A Study on Efficient Management of Changed Information in NTIS

Yejin Jo[1]*, ChulSu Lim[2]**

[1,2] Korea Institute of Science & Technology Information (NTIS Center, Daejeon, Republic of Korea)
** yejin.jo@kisti.re.kr*
***cslim@kisti.re.kr*

Information managed by information systems need some changes to efficiently manage and reflect updated data and correct errors.
This study takes three approaches to examine methods to improve the efficiency of information management. First, we analyze the causes of modification happened to AS-IS system and the procedures. To do that, we examine history log of request for updates. Second, we conduct a comparative survey of information management system. Third, we seek for suitable information management for NTIS to properly proceed its duty and manage information by examining aspects of data, tasks, and services. We identify changes in information on participants when examining items of projects, information on papers and patents needed updates to take care of the modification in items of outcomes. So the study shows on how to efficiently manage changed information to improve the accuracy of NTIS service is needed.

[1] Oh, In-Bae, et al. "Design of a History Data Management System for the Renewable Energy Resources." The KIPS Transactions: PartA 10.6 (2003): 757-768.

Graphical keyword service for research papers with text-mining method

Yejin Jo
NTIS Center
Korea Institute Science & Technology Information
Daejeon, Republic of Korea
yejin.jo@kisti.re.kr

Eun-Gyeong Kim, Yongju Shin
NTIS Center
Korea Institute Science & Technology Information
Daejeon, Republic of Korea
{eungyeong, yjshin}@kisti.re.kr

***Abstract*— This paper is for utilization of text mining method to provide visual keywords of the papers and reports. This study presents a visualization approach to secure intuitive understanding rather than abstract, keywords. The statistical examples of few technical papers are shown. The graphical methods in this paper will be helpful tools for researchers, the public who need to access expert literatures. The authors tried to draw graphical methods by using R programming language in this paper. In addition, we expect this work would contribute to the public who want to seek expert papers in easy and intuitive way.**

Keywords; Wordcloud, Text mining, Barchart, Keyword, R programming.

I. INTRODUCTION

The technological papers or technical reports are composed with few thousands of words. Therefore, text-mining method with R language can be used for search engine on these technological papers can be uploaded on the paper database. The conventional summary information such as highlights, keyword can be replaced with word cloud and bar chart. This method provides intuitive understanding for researchers. In addition, this technique has a useful point for public access to expert documents.

In a typical research paper, three to five keywords are provided. In this study, the authors try to replace it with graphical keywords. By using this method, a user can find the best matching papers with the barchart and wordcloud, and the user can grasp the keywords of the paper at a glance. Using this method, text mining technique was applied to the number of papers, and barchart and wordcloud were applied. By using this method, users can see the articles that match the keyword that they are looking for sequentially, and it is possible to grasp the keywords of each article at a glance.

In this research, we make following contributions in three different point of view.

- We propose currently the effective and feasible portal service for digital library to determine whether the patron need to read.
- By providing visualization service to readers, we have a target to help understanding to the research before reading.
- We support to discover and expand scientific knowledge by providing high-quality information for researchers at industry, academia and institute.

The remainder of this paper is organized as follows. Section 2 describes trend of studies regarding visualization and digital libraries. Section 3 includes how to operate experimental model with its operational flow. Section 4 presents analysis result by using text mining and drawing word cloud and bar chart. Section 5 expresses future works in our model. And section 6 includes conclusion of our study.

II. RELATED WORKS

A. *Visualization of Contents*

Studies on graphical abstract methods have been performed in various institutes. Google provides knowledge graph by constructing a knowledge database with semantic tag for refined result in academic search [4]. They utilized semantic tag for more accurate searching rather than Hyperlink.

B. *Digital libraries*

Wikipedia established Machine-readable knowledge DB named as DBPedia [5]. This DB was based on the ontology and PDF/s with semantic web technology. However, the converting process of knowledge to ontology requires large computational resource. Also, a triple conversion of knowledge written in text is not an easy task.

In the UK, British library performs as the central organization of information management system, and BLDSC(British Library Document Supply Center) is operated with the purpose of gathering information systematically. BLDSC provides various services with its DB of research reports, thesis, translated works, conference materials, or official publications.

BLDSC has the most massive amount of unrestricted report in the world by gathering information not only from the UK but also from government agencies of the US or international agencies. The materials it collects also include reports, papers, or technotes from private and public sectors.

INIST(the Institute for Scientific and Technical Information) of France gathers information on S&T from all around the world, but its main interest is gathering information on regions where national R&D is performed actively. Its duty is to promote gathering, refining, analyzing, and distributing international research results. In details, its main role is to build DB and provide bibliographic information as well as information service and information resource in order to allow research institutes to get access to information on S&T. Furthermore, as a leading scientific institute in Europe, INIST provides advanced information on STI to diverse clients including public institutes, academic personnel, and authorities in the field of social science and economics. It also provide services to countries of the world by providing multilingual DB of general information service.

NTIS(National Technical Information Service) of the US was established in 1970 and is under Department of Commerce. Its mission is "to provide accessible information in ways that promotes innovation and development in order to support economic growth and employment creation". To achieve the mission, it gathers and distributes information to the public and provides works or other services to federal agencies. The roles of NTIS are to promote users to get access to information from government agencies and to develop guidelines or process to transfer information from the agencies to NTIS efficiently. It also pursue to maintain the role of information repository by preserving unclassified information on S&T, engineering, and management permanently. In addition, it is developing ways to promptly gather and distribute information on S&T, engineering, and management created from all around the world.

Furthermore, there are many kinds of research regarding digital library specified in service such as the way of filter the contents or open access for their patrons [6], [7], [8].

III. Approaches

A. Wordcloud

The word cloud is a technique that the keywords are shown in text-based cloud [3], [9]. It can be understood in intuitive way. If a word is frequently mentioned, it is considered to an essential word in the paper. Therefore, important words are described in big size font. This process is described in Fig. 1.

The word cloud is parsing XML file to obtain text in body section. After that, it is converted to the text file, and formed a cloud shape in R program. The text file is used as a Corpus input file for data processing. The pre-preprocess steps for text mining are explained in following: i) Elimination of punctuation such as ".", ";' numbers. ii) All the alphabets are converted to the lower case; the stop words such as "be", "better" are eliminated. After that, the R program calculates frequency of each word, and converts it as a word cloud by using built-in library.

Also the font size of word cloud can be determined by using normalization factor.

$$FS_i = FS_M \times NF$$

$$\mathrm{NF} = \frac{Fi - LF}{HF - LF}$$

FS_i: Font Size at word of i-th frequency
FS_M: Font Size at word of most frequency
NF: Normalization Factor
HF: Highest Frequency
LF: Lowest Frequency

B. Barchart

The bar chart is a graphical method to show distribution of frequency of data. X-axis contains variable sections, and Y-axis shows statistical quantity of each sections as a height of bar. The results are separated in statistical rank, and Y-axis shows it as a frequency chart. Process for make bar chart is shown in Fig. 1.

The sum of all the relative frequency is 1. The high relative frequency means that it is important words in the paper, therefore it can be considered to keyword. But there are many words in the paper, we analyzed and extracted only the 15 frequent words.

$$\frac{Fi}{\sum_{i=1}^{n} Fi} = RFi$$

$$\sum_{i=1}^{n} RFi = 1, \qquad 0 \leq RFi \leq 1$$

n: the number of words
RFi: Relative Frequency at i-th step
Fi: Frequency at i-th step

C. Text preparation

The tests were conducted by using R program, and the statistical result was converted to the wordcloud and barchart. The visual results are produced by R program. Compile environment is written in Table 2.

TABLE I. Compile environment

Architecture	x86_64
System	x86_64, mingw32
OS	Windows 7 64bit
SVN version	70,800
version.string	R v3.3.1 (21th June, 2016)
Nickname	Bug in your hair

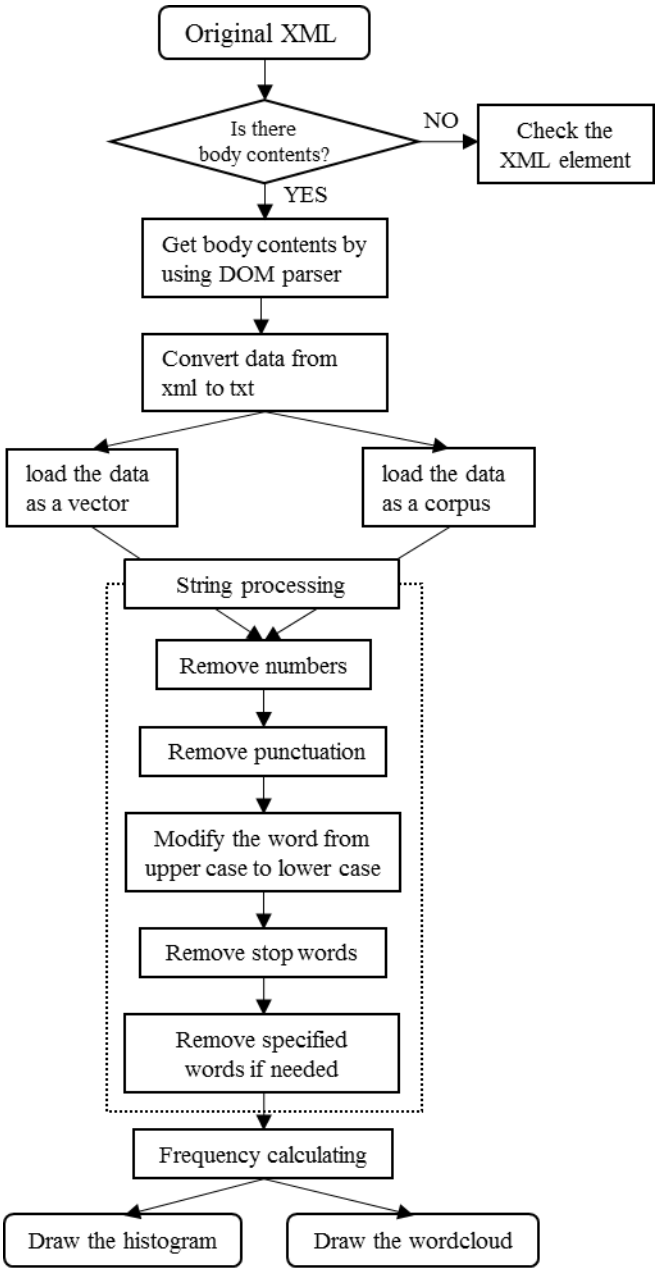

Figure 1. Drawing processes in wordcloud and barchart

IV. PROGRAM RESULTS

A. *Result of Paper #1*

The frequency of words in the literature is shown in Table 2. The table shows top 15 words that frequently used. As we mentioned in chapter 3, the text was analyzed in the string processing module. First step has removed numbers. Second step has removed punctuation. Third step has converted all the words from the upper case to the lower case. Forth step has removed stop words such as “the”, “as”. And final step has removed specified words if required. The total number of words is 359. The number of words that used in two or more times is 182. The number of words that used in more than 10 times is 23. The results are described as a bar chart and word cloud as shown in Fig. 2-3.

TABLE II. FREQUENCY, RELATIVE DEGREE OF WORDS.

Words	Frequency	Relative Frequency
nontextual	59	0.05051370
contents	55	0.04708904
information	44	0.03767123
pdf	29	0.02482877
report	29	0.02482877
tables	28	0.02397260
file	25	0.02140411
extract	19	0.01626712
images	19	0.01626712
research	16	0.01369863
xml	15	0.01284247
content	14	0.01198630
location	14	0.01198630
processor	14	0.01198630
search	13	0.01113014

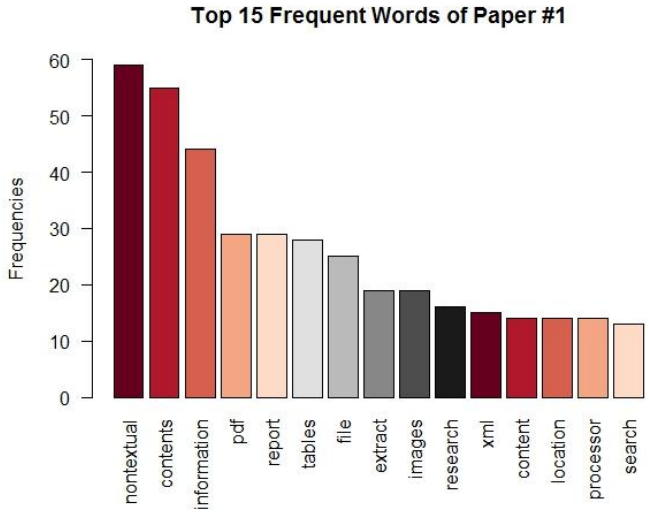

Figure 2. Barchart result from text mining.

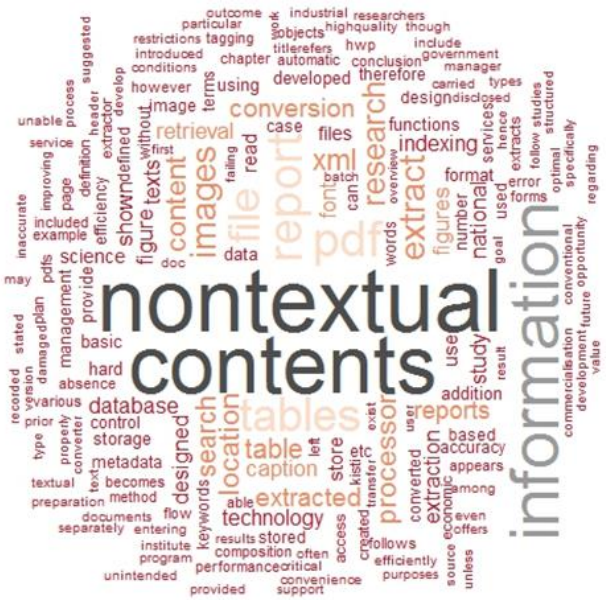

Figure 3. Wordcloud result from text mining.

Topic of this paper is “A method of extraction of non-text contents for extending the applicability of national R&D reports”. This paper proposed that how to extract non-textual contents such as image and table in the PDF format. The main keywords in bar chart and word cloud are successively showing the essential keywords.

