

SHANGHAI JIAO TONG UNIVERSITY

**学士学位论文**

BACHELOR’S THESIS



论文题目： Latency Impact of Docker Containers: A Closer Look

学生姓名: 章佐铭

学生学号: 5120309626

专 业: 计算机科学与技术

指导教师: 李超

学院(系): 电子信息与电气工程学院

**二甲醚清洁燃料均质压燃燃烧数值模拟研究**

摘要

均质充量压缩着火（HCCI）燃烧，作为一种能有效实现高效低污染的燃烧方式，能够使发动机同时保持较高的燃油经济性和动力性能，而且能有效降低发动机的NOx和碳烟排放。此外HCCI燃烧的一个显著特点是燃料的着火时刻和燃烧过程主要受化学动力学控制，基于这个特点，发动机结构参数和工况的改变将显著地影响着HCCI发动机的着火和燃烧过程。本文以新型发动机代用燃料二甲醚（DME）为例，对HCCI发动机燃用DME的着火和燃烧过程进行了研究。研究采用由美国Lawrence Livermore国家实验室提出的DME详细化学动力学反应机理及其开发的HCT化学动力学程序，且DME的详细氧化机理包括399个基元反应，涉及79个组分。为考虑壁面传热的影响，在HCT程序中增加了壁面传热子模型。采用该方法研究了压缩比、燃空当量比、进气充量加热、发动机转速、EGR和燃料添加剂等因素对HCCI着火和燃烧的影响。结果表明，DME的HCCI燃烧过程有明显的低温反应放热和高温反应放热两阶段；增大压缩比、燃空当量比、提高进气充量温度、添加H2O2、H2、CO使着火提前；提高发动机转速、采用冷却EGR、添加CH4、CH3OH使着火滞后。

关键词：均质充量压缩着火，化学动力学，数值模拟，二甲醚，EGR

**OPTICAL PROPERTIES OF COMPOSITE MATERIALS MADE FROM HYDROGEL AND BUTTERFLY WING SCALES**

**ABSTRACT**

Traditionally, many web services are held on virtual machines (VMs) provided by cloud computing suppliers. Since VMs bring about dramatic performance degradation compared to bare metal, the quality of service (QoS) is affected. Among all the QoS features, service latency is of crucial importance. With the prevalence of Docker, containers, also called “lightweight VM”, offer another choice to deploy web applications on the cloud. This paper takes the first to thoroughly analyze the impact of different Docker configurations on service latency. We conclude that the CPU quota configuration might lead to a long tail latency. Docker bridge could lead to a fixed amount of latency degradation instead of a percentage fallen. Using AUFS could bring about extra latency when opening a file or traversing the file system, and have no effect on writing data to a file.

**Key words:** Biomedical Sensor, Lepidoptera scales, Nature photonics, Optical sensor/indicator, Electric field sensitive, pH condition sensitive, Interpenetrating polymer network

**Contents**

1. Chapter One Introduction 1
2. Chapter Two Background & Motivation 4

2.1 Linux Bridge 4

2.2 AUFS 5

1 Chapter I ---------------------------------------------------------------------------------------1

* 1. Section------------------------------------------------------------------------1

1.1.1 Subsection-------------------------------------------------------------------------1

…

References ----------------------------------------------------------------------------------------2

# Introduction

Began from an open-source advanced container engine of dotCloud, a Platform-as-a-Service (PaaS) supplier, Docker is becoming one of the most promising virtualization platform. It significantly shortens the process of packing, shipping and running applications ([26], Merkel D., 2014: 2.). By packing all the dependencies of the application into several image layers, you can carry the package around and run it with simple commands on almost every laptop, personal computer, and even cloud center as long as running a Linux operating system.

Unlike traditional virtual machines, which use hardware-level virtualization, Docker containers employ system-level virtualization and share the same kernel with the host machine ([30], Soltesz S., 2007: 275.). Many researches have proved that containers have a better performance in most cases than VMs. Due to these performance reasons, many companies are trying to move services from virtual machines to containers ([13], He S., 2012: 15.). However, containers do add additional layers compared to bare-metal hardware, which leads to certain degree of performance degradation.

