read. To address this problem, we take an approach to reset such registers. After restoring the general register values, RawRes sends a signal to libsettop.so library at the end of the resume process, and its handler resets those particular registers.

To store the 88 MB heap area, RawRes compresses it into the suspend image with the other memory area (40 MB). Although RawRes can avoid the long loading time of DTV application, storing heap area results in the increase of suspend image. Upon a preliminary evaluation, the suspend image grows up by about 15 MB even after compressing 88 MB area. Since this augmentation makes the suspend image be read for a longer time during resuming process, the gain from avoiding the application loading time is offset. Therefore, it is important to reduce the size of the suspend image as much as possible.

The heap area is mainly used to keep temporal data for encoding and decoding. Therefore, most of data stored at suspend time are obsolete at resuming time, only small portion of the heap area is likely to be used at the early time after executing the DTV application. Isolated from the heap area, RawRes implemented in the kernel space (40 MB area) cannot determine which parts of the heap area are actually used. For this reason, we modified libsettop.so library to initialize the temporal data area of the heap memory with a special signature. Since the untouched pages filled with the special signature are not needed to be restored, the suspend procedure can exclude the pages from the image. In our evaluation, a large portion of untouched pages in the heap area are excluded, thereby shrinking the image size to 9MB.

Due to removing the long loading time of DTV application, RawRes is effective to reduce the overall startup time. RawRes, however, is difficult to be exploited without well-understanding of overall DTV system. For reliability and correctness, RawRes should be exploited at the early step to design software module that has charge of the management of hardware, such as settop.ko driver and libsettop.so library.

V. WARM RESTORATION

This section describes another approach named *Warm Restoration* (WarmRes). It suspends the DTV system with the buffer cache of OS and resumes to the warmed buffer cache state.

A. Overview

While RawRes method tries to suspend and resume the kernel together with the DTV application, WarmRes is an approach to suspend and resume the kernel with its buffer cache. To make WarmRes suspend image, we start the DTV system in normal way and halt the DTV application as soon as its screen is on. Then, a suspend image is built. After resuming with the suspend image, the DTV application should be restarted. Fig. 5 summarizes this procedure.

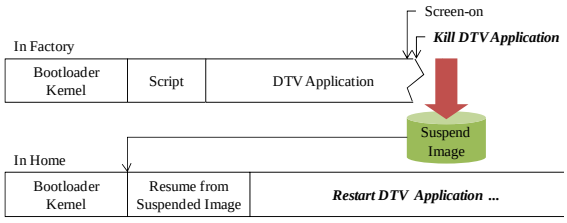

Fig. 5. The scenario of Warm Restoration

B. Warm Buffer Cache

Commodity OSes typically use a unified buffer cache to retain storage contents in the memory for I/O system performance. The unified buffer cache reduces a read latency when a user program accesses storage data that already resides in the memory. The DTV OS also takes advantage of the buffer cache, since the DTV system contains rich applications and contents such as widget or flash animations. For example, in the case of an application instantiation, the first startup requires the operation that reads the codes and data of the program image. After the buffer cache is warmed with the program image, succeeding startup of this application can be quickly executed because its codes, data, and the library data reside in the memory.

In addition to the buffer cache, demand paging is also a representative technique for efficient memory management of commodity OSes. The demand paging makes memory allocation to be delayed until requested data is actually needed. This mechanism uses the paging facility of a processor that has memory management unit, which is prevalent in modern embedded systems. When memory mapping operation or the allocation of heap is requested, an OS does not promptly allocate physical memory with remembering the requested regions. Because the page table entries of requested region are cleared, later accesses to this region cause page fault exception. The OS handles such page fault exception with lazy memory allocation in the page fault handler. If a page fault leads to some I/O operations for loading storage data, it is called a major page fault and disturbs the program execution in the run time.

WarmRes improves the DTV startup time by warming the buffer cache in advance and reducing major page faults. Because the DTV instantiation has deterministic working set during the startup procedure, the demand paging overheads caused by major faults are extravagant for the startup. WarmRes technique takes a snapshot of system image that has executed the DTV application till initial display time. This suspend image contains the buffer cache that includes the working set for the startup. WarmRes resumes the system with this image and restarts the DTV application. After the DTV system is resumed, the DTV application startup is boosted by subsequent buffer cache hits without major faults for its initial working set, including application codes, libraries, and multimedia contents.

The benefits of WarmRes technique are as follows. First, the resume process typically initiates right after the basic initialization of the kernel and device drivers, so that system configuration chores in the scripts are omitted; such initial scripts are typically hardware-independent and shell-based programs with some execution time. Second, this technique is more efficient than the normal startup process because locating from the suspend image is a sequential read operation using a low-level block I/O interface.