B. Paper #2

The frequency of words in the literature is shown in Table 4. The table shows top 15 words that frequently used. As mentioned in chapter 3, the text was analyzed in the string processing module. As for 5th step, the specific words are removed such as “drop”, “existing” in this paper. The number of words is 665. The number of words that used in two or more times is 301. The number of words that used in more than 10 times is 35. The results are described as a bar chart and word cloud as shown in Fig. 4-5.

TABLE III. FREQUENCY, RELATIVE DEGREE OF WORDS.

Words	Frequency	Relative Frequency
correlation	76	0.039521581
heat	43	0.022360894
pche	38	0.019760790
reynolds	38	0.019760790
cfd	26	0.013520541
cycle	26	0.013520541
pressure	26	0.013520541
range	25	0.013000520
channel	24	0.012480499
transfer	23	0.011960478
flow	20	0.010400416
hot	16	0.008320333
temperature	16	0.008320333
channel	14	0.007280291
experimental	14	0.007280291

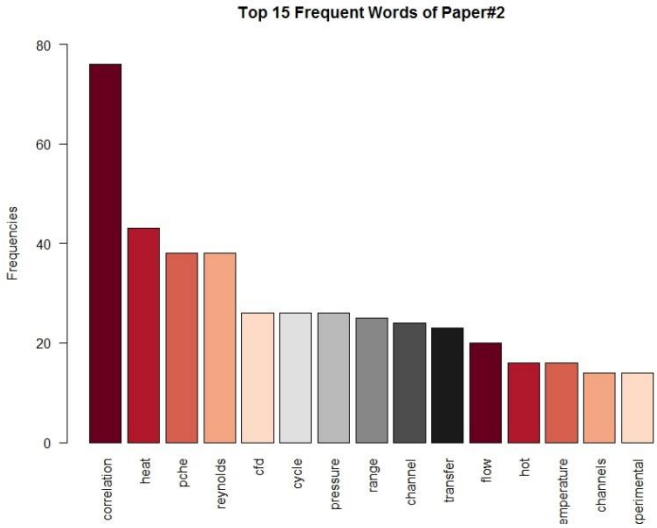

Figure 4. Barchart result from text mining.

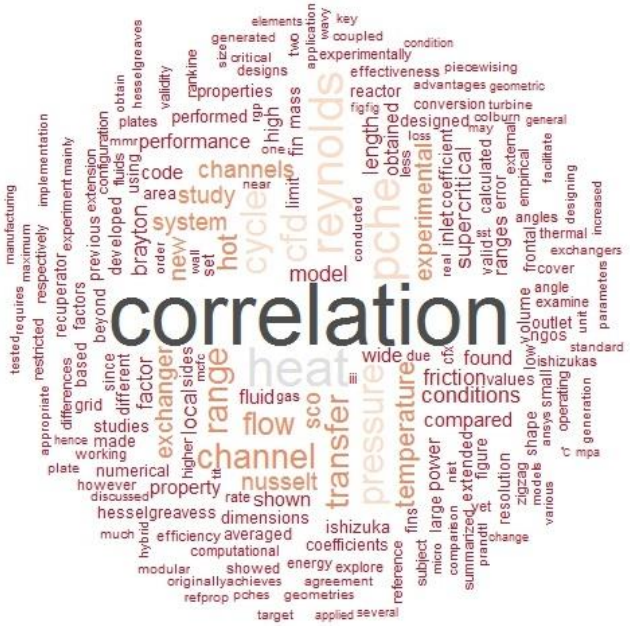

Figure 5. Wordcloud result from text mining.

The purpose of this paper is development of heat transfer and friction factor correlation with supercritical CO_2 fluid in the PCHE type heat exchanger. This paper is categorized to the mechanical engineering, computational fluid dynamics. Essential words such as “PCHE”, “Reynolds”, “CFD” were successfully captured in this analysis. The results are similar to the keywords, highlights of the paper.

C. Paper #3

The frequency of words in the literature is shown in Table 5. The table shows top 15 words that frequently used. As mentioned in chapter 3, the text was analyzed in the string processing module. As for 5th step, the specific words are removed such as “using”, “figure” in this paper. The purpose of this paper is for analyzing the process of research paper database.

TABLE IV. FREQUENCY, RELATIVE DEGREE OF WORDS.

Words	Frequency	Relative Frequency
report	35	0.060449050
management	23	0.039723661
national	18	0.031088083
system	14	0.024179620
information	14	0.024179620
quality	11	0.018998273
research	10	0.017271157
paper	6	0.010362694
meta	6	0.010362694
control	6	0.010362694
method	5	0.008635579
data	4	0.006908463
distribution	4	0.006908463
efficient	4	0.006908463
outcome	4	0.006908463

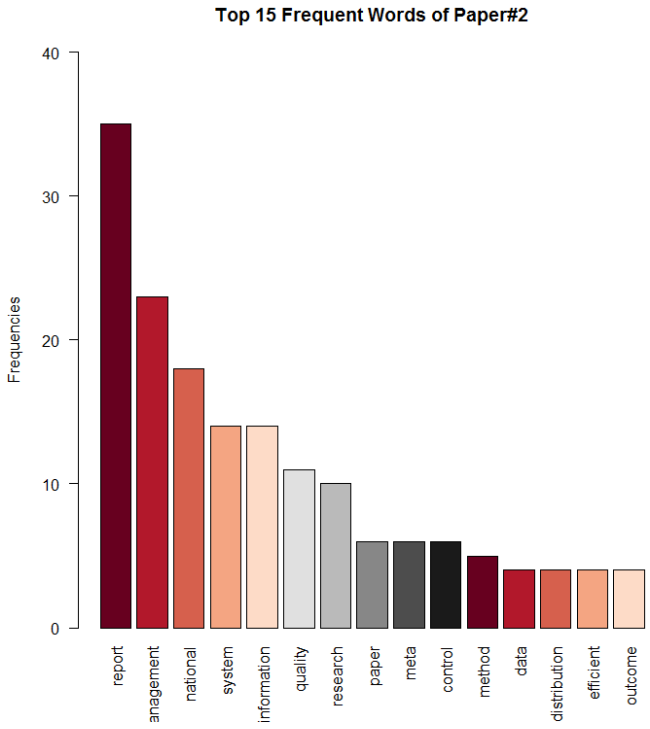

Figure 6. Barchart result from text mining.

Figure 7. Wordcloud result from text mining.

V. LIMITATAION AND FUTURE WORKS

As mentioned in chapter 1, this research provides the efficient and necessary service with the visualization methods rather than abstract. If the word clouds of other papers in similar subjects are connected to each other with network maps, it will be a helpful tool who seek the related literatures. It can be sorted by similarity, number of citations, and impact factor, and etc. The expert words combined with two or more words (ex: data mining, operating system) are missed out

from the text mining, because the R program ignores combined word with spacing. Also, there are the singular words even though including “s” such as “reynolds". Thus, there exists ambiguous word processing whether it is the plural or singular.

VI. CONCLUSION

Until now, the authors carefully studied the methods for reader-friendly services through text mining by analyzing research papers. Especially, in case of NDSL, it is a pan-national services. The authors will contribute to help understanding to the public or some of researchers by adding graphical summary before reading.

ACKNOWLEDGMENT

This research was funded by the Diffusion of National R&D Outcome funded by the Korea Institute of Science and Technology Information.

REFERENCES

[1] http://www.ndsl.kr/

[2] http://www.ntis.go.kr/

[3] https://en.wikipedia.org/wiki/Tag_cloud

[4] Singhal, A. (2012). Introducing the knowledge graph: things, not strings. Official google blog.

[5] Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., & Hellmann, S. (2009). DBpedia-A crystallization point for the Web of Data. Web Semantics: science, services and agents on the world wide web, 7(3), 154-165.

[6] Castelli, D., & Pagano, P. (2002, September). Opendlib: A digital library service system. In International Conference on Theory and Practice of Digital Libraries (pp. 292-308). Springer Berlin Heidelberg.

[7] Gao, F., Xing, C., Du, X., & Wang, S. (2007). Personalized service system based on hybrid filtering for digital library. Tsinghua Science & Technology, 12(1), 1-8.

[8] Speer, J. (2016). Open Access: Digital Research and Scholarship Services.

[9] Cui, W., Wu, Y., Liu, S., Wei, F., Zhou, M. X., & Qu, H. (2010, March). Context preserving dynamic word cloud visualization. In 2010 IEEE Pacific Visualization Symposium (PacificVis) (pp. 121-128). IEEE.

[10] K. S. Choi, K. N. Choi, J. S. Kim “A method of Extraction of Non-text Contents for Extending the Applicability of National R&D Reports” Indian Journal of Science and Technology Vol 8(S1), 340-345, January 2015.

[11] Kim, S. G., Lee, Y., Ahn, Y., & Lee, J. I. (2016). CFD aided approach to design printed circuit heat exchangers for supercritical CO 2 Brayton cycle application. Annals of Nuclear Energy, 92, 175-185.

[12] A´ gnes Boga´rdi-Me´szo¨ly, Takeshi Hashimoto, Shohei Yokoyama, Hiroshi Ishikawa “Frequency- and Content-Based Tag Cloud Font Distribution Algorithm”, International Journal of Computer, Electrical, Automation, Control and Information Engineering Vol:8, No:5, 2014.

[13] Bogárdi-Mészöly, Á., Hashimoto, T., Yokoyama, S., & Ishikawa, H. (2014). Frequency-and Content-Based Tag Cloud Font Distribution Algorithm. World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering, 8(5), 779-783.

[14] Kaser, O., & Lemire, D. (2007). Tag-cloud drawing: Algorithms for cloud visualization. arXiv preprint cs/0703109.

[15] Nguyen, Quang Minh, Thi Nhu Quynh Kim, Dion Hoe-Lian Goh, Yin-Leng Theng, Ee-Peng Lim, Aixin Sun, Chew Hung Chang, and Kalyani Chatterjea. "TagNSearch: searching and navigating geo-referenced collections of photographs." In International Conference on Theory and Practice of Digital Libraries, pp. 62-73. Springer Berlin Heidelberg, 2008.

Design for Most Compatible Booting Model of Integrated Memory-Disk based on ARM Linux

Ye-Jin Jo, Sang-Jae Nam, Ashok Kumar Sharma, Shin-Dug Kim

Scholl of Engineering
Department of Computer Science
Yonsei University, Seoul, Republic of Korea
yejinjo@yonsei.ac.kr
sjnam07@yonsei.ac.kr
i.ashok.s@gmail.com
sdkim@yonsei.ac.kr

Abstract. SCM such as STT-RAM, PCRAM, ReRAM can be used as DRAM or NAND Flash. Due to non-volatile characteristic of SCM, it as a main memory can be affected overhead when the boot code loaded and decompressed whenever system was turned on. Because of this, it is unnecessary to carry out for booting when the system is based on non-volatile memory. The purpose of this paper is to design a new booting model by analyzing the kernel ELF file and disassembling current booting sequence for most compatible startup model. To achieve this goal, we disassembled ELF file format including characteristics of kernel binary. To validate theoretical our model, time cost in each steps have been measured with several different methods. Conclusion has been made that our approach has improved the time cost by approximately 45.25%.

Keywords: booting sequence, non-volatile memory, kernel, ELF file

1 Introduce

Booting time is an important issue that the device has been smaller and increase portability from PC to smartphone and wearable device. Traditionally, whenever then system boot at the DRAM, it gets affected due to overhead of frequent coping and decompressing the kernel image. It can be unnecessary overhead that non-volatile main memory progress booting sequence like DRAM according to improve the technology of SCM. To solve these problems, this paper proposes fast boot by removing unnecessary overhead and providing most compatible booting system based on non-volatile main memory. To design fast boot and most compatible booting system, we define area for change in runtime by using kernel binary and kernel object. We design booting system to distinguish whether it is initialized or not by perceiving ELF file which has supported rule for linking and execution further. In order to verify this booting sequence, we measured the time in steps using different ways which provides the time reduction by 45.25% approximately.

adfa, p. 1, 2011.

2 Background

All kind of devices which are managed by operating system need to booting steps for communicating between hardware and software. For this the device prepare for using hardware resources such as process, memory, storage management through booting steps. In case of embedded system, it is different from PC environment. There are u-boot, flash memory, application processor rather than BIOS, HDD, CPU.

First step of booting process is the boot loader called u-boot. And it is divided into two stages which initialize hardware resources and prepare for kernel. When it run the hardware initialization part, u-boot make the interrupt vector table and set the supervisor mode, watch dog timer. Then it is disabled the interrupt and set CPU clock. U-boot fork memsetup.s for initializing the memory by cpu_init_crit function. It relocate armboot function on the RAM after returning start.s. In the next, u-boot is configure stack pointer for running C routine and fork start_armboot function for initializing resources on embedded board before returning main function.

Second step of booting is the running kernel. vmlinux includes additional data like debugging information can be changed the bzImage. bzImage consists of three parts about booting. First, head.o included initializing kernel code second, misc.o included decompressing code and third, piggy.o included kernel core code. After loading bzImage by bootloader execution, the code is decompressed and initialized for running kernel in piggy.o. The complex reason of the kernel image that's because it carry out of copying kernel image from storage and decompressing the code into the memory. But in case of non-volatile main memory, there are a lot of overhead reduction due to its non-volatile feature, which only provides one time loading and decompressing the kernel image during boot time.

3 Our Methodology

3.1 ELF Dynamic Loading Module (EDLM)

vmlinux executable binary file consists of program header table and section or segment and section header table. Program header table is the set of entry where one entry refer to one segment. This table includes segment information which contains type, location, size, virtual address and physical address. So called this table can be calculated the dynamic area which is changeable runtime. Also, section header table includes the section information which is location, features. This information can be used to support linker and debugger for interpreting ELF object. Linker recognize the location of section header table in front of ELF header and the stage of linking can be analyzed section or segment view. Loader finds the location of program header table by reading ELF header and get the information what segment loaded in memory by referring it.

vmlinux file is also the architecture of ELF binary format. This ELF binary involves file structure which can change the machine code by linker. ELF file has some

rules for linking. Because of this, non-volatile memory can be loaded the area which change in runtime. that is to say, By using vmlinux and bzImage and ELF binary and ELF object, program header table can be referred the address of dynamic area and section header table can be refer to whether it is dynamic are or not. So called by using vmlinux, bzImage, ELF binary and ELF object, the program header table figure out the segment address of dynamic area and section header table to determine dynamic or static.

