Communications and Network, 2013, 5, 292-297
http://dx.doi.org/10.4236/cn.2013.53B2054 Published Online September 2013 (http://www.scirp.org/journal/cn)

On the Optimization of Real Time Performance of Software Defined Radio on Linux OS

Zhen Wang, Limin Xiao, Xin Su, Xin Qi, Xibin Xu
State Key Laboratory on Microwave and Digital Communications, Tsinghua National Laboratory for Information Science and Technology, Research Institute of Information Technology, Tsinghua University, Beijing, China
Email: wangzhen11@mails.tsinghua.edu.cn

Received July, 2013

ABSTRACT

With the evolution of the communication standards, Software Defined Radio (SDR) is faced with an increasingly important problem to balance more and more complex wireless communication algorithms against relatively limited processing capability of hardware. And, the competition for computing resources exacerbates the problem and increases time-delay of SDR system. This paper presents an integrated optimization method for the real-time performance of SDR on Linux OS (operating system). The method is composed of three parts: real-time scheduling policy which ensures higher priority for SDR tasks, CGROUPS used to manage and redistribute the computing resources, and fine-grade system timer which makes the process preemption more accurate. According to the experiments, the round-trip data transfer latency decreases low enough to meet the requirement for TD-SCDMA via the application of the method.

Keywords: SDR; Real-time Priority; CGROUP; System Timer

1. Introduction

Software Defined Radio, which holds the promise of fully programmable wireless communication systems [1], has shown more and more importance in the development of wireless communication systems. An SDR based on general-purpose processor (GPP) enables us to dynamically modify software to realize different communication standards, instead of time-consuming hardware redesign, which consequently reduces the developing time [2]. In comparing with traditional wireless communication system based on DSP or FPGA, an SDR system based on GPP is a more flexible and convenient to develop for researchers [3,4], because high-level programming language (e.g. C/C++) is generally easier to develop and debug than Verilog or VHDL.

Although SDR offers many benefits to the development of wireless communication system, it cannot be ignored that the increasing computational expense brought about by the updates of wireless communication protocol, has posed immense challenge to the limited computing resources of hardware. Inevitably, the real-time performance of an SDR based on GPP would be affected by this problem. And worse still, when a system is composed of many SDR tasks, like C-RAN [5] (the whole function of BTS is designed to implemented on GPP), the competition for hardware capacity from these tasks would be beyond what the multi-core solutions could handle. Without efforts to manage and redistribute the computing resources, the problems mentioned above would make the real-time performance of the SDR system unguaranteed.

As known, in a personal computer, it is an important task for OS to manage and allocate the computing resources to different kinds of processes according to their priorities. Similarly, redistributing and rescheduling processes in an SDR system, which enables us to preserve enough computing resource to those time-critical tasks, is a main method to improve the real-time performance.

As a traditional process scheduling tool of Linux kernel, the principle and implementation of scheduling policy on Linux OS have been illustrated in many papers and books like Understanding The Linus Kernel [6]. Through applying different scheduling policies offered by Linux kernel to various processes, the tasks with real-time priority will be allocated more CPU time than other tasks and take precedence on execution. Considering the strict demand for latency from the time-critical processes in an SDR system, instead of scheduling all processes equally, adjusting priorities by selecting appropriate process-scheduling strategies, will make great contributions to improving the real-time performance without upgrading hardware.

Despite that scheduling strategy promoting the real-time performance of the SDR system, it still leaves much to be desired. For example, setting different priori-

ties to the processes is a relatively imprecise allocation to CPU time. And what's worse, without constraints to CPU consumption, a process may consume extra and even the whole CPU resources when no other processes could preempt it. There are many scenarios where this excess CPU share can cause unacceptable utilization or latency [7]. For an SDR system, it would introduce scheduling latency when a time-critical process has been woke up to preempt CPU resources against the process currently running.

Therefore, CGROUPS (Control Groups), which is a new resource management tool provided by the Linux kernel [8], should be taken into application. By imposing caps on the CPU usage time of other tasks, CGROUPS enables us to spare more CPU bandwidth to the SDR tasks and reduce the frequency of process switches. More specifically, the CPU share can be changed dynamically as needed.

As mentioned above, real-time priority guarantees that the time-critical process will occupy the computing resources when competing with other tasks; meanwhile, CGROUPS reduces the occurrence of those competitions by constraining the CPU usage of other tasks. In addition to that, a high-resolution timer [9] configured in Linux Kernel would minimize scheduling latency via making process switches more immediate; and in consequence, optimize the overall real-time performance of SDR system.

