Bid Data from Auctioning Application

K Saranya[1], Uma Maheshwari S[2], Sumaiya L[2], Soundharya A[2]

[1] Assistant Professor of Information Technology,
St. Joseph College of Engineering, Sriperumbudur, Chennai.
[2] Final Year Information Technology,
St. Joseph College of Engineering, Sriperumbudur, Chennai

Abstract--- In relational database service in the cloud usually achieves energy efficiency by using virtualization technology, which consolidates multiple independent database systems into a single physical machine while enforcing the hardware-level isolation among them. However, the disk I/O performance is inevitably hurt due to the resource contention on the shared device. While handling normal users' data in dayto-day life, RDBMS is sustainable due to its structure of rows and columns. But due to the changing paradigm in this digital era where almost every business requires every data from customers which can be optimized and monetized to improve their business. So, customer data are considered as the new gold, as almost every industry uses customers' data and other data to analyze their business flow where they can improve their service to their customers and dominate the industry.

Telecom industries follow the same strategy to get many data as possible then they can improve their business. Yet the problem arises when it comes to keeping their database sustainable while handling big data, as DBs are not designed to handle big data, some data loss may occur and even our database vulnerability may become obvious to end-user. In our case we propose a model to handle Big data in our Auctioning application where millions of users may bid within seconds ultimately a huge amount of data entry into a database, our model prevents data loss in a high-speed transition commit.

Keywords--- virtual machine, energy efficiency, big data, transaction commitment

I.INTRODUCTION

As we all know that data security is always an important acceptance of any organization that mainly relies on data protection regulation. Corporate spend a lot of money to protect the data from unauthorized access and data corruption else companies often face a slew of immediate financial consequences. Companies also face the real issue of losing the trust of their key customers and clients. Our proposed model is equipped with tools to protect the data from hackers and make sure only

authorized users can utilize the data. When a company faces a necessity to share its sensitive data with its client our encryption algorithm will make sure data is secured in 1st place. Our model makes sure that encrypted data are only accessed by authenticated users by monitoring their physical addresses. When this unique physical address is not matched with our registered information it is intimated to admin and company. Proposed model encompasses every aspect of information security from the physical security of hardware and storage devices to administrative and access controls, as well as the logical security of software applications. It also includes organizational policies and procedures. If one of the conditions is met, the flush procedure would be activated. Such management overheads complicate transaction I/O in a database system. To deal with these issues, synchronous method in the VM layer could be enforced, showing a Sync-Sync model. It allows each transaction commitment to directly produce a group of disk operations in the VMM to store the user data persistently. In this way, memory pages in the buffer pool are always clean and the database in VMs can bypass the complex management on dirty data.

Accordingly, the related CPU overhead of asynchronous manner can be freed and used in another places, such as parsing the upcoming user requests and thus to improve their response times. However, the downside of the synchronous approach is the storage performance, since each transaction commitment indicates that user request must wait until a group of disk operations is completed in the VMM. What is worse, synchronization in the VMM further delays the above operation. Due to server consolidation, multiple database VMs share the underlying storage device. Each of the VM has an independent image file and often resides in an uncertain storage position. Considering the individual differences of running VMs, their distributions of physical disk blocks are irregular and hard to investigate. Therefore, when concur rent synchronous operations are rushed from the respective VM, the storage device has no choice but receives them always as random I/O.

The resultant performance is poor and far from the sequential one, regardless of whether the magnetic disk or solid-state drive is used. We have implemented a VMSQL prototype in a virtualized database environment based on QEMU-KVM and InnoDB engine. The typical realistic workloads are used to test the effectiveness of VMSQL, the result demonstrates that VMSQL improves the performance of virtualized database system by 41.9% on average (up to 87.5%) when eight VMs are committing transactions simultaneously. Meanwhile, VMSQL brings about 13.8% overhead (up to 18.4%) on the performance of VM operations at the host layer. We believe this kind of overheads are moderate to the original system because we use the worst-case scenarios for pessimistic evaluation.