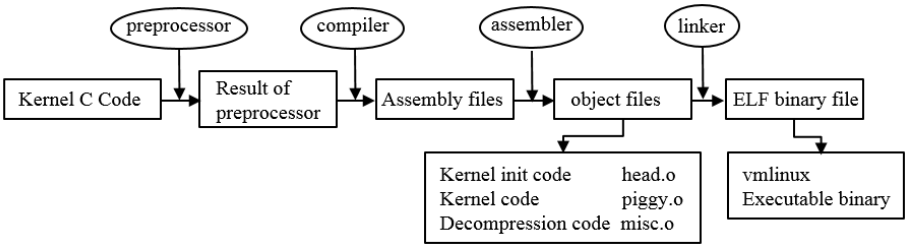

Fig. 1. The steps of fixing the executable compressed kernel from code.

3.2 Full Booting and Dynamic Booting

Full booting as well as current booting consists of multiple steps; bootloader step, kernel loading step, kernel initialization step, kernel running step, init process step. In case of embedded system, following steps are defined such as u-boot stage, kernel loading, decompression stage and then, mounting the root file system stage for running init process. This booting mode need to modify the kernel which gets panic, recover or update. In start, the system progress the full booting before IMD booting to avoid the situation of kernel panic or recover. As shown 3.1, IMD Booting is executed of ELF Dynamic Module for finding dynamic area by operating its three units; ELF verified unit, ELF calculated unit, dynamic loading module which identify the area of changing in runtime. vmlinux located in non-volatile memory is can extract the dynamic area and initialized bzImage which is executable kernel. The possible reason of IMD boot is that ELF format has unique rules for identify its section and segment. The dynamic area can be calculated and relocated based on the unique rules. that is to say, by using vmlinux and bzImage with forms of ELF binary and ELF object, the program header table can be solved the address of dynamic area and section header table can be recognized whether it is dynamic area or not.

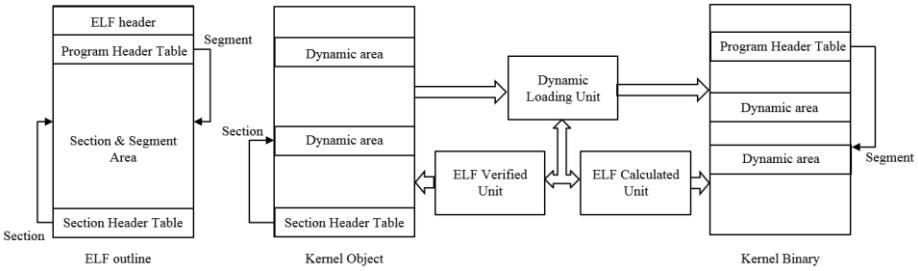

Fig. 2. The file architecture of vmlinux ; executable kernel binary and The dynamic module

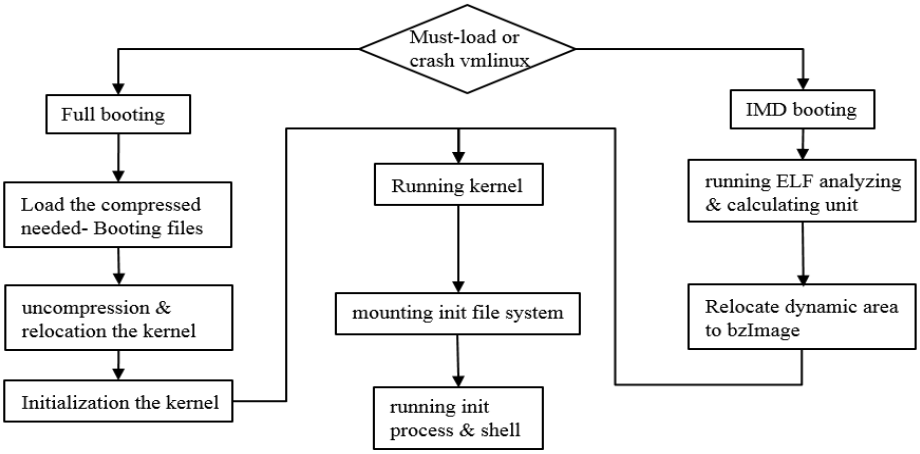

Fig. 3. The diagram of two booting module; Full Booting and Integrated Memory-Disk Booting

4 Experiment

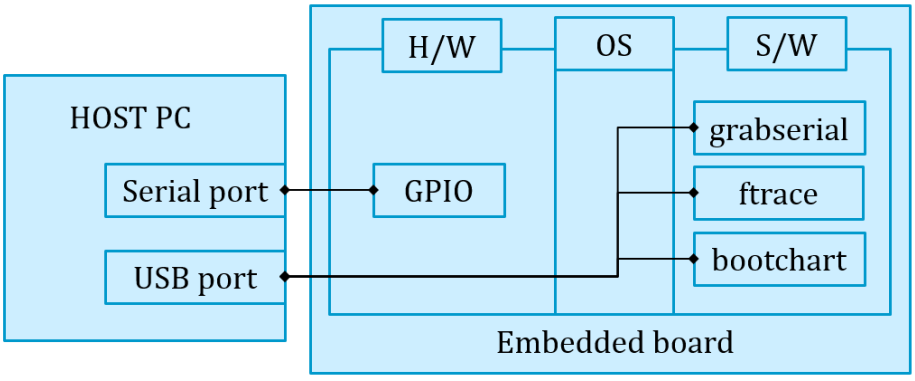

Fig. 4. The configuration of experimental environment

Table 1. The ways of measurement each steps

	Booting stage	Handling method
1st step	Bootloader	grabserial
2nd step	Kernel	ftrace
3rd step	User space	bootchart

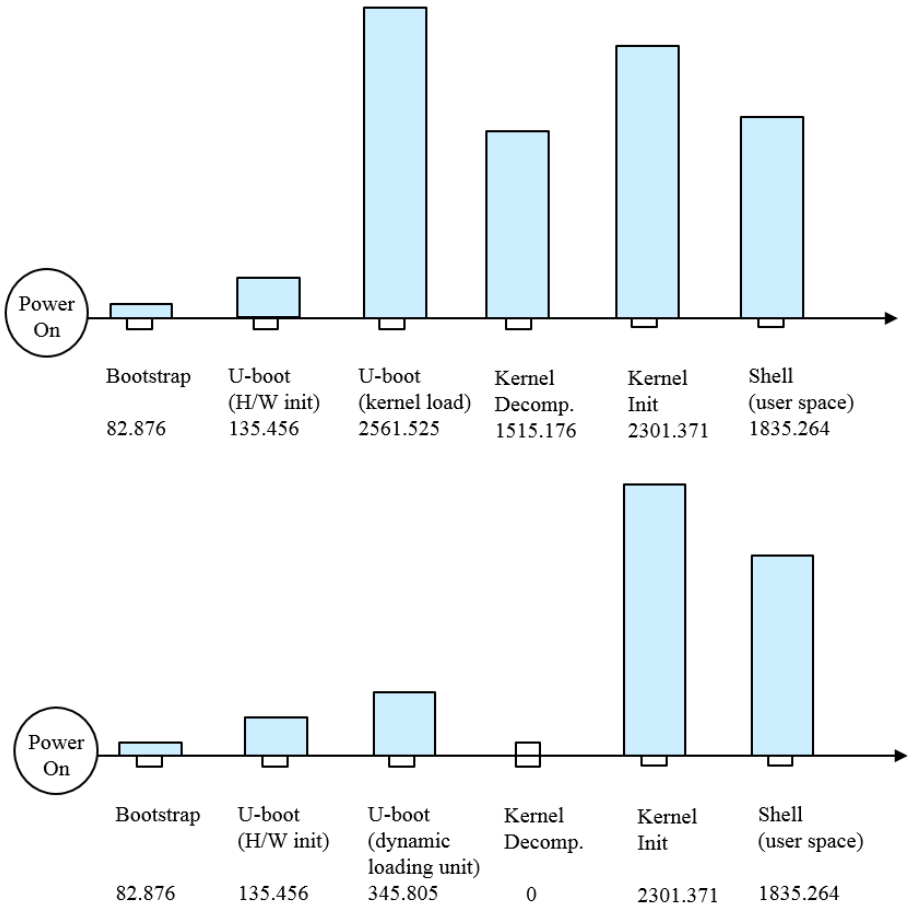

Fig. 5. The time measurement in Full boot and IMD boot (ms)

The system runs u-boot-s4412 and Embedded Linux Kernel 3.0.15 features Samsung Exynos 4412 Cortex-A9 Quad Core 1.4GHz with 1MB L2 cache, 1GByte LP-DDR2 SDRAM, 4GB eMMC Memory. Each of the booting steps is different from the way of estimated time. First, bootstrap step is estimated by grabserial. Second, H/W init and kernel load step is estimated by using grabserial. Third, kernel decompression

and dynamic loading step can be estimated by grabserial. Forth, kernel init step is estimated by using ftrace. Finally, shell step is estimated by bootchart. As shown figure 4, IMD booting can estimate the loading kernel step by distinguishing dynamic area based on section information and calculate the address based on segment address referring to kernel binary. As previously mentioned, by using vmlinux and bzImage and ELF binary and ELF object, program header table figure out the address of dynamic area and section header table can be refer to whether it is dynamic are or not. So on the basis of this point, we evaluated the dynamic kernel loading module operation. Consequently, the time of loading kernel decreases by approximately 86.51% from 2561.525 ms to 345.805ms and the total time reduces by approximately 45.25% from 8431.688ms to 4700.772ms.

5 Conclusion

So far, we probe the booting sequence based on ARM Linux and analyze the vmlinux so load only dynamic area which is changed the runtime. As a result, future memory can play a role of the main memory and it avoid the copying and decompressing mechanism from storage to memory unlike conventional memory based system. We verified that IMD booting reduce the total time by approximately 45.25%. We believe that it can reduce booting time and electric power. We are absolutely certain that the proposed methodology and our analysis will be help to reduce the startup time of many kind of embedded system such as smartphone, automobile, smart watch.

6 References

1. Benjamin C. L., Ping Z., Jun Y., Youtao Z., Bo Z., Engin I., Onur M., Doug B.: Phase-Change Technology and the future of main memory: IEEE J. Micro, 131-141 (2010)
2. Moinuddin K. Q., Vijayalakshmi S., Jude A. R.: Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In: ISCA '09: the 36th annual international symposium on Computer architecture, 24-33 (2009)
3. Justin M., Yixin L., Samira K. Jishen Z., Yuan X., Onur M.: A Case for Efficient Hardware/Software Cooperative Management of Storage and Memory:
4. Kunhoon B., Saena K., Suchang W., Jinhee C.:Boosting up Embedded Linux device : experience on Linux-based Smartphone: the 2010 Linux Symposium, 9-18 (2010)
5. Yang X., Sang N., Alves-Foss J.: Improving the Boot Time of the Android OS : IEEE J. Computer, 1-9 (2013)
6. Myungsik K., Jinchul S., Youjip W.: Selective Segment Initialization: Exploiting NVRAM to Reduce Device Startup Latency: IEEE E. S. Letter, 33-36 (2014)
7. Subramanya R. D., Sanjay K., Anil K., Philip L., Dheeraj R., Rajesh S., Jeff J.: System software for persistent memory. In: EuroSys '14: the Ninth European Conference on Computer Systems (2014)
8. http://www.elinux.org/Boot_Time

Design of Adaptive Hibernation and Booting Sequence for Memory-Disk Integrated System

Yejin Jo

The Graduate School
Yonsei University
Department of Computer Science

Design of Adaptive Hibernation and Booting Sequence for Memory-Disk Integrated System

A Masters Thesis
Submitted to the Department of Computer Science
and the Graduate School of Yonsei University
in partial fulfillment of the
requirements for the degree of
Master of Computer Science

Yejin Jo

December 2014

This certifies that the Masters Thesis
of Yejin Jo is approved.

_____

Thesis Supervisor: Shin-Dug Kim

_____

Joo-Seok Song: Thesis Committee Member

_____

Kyoung-Woo Lee: Thesis Committee Member

The Graduate School
Yonsei University

December 2014

Acknowledge

이 논문은 연세대학교 컴퓨터과학과의 석사과정을 밟으며 저의 연구 과정 및 결과를 포함한 최종 결과물입니다. 이 결과물을 얻고자 많은 사람들의 사랑과 응원이 있었습니다. 비단 저의 노력뿐만 아니라 많은 사람이 협력하여 완성되었음을 알리고자 이 Acknowledge를 기회삼아 감사하다는 말씀을 드리고 싶습니다.

우선 하나님께 감사드립니다. 그리고 부모님께 감사드립니다. 저를 지금까지 병치레 없이 무척이나 건강히 키워주셨으며 헌신적인 사랑을 주셨습니다. 또한 늦게 핀 꽃이 늦게 진다라는 말이 있다며 용기를 주셨던 사랑과 평안의 교회의 담임목사님 박상혁 목사님께 감사하다는 말을 드리고자 합니다. 이 용기는 삼수 후, 저의 대학 입학해인 2007년 이후 학문을 통찰력을 갖고 연구에 매진하는데 도움을 주셨습니다. 항상 할 수 있다며 용기주시고 기도를 아끼지 않으시는 박경애 목사님과 김광수 목사님 감사드립니다. 저의 든든한 버팀목이자 힐링 멤버들인 동진이, 지영이, 민지, 성진이, 유리, 형은이 등 이청멤버들도 감사드립니다. 이들의 헌신과 응원은 석사과정에서 어려움을 극복할 수 있도록 용기와 에너지를 주었습니다.

끊임없이 컴퓨터 학문의 미래를 고민하고 도전을 멈추지 않는 지도교수님, 저희들이 잘되어야 연구실도 잘된다며 항상 용기를 주셨으며 2년간의 연구생활에 중심이 되어 주셨던 김신덕 교수님께 감사드립니다. 또한 공정하고 연구의 논리에 맞도록 성심 성의껏 심사해주셨던 이 논문의 심사위원 송주석 교수님과 이경우 교수님께도 감사드립니다.

연구실의 맏언니로 궂은일도 마다하지 않으며 연구에 도움을 주었던 수경언니.. 정말 고맙습니다. 힘들 때는 용기를 기쁠 때는 함께 즐거워했던 수경언니 저의 연구실 생활의 활력소이자 버팀목이었습니다. 그리고 항상 일찍 오며 성실이 무엇인지 저에게 몸소 보여주었을 뿐만 아니라 연구에도 도움을 주었던 상재 정말 고맙다는 말을 하고 싶습니다. 그 외에도 금춘씨, 정근이, 민호, 각민이, 기현이 고맙습니다.