In this paper, real-time priority, CGROUPS and high-resolution timer are utilized in combination. According to the experiments, this integrated method could constraint the data transfer latency to meet the requirement of 3G standards like TD-SCDMA.

The rest of this paper is organized as follows. Section 2 describes the system model used to analyze the data transfer latency of an SDR system. Section 3 proposes the integrated method to optimize the real-time performance of SDR based on GPP. And then, we analyze the latency requirement for TD-SCDMA in section 4. In section 5, we design and conduct some experiments based on proposed system model to validate the effectiveness of the optimization method. Finally, in section 6, we draw conclusions based on analysis of experiment results.

2. System Model

To validate the effectiveness of the integrated method to optimize the real-time performance of SDR systems running on Linux OS, a system model based on GPP is designed and implemented.

As shown in **Figure 1**, the system model is composed of three parts: baseband processing board, adapter board and RF front end. We choose PC on Linux OS as the baseband processing platform, which is major responsible for data processing. The adapter board mainly implemented by FPGA serves as an important role in controlling the RF front end and transferring data between the PC and the RF front end. The RF front end acts as a transceiver.

In baseband processing board, our SDR task is composed of three main processes. Firstly, the physical-layer process is responsible for the PHY processing like modulation and demodulation. Secondly, the high-layer process is designed to implement the algorithms of Data Link Layer and Network Layer. Finally, the data interface process serves as a role of interacting with adapter board through USB 2.0. Besides, to better validate the effectiveness of resource management via use of the method proposed, there still exist some other tasks (the number of tasks is greater than the number of CPU cores of PC), which bring the increase of computing resources consumption.

In this model, the latency of data transfer between RF front end and baseband processing board reflects the real-time performance of the SDR system. Further, there are three factors that might have major effects on the latency. Firstly, because of the limited interface bandwidth of USB 2.0, transfer delay between RF front end and processing board is inevitable. Secondly, baseband processing board would spend a period of time in processing data from RF front end, which is called processing time. Finally, the scheduling latency under Linux OS would be introduced when process switches occur. Taking data interface process as example, when there need read data from RF front end, Linux kernel would spend some time in reallocating computing resources to data interface process from the process currently running on CPU.

3. Integrated Optimization Method

As mentioned above, the transfer delay, processing time and scheduling latency limit the real-time performance of an SDR system. Generally, transfer delay is fixed because the packet size and interface bandwidth have been defined by transfer protocol and hardware. And then, the processing time is mainly determined by the algorithmic complexity and computing capacity. Therefore, a general optimization method should be aimed to redistribute the

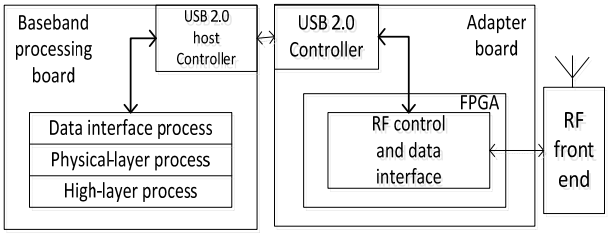

Figure 1. System model.

computing resources to reduce the cost of process scheduling and process switches.

Considering such features, three main methods will be taken in combinations, that is, real-time schedule policy for time-critical tasks, constraint of the CPU consumption on disturbance tasks via use of CGROUPS, and reconfiguration of fine-grain timer in Linux kernel.

3.1. Real-time Priority

On Linux OS, processes are divided into three categories: interactive processes, batch processes, and real-time processes [6]. Meanwhile, because of the features of different processes, there exist three scheduling algorithms: SCHED_NORMAL, SCHED_RR, and SCHED_FIFO. The SCHED_NORMAL policy is designed to schedule conventional, time-shared processes. Then, SCHED_RR and SCHED_FIFO are aimed at real-time processes.

In order to reduce process response time and avoid of process starvation, the Linux kernel schedules processes based on time sharing technology [10] and allows processes being preempted according to priority order. When a process gets ready to run in CPU, the kernel checks whether its priority is greater than the priority of the currently running process. If true, the execution of current will be interrupted and the scheduler is invoked to select the process of a higher priority to run. And additionally, the real-time processes enjoy higher priority than any other ordinary processes, so that, a real-time process will not be interrupted by ordinary process unless it has finished executing, while it can preempt other ordinary process if need as shown in **Figure 2**.

