2012 IEEE International Conference on Cluster Computing Workshops

High Availability Through Output Continuity

Wei Ye*, Yaozu Dong†, Ruhui Ma*,1, Alei Liang* and Haibing Guan*
* Shanghai Key Laboratory of Scalable Computing and Systems
Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240
Email: {yxysdcl, ruhuima, liangalei, hbguan}@sjtu.edu.cn
† Intel China Software Center, Shanghai, 200241, China
eddie.dong@intel.com

***Abstract*—Virtual machine (VM) based state machine approaches, i.e. *VM replication*, provide high availability without source code modifications; unfortunately, existing VM replication approaches suffer from excessive replication overheads. On the other hand, machine state replication is an overly strong (and therefore less efficient) requirement for high availability in the networked client-server system. In this paper, we propose a generic and highly efficient high availability solution through *output continuity*. Output continuity considers a backup server as a valid replica as long as the responses generated so far are the same between the backup and primary servers; the server state is propagated from the primary server to the backup server if and only if the previous outputs from the backup and primary servers are different.**

Keywords-fault tolerance; high availability; replication; state machine; checkpoint

I. INTRODUCTION

Surviving from server hardware failure is critical to achieve high reliability and availability for networked client-server systems. The state machine approach provides high availability by replicating the server state to multiple independent physical platforms (so as to survive from hardware failures such as fail-stop failures) [1]. Virtualization enables application agnostic high availability through Virtual Machine (VM) replication (i.e., replicating the virtual server or virtual machine from the primary node to the backup physical node). For instance, in lock-stepping [4], the primary VM and the backup VM execute in parallel for deterministic instructions, but lock-and-step for un-deterministic instructions, so as to achieve the exactly same state. In addition, continuous checkpoint [2] synchronizes the VM state from the primary VM to the backup VM with very high frequencies (e.g., up to 40 times/sec in Remus [2]) and buffer the outbound packets during each epoch until a successful checkpoint; consequently, the service can seamless fail-over to the backup VM when the primary host suffers from fail-stop failure.

Unfortunately, VM replication suffers from excessive performance overheads. Lock-stepping suffers from excessive replication overheads with the multi processor (MP) guest, where each memory access might be non-deterministic [4][6]. Continuous checkpoint suffers from extra latency and overheads due to frequent checkpoint. Longer checkpoint interval severely impacts the service throughput, while shorter interval means more frequent VM checkpoint, which consume a lot of CPU cycles and network bandwidth.

On the other hand, VM state replication is an overly strong (and therefore less efficient) requirement for high availability. From the client's point of view, high availability is achieved as long as the client can continue to receive valid response (according to the service semantics) after the primary server fails. Therefore, even if the primary and backup servers differ in their states, high availability can still be achieved as long as their outputs remain the same. In this paper, we propose an application agnostic high availability solution through *output continuity*. Output continuity considers a backup server as a valid replica as long as the responses generated so far are the same between the backup and primary servers; the server state is propagated from the primary to the backup if and only if the previous outputs from the backup and primary servers are different, so as to make sure that future outputs from the backup and primary will remain the same.

II. MOTIVATION

A client-server system can be considered as a request and response system. The client sends its request to server and the server respond to the client. The request and response packets form a stream of packet series (denoted as r and R respectively), as shown in equation 1 & 2. r_i and R_i denote the i_{th} request and response packet respectively.

$$r = \{r_0, r_1, r_2, ..., r_n, ...\} \quad (1)$$

$$R = \{R_0, R_1, R_2, ..., R_n, ...\} \quad (2)$$

The response of the current request is typically determined by the request stream consisting of prior requests; that is, the n_{th} response packet R_n is a function of the request stream $\{r_0, r_1, r_2, ..., r_n\}$, as shown in equation 3. Previous literatures [3] actually built server hot swap solution base on this assumption

[1]Corresonding Author

978-0-7695-4844-9/12 $26.00 © 2012 IEEE
DOI 10.1109/ClusterW.2012.23

for certain applications and usage model (e.g., Uniprocessor system based Apache server).