More importantly, this technique is completely stable in that it is platform-independent. A modern DTV system has rich hardware and thus includes more than hundreds or thousands of special registers. Warming the buffer cache involves with only OS features without relying on board-specific states and information. Because the DTV application is fully re-initiated after resume, the board-specific states can be completely re-constructed. Therefore, the suspend process does not need to save whole the DTV specific registers and the heap area. This stability is an important advantage in that many consumer electronics companies mass-produce various boards and regard time-to-market as a crucial factor.

VI. Evaluation

This section presents the startup time evaluation of RawRes and WarmRes on our DTV system. For baseline, we also evaluate the normal booting and the snapshot boot technique. As described in section 2, we use a commercial DTV system, which has an embedded CPU and 128 MB memory. NAND Flash memory used as storage shows about 5 MB/s read throughput including kernel-level and user-level overheads such as file systems, page faults, copy-to-user, and so forth. Since the DTV application does not write to the storage during the startup, we do not consider the write throughput of flash memory.

A. Startup Time

Fig. 6 shows the startup time of our DTV system using the snapshot boot technique. In this evaluation, we snapshot only the kernel image and resume with it. After resuming, we start the DTV application. The x-axis of the figure shows the elapsed time from power-on to screen-on. Although the flash memory read throughput of the DTV system is 5 MB/s, we also simulate flash memory with 20 MB/s read throughput to anticipate the effectiveness of flash memory throughput achieved by the proposed schemes. In the process of resume, flash memory read throughput is critical, since most of overall startup time strongly depends on the time to read a suspend image. The upcoming DTVs will adopt a faster flash memory that shows up to about 20 MB/s read throughput. Therefore,

we estimate the elapsed time for our technique on 20MB/s flash memory by simulating the I/O-dependent part of the measured startup time.

Each item in the figure represents the time spent in each part of the startup procedure. The consumed time before resuming is not different for all cases. The Bootloader, Kernel loading, and Kernel booting take almost constant time. Resume(I/O) shows the time to read the suspend image, and Resume(non-I/O) denotes the time to uncompress the suspend image and to restore device registers and the heap area.

In the snapshot boot technique, the startup time is 11.12 seconds using the flash memory of the DTV system. With the 20 MB/s flash memory, the startup time is 10.25 seconds. The former is longer than that of the normal booting by 0.34 seconds. This result describes that resuming only the kernel is rather inefficient compared to the normal booting. The estimated result of 20 MB/s flash memory, however, shows that the snapshot boot technique can reduce the startup time by 0.53 seconds compared to the normal booting.

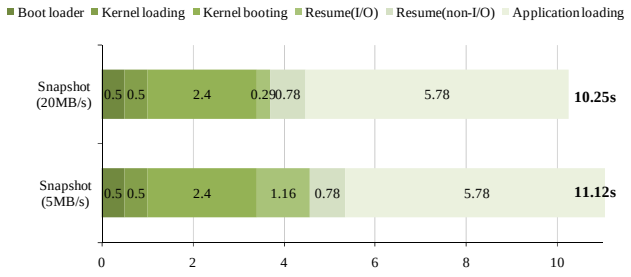

Fig. 6. The startup time (second) using Snapshot

Fig. 7 shows the startup time evaluation result using RawRes. In this evaluation, the Resume(I/O) time of 20 MB/s flash memory is 0.49 seconds and that of 5 MB/s flash memory is 1.94 seconds. The total startup time of DTV system is taken about 4.76 and 6.21 seconds, and these results are 6.02 and 4.57 seconds reduction compared with the normal startup time. The size of suspend image is about 9.73 MB. The main startup time reduction comes from avoiding the DTV application loading. RawRes does not need to restart the DTV application after resuming because the suspend image includes it already.

Fig. 7. The startup time (second) using Raw Restoration

We also evaluate WarmRes approach as shown in Fig. 8. The consumed time before resuming is the same as that of RawRes. The Resume(I/O) time in WarmRes is taken about 0.44 seconds in 20 MB/s flash memory and 1.76 seconds in 5 MB/s flash memory. The size of the suspend image is about 8.73 MB. Differently from RawRes, the DTV application should be restarted with warmed cache. Therefore, the application loading time is added. It is about 3.39 seconds until the screen is on. Compared with the normal DTV application loading time, the startup time is reduced by 2.39 seconds due to the warmed buffer cache, since WarmRes can avoid major page faults and storage access overheads. The total startup time of DTV system takes about 8.27 seconds for 20 MB/s flash memory and 9.59 seconds for 5 MB/s flash memory. The time reduction is 2.51 and 1.19 seconds, respectively.