Considering such feature of process scheduling, we can specify a real-time scheduler for these time-critical processes in order to get the real-time performance improved. In an SDR system, data interface process running in baseband processing board is designed to interact with RF front end, which has strict requirement for latency; that is, it must read or write data in specified time slot following air interface specifications. Otherwise, if the data transfer is delayed, the speed and quality of data processing will be diminished, as a result, the real-time performance of SDR system will be affected.

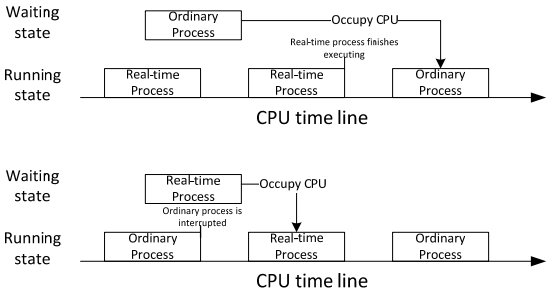

Figure 2. Process preemption.

Therefore, the data interface process should be specified as real-time process, which makes sure that the data interface has a higher priority than other ordinary processes running on the same platform. The data transfer will not be interrupted by other processes, and what's more, when some data packages need to be read from RF front end, the data interface process will seize the CPU even if an ordinary process is currently running.

3.2. Computing Resource Redistribution

CGROUPS is an abbreviation of Control Groups, which provides a mechanism for aggregation and partitioning sets of tasks, and all their future children, into hierarchical groups with specialized behavior [11]. A set of tasks (processes or threads) are associated with a set of parameters for one or more subsystems. A subsystem is a module that makes use of the task grouping facilities provided by CGROUPS to treat groups of tasks in particular ways and redistribute computing resource such as CPU, memory, IO of block device as designed. There are five main subsystems in CGROUPS and the details will be introduced below.(**Table 1**)

The initial goal of CGROUPS is to provide a unified framework for resource management, not only integrating the existing subsystems, but also providing the interface to the new subsystems which may be developed in the future. Nowadays, CGOUPS has been taken into applications in a variety of scenarios, especially in some network services such as Taobao-a leading online C2C auction company in China [12], where it is used as a tool of redistributing computing resource for large servers and OS-Level virtualization [16].

What's important for SDR systems, is that we can use the cpu subsystem to assign a specific CPU shares to processes running on the same OS and place constraints to CPUS usage of those less important tasks. In cpu subsystem, there are two main configuration options to allocate CPU resource: cpu.cfs_period_us and cpu.cfs_quota_us [13].

Reconfiguration of CPU bandwidth by set certain value (from 1000 to 1000000 microseconds) to cpu.cfs_quota_us and cpu.cfs_period_us, works immediately and

Table 1. Subsystem of CGROUPS and function.

Subsystems	Function
cpu	Allocate the cpu occupancy for a set of tasks
cpuset	Assign cpus and mem to a set of tasks
memory	Memory resource controller
devices	Device whitelist controller
blkio	Set limits to the IO of block device

efficiently. The cpu.cfs_quota_us specifies a maximum CPU time that a set of tasks could use in a period of time which is set in cpu.cfs_period_time. For example, when the cpu.cfs_quota_us is set to 10000 while the cpu.cfs_period_us is 100000, tasks in this group will use 10ms (milliseconds) in 100ms at most, which means the usage of CPU is limited to 10 percent.

Through establishing constrains to CPU usage via using cpu.cfs_quota_us and cpu.cfs_period_us, we can prevent the over much consumption from some processes and spare more resources to the time-critical processes in our SDR system. Via use of CGROUP, we could better isolate the execution of tasks in SDR system than multi-cores solutions, because the competition for computing resource could not be relieved when the number of tasks is greater than the number of cores.

3.3. Fine-grain Timer in Linux Kernel

The passing of time is important to the Linux kernel, because there exist lots of kernel functions which are time-driven except of event-driven. These periodic tasks occur on a fixed schedule, depending on the System Timer which is a programmable piece of hardware that issues an interrupt at a fixed frequency [10]. Then, system time gets updated and those tasks get performed by interrupt service routine handling for this timer.

Indeed, System Timer plays as similar role as the alarm clock of a microwave oven [10], which makes users aware of that the cooking time interval has elapsed, while System Timer reminds computers that one more time interval has elapsed based on some fixed frequency established by the kernel. The frequency of the system timer is programmed on system boot based on a static preprocessor define, HZ.