마지막으로 고등학교 친구들인 백승조, 신유정, 정다이 고맙다고 말하고 싶습니다.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

ABSTRACT

Design of Adaptive Hibernation and Booting Sequence for Memory-Disk Integrated System

YEJIN JO
Dept. of Computer Science
The Graduate School
Yonsei University

As new smart devices emerge, the need for research on booting time for removing overhead has arisen. This research proposes a design of a more compatible booting model for memory-disk integrated architecture by analyzing conventional booting sequence and eliminating kernel executional overhead.

We remove booting overhead by using non-volatile main memory, which can maintain its data even when powered off, and propose a method for

synchronization of the register and cache that has a volatile feature.

Recently, Storage Class Memory (SCM) systems, such as STT-RAM, PCRAM, and ReRAM, have been used in place of DRAM and NAND flash memory. Owing to the non-volatile characteristic of SCM, its use in main memory can reduce the overhead during boot code loading and decompression whenever the system is switched on. Thus, a system with non-volatile memory suffers from less overhead during booting than a system with Dynamic Random Access Memory (DRAM). The purpose of this study is to design a new booting model by analyzing the kernel ELF file and disassembling the current booting sequence to establish a more compatible startup model. To achieve this goal, we disassembled the ELF file format, including the characteristics of the kernel binary. To verify the effectiveness of our theoretical model, the time cost in each step was measured by using several different methods in our experimental setup. The results show that our approach reduces the time cost by approximately 45.25%.

Keyword : Non-volatile Memory, Booting Sequence, Fastboot, Phase-change Memory, Kernel ELF, Memory-Disk Integration

CHAPTER 1

INTRODUCTION

Booting time is an important issue in devices that are smaller and have increased portability, from PCs to smartphones and wearable devices. As new smart devices are developed, we need to eliminate the startup overhead that occurs from powering-on to preparing for use. Traditionally, systems booted on DRAM, which negatively affected the booting time due to the overhead of frequent copying and decompressing of the kernel image. This unnecessary overhead can be avoided by using a non-volatile main memory, like Storage Class Memory (SCM), for the boot sequence instead of DRAM. SCM systems, such as STT-RAM, PCRAM, and ReRAM, can be used in main memory or storage. As shown in Table 1, MRAM has not only a lower read latency than the others, but it does not need to perform an erase operation before modifying data.

	DRAM	Flash(NOR)	Flash(NAND)	MRAM	PRAM
Non-volatile	N	Y	Y	Y	Y
Read Latency (ns)	30	10	50	3-20	20-50
Write Latency (ns)	15	1μs	1ms	3-20	50
Erase Latency (ns)		10ms	0.1ms		120
Accessable Unit	byte	byte	page/block	byte	byte
Endurance	10^{16}	10^{5}	10^{5}	$>10^{15}$	10^{8}

Table 1. Characteristics of memories

Owing to the non-volatile character of SCM, its use in main memory can reduce the overhead during boot code loading and decompression whenever the system is switched on. This research proposes a fast boot sequence by removing unnecessary overheads and providing a more compatible booting system based on non-volatile main memory. To design this fast and more compatible booting system, we propose a change in the runtime system by using kernel binaries and provide a method for synchronization between the memory and the cache. Even it has volatility, such as L1 cache, last-level cache, and register, regardless of memory. We design a booting system to determine whether it has initialized by reading the Executable and Linkable Format (ELF) file that stores the rules for further linking and execution.

As new smart devices emerge, the need for research on booting time for removing overhead has arisen. This research proposes a design of a more compatible booting model for memory-disk integrated architecture by analyzing the conventional booting sequence and eliminating kernel executional overhead.

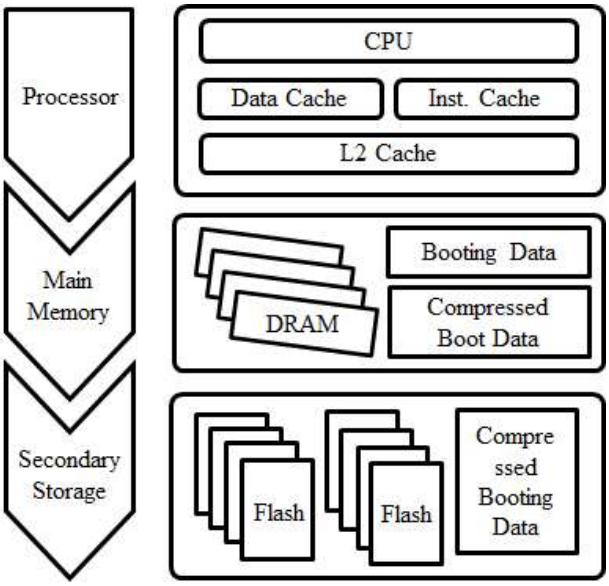

Figure 1. The Conventional Memory System in the view of Booting Data

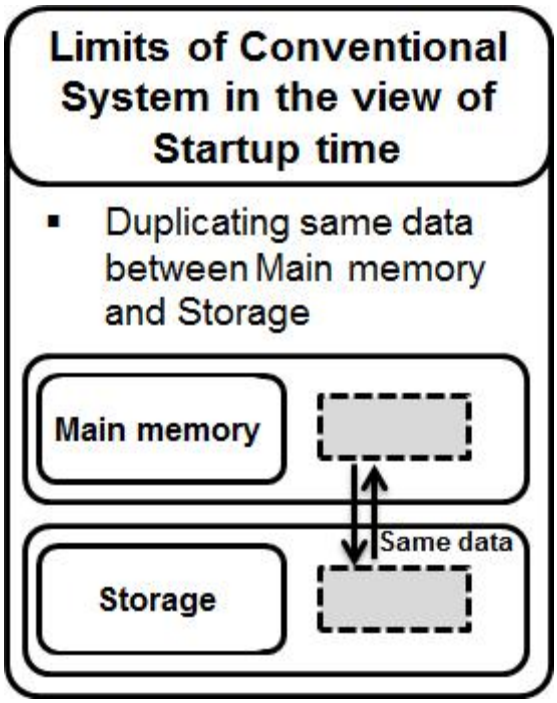

Figure 2. The Limitation of Conventional Booting in the view of Memory Hierarchy

As shown in Figures 1 and 2, conventional memory systems have some booting sequence limitations in view of memory hierarchy. Additionally, there is the same booting data between the main memory and the storage. We regarded this as unnecessary overhead in view of integrated memory-disk architecture and removed waste by analyzing the current booting sequence and implementing hibernation in the kernel.

We removed the booting overhead by using non-volatile main memory, which can maintain its data even when powered off and proposed a method for synchronization of the register and cache that has a volatile feature.

The purpose of this study is to design a new booting model by analyzing the kernel ELF file and disassembling the current booting sequence to establish a more compatible startup model. To achieve this goal, we disassembled the ELF file format, including the characteristics of the kernel binary. To verify the effectiveness of our theoretical model, the time cost of each step was measured using several different methods in our experimental setup. We proposed many kinds of experimental methods for measuring the time cost, from powering-on to loading the shell.

CHAPTER 2

RELATED WORK

In this chapter, we explain several studies on integrated memory-disk architecture and fastboot. First, we introduce the concept of integrated architecture, and we then explain how to reduce the booting time by using NAND flash memory.

2.1 Integrating Memory Management Techniques

This research shows that processes and files can be allocated in the same size to improve management between the main memory and storage. Address space can be managed linearly on the same level rather than in the hierarchical model. As shown in Figure 3, the files and processes can be allocated to the same physical address space and managed by virtual memory.

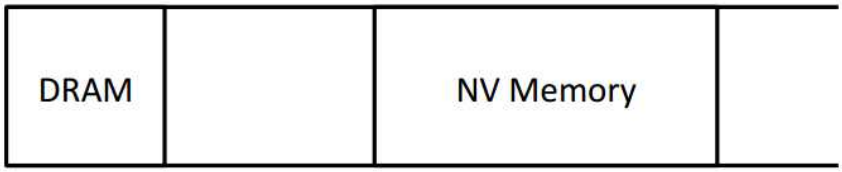

Figure 3. Linear Physical address space between DRAM and Non Volatile Memory

As shown in Figures 3 and 4, the same memory hierarchy is managed by allocating the same units of processes and files to the virtual memory and using the same physical memory address space between DRAM and Non-Volatile memory.

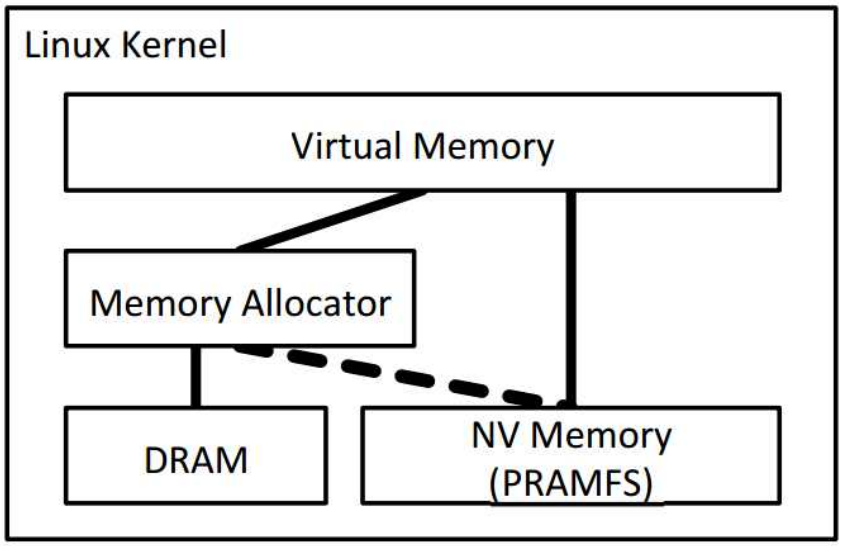

Figure 4. Physical address space and conceptual diagram of Integrating memory management

2.2 Fastboot Techniques

The following is one of the fastboot techniques for analyzing and optimizing the kernel image. This research shows information about each section of the kernel and initializes the section only if it is not read-only, such as the init section and the data section, as shown in Figure 5.

Section	Attributes	Address (Hex)	Size (KB)
.text.head	CONTENTS, ALLOC, LOAD, **READONLY**, CODE	c0008000	1.0
.init	CONTENTS, ALLOC, LOAD, CODE	c00083e0	167.0
.text	CONTENTS, ALLOC, LOAD, **READONLY**, CODE	c0032000	5008.2
.text.init	CONTENTS, ALLOC, LOAD, **READONLY**, CODE	c05160e4	0.2
__ksymtab	CONTENTS, ALLOC, LOAD, **READONLY**, DATA	c0517000	19.8
__ksymtab_gpl	CONTENTS, ALLOC, LOAD, **READONLY**, DATA	c051bf40	7.9
_ksymtab_strings	CONTENTS, ALLOC, LOAD, **READONLY**, DATA	c051dea8	60.0
__param	CONTENTS, ALLOC, LOAD, **READONLY**, DATA	c052ce80	4.4
.data	CONTENTS, ALLOC, LOAD, DATA	c052e000	164.1
.init.rodata	CONTENTS, ALLOC, LOAD, **READONLY**, CODE	c0557040	0.3
.bss	ALLOC	c0557140	248.2

Figure 5. The classified kernel attributes

As shown in Figure 6, the proposed system loads the read-only and rewritable sections after decompressing the zImage in DRAM from the disk or executing the booting by loading only the rewritable section on the non-volatile main memory from the disk.

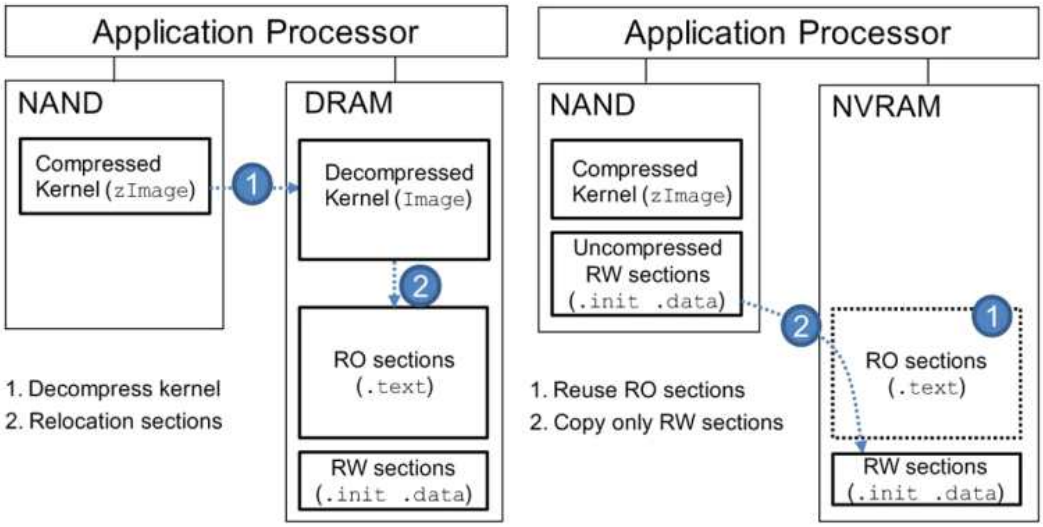

Figure 6. Startup Procedures

2.3 Hibernation Techniques

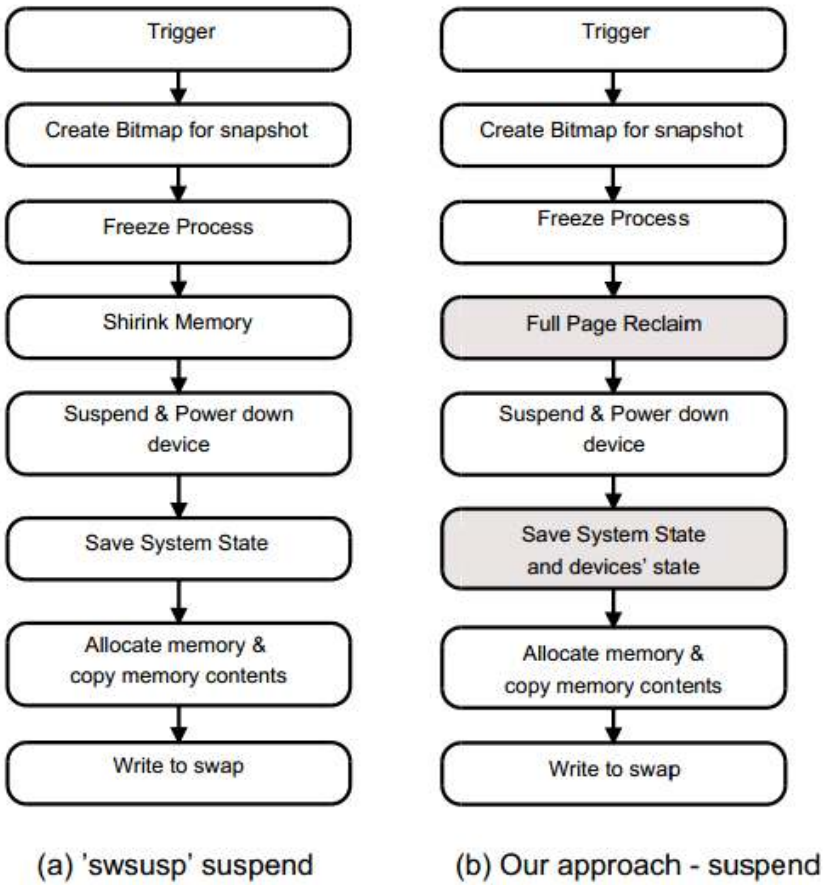

Figure 7. 'swsusp' suspend and 'reclaim and save' suspend

As shown in Figure 7, this research proposes “swsusp” suspend and “reclaim and save” suspend by modifying the shrink memory stage to achieve full page frame reclamation, which is associated with minimizing

the size of the snapshot image by reclaiming most of the memory required except for the essential code and data required by hibernation.