II. RELATED WORKS

Li et al propose HypeGear which shares the similar disk model with VMSQL in a virtualized environment. It uses Sign Post in VM to apply synchronous operation into guest file system and a block-level in-memory cache called GearCache is interposed into VMM to coordinate the write flows among multiple co-located VMs. Both models aim to improving the CPU consumption at VM layer and random I/O at VMM layer. But Hype Gear differs from VMSQL in two aspects as follows. First, Hype Gear mainly focuses on the guest file system. It is a relatively general method across different applications which located on file system, regardless of their specific I/O internals. This character istic results in that HypeGear is hard to cover the scope of RDBMS, because a typical database application often access the underlying storage without the involvement of file system. Instead, VMSQL is a dedicated method for vir tualized RDBMS. Its synchronous manner considers about the internal details of a typical RDBMS seriously, who often shows the very different I/O behavior with another appli cations. Second, Gear Cache is located in memory, which makes HyperGear is not an ACID-compliant solution on RDS. A crash event in VMM could lead data to loss. By leveraging a WAL mechanism (namely TransTown) in the underlying persistent storage, VMSQL not only provides the improvement on random I/O, but also maintains the ACID features for RDS VM. Recently, Chen et al devise three methods to improve the extra synchronous operations brought by QCOW2 image format [26]. Meanwhile, the features of QCOW2, such as fulldisk encryption, de-duplication and snapshot, are also persevered. This work is an improvement at the host level of cloud infrastructure. It concentrates on the format of guest image file

and there is no overlap with VMSQL. Therefore, VMSQL could exploit this improved guest image format to provide

the better disk I/O performance of RDS.

III.BACKGROUND AND MOTIVATION NATIVE SQL DATABASE SYSTEM

Native SQL Database System A typical RDBMS includes two main components: a query processor and a storage engine . The query processor parses and executes SQL queries which are carried in the user request; this procedure mainly consumes the CPU resource. The storage engine is responsible for storing user tables and indexes; it responds to the parsed requests from the query processor to read or write data on a storage device. Generally, when the query processor has parsed a com mitted transaction (denoted by Tc), the storage engine pro ceeds this operation with one of the following two ways: (1) flushes Tc immediately to the persistent storage; (2) allows Tc to wait in a buffer for a time period, to be merged with other committed transactions to yield a better transaction throughput. We call the first one synchronous due to its obedience with the user or application semantic. Instead, the second one is called asynchronous because the buffer will delay the real transaction commitment. By evaluating the synchronous and asynchronous path respectively, Table 1 exemplifies their differences on both CPU consumption and transaction performance in the Inn oDB storage engine of a MySQL server (version 5.6.31). The server has one Intel i7-4790 3.6GHZ processor with hyper threading (HT) enabled8 , 32GB DDR3 RAM, 1TB WDC (Western Digital Corporation) hard disk drive and dual Full duplex Intel Pro/1000 Gbit/s NIC. CentOS 7 (1511) with Linux kernel 4.1.25 is used as the host OS. In this experi ment, we use tpcc-mysql9 in another separated physical machine, which is equipped with the same HW/SW con figurations with the former MySQL server, to benchmark the InnoDB engine. Two machines are connected via a 10G switch.

Virtualized SQL Database System

Mainly for keeping the VM’s write semantic, the VMM often uses synchronous method to deal with this operation. Therefore, both models of Sync-Sync and AsyncSync can be

employed in current virtualized RDS. To make a case here, we use QEMU 2.5.011 with KVM support in Linux kernel to create four VMs in the MySQL server which is used in Block IO Controller12 (blkio for brevity) is leveraged, to allow colocated VMs sharing the underlying disk bandwidth fairly. Each VM is assigned two virtual CPUs, 2GB memory and a 100GB image fileis allocated using the QCOW2 format. CentOS with kernel is installed for guest OSes. The driver of virtual disk and network in the guest OS uses virtio. The cache model of QEMU uses none mode to act as the synchronous manner in the VMM. Other HW/SW configurations and experimental environments are identical to the aforesaid MySQL server, except that four tpcc_mysql instances are booted together in the client machine to connect to their respective virtualized MySQL servers. In addition, each tpcc_mysql instance has a dedicated CPU core in the client machine by using taskset utility. The last twelve rows in Table 1 show the results. Based on them, we make the following observations in the virtualized system:

(1) Due to four VMs sharing the underlying hardware, the transaction performances (TpmC) of all configurations are deteriorated greatly compared with the native case. AWait demonstrates this change mainly derives from the random I/O, since the value of AWait in the host layer is increased by about 9.3 times from the averaged 14.4 ms to 134.2 ms.

(2) As t is increased, the transaction performances of Async-Sync models, which include both of Async0 and Async2, are degrading due to the context switching be tween InnoDB I/O threads. In contrast, the synchronous path (SyncSync model) presents a more steady down side than its counterparts. The main reason is similar with the native case: the CPU cycles, which are decoupled from the asynchronous path, limit the cost of thread context switchings .

IMPLEMANTATION

In this section, we discuss the implementation of VMSQL on QEMU-KVM hypervisor and

MySQL InnoDB storage engine. Under minor adjustment, VMSQL can also be applied to other hypervisors and database systems13. We describe the detailed implementation in guest and host system respectively.