$$R_n = f_n(r_0, r_1, r_2, ..., r_n) \quad (3)$$

$$R_n = g_n(r_0, r_1, r_2, ..., r_n, u_0, ..., u_m) \quad (4)$$

However, in many cases, the response of the current request is determined both the prior request stream and the execution of un-deterministic instructions (e.g. I/O, interrupt and access of Time-Stamp Counter) in the server application. Consequently the response packet R_n can be considered as a function of both the prior request stream and the execution result of un-deterministic instructions, as shown in equation 4 (where the execution result of an un-deterministic instruction is denoted as a variant u_i). The variant u_i is different by nature between the primary and backup servers. More importantly, the result of memory access in an MP system is typically un-deterministic, which means the un-deterministic instructions are pervasive in modern server system.

On the other hand, from the client's point of view, every response stream delivered by equation 4 (no matter whatever u_i is), is a valid response according to the application semantics, and we call the stream delivered by equation 4 as a *semantic stream* and packet R_k^n is an *semantic packet*. All possible *semantic stream*s form a set D_R , as depicted in equation 5, and we denote the *semantics stream* R^m as a semantic response stream from an instance VM *m*.

$$D_R = \{R^1, R^2, R^3, ..., R^n, ...\} \quad (5)$$

Consequently, as long as the response stream R received by the client is belongs to D_R, i.e. $R \in D_R$ (i.e., it is a *semantic stream*), the client-server system is working correctly according to the service semantics. Therefore, if a server system can continue to generate a *semantics response stream* even after hardware failures, the client-server system can be considered as a high availability system.

Output continuity implements high availability by constructing a backup VM whose output packets are identical to the primary VM (before failover), and performs on-demand VM checkpoint if they are different. When the primary VM fails, the replica (backup VM) can take over and send the response to the client. From the client's point of view, the response stream consists of packets from the primary server *p* (packets 1 to k of R^p) and the packets from backup server *b* (starting from packets k+1 of R^b), as shown in equation 6, where the failover is achieved by switching output packets from primary server to backup server at the $k+1^{th}$ packet.

$$C = \{R_1^p, ..., R_k^p, R_{k+1}^b, ...\} \quad (6)$$

III. DESIGN

Even though the execution of un-deterministic instructions may cause immediate difference in VM machine states, the primary and backup servers may still generate identical outputs in short term (referred to as ***output packets similarity*** in the paper). For instance, the TCP timestamp typically uses system jiffies, and its access is un-deterministic. However, the timestamp becomes different only after sufficiently large guest time drift has been accumulated between the two VMs.

Output continuity implements an efficient and generic server replica by taking the advantage of ***output packets similarity***, eliminating excessive lock-and-stepping of un-deterministic instructions and frequent VM checkpoint overheads. The backup VM is forced to generate same response with primary VM before failover, so that the packets it generated after failover (together with previous packets received by client) is a *semantics response*,

To achieve this goal, all the packets generated in the backup VM must meet equation 7, and a new machine state checkpoint is imposed if an output packet divergence happens between primary VM and backup VM, i.e. violation of equation 7.

$$\forall i \le k, R_i^b = R_i^p \quad (7)$$

The architecture of output continuity is shown in Figure 1. The input packet from client is forwarded from the primary VM to the backup VM, so that both primary VM and backup VM can have same input. The output packet of backup VM, i.e. R_i^b , is then forwarded to the primary VM so that the continuity manager can check if R_i^b is identical to the packet generated by primary VM, i.e. R_i^p . If they are identical, the continuity manager releases the packet and informs backup VM to discard the packet; otherwise if a divergence is observed, the continuity manager forces a new checkpoint to forward the primary VM state to the backup VM, and release all the buffered packets in the primary VM once the checkpoint is completed successfully.

