LOCK-FREE FIFO BASED PARALLEL HTB IN CLOUD COMPUTING

[1] T.PRANAV, [2] M.PRAVIN KUMAR, [3]DR.C.NALINI
[1,2]Student
[3]Professor
Department of Computer Science & Engineering
Bharath University

ABSTRACT:

Cloud computing is becoming more and more popular in IT industry nowadays. Those famous companies including Amazon, IBM, HP, Google and Microsoft are creating and deploying Clouds in various locations around the world. Technically, Cloud Computing refers to both the applications delivered as services over the Internet. In the cloud, there might be tens of thousands or even more users accessing resource simultaneously, which give an extremely high pressure on the cloud. An effective traffic control mechanism which can both control the network traffic and make full use of network bandwidth is Hierarchical Token Buckets (HTB). It is used to control the outbound bandwidth on a given link. HTB ensures that the traffic rate for each class is at least the amount assigned to it. And the main difference between HTB and other queue discipline is when a class requests less than the amount assigned, the remaining bandwidth can be "borrowed" by other classes which request more. This solution is very suitable for service provider such as cloud computing: the basic requirements are guaranteed when there are many concurrent users (based on their payment), and when there are free resources, users can enjoy a better experience. Unfortunately, the existing HTB implementation can afford 0.5Gbps speed at most, making it impossible to be utilized in the cloud. With the popularity of multi-core processors, a possible improvement is making the original sequential HTB into parallel, which might raise its processing speed.

Keywords: Cloud Computing, Network Traffic Control, Hierarchical Token Buckets (HTB), Lock Free FIFO(First In First Out)

INTRODUCTION:

An effective traffic control mechanism which can both control the network traffic and make full use of network bandwidth is Hierarchical Token Buckets (HTB). It is used to control the outbound bandwidth on a given link. HTB ensures that the traffic rate for each class is at least the amount assigned to it. And the main difference between HTB and other queue discipline is when a class requests less than the amount assigned, the remaining bandwidth can be "borrowed" by other classes which request more. This solution is very suitable for service provider such as cloud computing: the basic requirements are guaranteed when there are many concurrent users (based on their payment), and when there are free resources, users can enjoy a better experience. Unfortunately, the existing HTB implementation can afford 0.5Gbps speed at most, making it impossible to be utilized in the cloud. With the popularity of multi-core processors, a possible improvement is making the original sequential HTB into parallel, which might raise its processing speed. Network applications have two inherently features that are suitable for parallelization:

1) They have naturally layered structures that can be organized into a functional pipeline; and

2) Packets belonging to different flows can be processed in parallel. Besides these, software based network application on multi-core platform is also economic:

It is not only cheaper than special hardware such as ASIC, but also more scalable. In this paper, we propose a parallel HTB based on multicore processors. The traditional operations on HTB structures are modified to reduce the strong dependency in the sequential code. Then lock-free structures are applied selectively to make multi-core parallelization easier and manageable. Based on this, the parallel HTB can run in a 1- way 2-stage pipelined fashion on a multi-core processor, which not only increases the processing speed significantly, but also performs well in stability.

RELATED WORK:

As cloud computing is a relatively new concept, it is still at the early stage of research. Most of the published works focus on general description of cloud, such as its definition, advantages, challenges, and future [1]. In detail, security is a very popular and important research field in cloud computing. Some researches focus on the data confidentiality and integrity in cloud computing. So the research on this field is meaningful. Our work is based on HTB, a packet scheduler implemented in the Linux kernel [7].While there are no concept of "class" in SFQ and TBF, they could not process multitype of data flows. PRIO is an optimization to simple FIFO queue discipline. CBQ is an algorithm which can control rates for different traffics, and HTB is based on