Docker was born to replace virtual machines to some extent. Nowadays, the widely known Infrastructure-as-a-Service (IaaS) platforms like Amazon EC2 uses virtual machines to run applications like cache and database. Most of these applications not only focus on throughput, but also favor real-time low latency. However, most related work of Docker focus mostly on containers’ influence on throughput instead of the latency degradation. Since Docker provides many choices of resource isolation, in this paper, we will do research on how these parameters will affect the latency performance of real time applications.

## Motivation

Many modern web services like Google and Facebook are interactive. Responses should be returned very soon otherwise users might complain. Also, these services are dynamic. Data centers process huge amount of data based on the user input and response in very limited time. For example, it requires thousands of Memcached machines to do a simple request through Facebook servers ([27], Nishtala R., 2013: 385.) and tens of thousands of index servers to do a Bing search ([17], Jalaparti V., 2013: 219.). In these cases, not only the throughput is of crucial importance to serve as many clients as possible concurrently, latency should also be taken into account to provide users with the best interactivity. Each additional time cost in one of the service backend layers would increase the overall latency. If all of these separated services require software virtualization layer, a millisecond's latency might be amplified to several thousand milliseconds, thus greatly influencing the overall performance of the service.

Tail latency is another issue to care about ([6], Dean J., 2013: 74.). In Map-Reduce task ([7], Dean J., 2008: 107.), a program is processed by hundreds of machines. The result of each map machine is passed to a central reduce machine, so the reduce one has to wait all map ones before moving on. In this case, a map work done over one minute encumbers the whole system even if others are finished in several seconds. Assume that a task has one hundred sub tasks and each sub task has a 99% probability to finish in one microsecond while 1% to finish over 1 second. Then the overall performance of this job is 63.3% probability to finish over one second, which is a rather bad performance. The one-in-one-thousand situation becomes a common case.

The CTO of GigaSpaces claimed a list of interesting phenomenon. He pointed out that latency is a serious matter that can lead to huge profit lost in many companies. Every 100ms of latency would cost Amazon 1% of lost in sales. Also, every extra of 0.5 seconds wasted on generating a search page can drop Google's network traffic 20%. Moreover, if a broker's electronic trading platform can not catch up others’ and gets 5 milliseconds behind the competition, they would lose $4 million in revenues per millisecond. Even these latencies seem relatively small, people hate waiting. They feel repulsed by these less interactive services, quickly click away and finally do other things like turning to the opponents' services. People are talking about how to scaling up the capacity of their services, but they sometimes neglect the importance of building low-latency ones. Service suppliers should try their best to decrease service latency, increase interactivity, and finally lower the customer defection rate ([4], Colgate M., 1996: 23.).

Despite the increasing need of virtualization technologies to decrease latency, Docker doesn't seem to be focusing on this part. In fact, although Docker provides us with a simple way to deploy applications, the technologies it employs are not so latency-friendly. Like what has been mentioned in IBM's technical report ([10], Felter W., 2015: 171.), Docker containers take the Linux bridge as the method of network isolation. However, it shows that Docker containers even perform worse in transmission throughput and also have a longer network latency compared to KVM ([18], Kivity A. 2007: 225.). Actually, all technologies used by Docker are not new ones. Most of them have already existed since the year of 2007, and the concept of container also occurs at that time ([30], Soltesz S., 2007: 275.). Docker container is just a combination of these simple technologies. With the concept of Docker images and the emergence of Docker Hub, Docker quickly win the eyes of system deployers. However, since most of these technologies are provided by old versions of Linux Kernel and they focus on resource isolation instead of latency, it will take Docker a long time to find ways to replace those inefficient technologies and thus decreasing latency lost.

Unlike Google or Facebook, which has dedicated data centers for their services, most small companies cannot afford the cost of hardware and the following maintenance. They can only deploy services on cloud centers like Amazon EC2 ([3], Shankar S., 2009.) and Microsoft Azure ([5], Copeland M., 2015:27.). As we have mentioned above, these cloud centers use virtual machines to provide hardware virtualization and have a significant performance cost compared to bare metal. The occurrence of Docker thus providing another choice for these customers. Many cloud center service suppliers provide container services in recent years. To simplify the deployment of applications, these small companies are considering to use Docker cloud. Since the additional layer of virtual machine brings about significant performance lost ([16], Huber N., 2011:563.) and is part of the source reason of long latency, it is very important for them to know the trade off between the convenience and latency performance degradation of using Docker to deploy latency-sensitive applications.