As shown in Figure 8, this research proposes snapshot image methods, including a minimal number of pages, and indicates that it can enter the running state simply by restoring the essential snapshot image. The other pages requested by users will be restored on demand by the Linux memory management mechanism.

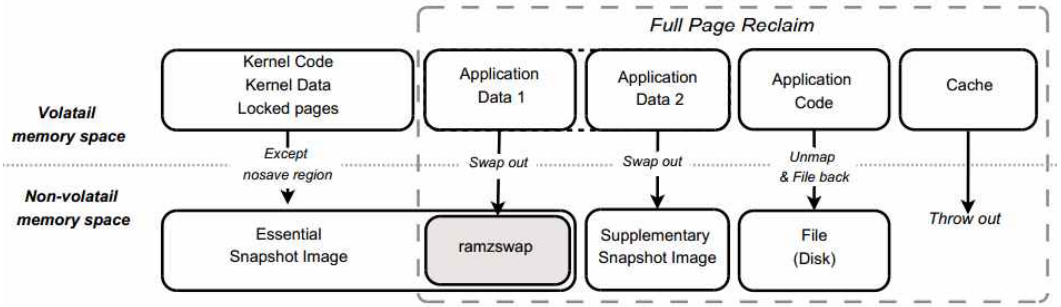

Figure 8. The methods for snapshot image

As shown in Figure 9, the swsusp resume is started after all devices are initialized, and at the suspend device stage, it places them in the resumable stage. Thus, if there are more devices or the device complexity increases, the time required to initialize and suspend the device will increase. However, as shown in Figure 9-(b), the device initialization and suspend stages are removed and the restore device state is added to the restore system state stage.

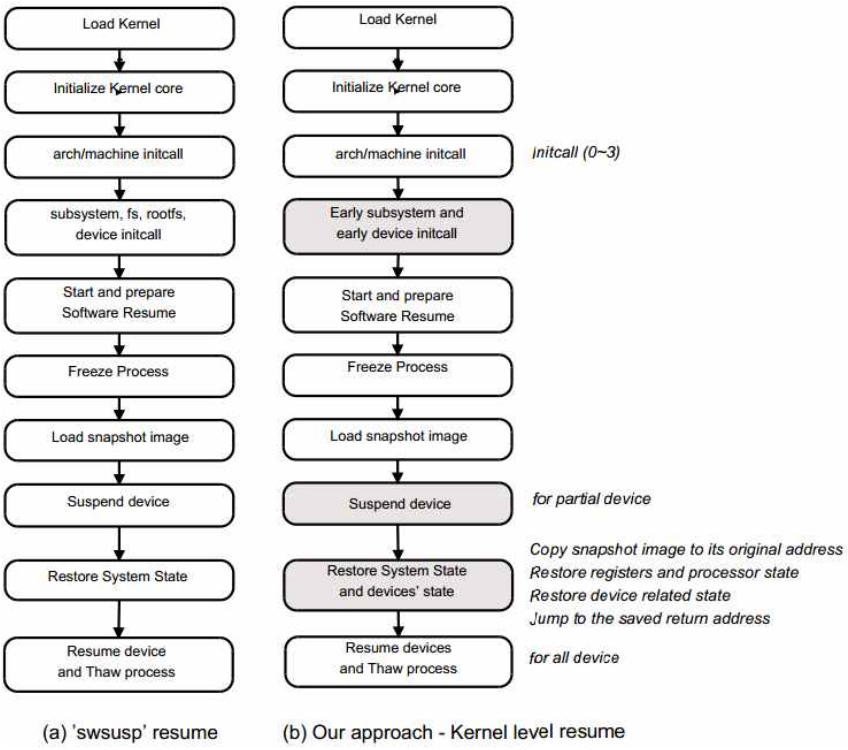

Figure 9. booting sequence flow by hibernation

2.4 Hibernation and Fastboot Hybrid Techniques

One of the methods for fast loading of the OS that recovers the boot image from the disk to the memory is called hibernation. The hibernation is a suspend-resume technique that removes the need to initialize the Linux kernel and can be combined with approaches to enhance secondary disk read speed and shrinking of the suspend image.

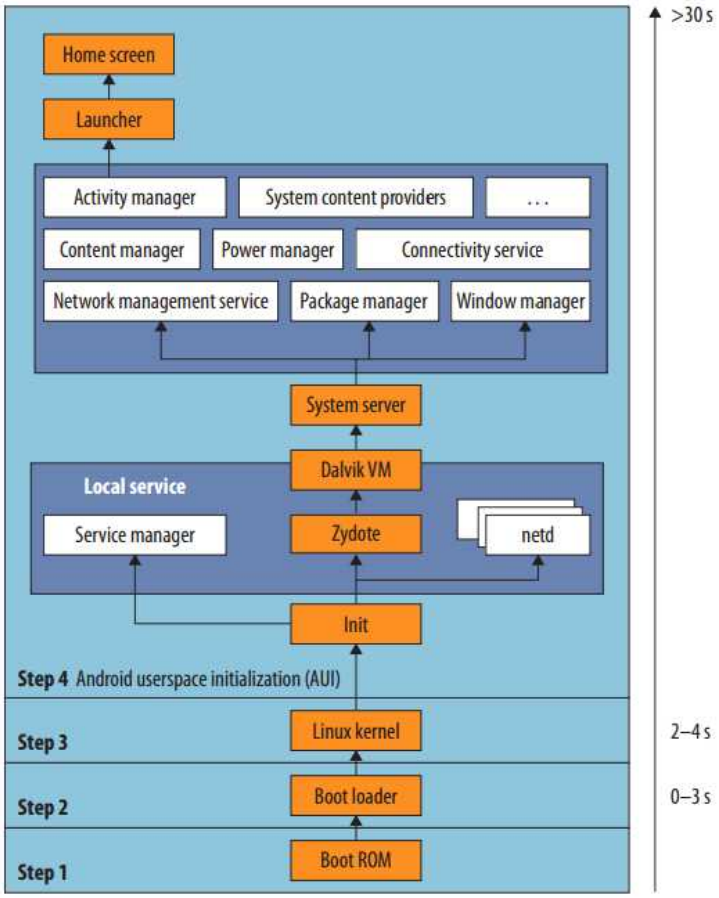

Figure 10. Android Booting Sequences

As shown in Figure 10, this research shows that upgrading the performance by optimizing the Android user-space initialization occupies the highest percentage of the total boot time.

According to this paper, the booting time is decreased by not only reducing the booting time through AUI stage hibernation, but also optimizing the U-boot stage.

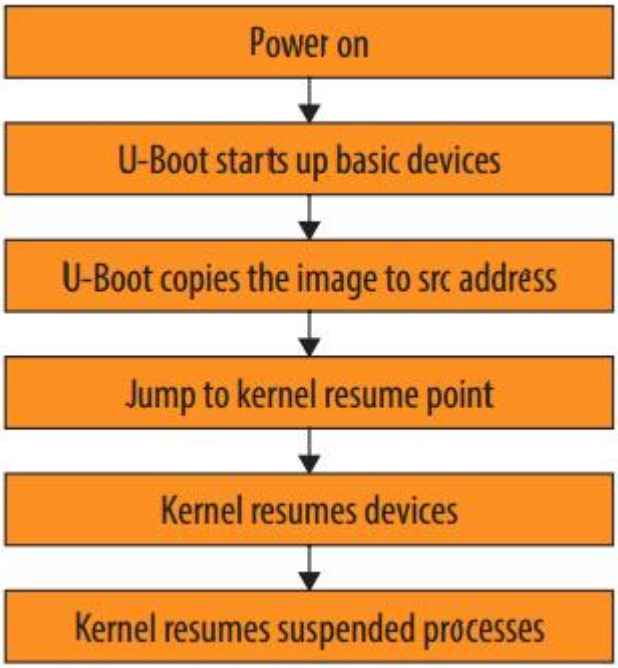

Figure 11. Booting Steps in hibernation and fastboot Hybrid technique

Figure 11 gives a flow chart for the hibernation and fastboot hybrid technique. This keeps both the U-boot image and the kernel in the disk for hibernation.

Therefore, we suggest using the hybrid technique for achieving quick booting by optimizing the U-boot and kernel stage.

CHAPTER 3

Backgrounds

In this chapter, we explain the multiple steps of the conventional booting sequence and analyze each step based on ARM Linux.

All devices managed by an operating system need to perform booting steps to start the operating system, which facilitates communication between the hardware and the software. During booting, the device prepares its hardware resources, such as processors, memory, and storage management modules, for use. In the case of embedded systems, the computer environment differs from that of a PC or a hand-held device in that they use U-boot, a flash memory, and an application processor rather than BIOS, a HDD, and a CPU.

The first step in the booting process is running the bootloader, which is called U-boot. The work of U-boot is divided into two stages: (a) Initializing the hardware resources and (b) preparing them for the kernel. During the hardware initialization stage, U-boot defines the interrupt vector table and sets the supervisor mode and the watchdog timer. It then disables interrupts and sets the CPU clock. It then forks memsetup.S to initialize

the memory using the cpu_init_crit function. It relocates the armboot function on the RAM after returning from start.s. In the next stage, U-boot configures the stack pointer to run C routines and forks the start_armboot function to initialize the resources on the embedded board before returning to the main function.

The second step of booting involves the running of the kernel. The vmlinux file includes additional data, like debugging information, which can change the bzImage. The bzImage consists of three objects used in booting: head.o, which includes the initializing code for the kernel, misc.o, which includes the decompressing code, and piggy.o, which includes the kernel core code. After the bzImage is loaded by the bootloader, the code is decompressed and initialized for running the kernel in piggy.o. The complex nature of the kernel results from copying the kernel image from the storage and then de-compressing the code in the memory, which leads to overhead. In the case of non-volatile main memory, this overhead is reduced due to its non-volatile feature, which involves loading and decompressing the kernel image only once during the boot time.

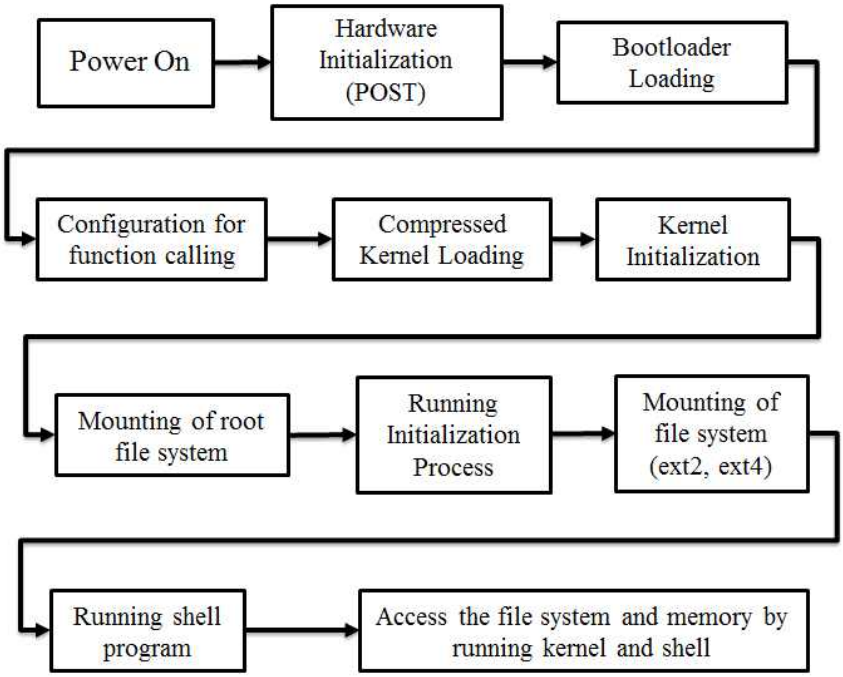

Figure 12. The Common Booting Sequences based on PC

CHAPTER 4

Overall Methodology

This chapter gives a full explanation of methodologies of how to improve the performance for the booting sequence and hibernation by using a non-volatile memory disk system.

There are many methods for reducing overhead in engineering. For example, overhead can be removed by eliminating unnecessary functions or reordering by priority. In addition, the performance can be upgraded through parallel processing. Finally, overhead can be removed and faster programs can be achieved by optimizing each function. In this research, we improve the performance by optimizing the functions and algorithm.

4.1 Overall Model

In this chapter, we explain several researches about fast boot and integrated memory-disk architecture.