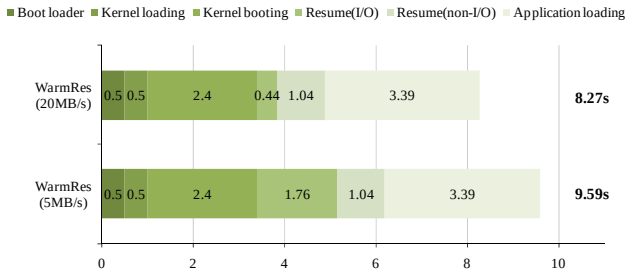

Fig. 8. The startup time (second) using Warm Restoration

B. Discussion

Our evaluation illustrates that RawRes achieves shorter startup time than WarmRes. RawRes, however, should recover all register states of the DTV-specific devices and the heap area managed by libsettop.so library. This burdens heavy engineering cost because the modification of the kernel and library requires the complete understanding of all device specifications and memory management mechanisms. Moreover, if the DTV board is replaced with another model, a large part of suspend-resume, device driver codes, and libsettop.so library should be modified.

On the other side, although WarmRes takes more time to start the DTV system, it is advantageous in higher portability. WarmRes implementation is confined to the kernel, and therefore it does not require the modification of device driver and library. More importantly, it is not dependant on the platform devices. Actually the DTV boards are diverse in commercial product lines, while the kernel version is hardly changed. Therefore, the porting cost is much lower than that of RawRes. Another advantage of WarmRes is reliability. Since it benefits from the warmed buffer cache and starts the DTV application again, WarmRes guarantees stable execution. Table I summarizes the comparison between RawRes and WarmRes.

From the evaluations, our two approaches are tightly associated with flash memory read throughput and decompressing time. As the estimation for 20 MB/s flash memory shows, when a DTV system adopts a faster flash

memory, it will increase the effect of the proposed approaches. With regard to the decompressing time, it is relevant to the computing power of CPU. As having increasingly provided computation-intensive non-television services, DTV systems require a higher performance CPU. These upgrades of hardware specification will redouble the reduction of the startup time using our scheme.

TABLE I
THE COMPARISONS BETWEEN RAW RESTORATION AND WARM RESTORATION

	RawRes	**WarmRes**
Startup Time	6.21 seconds	9.59 seconds
Image size	9.73 MB	8.73 MB
Modification	Kernel Device drivers Library	Kernel
Portability	Low	High

VII. CONCLUSION

In the last few years, the popularity of DTV increases more and more. DTV is advantageous in that it allows higher-quality images, sound, and special services such as multiplexing, EPG, and so forth. Notwithstanding the advantages of DTV, there are common limitations, most of which come from the software complexity.

In this paper, we propose two approaches to improve the startup time of DTV. RawRes is an approach to suspend DTV system states with its application. Then, it resumes with the suspend image instead of booting. RawRes reduces about 50% of the normal startup time by avoiding the DTV application loading. On the other hand, WarmRes is an approach to exploit the warmed buffer cache of OS while decreasing engineering cost by confining all implementations to the device-independent part of the DTV system. Although the startup time reduction is less than that of RawRes, WarmRes can be applied to a DTV system with ease because its implementation is lightweight.

We also analyze which points are bottlenecks to improve the startup time more. If the read throughput of storage such as flash memory to read a suspend image and the computing power to decompress a suspend image are faster, we can achieve more improvement of the startup time. We believe that the proposed mechanism and our analysis will be practical to enhance the startup time of a large embedded system such as DTV, PVR, mobile handset, and so forth.

REFERENCES

[1] Y. Wu, S. J. Hirakawa, and U. H. Reimers, "Overview of Digital Television Development Worldwide," *Proceedings of The IEEE*, vol.94, no.1, pp.8-21, Jan. 2006.

[2] Tim R. Bird, "Methods to Improve Bootup Time in Linux, Sony Electronics," In *Proc. of the Linux Symposium*, 2004.

[3] Hiroki Kaminaga, "Improving Linux Startup Time Using Software Resume," In *Proc. of the Linux Symposium*, 2006

[4] System Power Management States. *Documentation/power/states.txt*

[5] Chanju Park, Kyuhyung Kim, Youngjun Jang, and Kyungju Hyun, "Linux Bootup Time Reduction for Digital Still Camera," In *Proc. of the Linux Symposium*, 2006

[6] *http://www.broadcom.com/products/Consumer-Electronics/Digital-TV-Solutions/BCM3560*

[7] Vitaly Wool, "Optimizing boot time for Embedded Systems," In *Proc. of FOSDEM*, 2006

[8] *http://sourceware.org/redboot/*

[9] *http://www.mips.com/*

[10] Ling Xing, Jianguo Ma, Xian-He Sun, Youping Li, "Dual-Mode Transmission Networks for DTV," *IEEE Transactions on Consumer Electronics*, Vol. 54, No. 2, MAY 2008.

[11] *http://www.samsung.com/us/aboutsamsung/ir/ireventpresentations/conferenceactivities/downloads/2008/fkii_20080416.pdf*

[12] Charles W. Rhodes, "Interference to DTV Reception due to Non-Linearity of Receiver Front-Ends," *IEEE Transactions on Consumer Electronics*, Vol. 54, No. 1, FEBRUARY 2008