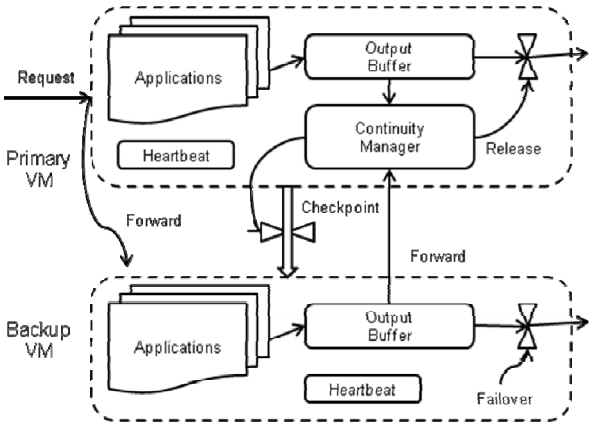

Figure 1. Architecture of Output Continuity.

A. Recovery model

Heartbeat based fail-stop failure detection is general enough for modern platform. Modern X86 servers, such as Intel Xeon series, implement reliability, availability and serviceability (RAS) features. Hardware RAS provides software the capability to successfully detect the (unrecoverable) hardware failure (such as memory, cache and PCIe traffic failure) before the failure is propagated. Consequently the possibility of Byzantine failures is significantly reduced and becomes fail-stop failure. Therefore, our solution base on fail-stop failure is general enough for wide adoption of high reliability and availability.

Backup VM can successfully take over the control, if the primary VM fails after releasing k^{th} packet and before $k+1^{th}$ packet, by releasing packets from the backup VM at the $k+1^{th}$ packet without noticeable difference to client. If the fail-stop failure happens at the time when the primary VM is releasing $k+1^{th}$ packet, the client may see an incomplete packet *k+1*, or even lose the packet *k+1*. We rely on the network stack and application to recover from this type of error, which may happen in typical network as well. Consequently output continuity can achieve high reliability and availability.

B. Local Disk Support

Delta local disk state checkpoint is used for efficient disk I/O checkpoint. Local disk is vastly used in data centers, providing cheap solution of storage such as Google File System [9]. The state of a virtual disk, stored in local disk, is part of the machine state, consequently it requires checkpoint as well between primary VM and backup VM in output continuity. However the state or contents of entire virtual disk of a guest is usually too large to be transferred. Similar to the delta memory checkpoint technology, output continuity caches disk I/O (as delta local disk state) internally and does delta disk state checkpoint at the time of VM checkpoint and release the disk packet cache to physical disk after each successful checkpoint. An alternative solution is to guarantee instant disk packets release if they are identical, which requires forwarding disk packets so that the continuity manager can decide if they can be released immediately.

We argue delta local disk state checkpoint is efficient in output continuity for a couple of reasons: 1) the *similarity* of disk I/O packets between two different VMs has much higher possibility than network I/O packets, and hence disk packets can likely be released in place which won't bring additional overhead to VM checkpoint. 2) the bandwidth spent for disk I/O packets forwarding is much less than the network packets a virtual server may generate in many server usage case, consequently the overhead due to disk I/O packet forwarding is negligible as well.

IV. OPTIMIZATIONS

Improving the ***output packets similarity,*** or duration before a different output packet is generated between primary VM and backup VM, is critical to the performance of output continuity. The performance of output continuity highly depends on the overhead spent in the VM checkpoint. The longer the output packet similarity is the less checkpoint number is, and the more dirty memory pages are transferred per checkpoint. However the dirty memory pages does not increase linearly with the interval the VM runs, and actually it increases roughly logarithmically with the execution interval (per our preliminary observations), as shown in table I. Consequently, the longer the output packet similarity, the less total checkpoint overhead and the better performance is.

TABLE I. DIRTY PAGE # VS. EXECUTION INTERVAL

Benchmark	Clients	Interval(ms)		
		2000	***200***	***20***
SPECweb_Banking	10	2356	805	156
	100	6646	2458	744
SPECweb_Ecommerce	10	1731	379	69
	100	5759	2075	424
SPECTweb_Support	10	1611	431	191
	100	5026	1871	753
Webbench	100	15199	4359	790
	1000	24920	5462	1133

Optimizations to reduce the coupling between output packets and execution result of un-deterministic instructions may greatly improve the ***output packets similarity***. Execution of an un-deterministic instruction in different VM could lead to immediate machine state deviation. However how the deviation may be propagated to the output packet highly depends on the OS and application software implementation. The major un-deterministic instructions include 1) time stamp access such as TSC, 2) external interrupt, 3) randomization number access,

and 4) shared memory access among different processors.