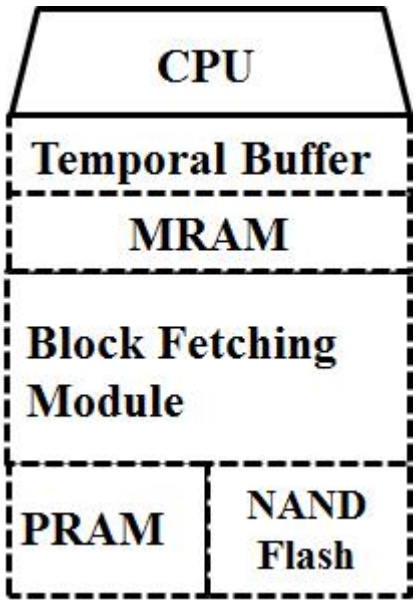

Figure 13. Memory Architecture for removing booting overhead

4.2 ELF Dynamic Loading Module

On Linux systems, vmlinux is an executable binary file that consists of a program header table and a section or segment and section header table. The program header table is a set of entries in which one entry refers to one segment. This table includes segment information, which contains the type, location, size, virtual address, and physical address of the segment. Therefore, this table can calculate the dynamic area, which has a changeable runtime. The section header table contains the section

information, which includes the location of the section and its features. This information can be used by the linker and debugger to interpret the ELF object. The linker recognizes the location of the section header table in front of the ELF header, and the link stage can analyze the section or view the segment. The loader finds the location of the program header table by reading the ELF header; on this basis, it determines which segment is loaded in the memory.

The vmlinux file also includes the architecture of the ELF binary format. This ELF binary involves the file structure, which can change the machine code using the linker. The ELF file contains rules for linking. Because of this, the non-volatile memory can be loaded in the area that changes during runtime; thus, by using vmlinux, bzImage, ELF binary, and ELF object, the program header table can refer to the address of a dynamic area, and the section header table can determine whether this is a dynamic area.

The dynamic loading module identifies the areas that change during runtime. The vmlinux located in the non-volatile memory can extract the dynamic area and initialize the bzImage, which is an executable kernel image. The possible reason for the IMD boot is that the ELF format has unique rules for identifying its sections and segments.

The ELF dynamic module finds the dynamic area by running three units: The ELF verified unit, the ELF calculated unit, and the dynamic loading module, which identifies the areas that change during runtime. The vmlinux located in the non-volatile memory can extract the dynamic area and

initialize the bzImage, which is an executable kernel image. The possible reason for this method is that the ELF format has unique rules for identifying its sections and segments.

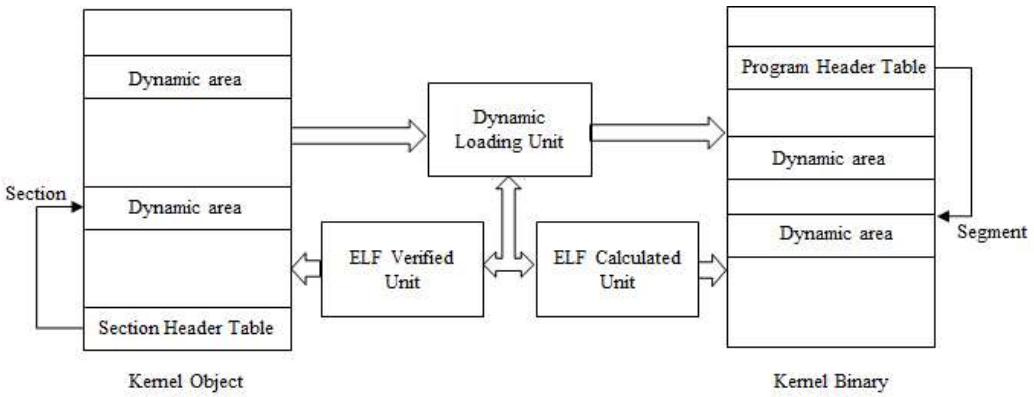

Figure 14. The Kernel Runtime Loading Module

4.3 Kernel Reloadable Optimization Module

The objective of the general hibernation technique is to copy the compressed kernel, decompress its image, initialize and run the kernel image, and finally restore the user space area of the virtual memory. That is to say, it restores the user space area after preparing to run the kernel so that the unnecessary overhead remains while copying the kernel image and decompressing its files. However, we propose a means of restoring the kernel space of the virtual memory in our research. We used the kernel dynamic optimization module in charge of the hibernation aspect of the kernel.

4.3.1 Kernel Reloadable Array Module

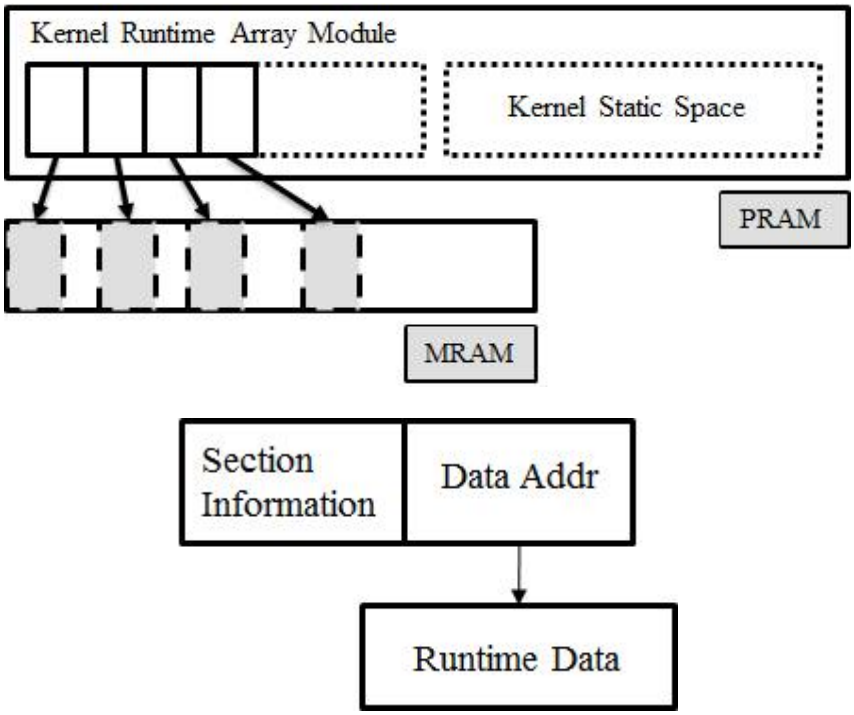

Figure 15. The Kernel Runtime Array Module in view of Virtual Memory and its Data Structure at Module

As shown in Figure 15, the kernel reloadable array module means that relocated data, such as metadata that is frequently accessed or runtime data, can initialize the memory for fast execution. However, the problem of array exists. In the case of metadata, it can be frequently changeable whenever

startup occurs. If the data is being deleted, an empty area exists that necessitates the use of migration.

4.3.2 Kernel Runtime List Module

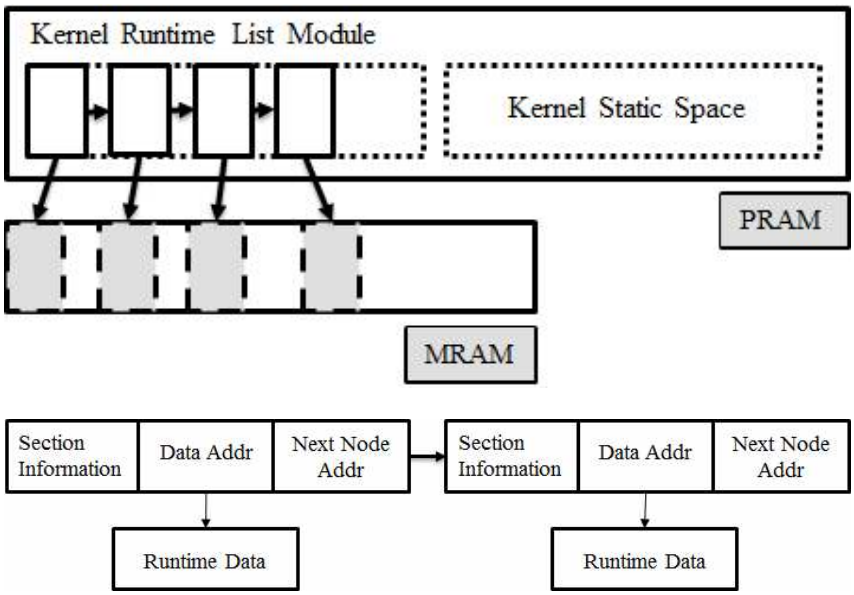

Figure 16. The Kernel Runtime List Module in view of Virtual Memory and its Data Structure at Module

In the case of frequently modified data, such as metadata, the data structure for deleting can be improved by adding a linked list to overcome the problem of frequent modifications by using the list. However, the

problem of booting time remains. For example, if the size of the section is bigger, it needs more time for searching, increasing its booting time. This problem needs to be overcome.

4.3.3 Kernel Runtime Hash Module

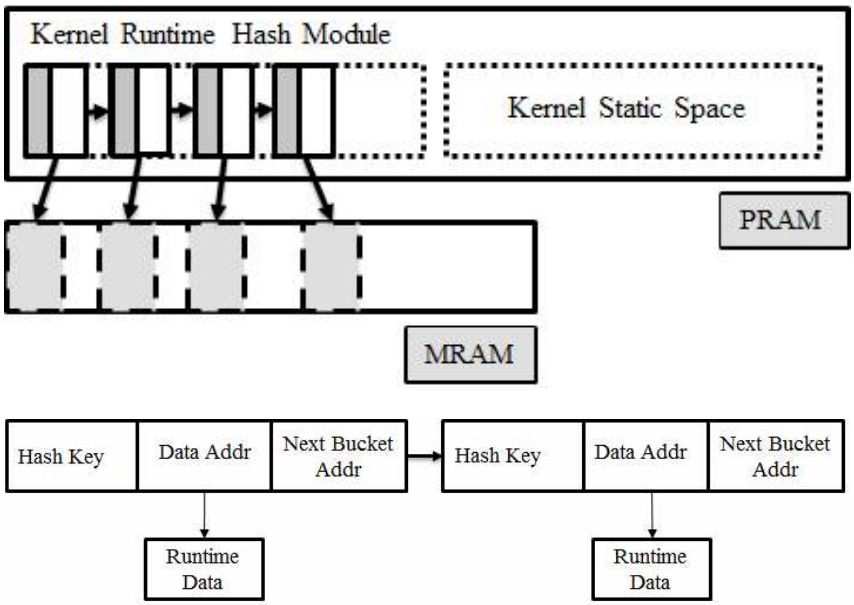

Figure 17. The Kernel Runtime Hash Module in view of Virtual Memory and its Data Structure at Module

Therefore, we propose that the kernel reloadable hash module can overcome two problems: Increased search time for bigger data and the need for migration to fill empty spaces after deleting and adding. As shown in

Figure 17, a problem remains in the hash; for example, in the worst case, it is similar to the list when there is only one hash node. We concluded that the best structure can be achieved by using hashing with chaining.

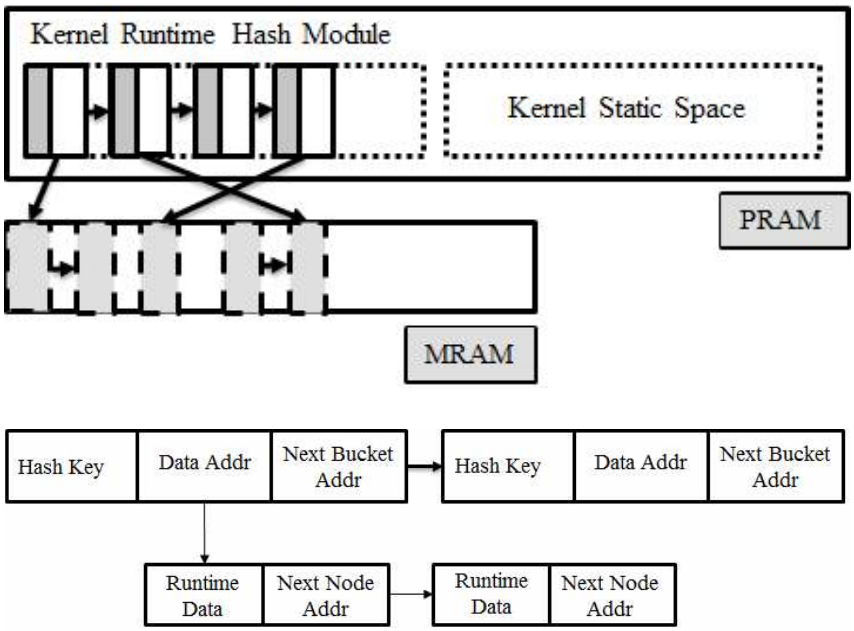

Figure 18. The Kernel Runtime Hash Module by using chaining in view of Virtual Memory and Data Structure at Module

4.4 Recovery Synchronization Module

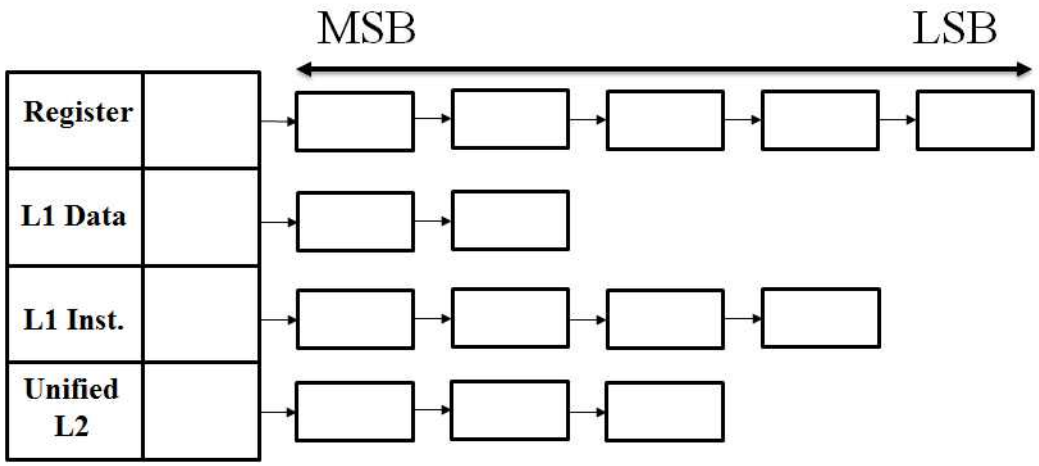

Figure 19. The Data Structure for recovery synchronization module

The conventional memory hierarchy is normally structuralized as multiple levels of cache, main memory, and secondary disk.