Previous researches mainly focus on the throughput of CPU, memory and I/O. Some of them talks about memory footprint and the latency brought about by Docker network bridge and methods to shorten this latency. However, these methods are not suitable for public cloud. This paper is intended to solve the problem from the customers’ perspective. Although customers can not the change the services provide by cloud service suppliers, they have the choice to choose their start up configurations and the policies to build their services. We focus on the effect of Docker containers on latency with respect to various configurations. We analyze the effect of Docker container configurations to web service situation. This analysis provides customers with the potential latency cost of Docker containers and helps them to build services with the awareness of these possible degradation.

## Related works

The appearance of Docker is in the year of 2012. However, the history of Linux containers is more than just several years. In the year of 2001, as an initial implementation of “virtual private servers”, Linux-VServer project came into existence. However, it has never been merged to the mainstream Linux operating systems. There are other Linux containers like OpenVZ, which is mainly used to host web applications, that also doesn't share a position in the mainstream Linux. Finally, in the year of 2007, as many features including namespaces and chroot are added to Linux kernel, Linux Containers (LXC) was finally added to the mainstream Linux and becomes the most widely used containers since then.

Several institutions and researchers have published related performance evaluation work on Docker. Most of them focus on throughput, while a few are concerned with latency. Researchers from IBM ([10], Felter W., 2015: 171.) use KVM as a representative hypervisor and Docker as a representative container and compare the performance of bare metal, virtual machine and container. They use various workloads to stress CPU, memory, and I/O resources. They have found that containers overwhelm virtual machines in almost every case concerning throughput. After these workloads, the research also shows experiments on some real world applications including MySQL and Redis Cache. Both these real world applications exhibit a better performance for Docker containers than virtual machines. The report also reveals that the startup time of KVM is 50x slower than Docker containers. Kavita ([2], Agarwal K., 2015: 8.) tries to increase the number of containers on a host machine with the same kind of workload. He finds that the overall density of containers on a machine is highly dependent on the most demanded resource. He also concludes that virtual machines have significantly higher overheads than containers concerning memory footprint. He uses Kernel Same Page Merging (KSM), a memory de-duplication technology, and finds a 60 times opportunities to lower the memory cost of a virtual machine compared to containers. Canonical does a similar work as Kavita comparing LXD and KVM. All these virtual machines are running Ubuntu 14.04 operating system. Experiment measurements reveals that on a host machine containers have 14.5x higher density than virtual machines. The density bound is mainly caused by memory limitation. Also, the work shows a 57% reduction in network latency than virtual machines. But it doesn't show that the LXD is using Linux bridge technology.

Eder ([9], Eder J., 2015.) did a very simple work using kernel bypass ([22], Liu J., 2006: 29.). He concluded that applications running in a container does not have an obvious impact on its network latency performance. He uses *OpenOnload* together with *netperf* to realize the bypass. The results are shown and compared with their average, mean and 99th-percentile round trip latency. From that report, he concludes that there are almost no performance degradation using Docker containers. However, this test is only suitable for private Docker cloud rather than public Docker cloud. This is because kernel bypass requires direct access to *Network Interface Card* (NIC), which has potential security problems in public cloud since one can modify the content of other containers as long as they want. However, in a private Docker cloud, kernel bypass can be a very good choice. Conventionally, once a packet is sent, it has to go through user space, kernel space and finally arrive at the NIC. With kernel bypass, the packet can be directly sent from user space to NIC, which saves some time.