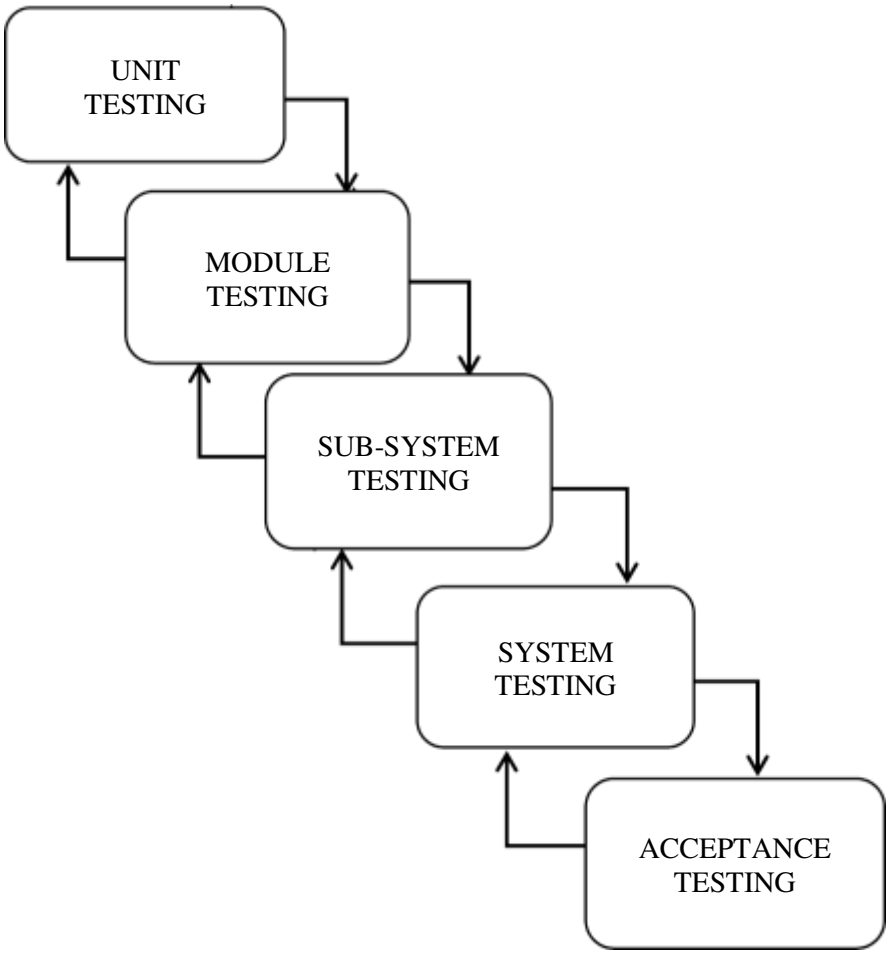
(i) Guest System

There are two ways to deploy InnoDB storage engine on a typical OS: (1) file system and (2) raw block device. To synchronize in the first case, we mount the file system, which

stores the transaction log file of InnoDB, with the parameter sync. In this way, guest file system enforces the synchronized write operations on the upper database, regardless of the specific I/O manner it adopts. As for the raw block device, the value of log_trx is configured to 1, allowing each transaction commitment to be passed into the persistent storage at once, without any delay in the log buffer or log file buffer. Besides, a virtual block device denoted by Bv should be added to the VM, which operates the function of Logging Frontend to identify the transaction semantic from the database. To do this, in the VMM layer we first create a new physical block device denoted by Bp, which acts as the infrastructure of TransTown, by using fdisk command. Then, we use the boot script of VM to import Bp into VM kernel. A new pair of block device back-end and front end driver, namely the mode of split driver, would au tomatically establish the connection between the VM.