The main benefit of a higher HZ is the greater accuracy in process preemption, consequently, the scheduling latency decreased, which improve the real-time performance of SDR system. As mentioned above, the System Timer interrupt is responsible for decrementing the running process's timeslice count. When the count reaches zero, a flag called NEED_RESCHED is set and the kernel runs the scheduler as soon as possible. Now assume that a given process is running and has 2 ms of its timeslice remaining. In 2 ms, the scheduler should preempt the running process and begin executing a new more important process. Unfortunately, this event does not occur until the next timer interrupt, which might not be in 2 ms. At worst the next timer interrupt might be 1/HZ of a second aware. With Hz = 100, a process can get nearly 10 extra ms to run; by contrast, with Hz = 1000, the extra time would be limited under 1ms.

Due to the decrease of latency created by preemption delay, the real time performance of SDR system would get improved by greater accuracy in process preemption. Taking data interface process as example, even if it has a higher priority, the delay of data transfer is still out of control when scheduling latency is introduced. In conclusion, fine-grain system timer makes these time-critical processes wait less time to seizing the CPU and response more timely.

4. Latency Requirement For TD-SCDMA

As a 3G standard, TD-SCDMA has a stricter demand for data transfer latency. In order to calculate the maximum transfer delay that could be tolerated, some detail of physical channel signal format need to be elaborated in advance.

The radio frame of 1.28 Mcps TD-SCDMA has a length of 10ms, composed of two 5ms subframes. In each subframe, there are seven traffic time slots and three special time slots as shown in **Figure 3**. The 5ms subframe contains 6400 chips. The traffic time slots are 864 chips long. A physical channel is transmitted in a burst, which is transmitted in a particular timeslot within allocated radio frames.

Using the subframes structure, TD-SCDMA can operate on both symmetric and asymmetric mode by properly configuring the number of downlink and uplink time slots [15]. **Figure 4** takes the H-ARQ [15] as an example of asymmetric mode. The HS-DSCH related ACK/NACK must be transmitted on the associated HS-SICH in the next but one subframe. The time between the last HS-DSCH and the HS-SICH would be spent in processing and uplink data. If we take the first downlink time slot as zero time reference, responses should be ready before 3.45ms, which is the total time of downlink (including the DwPTS). And 3.45 ms is the threshold of arrival time.

In other words, if the uplink data has not been ready before the time for the uplink time slot transmission comes, it means the response failure. In the next section,

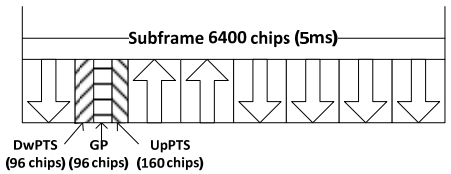

Figure 3. Subframe structure.

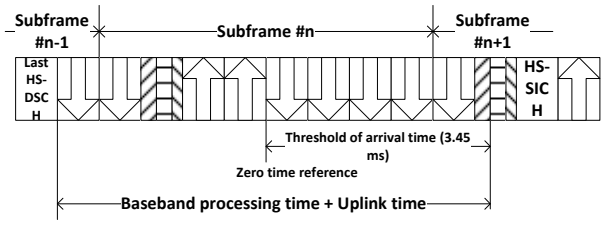

Figure 4. H-ARQ.

experiments are proposed to measure the round-trip latency and calculate the ratio of latency over 3.45 ms, i.e. response failure.

5. Experiment Result and Analysis

To validate the effectiveness of the proposed methods, we conduct some experiments based on the system model mentioned in Section 2.

In order to measure the round-trip time of the SDR platform, the whole procedure of data transmission between PC and RF front end is illustrated as **Figure 5**. It begins when the downlink data is received from the air interface and ends when the uplink response is ready to be sent by RF front end. Firstly, Downlink data arrive at PC a little later than the air interface because of the delay of USB2.0 transmission and process schedule, which is shown in the third and fourth line of **Figure 5**. The PC processes the downlink data and then sends the associated response i.e. ACK/NACK as shown in the fifth and sixth line of **Figure 5**.

As the analysis in section 4, the round-trip latency must be constrain to 3.45ms at least, otherwise, some uplink responses would not be ready in the RF front end before the uplink time slots, i.e. the response fail. In experiments, time between the first downlink time slot and the arrival time of the uplink responses, would be recorded as round-trip latency to calculate the ratio of response failure.