On the other hand, IBM’s report ([10], Felter W., 2015: 171.) shows that Docker container has a significant impact on overall network performance. The report uses *nuttcp* to measure throughput and also *netperf* to gauge latency. The report shows that there is over 80% degradation on network round trip latency and also consumes more CPU cycles transferring a single byte using Docker containers than natively. So why there exists such a big difference between Eder’s work and IBM’s report? The key point lies in the fact that one uses kernel bypass while the other doesn’t. Using kernel bypass in a Docker container leads to shorter latency than Docker bridge or Docker host and even faster than bare metal without kernel bypass. From the public cloud perspective, since it is not allowed for customers to use kernel bypass due to security reasons ([8], Dua R., 2014: 610.), IBM’s work is more valuable in this case. However, IBM’s report only uses a single group of comparison. It doesn’t incorporate more comparison groups to further develop the relationship between round trip latency and other variables like the size of each packet transmitted.

There are also works showing that containers don’t have a significant performance lost concerning network performance compared to bare metal. Xavier ([37], Xavier M. G., 2013: 233.) uses Xen as an representative of virtual machines and compare its performance to various kinds of containers including LXC, OpenVZ, and VServer. He presses these technologies with various kinds of well-known benchmarks and draws to the conclusion that containers outperform virtual machines in every high performance cases. However, different from other researches, his work doesn’t show a high performance lost in I/O cases. This is because these old-type containers don't employ technologies including AUFS or Linux bridge. In our work, we find that these two technologies are key reasons to the latency lost in Docker containers.

## Contributions

This thesis makes the following contributions:

First and foremost, we investigate the latency slowdown caused by CPU configurations. We explore and compare two kinds of configurations, the first one is CPU shares and the second one is CPU quota. We find that CPU shares almost has no impact on the performance lost while it cannot limit the CPU usage of a container when only one container is running on a CPU. On the other hand, CPU quota can successfully limit the CPU usage of a container, but it has the potential to lead to a rather long tail latency.

Secondly, we build a research platform to evaluate the network latency performance of Docker containers. The platform employs a client-server architecture. The server is hosted in a Docker container and we measure the round trip latency of a client request. We choose two situations, the first is server sending data and the second is server receiving data. We compare using Docker host and Linux bridge configurations. We draw the conclusion that containers do have some impact on the performance lost using Linux bridge compared to directly using the host machine’s port. However, this performance is not as exaggerated as described in IBM’s report that Docker bridge causes 80% performance lost compared to bare metal. In fact, this is more like a fix-length performance degradation. The smaller the transmitted message is, the more relatively significant the performance slowdown is.

Thirdly, we analyze the latency impact of Docker using AUFS to do file operations. We find that Docker containers do not have impact on performance when writing an existing file. However, when it comes to operations related to the file system instead of a single file, situation changes. Operations like opening a file would lead to extra latency due to locating the file in multiple AUFS layers and the extra cost of creating the copy-on-write layer. When listing a directory which has many hidden files in low layers, the hidden files will also be scanned instead of just the superficial ones. The total scanning time is linear to the sum of the number of hidden and superficial files.

## Organization of This Paper

The following sections are organized as follows: Section 2 introduces some background information about Docker related technologies. Section 3 carries out experiments and gives analysis about their affects to service latency. We discuss related works in Section 4 and give a final conclusion in Section 5.

# Background

## Docker

Docker, an open-source advanced container implemented by dotCloud, is making a huge impact on the filed of cloud computing. Docker wraps the whole runtime environment into the unit of Docker containers to divide and schedule resources. It is a platform designed for developers and system administrators to build, ship and release distributed applications. It is also a cross-platform, portable and easy of use container solution. Docker is implemented in Go language and its source code is hosted on Github. Docker provides developers a fast and automatic way to deploy applications. It incorporates many operating system-level Linux kernel technologies like namespaces and control groups to provide resource isolation, resource limitation and security.

### Container and virtual machine

Many people are familiar with virtual machines. Starting from our daily life, we might run Ubuntu Linux on Windows PC using VMware or play games in a windows virtual machine ran using Parallel Desktop [8] on Mac OS. These are all hardware-level virtual machines. They uses software to simulate the instructions that might be used by the operating system run in the virtual machine as if they are just operating on the bare metal. Hardware-level virtual machines like VMWare and Parallel Desktop are more used on personal PC?s, while Xen [9] and KVM [10] are more used on servers and public/private clouds.