There are characteristics of hierarchy even after integrating the memory-disk, such as the register and the cache, because of volatility. Therefore, we propose the use of recovery synchronization module for this issue.

That is to say, even though a non-volatile memory-disk is used, it needs to synchronize between the register and the cache. If this module is not

present, the same result achieved by a conventional memory system booting is found, because the program counter was initialized when the power was off. As shown in Figure 19, we designed each level to be associated with a hash key and adopted a last-in-first-out stack structure.

CHAPTER 5

EXPERIMENTAL RESULTS

In this chapter, we describe and analyze the experimental outcome. In particular, we explain the environment of the experiment and its methodology for estimating the booting time by stages.

There are two methods for estimating booting time. One way is using an oscilloscope for obtaining the best measurements. The other way is using software, such as grabserial, Linux Trace Toolkit, Kernel Function Trace, printk function, Oprofile, or bootchart.

An oscilloscope is used for receiving the signal when the GPIO interrupt occurs at each booting step. Alternately, using software obtains each value at each booting stage. In this paper, we propose the second method for measuring booting time to obtain the value at each booting step.

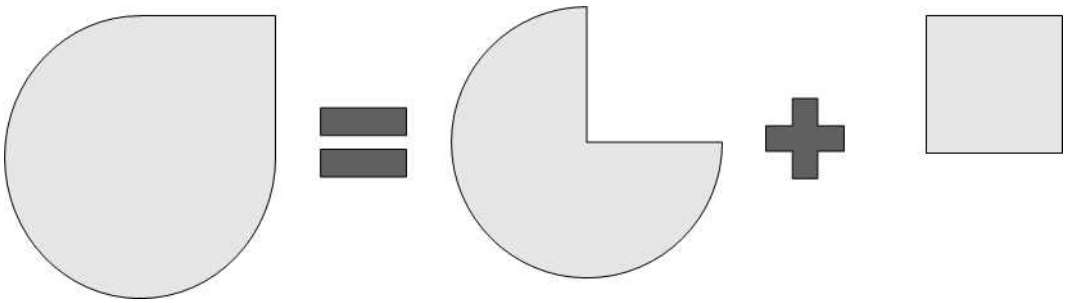

Figure 20. The way for measuring the area when it cannot get the value directly

As shown in Figure 19, we can obtain the total area by dividing the obtainable area. Similarly, the total booting time can be used to acquire the value by separating the obtainable steps, such as the BIOS stage, the U-boot stage, the kernel and init stage, and the user program stage. In this paper, we propose a method for assessing the total booting time and record the experimental results in a graph.

5.1 Experimental Methodology

The system runs the U-boot-s4412 and Embedded Linux Kernel 3.0.15 features on a Samsung Exynos 4412 with a Cortex-A9 Quad Core 1.4 GHz processor and a 1 MB L2 cache, 1GB LP-DDR2 SDRAM, and a 4-GB eMMC memory. The time for each of the booting steps is estimated independently. For three of the five steps, it is estimated using grabserial: first, for the bootstrap step; second, for the H/W init and kernel load step; and third, for the kernel decompression and dynamic loading step. Next, the time for the kernel init step is estimated using ftrace. Finally, the time for the shell step is estimated using bootchart.

There are many methods for estimating the booting time. The first way to measure the startup time is by using an oscilloscope (the hardware method). The second way is by using software to measure each step.

In this research, we estimated the booting time by using software.

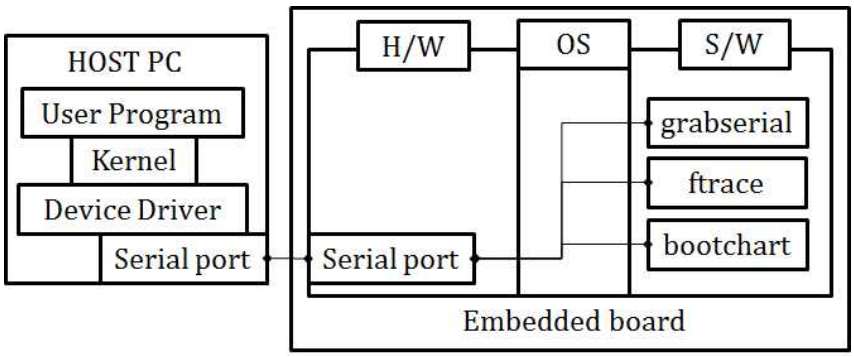

Figure 21. Experimental Methodology

As shown in figure 21, we propose a method for measuring booting time by using tools for to obtain the value at each booting step.

As previously mentioned, we can acquire the total area by dividing the obtainable area. Similarly, the total booting time can be used to attain the value by separating the obtainable steps, such as the BIOS stage, the U-boot stage, the kernel and init stage, and the user program stage. In this paper, we propose a method for assessing the total booting time and record the experimental results in a graph.

5.2 Dynamic Space Time Measurements

In this section, we estimate the execution time of each structure's dynamic kernel area. The current memory system loaded the kernel from the disk, and we eliminated the static section of binary data. Figure 21 shows the results of the execution of the frequently changeable data, such as metadata or runtime data, on the kernel image. The x-axis indicates the size of the data, and the y-axis values (i.e. 0.05, 0.1, 0.15, and 0.2) represent the execution time of the data. According to our results, the hash structure is faster than array or list structures, and the bigger data requires more loading time.

Figure 22. The execution time on each structures dynamic area and its size of the kernel

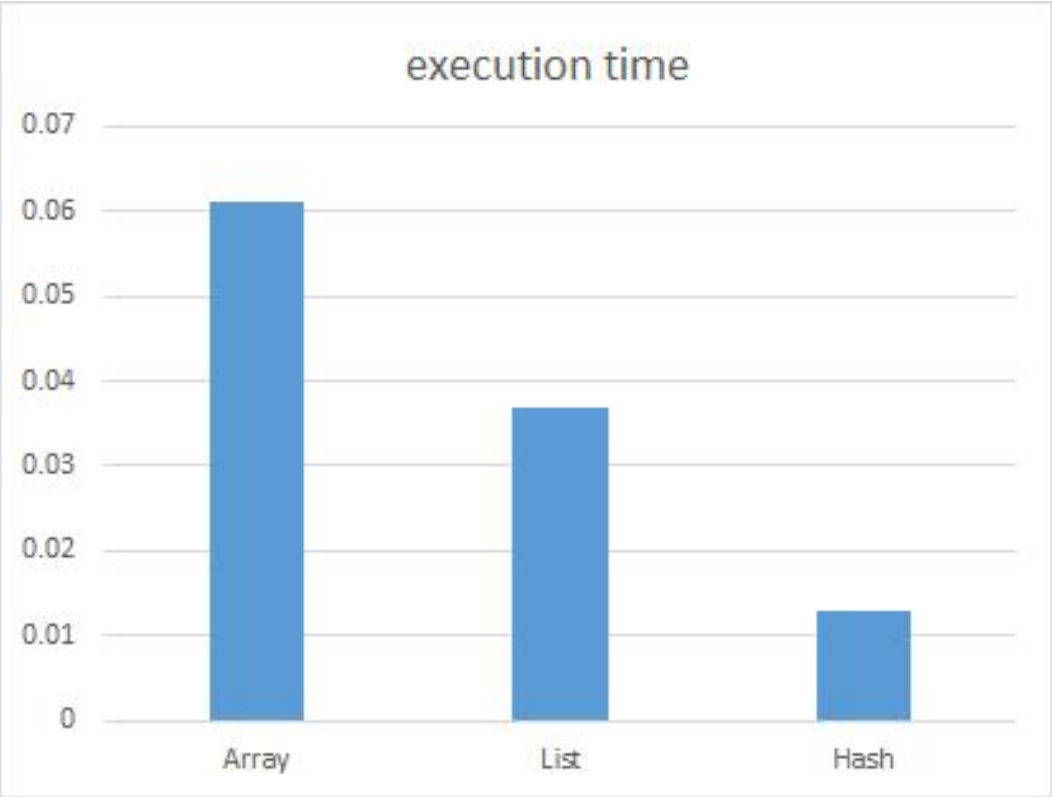

Figure 23. The execution time on each structures dynamic area of the kernel

As previously mentioned, the x-axis indicates the size of the data, and the y-axis values (i.e. 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, and 0.07) represent the execution time of the data. According to our results, the hash structure is faster than the array or list of runtime data.

5.3 Booting Time Measurements

As previously mentioned, we can obtain the total area by dividing the obtainable value. Similarly, the total booting time can be used to acquire the value by separating the obtainable steps, such as the BIOS stage, the U-boot stage, the kernel and init stage, and the user program stage. We propose a method for assessing the total booting time and record the experimental results in a graph.

The x-axis indicates each booting step, such as bootstrap, U-boot (hardware initialization), U-boot (loading the kernel), decompressing the kernel, initializing the kernel, and finally running the shell. The y-axis represents the booting time.

According to our results, the steps of loading the kernel, initializing the kernel, and running the shell were markedly decreased by the optimization.

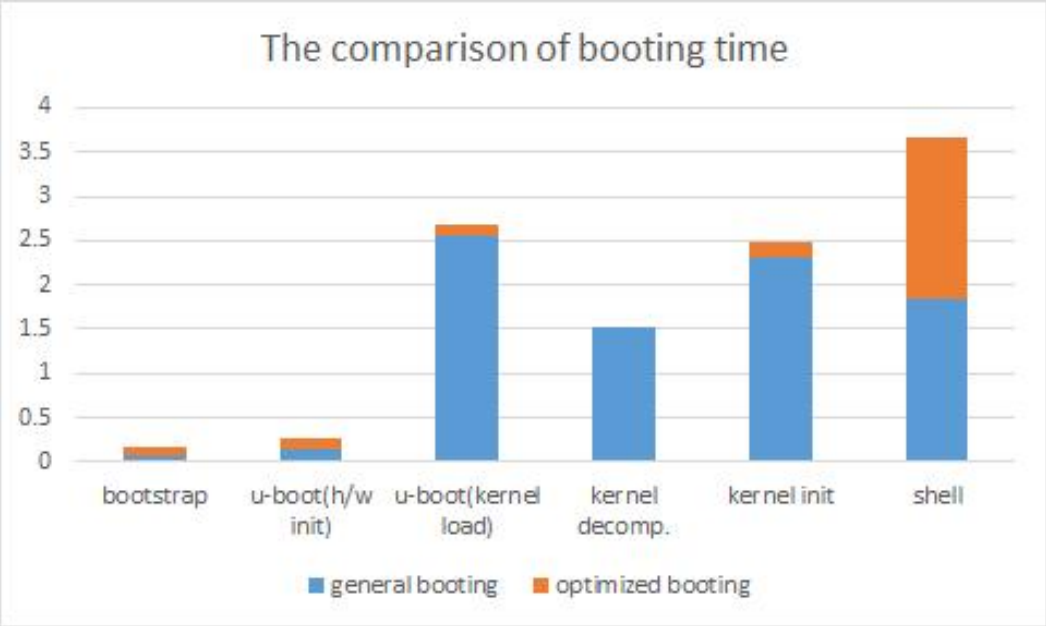

Figure 24. The Comparison between general booting time and optimized booting time

5.4 Memory I/O Measurement

As previously mentioned, we can obtain the total area by dividing the obtainable value. And we can estimate time of each booting steps by using several tool such as grabserial and ftrace.

The x-axis indicates each booting step, such as bootstrap, U-boot (hardware initialization), U-boot (loading the kernel), decompressing the kernel, initializing the kernel, and finally the running shell. The y-axis represents the booting time.

According to our results, the steps of loading the kernel, initializing the kernel, and running the shell were markedly decreased by the optimization.

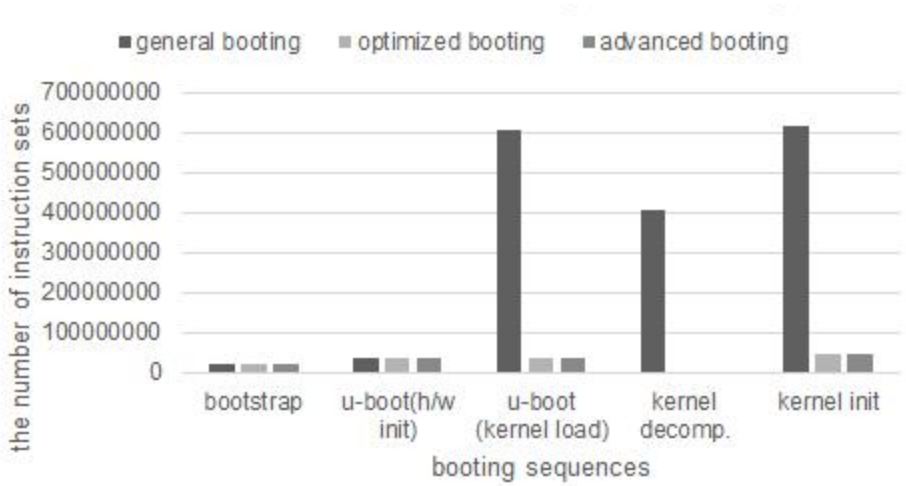

Figure 25. The trace data of booting sequence

CHAPTER 6

CONCLUSION

In this study, we evaluated booting sequence based on ARM Linux and analyzed the vmlinux loading of the kernel in the dynamic area only, which can be changed during runtime. We proposed the use of SCM for the main memory to avoid the copying and decompressing mechanism from storage to memory, unlike in the conventional memory-based system. We can obtain the total area by dividing the obtainable area. Similarly, the total booting time can be used to acquire the value by separating the obtainable steps, such as the BIOS stage, the U-boot stage, the kernel and init stage, and the user program stage. In this paper, we propose a method for assessing the total booting time and record the experimental results in a graph. We verified that IMD booting reduces the total time by approximately 45.25%.

We concluded that it can reduce booting time and electric power consumption. Even though, it is too short time and energy saving, we expect that we improve our life because of increasing our smart device

tremendously. We are confident that the proposed methodology and analysis will help to reduce the startup time of many embedded systems, such as those in smartphones, automobiles, and smart watches.