In order to quantitatively analyze the influence of the competition for computing resource on performance of SDR system, we run our data interface program with the existence of four infinite loops as disturbance programs on PC (PC has two CPU cores). Consequently, the ratio of response failure is 0.7841, on the contrary to 0.00024 without any interference on the same platform. It proves that the existence of resource competition will make system performance especially latency deteriorated because less computing resources would allocated to data interface process, when the number of tasks is greater than that of CPU cores.

When data interface process has read the last data packet and is waiting for the next, for avoiding of process starvation, the kernel will allocate CPU to disturbance tasks. But the issue is, when the data interface process is woke up to read data from adapter board, the kernel has to cost a period of time to reallocate CPU to it as analyzed in section 3.3. Therefore, even after we set real-time priority to data interface process, the ratio of round-trip latency over 3.45ms is only reduced to 0.582, which still far more than 0.00024 in ideal environment.

Next, a series of experiments are conducted to validate the effectiveness of the proposed methods. We set real-time priority to data interface process, and then, record the response failure ratios in different CPU usage constraints (i.e. no constraints, 80%, 60%, 40%, 20%, 10%) to disturbance programs by reconfiguring cpu.cfs_quota_us in cpu subsystem. The value of system timer frequency is set as 128 Hz, 512 Hz, 1024 Hz.

The results are shown in **Figure 6**. The constraints to CPU usage are reflected in x-axis and the ratios of response failure are drawn in y-axis; meanwhile, system timer of 128Hz, 512Hz and 1024Hz are represented by

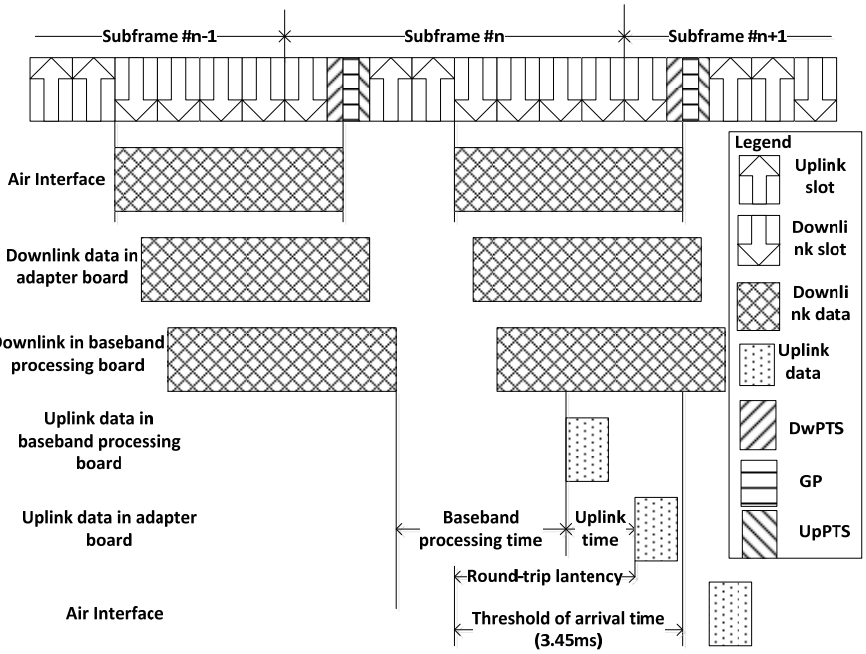

Figure 5. Procedure of data transmission in experiments.

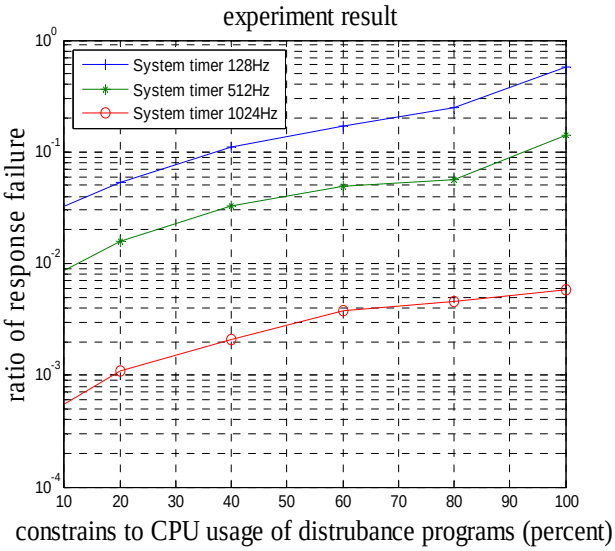

Figure 6. Experiment result.