- Performance and applicability

Best effort synchronization, such as lock-and-stepping some of the hot and highly-coupled un-deterministic instructions, may significantly reduce the output difference, e.g. implementing per process best effort timestamp access synchronization, best effort synchronization on guest OS scheduling, and randomization number generation (by means of trap-and-emulation or paravirtualization) to reduce the possibility of next output packet divergence.

However, the characteristics of coupling between output packets and machine state varies per application basis, and we argue the best effort synchronization can serve a large portion of application if not most of them. Concurrent shared memory access is one of the potential coupling sources, we argue most of those potential coupling is in kernel side, and can be addressed by paravirtualizing the guest OS system call.

A. TCP specific optimization

TCP connection is the paradigm of reliable connection in modern client-server system. TCP stack may add additional head to the network packets such as TCP heads which may include time stamp, and a real server process such as Web server may have many concurrent connections, i.e. multiple separately generated output packets from different threads, consequently the output difference may happen more often. We discuss and project following technologies to improve ***output packets similarity*** for TCP connection.

◆ *Time stamp*

TCP connection supports an optional timestamp, which may lead to different output packet easily. Applications may create a TCP connection w/ or w/o time stamp per usage model. In the case of TCP connection using time stamp, the packets come from different VM may be different in time stamp even though their contents are the same. This is because the time stamps such as TSC or system jiffies (delivered from TSC and other timer source) in two different VM may be different as a result of un-deterministic instructions.

We argue the timestamp introduced output packet divergence can be greatly reduced by coarse grain time stamp. The typical time stamp used in TCP head comes from system jiffies in Linux, which is based on OS tick in the unit of 1ms, 4ms or 10ms depending on the tick frequency the guest OS uses. The TCP stack may observe an accumulated tick difference after certain amount of ticks (tens or hundreds or even more) the guest OS has executed, depending on the time virtualization policy and accuracy. On the other hand, TCP stack actually does not require high accuracy of the timestamp. Rather it is mostly used to identify a timeout or a time stamp of an event log. We argue we can enhance guest OS TCP software stack to use coarse grain time stamp such as in the unit of 100ms to greatly increase ***output packets similarity***.

◆ *Randomization number generation*

Randomization number generation is another challenge to TCP packets. Randomization number is widely used in modern OS, and TCP software stack uses OS randomization number generation mechanism to generate the TCP sequence number. OS may use any dynamically changing hardware components such as TSC, thermal sensor result or even past keyboard event as seed to provide a randomization number, which is unlikely identical between 2 VMs running on different host platform even though their previous states are identical. Consequently deviation in randomization number generation mechanism may seriously impact the ***output packets similarity***.

We argue this can be solved by paravirtualizing randomization number generation API in OS to provide identical randomization number between primary VM and backup VM. We argue the overhead of randomization number generation synchronization should be very small because it is unlikely to be used in performance critical path. Consequently the ***output packets similarity*** of TCP packets can be greatly improved.

◆ *Replicating TCP connection only*

In many cases, we may optimize performance of output continuity by replicating TCP connection only for high reliability and availability and leave it to application to recover from those by nature un-reliable connections such as UDP. Although output continuity can provide successful failover for different protocol packets, it may be not necessary from the performance point of view. In network based distributed system, different protocol packets may be on going in parallel; however applications likely only use TCP for reliable connections.

We argue we may optimize output continuity to replicate TCP connection only for vast usage models. Replicating TCP connection only means that a response stream R^b, represented in equation 9, can be treated as a replica of stream R^p, represented in equation 8, if and as if $\forall i, R^b_{ti} = R^p_{Ti}$, ignoring other protocol packets. Base on this, the continuity manager only needs to do VM checkpoint when the TCP packet from primary VM and backup VM has divergence. Consequently, it can greatly improve ***output packets similarity*** as well.

$$R^p = \{.., R^p_{T1}, ..., R^p_{T2}, .., R^p_{T3}, ...\} \quad (8)$$

$$R^b = \{.., R^b_{t1}, ..., R^b_{t2}, .., R^b_{t3}, ...\} \quad (9)$$

V. RELATED WORK AND FUTURE WORK

Surviving from hardware fail-stop failure is becoming more critical in the paradigm of high reliability and availability. With the advance of modern hardware technologies such as Reliability, Availability and Serviceability (RAS) features, most of the hardware failures will be able to be detected by software, consequently most of the hardware failures will be fail-stop failures.