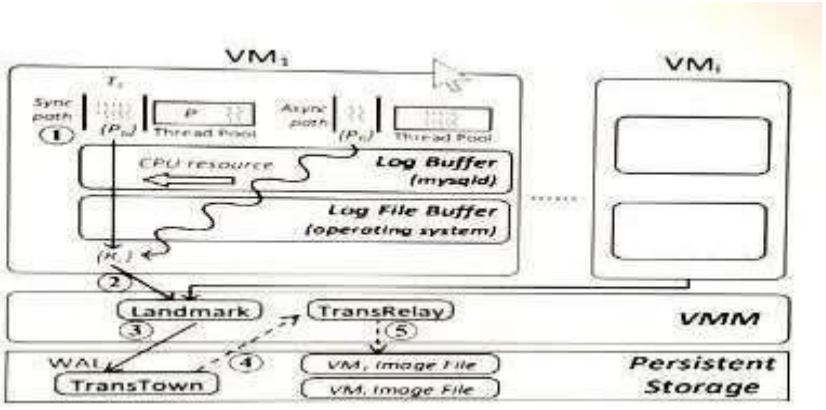
(ii) Host System

In QEMU-KVM hypervisor, when the VM issues I/O re quests to the virtual block device, a process in host user space, called QEMU I/O process, takes over this I/O request and translates it into a physical operation. Then, the process uses read() and write() system calls to deliver this physical operation to the associated image file, which is the same as operating a regular file in the host file system. We implement TransTown by intercepting the above procedure. According to the design of TransTown, its storage space is pre-allocated to provide continuous disk blocks while keeping its small capacity to fit in the host page cache. We use three regular files in host file system, denoted by Ft, Fm and Fc, to act as the TransTown's slot array, the associated meta- data and its CB respectively. All of them use the fallocate() system call in ext4 file system to pre-allocate the on-disk space, reducing fragmentation.



Overview of VMSQL design

Three new components, namely Landmark, TransRelay and TransTown, are added into a typical RDS system. All of them are marked with the rounded gray rectangle. Numeric labels with circle refer the general procedure of a committed trans action in VMSQL. The solid lines with arrow indicate the synchronous operation of VMSQL while the dotted ones indicate the asynchronous operation which is decoupled from the original procedure of transaction commitment.



Generally, VMSQL employs the Sync-Async idea to handle transaction commitment in a virtualized database system. Synchronous operations in the VM layer are en forced and allow the associated data of a transaction to quickly pass through the VM space. When it is stored in the persistent storage, the acknowledgment is returned and informs the corresponding application. By doing so, VM semantics are strictly ensured and the consistency between the user and database can be maintained across a crash. Meanwhile.

IV. CONCLUSION AND FUTURE WORK

In the big data era, traditional RDBMS confronts with the exponentially increased user queries and transaction commitments, especially in a RDS environment which is built with virtualization. The key components in a RDS, including the query processor and storage engine, amplify their CPU and storage overheads respectively due to the continuously incoming data and the hardware resource contention among co-located RDSs. To address this issue, we propose VMSQL, a disk I/O model based on Sync-Async procedure. With the scale of running eight VMs in a host machine, our evaluation verifies the performance advantage of VMSQL over the original models. In the meanwhile, the ACID features of a traditional RDBMS is retained. However, VMSQL brings moderate overhead on several ordinary VM operations in VMM management, including VM close, VM migration and VM reboot after a crash event. In our future work, we plan to:

(1) run VMSQL in the server-level hardware, such as Xeon CPU and SSD/HDD hybrid

torage device, to allow VMSQL becoming a more general method across the different infrastructures of data center;

(2) applying VMSQL idea to the in-memory databases such as Redis, which also uses logging like technology to persist the data, to expand the applicablescope of our work .

REFERENCES

[1] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fet zer, "SCONE: Secure Linux Containers with Intel SGX," in 12th USENIX GA: USENIX Association, November 2016, pp. 689–703.

[2] K. Ota, M. Dong, J. Gui, and A. Liu, "QUOIN: Incentive Mecha nisms for Crowd Sensing Networks," IEEE Network, vol. 32, no. 2, pp. 114–119, March 2018.

[3] A. Floratou, N. Megiddo, N. Potti, F. Ozcan, U. Kale, and ¨ J. Schmitz-Hermes,

"Adaptive Caching in Big SQL Using the HDFS Cache," in Proceedings Cloud Computing (SOCC 2016). New York, NY, USA: ACM, 2016, pp. 321–333.

[4] J. Wu, M. Dong, K. Ota, J. Li, and Z. Guan, "Big Data Analysis Based Secure Cluster Management for Optimized Control Plane in Software-Defined Networks," IEEE, vol. 15, no. 1, pp. 27–38, March 2018.

[5] M. Ben-Yehuda, O. Agmon Ben-Yehuda, and D. Tsafrir, "The Nom Profit-Maximizing Operating System," in Proceedings International Conference on Virtual Execution Environments (VEE 2016). New York, NY, USA: ACM, 2016, pp. 145–160.

[6] L. Cheng, J. Rao, and F. C. M. Lau, "vScale: Automatic and Efficient Processor Scaling for SMP Virtual Machines," in Proceedings of the Eleventh European Conference on Computer Systems (Eurosys 2016). New York, NY, USA: ACM, 2016, pp. 2:1–2:14.

[7] H. Li, K. Ota, M. Dong, and M. Guo, "Mobile Crowdsensing in Software Defined Opportunistic Networks," IEEE Communications Magazine, vol. 55, no. 6, pp. 140–145, 2017.