CBQ. [8] parallelizes the network stack in Linux user space in a pipelined fashion, which provides an environment for HTB to be ported from Linux kernel space and parallelized. [2]The data that is stored and/or transmitted on the Internet has been called "the blood of the IT". Along with the infrastructure and network based applications, data storage has been recognized as one of the major dimensions of information technology. The prosperity of Cloud Computing requires the moving from server-attached storage to distributed storage. [3]Today, we have the ability to utilize scalable, distributed computing environments within the confines of the Internet, a practice known as cloud computing. In this new world of computing, users are universally required to accept the underlying premise of trust. Within the cloud computing world, the virtual environment lets users access computing power that exceeds that contained within their own physical worlds. Typically, users will know neither the exact location of their data nor the other sources of the data collectively stored with theirs. [5]Many cloud storage providers declare that they store multiple replicas of clients' data in order to prevent data loss. However, currently there is no guarantee that they actually spend storage for multiple replicas. Recently a multiple-replica provable data possession (MR-PDP) protocol is proposed, which provides clients with the ability to check whether multiple replicas are really stored at the cloud storage servers.[4]Cloud Computing has been envisioned as the next generation architecture of IT Enterprise. In contrast to traditional solutions, where the IT services are under proper physical, logical and personnel controls, Cloud Computing moves the application software and databases to the large data centers, where the management of the data and services may not be fully trustworthy.[7]computing is being transformed to a model consisting of services that are commoditised and delivered in a manner similar to utilities such as water, electricity, gas, and telephony. In such a model, users access services based on their requirements without regard to where the services are hosted. Several computing paradigms have promised to deliver this utility computing vision and they include Grid computing, P2P computing, and more recently Cloud computing. [6]A discrete metric trust management model based on cloud model is proposed to solve the problem of fuzziness and randomicity in description and reasoning of trust relationship in open networks. Base-cloud and acceptance factor of trust are used to implement the reasoning mechanism of trust cloud, which can deal with the trust recommendation and synthesis of multiple trust paths, and implement the propagation of trust relationship. The simulation results show that the proposed model can lead to higher cooperation successful rate compared with the previous models.[8]The industry wide shift to multi-core architectures arouses great interests in parallelizing sequential applications. [9]Privacy is an important issue for cloud computing, both in terms of legal compliance and user trust, and needs to be considered at every phase of design. In this paper the privacy challenges that software engineers face when targeting the cloud as their production environment to offer services are assessed, and key design principles to address these are suggested.[10]This paper introduces a practical security model based on key security considerations by looking at a number of infrastructure aspects of Cloud Computing such as SaaS, Utility, Web, Platform and Managed Services, Service commerce platforms and Internet Integration which was introduced with a concise literature review. The purpose of this paper is to offer a macro level solution for identified common infrastructure security requirements. This model with a number of emerged patterns can be applied to infrastructure aspect of Cloud Computing as a proposed shared security approach in system development life cycle focusing on the plan-built-run scope.

PARALLEL HTB:

This section presents the algorithm and implementation details on parallelizing HTB by applying lock-free design principles for high speeds. We first describe the basic idea of pipeline based HTB. Then, the new algorithm is introduced to eliminate the data race conditions. Based on this, the lock-free FIFO is designed, and a 2-stage pipeline is constructed. HTB has two main operations, *enqueue* and *dequque*. The “enqueue” is responsible for finding the leaf class that a

packet belongs to, and inserting the packet into a queue of the leaf class. The “dequeue” calculates the sending mode (*can_send*, *borrow*, *cannot_send*) of each class and chooses a proper class which can send a packet. When a leaf class

needs to borrow bandwidths, the tree is traversed

extensively to find an appropriate ancestor to borrow. Between *enqueue* and *dequque*, the packet queue potentially involves data racing conditions. Our basic idea is mapping these two onto different CPU cores and executing them in

pipeline, thus the HTB task can be parallelized.

By applying fast lock-free FIFOs, application-level software pipelining can be efficiently implemented. In parallel HTB, the FIFO is the packet queue for each of the leaf class. Our

designing goal is that the only critical region that might be accessed by *enqueue* and *dequeue* concurrently is the FIFOs, and because the FIFOs are lock-free, there will not be extra lock/unlock time on accessing them. There will be FIFOs with the same number of leaf classes connecting *enqueue* and *dequeue* operations. An efficient lock-free FIFO implementation makes such massive application-level pipelining possible. core pipelining using

application-level FIFOs *B. Eliminate Locks* The main obstacle for realizing the pipeline is concurrency on the global resource. Because in the original HTB, *enqueue* and *dequeue* will both access the global HTB trees, the two operations can't be ispatched to 2 different cores. This requires new algorithm for *enqueue*/*dequeue* operation.

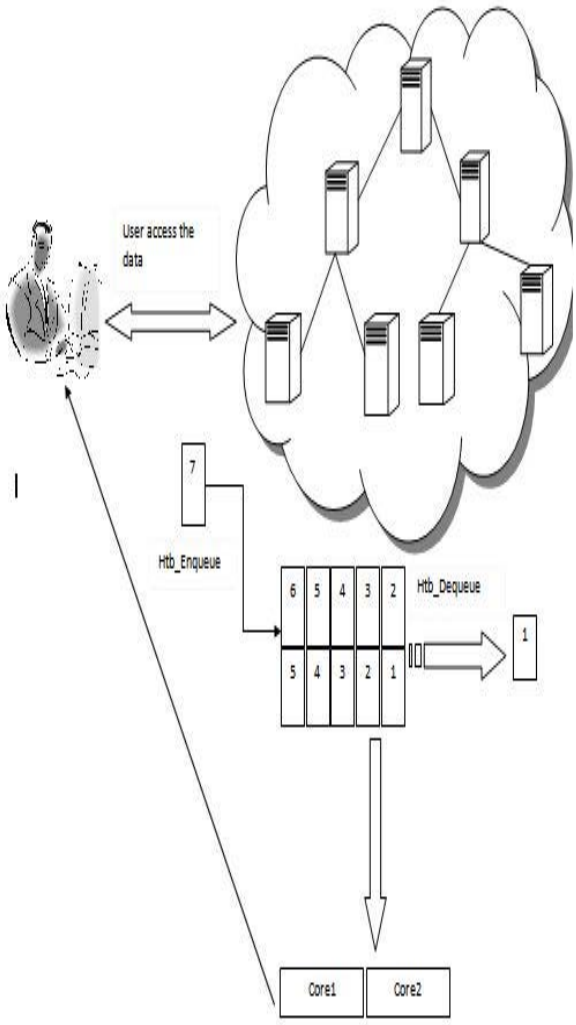
Lock-free FIFO:

HTB has two main operations, enqueue and dequque. The "enqueue" is responsible for finding the leaf class that a packet belongs to, and inserting the packet into a queue of the leaf class. The "dequeue" calculates the sending mode (can_send, borrow, cannot_send) of each class and chooses a proper class which can send a packet. When a leaf class needs to borrow bandwidths, the tree is traversed extensively to find an appropriate ancestor to borrow. Between enqueue and dequque, the packet queue potentially involves data racing conditions. Our basic idea is mapping these two onto different CPU cores and executing them in pipeline, thus the HTB task can be parallelized.

Now the structure that might be accessed by enquene and dequeue at the same time is only the packet queue. Using lock-free FIFOs as the packet queue will totally eliminate locks in the parallel HTB. For the two-core software pipeline shown, at any time enqueue can insert only one packet to a leaf class, and dequeue can fetch only one packet from a leaf class. Therefore, the lock-free FIFO should be a one producer and one consumer type. For multi-producer and multi-consumer cases, the TCP connection affinity can be used to guarantee that a FIFO for each class still has one producer and one consumer for multiple pipelines. Unlike lock-free FIFOs designed for fast core-2-core communication, minimal delay is the design goal for HTB FIFO. Since there are FIFOs for each leaf class, the chance of enqueue and dequeue operating on the same FIFO is quite low. Therefore, the advanced cache-line distance and cache-line aggregation techniques introduced in seems irrelevant, and a simpler version of an array of FIFOs is needed. Lists an array of lock-free FIFOs for the 800 class packet queues. Please note that _ as proved in, the cache coherence protocol implemented on a modern multi-core guarantees that an aligned access is linearizable, and stores are not visible to remote-core's loads until the stores are no longer speculative. Therefore, for each FIFO the consumer dequeues values in the same order that they were enqueued in the producer's program execution order. The head and tail of a FIFO are owned by two different cores and are modified independently (Lines 6 and 15). We use two arrays to separate the head and the tail of a FIFO into separate cache lines to avoid cache trashing. The FIFO status is detected using element value directly without comparing the value of head and tail. Since a FIFO element contains a pointer to a packet, a NULL can be used for detecting an empty queue and a non-NULL for a full queue. Such domain knowledge makes the FIFO design simple and efficient.

```
FIFO_put (FIFO_ELEMENT *data, int i) {
 head = queue_head[i];
 if (NULL != queue[i][head])
 return FLASE; //! The queue is full
 queue[i][head] = *data;
 queue_head[i]++; //! mod add
 return TRUE;
 }
```

SYSTEM ARCHITECTUREE:



MODULES:

- User Interface Design.
- Parallelizing on HTB.
- Packet scheduling on HTB.
- Htb_Enqueue.
- Htb_ Dequque.

❖ **USER INTERFACE DESIGN:**

In this module user has to create an account for only allowing right persons to access the resources. All the details will be stored in database which is placed in server. If he entered correct user name and password then he will be able to access the cloud. Logging in is usually used to enter a specific page, which trespassers cannot see. Logging out may be performed explicitly by the user taking some action, such as entering the appropriate command, or clicking a website link labeled as such. It can also be done implicitly, such as by the user powering off his or her workstation, closing a web browser window, leaving a website, or not refreshing a webpage within a defined period. In the case of web sites that use cookies to track sessions, when the user logs out, session-only cookies from that site will usually be deleted from the user's computer. In addition, the server invalidates any associations with the session, making any session-handle in the user's cookie store useless.

❖ **PARALLELIZING ON HTB:**

In this module we Parallelizing HTB by applying lock-free design principles for high speeds. In parallel HTB, the FIFO is the packet queue for each of the leaf class. Our designing goal is that the only critical region that might be accessed by enqueue and dequeue concurrently is the FIFOs, and because the FIFOs are lock-free, there will not be extra lock/unlock time on accessing them. There will be FIFOs with the same number of leaf classes connecting enqueue and dequeue operations. An efficient lock-free FIFO implementation

makes such massive application-level pipelining possible. Mapping operations of enqueue and dequeue onto different CPU cores and executing them in pipeline.