Virtualization enables application agnostic high reliability and availability solution through Virtual Machine (VM) replication. However existing solutions such as lock-stepping [4] and continuous VM checkpoint [2] suffer from excessive performance overhead. TCP specific solutions and application specific solutions are proposed, but they suffer from usage model limitation and application modification. ST-TCP [8] and CoRAL [7] modify TCP protocol to tolerate TCP server failures, and FT-TCP [5] modifies selected applications to be free from un-deterministic instructions respectively.

Output continuity provides a new highly efficient and generic application agnostic solution to high reliability and availability for networked client-server system. It overcomes the performance overhead in VM lock-and-stepping due to excessive memory access instruction replay in multiple processor system [4][6]. It also eliminates the network latency and overhead of excessive VM checkpoint in continuous VM checkpoint [2]. We are currently on the way to fulfill our work, analyzing the coupling characteristics of applications, and collaborate with industry to tune and optimize the solution to achieve the best performance.

ACKNOWLEDGEMENT

This work was supported by National Natural Science Foundation of China (No.60970107, 60970108), National High Technology Research and Development Program (863Program) of China (No.2012AA010905), International Cooperation Program of China (No. 2011DFA10850), and China Postdoctoral Science Foundation (No. 2012M511096).

REFERENCES

[1] Fred B. Schneider, Cornell Univ, "Implementing fault-tolerant services using the state machine approach: a tutorial," *ACM Computing Surveys*, vol. 22, no. 4, pp. 299-319, Dec 1990

[2] Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, "Remus: High Availability via Asynchronous Virtual Machine Replication", in *Proceedings of the 5th conference on Symposium on Networked Systems Design and Implementation*, ser. NSDI '08, San Francisco, CA, April 16-18, 2008, pp. 161-174,

[3] N Burton-Krahn, "HotSwap-Transparent server failover for Linux," in *Proceedings of the 16th USENIX conference on System administration*, Philadelphia, PA, November 03-08, 2002, pp. 205-212

[4] George W. Dunlap , Samuel T. King , Sukru Cinar , Murtaza A. Basrai, Peter M. Chen, "Revirt: Enabling intrusion analysis through virtualmachine logging and replay," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, ser. OSDI '02, New York, USA, Winter 2002, pp. 211-224

[5] Dmitrii Zagorodnov, Keith Marzullo, Lorenzo Alvisi, Thomas C. Bressoud, "Engineering fault-tolerant TCP/IP servers using FT-TCP," in *Proceeding of IEEE International Conference on Dependable Systems and Networks*, ser. DSN '03, San Francisco, California, 2003

[6] George W. Dunlap , Dominic G. Lucchetti , Michael A. Fetterman , Peter M. Chen, "Execution replay of multiprocessor virtual machines," in *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, Seattle, WA, USA, March 05-07, 2008, pp. 121-130

[7] Navid Aghdaie, Navid Aghdaie, "CoRAL: A transparent fault-tolerant web service", *Journal of Systems and Software*, vol. 82, no. 1, pp. 131-143, January, 2009

[8] M. Marwah, S. Mishra, and C. Fetzer, "TCP server fault tolerance using connection migration to a backup server," in *Proceeding of IEEE International Conference on Dependable Systems and Networks*, ser. DSN '03, San Francisco, CA, 2003, pp. 373-382

[9] Sanjay Ghemawat, Howard Gobioff and Shun-Tak Leung, "The Google file system," in *Proceeding of 19th ACM Symposium on Operating Systems Principles*, ser. SOSP '03, Bolton Landing, NY, December 2003, pp. 29-43