On the other side is the container-based virtualization, which is also called operating system-level virtualization. OS-level virtualization come into people?s eyes might because of the chroot [11] mechanism introduced in Unix-like operating systems. Chroot was traditionally used to run multi services in a multiuser environment and leave one will not affect each other running applications on the same machine. Start from chroot, a lot of prototype containers were implemented like the famous Linux Container (LXC) [12], FreeBSD jail [13] and OpenVZ [14]. Although Docker became famous in recent years, container is not a new concept since the container technologies mentioned above has appeared traced back to the year of 2007. It might be the mature of cloud environment, the spring of web applications, and also the completeness of Docker environment that makes Docker so popular. Unlike hardware-level virtual machines, OS-level virtual machines use the host operating system, but each only has its own files. A container contains all the files it needs to run a program, it might has its own libraries, /boot directory, /usr directory, /home directory and so on. The whole running container can even have only a single file if you like ? the binary program you want to run if it doesn?t have any dependency. Also, programs running inside a container cannot see processes outside the container, including those running directly on the native host and also applications running in other containers. This is implemented using the linux namespace mechanism. Containers also use chroot mechanism to limit the resource like cpu usage, memory usage, I/O usage cost by a container.

IBM?s report shows that in almost all cases (except for network latency), Docker containers performs better than virtual machines and is very competitive to native. This is because virtual machines have to use software to simulate hardware. It might cost several instructions to simulate a single instruction using software, thus dramatically drag down the performance. While containers? processes are just running on host operating systems, except for the fact that they are isolated and resource limited. Containers also exhibits a much shorter start time than virtual machines, while most virtual machines use tens of seconds or even several minutes to start, it only takes a container several seconds or even less than a second. It also takes much less memory footprint [15] to run processes than virtual machines because containers do not need those extra files to start a whole operating system.

Although containers do bring us about better overall performance than virtual machines, there also exists a lot of problems to be solved. The most important part is security. As I have mentioned above, containers use the host operating system, which means that any security threats in host operating system might be made use of to attack the host and other containers running on the same machine. The host operating system can see everything running inside a container, and the content might not be private for users. Once a container process successfully take the administration of the host operating system, it can operate all other container processes at its will.

Another current limitation for containers is that most of them can only run on Linux. Since running a process running in container requires the host operating system, running Linux processes on Windows systems may not be practical. Also, since containers do not have the authority to modify its host operating system, it is not possible to load kernel modules dynamically, thus limiting its ability.

