

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 5

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 ([4], 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 ([6], 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 ([3], 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 ([5], Nishtala R., 2013: 385.) and tens of thousands of index servers to do a Bing search\cite{jalaparti2013speeding}. 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\cite{dean2013tail}. In Map-Reduce task\cite{dean2008mapreduce}, 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 phenomenons. 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 seems relatively small, people hate waiting. They feel repulsed by these less interactive services, quickly click away and finally do other thing, 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\cite{colgate1996customer}.

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 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\cite{felter2015updated}, 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\cite{kivity2007kvm}. 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\cite{soltesz2007container}. 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 mantaince. They can only deploy services on cloud centers like Amazon EC2. As we have mentioned above, these cloud centers use virtual machines to provide hardware virtualization and has a significant performance cost compared to bare metal. The occurrence of Docker thus provide another choice for these customers. Many cloud center service suppliers like Amazon EC2\cite{amazon2010amazon} and Microsoft Azure\cite{copeland2015overview} 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\cite{huber2011evaluating} 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 throughput of CPU, memory and I/O. Some of them talks about memory footprint and the latency bring about by Docker network bridge and methods to shorten this latency. However, these methods are not suitable for public cloud. This paper 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 to latency with respect to various configurations. We analyze the effect of Docker container configurations to web service situation. This analysis provide customers with the potential latency cost of Docker containers and help them build services with the awareness with these possible degradation.

# Background & Motivation

LXC\cite{helsley2009lxc} and Docker libcontainer\cite{vspavcek2015docker} use Linux Namespaces\cite{wright2003linux} together with Control Groups (CGroups)\cite{menage2007linux} to realize resource isolation and limitation. Resources like CPU, memory and process id (PID) are no longer global, but belongs to a particular namespace. Processes outside a namespace are transparent to ones inside the namespace. Inside processes also has no access to outside resources, thus providing certain level of security.

## Linux Bridge

Bridge mode is the default network setting of Docker, which makes use of the Linux bridge feature\cite{tseng2011network}. When using this mode, each container is allocated a network namespace and separate IP. Once the Docker daemon starts, it creates a virtual network bridge named docker0 on the host machine. All Docker containers created on this machine will be connected to docker0. Virtual network bridge works like a physical switch, thus all containers on the host machine are connected in a two-layer network through the switch. Docker chooses a private IP different from the host IP and allocate it to docker0. It also selects a sub net defined in RFC1918. Each container on the host machine is assigned an unused IP from this sub net pool.

## AUFS

Another Union File System (AUFS) is a kind of Union File System\cite{pendry1997union}. Frankly speaking, it is a file system that supports to mount different directories to a single virtual filesystem. Further deep inside, AUFS supports to set the readonly, readwrite, whiteout-able authority of every member category, just like Git Branch. Also, the layer concept in AUFS supports to logically and incrementally modify the readonly branch without affecting the readonly part. Generally speaking, there are two uses of Union FS. On one hand, it can mount multiple disks to a single directory without the help of LVM\cite{hasenstein2001logical} or RAID\cite{gibson1992redundant}. On the other hand, it enables the cooperation of a readonly branch and a writeable branch. Docker uses AUFS to build container images.

**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. Jkjk
3. 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.
4. Merkel D. Docker: lightweight linux containers for consistent development and deployment[J]. Linux Journal, 2014, 2014(239): 2.
5. 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.
6. 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.

**…**