REFERENCES

[1] Benjamin, C. L., Z. Ping, Y. Jun, Z. Youtao, Z. Bo, I. Engin, M. Onur, and B. Doug. “Phase-Change Technology and the future of main memory.” IEEE J. Micro. (2010): 131 – 141. Print.

[2] Moinuddin, K. Q., S. Vijayalakshmi, and A. R. Jude. “Scalable High Performance Main Memory System Using Phase-Change Memory Technology.” ISCA '09: The 36th Annual International Symposium on Computer Architecture. (2009): 24 – 33. Print.

[3] Justin, M., L. Yixin, K. Samira, Z. Jishen, X. Yuan, and M. Onur. “A Case for Efficient Hardware/Software Cooperative Management of Storage and Memory.” Proceedings of the 5th Workshop on Energy-Efficient Design (WEED). (2013): 1 – 7. Print.

[4] Kunhoon, B., K. Saena, W. Suchang, and C. Jinhee. “Boosting up Embedded Linux Device: Experience on Linux-based Smartphone.” The 2010 Linux Symposium. (2010): 9 – 18. Print.

[5] Yang, X., N. Sang, and J. Alves-Foss. “Improving the Boot Time of the Android OS.” IEEE J. Computer. (2013): 1 – 9. Print.

[6] Myungsik, K., S. Jinchul, and W. Youjip. “Selective Segment Initialization: Exploiting NVRAM to Reduce Device Startup Latency.” IEEE E. S. Letter. (2014): 33 – 36. Print.

[7] Subramanya, R. D., K. Sanjay K, K. Anil, L. Philip, R. Dheeraj, S. Rajesh, and J. Jeff. “System software for persistent memory.” EuroSys '14: The Ninth European Conference on Computer Systems. (2014)

[8] Chang, Che-Wei, Chuan-Yue Yang, Yuan-Hao Chang, L. Philip, and Tei-Wei Kuo. “Booting Time Minimization for Real-Time Embedded Systems with Non-Volatile Memory.” IEEE J. (2014): 847–859. Print.

[9] Baek, S. J., J. M. Choi, D. H. Lee, and S. H. Noh. “Energy-efficient and high-performance software architecture for storage class memory.” ACM Transactions on Embedded Computing Systems. (2013)

[10] Jung, J. Y., and S. Y. Cho. “Memorage: Emerging persistent RAM-based malleable main memory and storage architecture.” ICS '13: The 27th International ACM Conference on International Conference on Supercomputing. (2013): 115–126. Print.

[11] Badam, A. “How Persistent Memory Will Change Software Systems.” Computer J. (2013): 45–51. Print.

[12] Dhiman, G., R. Ayoub, and T. Rosing. “PDRAM: A hybrid PRAM and DRAM main memory system.” Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE. (2009): 664–669. Print.

[13] Freitas, R. F., and W. W. Wilcke. “Storage-class memory: The next storage system technology.” IBM Journal of Research and Development. (2008): 439–447. Print.

[14] Burr, G. W., B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy. “Overview of candidate device technologies for

storage-class memory.” IBM Journal of Research and Development. (2008): 449–464. Print.

[15] Oikawa, Shuichi. “Integrating memory management with a file system on a non-volatile main memory system.” SAC '13: The 28th Annual ACM Symposium on Applied Computing. (2013): 1589–1594. Print.

[16] Chang, Bing-Jing, Yuan-Hao Chang, Hung-Sheng Chang, Tei-Wei Kuo, and Hsiang-Pang Li. “A PCM translation layer for integrated memory and storage management.” CODES '14: Hardware/Software Co-design and System Synthesis. (2014)

[17] NcPRAM, http://www.samsung.com/global/business/semiconductor/products/fusionmemory/Products_NcPRAM.html

[18] 256K x 16-Bit 3.3-V Asynchronous Magnetoresistive RAM, MR2A16A http://www.freescale.com/files/microcontrollers/doc/data_sheet/MR2A16A.pdf

[19] Condit, Jeremy, Edmund B. Nightingale, Christopher Frost, Engin Ipek, Benjamin Lee, Doug Burger, and Derrick Coetzee. “Better I/O through byte-addressable, persistent memory.” SOSP '09: ACM SIGOPS 22nd Symposium on Operating Systems Principles. (2009): 133–146. Print.

[20] Coburn, Joel, Adrian M. Caulfield, Ameen Akel, Laura M. Grupp, Rajesh K. Gupta, Ranjit Jhala, and Steven Swanson. “NV-Heaps: Making persistent objects fast and safe with next-generation, non-volatile memories.” ASPLOS XVI: Architectural Support for Programming Languages and Operating Systems. (2011): 105–118. Print.

[21] Wu, Xiaojian, and A. L. Narasimha Reddy. “SCMFS: A file system

for storage class memory.” SC '11: High Performance Computing, Networking, Storage and Analysis. (2011)

[22] Bailey, Katelin, Luis Ceze, Steven D. Gribble, and Henry M. Levy. “Operating system implications of fast, cheap, non-volatile memory.” HotOS'13: USENIX conference on Hot topics in operating systems. (2011)

[23] Park, Hyunsun, Sungjoo Yoo, and Sunggu Lee. “Power management of hybrid DRAM/PRAM-based main memory.” DAC '11: Design Automation. (2011): 59 – 64. Print.

[24] Kultursay, E., M. Kandemir, A. Sivasubramaniam, and O. Mutlu. “Evaluating STT-RAM as an energy-efficient main memory alternative.” ISPASS Performance Analysis of Systems and Software. (2013): 256 – 267. Print.

[25] Fang, Kun, Long Chen, Zhao Zhang, and Zhichun Zhu. “Memory Architecture for Integrating Emerging Memory Technologies.” PACT: Parallel Architectures and Compilation Techniques. (2011): 403 – 412. Print.

[26] Duo, Xianlu Luo, Liu Kan Zhong, Dan Zhang, Yi Lin, and Jie Dai Weichen Liu. “Improving PCM endurance with randomized address remapping in hybrid memory system.”

[27] Oikawa, Shuichi, and Satoshi Miki. “File-Based Memory Management for Non-volatile Main Memory.” COMPSAC '13: Computer Software and Applications Conference. (2013): 559 – 568. Print.

[28] Joe, I. H., and S. C. Lee. “Bootup time improvement for embedded Linux using snapshot images created on boot time.” ICNIT:

Next-Generation Information Technology. (2011): 193 – 196. Print.

[29] Lee, Y. K., H. B. Park, and C. Jeon. “Fast booting based on NAND flash memory.” RACS, 12: Research in Applied Computation Symposium. (2012): 451 – 452. Print.

[30] MRAM Technical Guide, http://cache.freescale.com/files/microcontrollers /doc/brochure/BRMRAMTECHGUIDE.pdf

[31] Fast Non-Volatile RAM Product Superior price and performance from a source you can trust http://cache.freescale.com/files/memory/doc/fact_sheet/ BRMRAMSLSCLTRL.pdf

[32] Oikawa, Shuichi. “Independent Kernel/Process Checkpointing on Non-Volatile Main Memory for Quick Kernel Rejuvenation.” 27th International Conference. (2014): 233 – 244. Print.

[33] Bheda, R. A., J. A. Poovey, J. G. Beu, and T. M. Conte. “Energy-efficient Phase Change Memory-based main memory for future high performance systems.” IGCC 11: Green Computing Conference and Workshops. (2011): 1 – 8. Print.

[34] Mutlu, O. “Memory scaling: A systems architecture perspective.” IMW: Memory Workshop. (2013): 21 – 25. Print.

[35] Storage Systems for Non-volatile Memory Devices

[36] Choi, I. S., S. I. Jang, C. H. Oh, Charles C. Weems, and S. D. Kim. “A Dynamic Adaptive Converter and Management for PRAM-Based Main Memory.” Microprocessors and Microsystems 37.6 (2013): 554 – 561. Print.

[37] Lu, Ning, I. S. Choi, and S. D. Kim. “A flash-aware write buffer

scheme to enhance the performance of superblock-based NAND flash storage systems." Microprocessors and Microsystems 37.3 (2013): 345–357. Print.

[38] Park, S. H., J. W. Park, S. D. Kim, and Charles C. Weems. "A Pattern-Adaptive NAND Flash Memory Storage Structure." IEEE Transactions on Computers 61.1 (2012): 134–138.

[39] http://www.elinux.org/Boot_Time

[40] http://free-electrons.com/doc/training/boot-time/boot-time-slides.pdf

국문요약

메모리-디스크 통합 구조를 위한 부팅 및 하이버네이션 기법 설계

스마트 워치나 스마트 글래스 같이 웨어러블 디바이스의 기술이 증가함에 따라 사용자로 하여금 기기를 즉시 사용 가능하도록 부팅시간에 대한 시간 절약 연구가 필요해지고 있다. 공학적으로 시간측면의 성능을 높이고자 오버헤드를 줄이려면 프로그램 상에서 각 함수별로 불필요한 기능을 없애거나 기능별로 우선순위에 따라 실행 순서를 재배치 한다든가 혹은 연기하여 재 정렬하는 방법이 있다. 또한 서로 연관이 없는 기능들끼리 병렬적으로 처리하는 방법이 있고 마지막으로 필요한 기능일지라도 각 함수별로 알고리즘의 성능을 높이거나 자료구조를 바꾸어 프로그램을 최적화하는 방법 등이 있다. 이 여러 가지 방법론중에서 본 논문에서는 필요한 기능을 묶어 부팅 프로시져간의 알고리즘의 성능을 높여 최적화하도록 설계하였다. 일반적인 부팅 절차는 파워가 인가

되면 BIOS가 실행되어 이 BIOS는 하드웨어 적으로 체크를 하고 부트로더를 실행시킨다. 이 부트로더는 stack영역을 초기화 하여 함수를 불러들일 준비를 하거나 ARM프로세서의 경우 machine type을 리턴 받는 등의 커널을 로드 하기 위한 작업들을 수행한다. 또한 리눅스 커널이 로드가 되면 압축을 해제하고 초기화를 한 후 실행을 한다. 즉, 커널이 초기화를 한후 init process가 실행이 되는데 이는 각종 디바이스 드라이버가 초기화하고 shell을 실행하는 등 사용자 프로세스나 하드웨어 자원을 사용가능하도록 마지막 준비 작업을 하여 컴퓨터가 정상적으로 작동할 수 있는 상태를 만든다.

기존의 DRAM과 NAND Flash나 HDD와 같이 메모리 디스크 계층 구조 시스템은 운영체제 및 부트로더와 같이 사용자 프로그램의 실행을 위한 시스템 프로그램이 디스크에서 메인 메모리로 복사하여 압축을 해제하고 초기화 하는 일련의 단계가 반복 실행되어져야만 했다. 이런 불필요한 반복단계를 오버헤드라 간주하고 이 오버헤드를 줄이고 이러한 문제를 극복하고자 비휘발성 메모리를 메인 메모리로서 접근하여 반복 실행되는 것은 비휘발성 메모리를 통해 더 이상 반복 할 필요가 없어지도록 설계하였다. 또한 비휘발성 메모리는 소자에 따라 특성이 있는데 예를 들어, PRAM의 경우 데이터를 쓰고 다시 다른 데이터를 쓰고자 할 경우 중간에 지우는 연산이 필요하고 이 수정 연산이 바로 불필요한 오버헤드로 작용하여 이를 줄이고자 한다. 또한 비휘발성 메모리는 쓰기 연산을 수행하는 횟수에 제한이 있는데 이 문제를 Endurance 문제라 명칭하고 이 문제를 해결하고자 Kernel Dynamic Optimization Module을 설계하였다 이 문제를 극복하기 위해 현재 활발히 연구가 진행되고 있고

보급을 앞두고 있는 PRAM과 그에 반해 연구가 필요하고 비용이 큰 MRAM을 하이브리드 방식으로 구성함으로서 중간에 지우는 연산이 필요한다든가 쓰기 횟수가 정해져 있는 Endurance문제를 극복하고자 하였다. 또한 CPU내의 레지스터나 레벨별 캐시의 경우 여전히 휘발성이기에 전원이 차단되면 데이터는 제거되기에 결국 다시 부팅을 하면 PC값에 의해 처음부터 부팅이 실행되기에 문제가 극복되지 않는다. 결국 현재와 같은 모델의 구성이 되는데 이 문제를 극복하고자 endurance문제로 해방되어진 MRAM내에 Recovery Synchronization Module을 설계하여 컴퓨터의 전원이 차단되었을 시 이 모듈에 의해 캐시 및 레지스터 값이 저장되고 전원이 공급되었을 때 복구 하는 동기화 방식을 사용하여 문제를 극복하였다.

설계한 통합 메모리 기반 부팅 시스템은 트레이스 파일 및 시뮬레이터를 사용하여 성능을 평가하였다. 부팅시간을 측정하는 방법은 GPIO인터럽트신호를 오실로스코프로 받아 측정하는 하드웨어적인 방법과 Linux Trace Toolkit 이나 Kernel Function Trace와 같이 툴을 이용하는 소프트웨어적인 방법 이렇게 두가지로 나누어진다. 본 연구에서는 grabserial, ftrace 및 bootchart를 이용한 소프트웨어 적인 방법으로 부팅시간을 측정하였다. 전체 시간을 한번에 구할 수 없기 때문에 각각의 부팅 단계별로 서로 다른 툴을 사용하여 각각의 시간을 측정하고 그 시간을 더하는 방법을 제시하였다. 즉, 측정가능한 단계별로 나누어 각각의 단계의 시간을 측정한다음 모두 더하여 전체 시간을 측정하였다. 이 방법은 하드웨어적인 방법보다 비용이 적게 들뿐만 아니라 각각의 단계별로 얼마나 시간이 절약되었는지 알수 있는 장점이 있다.

실험결과를 비추어 보아 하이브리드 형식의 PRAM의 부팅시간 커널 영역뿐만 아니라 유저영역 또한 부팅 시간이 감소되었음을 확인할 수 있었다. 이 결과를 통해 이 논문에서 설계한 방법 및 실험 결과는 스마트폰이나 웨어러블 디바이스 같이 스마트 기기들의 부팅 시간을 단축할 수 있음을 기대한다.

핵심어 : 비휘발성 메모리, 부팅 절차, 빠른 부팅, 상변화 메모리, 커널, 메모리-디스크 통합, 커널, 실행 및 링크 가능 포맷