❖ **PACKET SCHEDULER HTB:**

In this we are scheduling the packet using the FIFO. The Packet schedulers determine the order of packets transmission in a network. First-In, First-Out (FIFO) queuing is the most basic queue scheduling discipline. In FIFO queuing, all packets are treated equally by placing them into a single queue, and then servicing them in the same order that they were placed into the queue. It does not emphases throughput since long processes are allowed to monopolize CPU. FIFO queuing is also referred to as First- Come, First-Served (FCFS) queuing.

❖ **HTB_ENQUEUE:**

In this module we are going to execute Enqueue operation in which, the "enqueue" is responsible for finding the leaf class that a packet belongs to, and inserting the packet into a queue of the leaf class.

Enqueue will activate the sending service for a leaf class if its packet queue length changes from 0 to 1 after inserting a packet

❖ **HTB_ DEQUQUE:**

The "dequeue" calculates the sending mode (can_send, borrow, cannot_send) of each class and chooses a proper class which can send a packet. When a leaf class needs to borrow an appropriate ancestor to borrow.

Dequeue walks the same tree downward in the network, following the borrow link, finding the leave node, and then sending the packet out. Now the packet queue of the leaf changes to 0. So its service has to be deactivated and the borrow link has to be removed, though in fact this class (marked as yellow) can still borrow from its parent.

EXPERIMENT AND RESULT:

In our evaluation, we design the HTB bandwidth tree for a data center in the cloud. As shown in Figure 6, the leaf nodes represent different types of service for each user. There are two types of services: 0.5Mbps/1Mbps and 2Mbps/12Mbps. The first service is for the common application such as web service, and the second one is for those requiring more bandwidth, e.g. streaming service. The number of leaf nodes in Figure 6 is 800. Because the pressure on parallel HTB is higher for more leaf nodes, the leaf nodes number is doubled to 1600 in Exp.5 and Exp.6 to

test potential capacity of parallel HTB, and the whole bandwidth increases to 2Gbps. For the evaluation platform, the Intel Core 2 Quad processors, Xeon E5410 are used in our experiments. The processor runs at 2.3GHz, and has two 6MB L2 caches with 4B cache-line size and 1333MHz FSB. The system is configured to run the 64-bit Linux 2.6.x kernel and the code

is compiled by the GCC 4.1.2 with –O2 option.

CONCLUSION:

HTB is proposed for effective and stable traffic control in the cloud. Based on new algorithms on accessing key data structures and the usage of lock-free FIFO, the parallel HTB can run in a pipelined fashion. The theoretical analysis and evaluation results both indicate that parallel HTB is more suitable for cloud computing, due to its excellent performance on both line rate and stability.

REFERENCES:

[1] Zheng li, Nenghai fu, Zhuo Hao "MOE-microsoft key laboratory of multimedia computing & Communication University & science & technology" of china, Hefei, china.

[2] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring data storage security in cloud computing", Proc. of IWQoS'09, Charleston, South Carolina, USA, 2009, pp.1-9, doi: 10.1109/IWQoS.2009.5201385.

[3] N. Leavitt, "Is cloud computing really ready for prime time", IEEE Computer Society, vol.42, Issue.1, 2009, pp. 15-20. [4] L.M. Vaquero1 et al., "A Break in the Clouds: Towards a Cloud Definition", ACM SIGCOMM, vol.39, no.1, 2009, pp. 50.

[4] J. Heiser and M. Nicolett, "Accessing the Security Risks of Cloud Computing", Gartner Inc.,Stanford,CT,2008,http://www.gartner.com/DisplayDocument?id=685308.

[5] C.L.Zhang and Y.Liu, "A Cloud-based Discrete Metric Trust Management Model in Open Networks", Journal of Internet Technology, vol. 10, no.1, 2009, pp.79-82.

[6] HTB Home, http://luxik.cdi.cz/devik/qos/htb/

[7] J. Wang, H. Cheng, B. Hua, and X. Tang, "Practice of Parallelizing Network Applications on Multi-core Architectures", Proc. of ACMICS'09, New York, June, 2009, pp.204-213, doi: 10.1145/1542275.1542307.

[8] M. Armbrust et al., "Above the Clouds: A Berkeley View of Cloud Computing", epartment of Electrical Engineering and ComputerSciences, University of California at Berkeley, Report No. UCB/EECS-2009-28, CA, USA, 2009.

[9] Z. Hao and NH. Yu, "A Multiple-Replica Remote Data Possession Checking Protocol with Public Verifiability", in the Second International Symposium on Data, Privacy, & E-Commerce, Buffalo, USA, Sept., 2010.