blue curve, green curve and red curve respectively. The System Timer of 1024 Hz performs better than that of 512 Hz and 128 Hz (default value in PC), because that increasing the timer frequency to 1024 Hz lowers the worst-case scheduling delay to just nearly 1 m. Besides, along with more stringent constrains to CPU usage of disturbance programs, the ratio of response failure has decreased obviously, which proves that via using CGROUPS, we can spare more computing resource to our SDR tasks. Better yet, when the CPU consumption of disturbance programs is limited to 10 percent and the frequency of System timer is raised to 1024 Hz from 128 Hz, the round-trip latency will achieve a satisfactory ratio of response failure, 0.00056, that is magnitude equal to that in environment lack of competition and lower enough to meet the requirement of TD-SCDMA.

6. Conclusions

In this paper, we use real-time priority, CGROUPS and high-resolution system timer in combination as above, to optimize the real-time performance of SDR system. Taking TD-SCDMA as example, we justify it by experiments that the integrated use of these methods can provide a latency guarantee to data transfer to meet the requirement for this 3G system even under the competition for computing resource from other tasks.

7. Acknowledgements

This work was supported by State Key Laboratory team building project ”the key technology research of integrated wireless communication”, S&T Major Project (2012ZX03003007-004) and National Science Foundation of Beijing (4110001).

REFERENCES

[1] K. Tan, J. Zhang, J. Fang, H. Liu, Y. Ye, S. Wang, Y. Zhang, H. Wu, W. Wang and G. M. Voelker. Sora: High Performance Software Radio Using General Purpose Multi-core Processors. In NSDI 2009.

[2] S. Mamidi, E. R. Blem, M. J. Schulte, *et al*., (2005, September). Instruction Set Extensions for Software Defined Radio on a Multithreaded Processor. In Proceedings of the 2005 International Conference on Compilers, Architectures and Synthesis for Embedded Systems, pp. 266-273.

[3] J. Zhang, K. Tan, S. Xiang, Q. Yin, Q. Luo, Y. He, J. Fang and Y. Zhang, “Experimenting Software Radio with the SORA Platform,” *In ACM SIGCOMM Computer Communication Review*, Vol. 40, No. 4, 2010, pp. 469-470. doi:10.1145/1851275.1851268

[4] P. Guo, X. Qi, L. Xiao and S. Zhou, “A Novel GPP-based Software-Defined Radio architecture,” *In Communications and Networking in China* (*CHINACOM*), 2012, 7th *International ICST Conference on*, pp. 838-842.

[5] C-RAN http://labs.chinamobile.com/cran/

[6] D. P. Bovet and M. Cesati, Understanding the Linux Kernel. O’Reilly Media, 3 Edition.

[7] P. Turner, B. B. Rao and N. Rao, “CPU Bandwidth Control for CFS,” *In Proceedings of the Ottawa Linux Symposium-OLS*, Vol. 10, 2010, pp. 245-254.

[8] H. Ishii, Fujitsu’s Activities for Improving Linux as Primary OS for PRIMEQUEST. Fujitsu Science Technology Journal, Vol. 47, No. 2, 2011, pp. 239-246.

[9] S. T. Dietrich, D. Walker. The evolution of real-time linux. In Proc. 7th Real-Time Linux Workshop, 2005, pp. 3-4.

[10] R. Love. Linux Kernel Development. Addison-Wesley, 3 Edition.

[11] CGROUPS. http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt

[12] The Practice of Resource Control Using Cgroup in Tao-Bao main servers. http://wenku.baidu.com/view/19668a5677232f60ddcca113.html

[13] CPU subsystem. https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enteprise_Linux/6/html/Resource_Management_Guide/sec-cpu.html.

[14] C. S. Wong, I. K. T. Tan, R. D. Kumari, J. W. Lam, W. Fun. (2008, August). Fairness and Interactive Performance of O (1) and CFS Linux Kernel Schedulers. In Information Technology, ITSim 2008. *International Symposium on*, Vol. 4, pp. 1-8.

[15] 3GPP TS 25.221 V9.4.0 “Physical Channels and Mapping of Transport Channels onto Physical Channels,” November, 2011.

[16] M. Rosenblum, The Reincarnation of Virtual Machines. Queue, Vol. 2, No. 5, p. 34. doi:10.1145/1016998.1017000