**REFERENCE**

体例参照GB-T 7714-2005《文后参考文献著录规则》，用中括号的数字连续编号，顶格书写，字体为12号Times New Roman，1.5倍行距。

Times New Roma.n, 14号字加粗居中，上下各空一行。

1. Interesting
2. Agarwal K, Jain B, Porter D E. Containing the hype[C]//Proceedings of the 6th Asia-Pacific Workshop on Systems. ACM, 2015: 8.
3. Shankar S. Amazon elastic compute cloud[J]. 2009.
4. Colgate M, Stewart K, Kinsella R. Customer defection: a study of the student market in Ireland[J]. International Journal of Bank Marketing, 1996, 14(3): 23-29.
5. Copeland M, Soh J, Puca A, et al. Overview of Microsoft Azure Services[M]//Microsoft Azure. Apress, 2015: 27-69.
6. Dean J, Barroso L A. The tail at scale[J]. Communications of the ACM, 2013, 56(2): 74-80.
7. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.
8. Dua R, Raja A R, Kakadia D. Virtualization vs containerization to support paas[C]//Cloud Engineering (IC2E), 2014 IEEE International Conference on. IEEE, 2014: 610-614.
9. Eder J. Accelerating red hat enterprise linux 7-based linux containers with solarflare openonload[R]. Red Hat Enterprise, 2015.
10. Felter W, Ferreira A, Rajamony R, et al. An updated performance comparison of virtual machines and linux containers[C]//Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On. IEEE, 2015: 171-172.
11. Gibson G A. Redundant disk arrays: Reliable, parallel secondary storage[M]. Cambridge, MA: MIT press, 1992.
12. Hasenstein M. The logical volume manager (LVM)[J]. White paper, 2001.
13. He S, Guo L, Guo Y, et al. Elastic application container: A lightweight approach for cloud resource provisioning[C]. Advanced information networking and applications (aina), 2012 ieee 26th international conference on. IEEE, 2012: 15-22.
14. Helsley M. LXC: Linux container tools[J]. IBM devloperWorks Technical Library, 2009.
15. Hopper T. Cumulative Distribution Function[J]. Month, 2014.
16. Huber N, von Quast M, Hauck M, et al. Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments[C]//CLOSER. 2011: 563-573.
17. Jalaparti V, Bodik P, Kandula S, et al. Speeding up distributed request-response workflows[J]. ACM SIGCOMM Computer Communication Review, 2013, 43(4): 219-230.
18. Kivity A, Kamay Y, Laor D, et al. kvm: the Linux virtual machine monitor[C]//Proceedings of the Linux symposium. 2007, 1: 225-230.
19. Kratzke N. About Microservices, Containers and their Underestimated Impact on Network Performance[J]. CLOUD COMPUTING 2015, 2015: 180.
20. Li C, Ding C, Shen K. Quantifying the cost of context switch[C]//Proceedings of the 2007 workshop on Experimental computer science. ACM, 2007: 2.
21. Li J, Sharma N K, Ports D R K, et al. Tales of the tail: Hardware, os, and application-level sources of tail latency[C]//Proceedings of the ACM Symposium on Cloud Computing. ACM, 2014: 1-14.
22. Liu J, Huang W, Abali B, et al. High Performance VMM-Bypass I/O in Virtual Machines[C]//USENIX Annual Technical Conference, General Track. 2006: 29-42.
23. Love R. Kernel korner: CPU affinity[J]. Linux Journal, 2003, 2003(111): 8.
24. Menage P, Seth R, Jackson P, et al. Linux control groups[J]. 2007.
25. Menascé D A. Virtualization: Concepts, applications, and performance modeling[C]//Int. CMG Conference. 2005: 407-414.
26. Merkel D. Docker: lightweight linux containers for consistent development and deployment[J]. Linux Journal, 2014, 2014(239): 2.
27. Nishtala R, Fugal H, Grimm S, et al. Scaling memcache at facebook[C]//Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13). 2013: 385-398.
28. Pendry J S, Sequent U, McKusick M K. Union mounts in 4.4 BSD-lite[J]. AUUGN, 1997: 1.
29. Slee M, Agarwal A, Kwiatkowski M. Thrift: Scalable cross-language services implementation[J]. Facebook White Paper, 2007, 5(8).
30. Soltesz S, Pötzl H, Fiuczynski M E, et al. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors[C]//ACM SIGOPS Operating Systems Review. ACM, 2007, 41(3): 275-287.
31. Špaček F, Sohlich R, Dulík T. Docker as platform for assignments evaluation[J]. Procedia Engineering, 2015, 100: 1665-1671.
32. SPECweb2009 E. commerce workload, 2009[J].
33. Strauch C, Sites U L S, Kriha W. NoSQL databases[J]. Lecture Notes, Stuttgart Media University, 2011.
34. Tseng H M, Lee H L, Hu J W, et al. Network virtualization with cloud virtual switch[C]//Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on. IEEE, 2011: 998-1003.
35. Tsirtsis G. Network address translation-protocol translation (NAT-PT)[J]. Network, 2000.
36. Wright C, Cowan C, Smalley S, et al. Linux security modules: General security support for the linux kernel[C]//null. IEEE, 2003: 213.
37. Xavier M G, Neves M V, Rossi F D, et al. Performance evaluation of container-based virtualization for high performance computing environments[C]//Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on. IEEE, 2013: 233-240.
38. Xavier M G, Neves M V, Rossi F D, et al. Performance evaluation of container-based virtualization for high performance computing environments[C]//Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on. IEEE, 2013: 233-240.
39. Xu Y, Musgrave Z, Noble B, et al. Bobtail: Avoiding long tails in the cloud[C]//Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13). 2013: 329-341.
40. Zawodny J. Redis: Lightweight key/value store that goes the extra mile[J]. Linux Magazine, 2009, 79.

**…**