

# Kemari: Virtual Machine Synchronization for Fault Tolerance

Yoshiaki Tamura

Koji Sato

Seiji Kihara

Satoshi Moriai

NTT Cyber Space Laboratories  
NTT Corporation

1-1 Hikarinooka, Yokosuka, Kanagawa, 239-0847, Japan  
{tamura.yoshiaki, sato.koji, kihara.seiji, moriai.satoshi}@lab.ntt.co.jp

## 1 Introduction

In recent years, Internet services have been growing in number and functionality. They are typically hosted on a number of commodity servers, and reducing the running cost of these servers is a crucial problem for service providers.

One effective solution is to consolidate multiple servers into fewer physical servers using virtual machines (VMs). Decreasing the number of servers in this way results in better utilization of resources such as floor space, power, and management. The down side is that it creates a single point of failure. Although high-availability architectures have been considered as a way of overcoming this shortcoming, they typically require specially designed hardware or modifications to applications.

In this paper we propose Kemari, a cluster system that synchronizes VMs for fault tolerance. Kemari offers a feasible approach to fault tolerance that does not require the use of specific hardware or modification of applications. This paper contains the design, implementation and evaluation of our system.

## 2 Design and Implementation

### 2.1 Synchronization Model

Kemari aims to keep VMs transparently running in times of hardware failures. To achieve this objective, we need to (1) synchronize the state of VMs on multiple hardware, (2) detect hardware failures and (3) switch to a secondary VM. The synchronization mechanism of Kemari is described in this paper.

There are two main approaches to synchronizing the state of VMs across physical machines. The first approach is lock-stepping [5], in which external events are logged on the primary VM and replayed on the secondary VM. Although this approach enables efficient synchronization, it requires complicated implementation

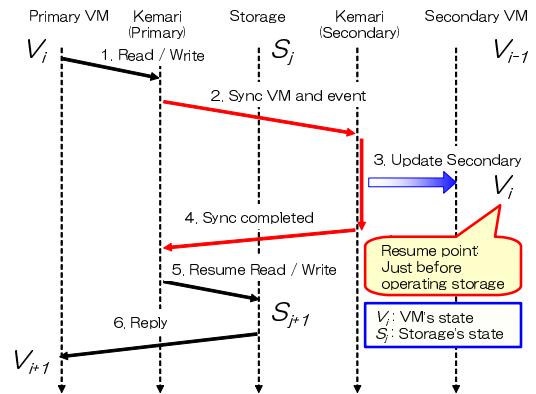


Figure 1: Synchronization steps initiated by an event sent to storage

to handle differences between processor families. The second approach is continuous checkpointing [1], in which the state of the primary VM is frequently transferred to the secondary VM. This approach can be implemented with less complexity. However, outputs to devices are delayed for a specific period of time because they are buffered to keep the state of the VMs and the attached devices consistent.

We designed Kemari to take advantage of both approaches. It transfers the state of the primary VM to the secondary VM when an event occurs. This approach enables the primary VM and the secondary VM to be kept in sync with less complexity compared to lock-stepping. Furthermore, Kemari does not require external buffering mechanisms which impose output latencies, which is necessary in continuous checkpointing. Kemari synchronizes the VMs when the primary VM is about to send an event to devices such as storage and networks. Figure 1 shows the synchronization steps initiated by an event sent to storage.

When the primary VM tries to read or write to the storage, it outputs an event to the storage. Since this

event will change the state of the storage, Kemari needs to pause the primary VM, and update the secondary VM to the current state of the primary VM. After this synchronization, it resumes the primary VM and transmits the trapped event to the storage.

With this synchronization model, the secondary VM can continue transparently when the primary VM fails. If a hardware failure is detected, Kemari switches to the secondary VM, which will start running from the latest synchronized point, where the secondary VM and the virtual disk are consistent.

Events of networks are handled in a similar manner as those of storage. Applications using reliable protocols, such as TCP, will transparently continue after switching to the secondary VM. Applications using unreliable protocols, such as UDP, need to keep the network status consistent by themselves.

## 2.2 Implementation

We implemented Kemari based on Xen virtual machine monitor (VMM) [4]. A VM is called *domain* in Xen. There are two types of domains: the privileged domain and the unprivileged guest domains. The privileged domain, also called domain 0, can access the devices. Guest domains need to request I/O through virtualized devices such as virtual disks and virtual networks, which domain 0 provides. The guest domain’s side of the virtual device is called the *frontend*, and the domain 0 side is called the *backend*. To connect the frontend and the backend, Xen provides a notification mechanism called *event channel*.

Kemari is implemented across VMM and userland. The VMM part of the primary Kemari captures events sent through the event channel to kick start the synchronization process. When Kemari detects an event from the frontend, it *pauses* the guest domain, searches for dirty pages in the guest domain created since the previous synchronization, and notifies the userland part of Kemari to send the pages and the *vcpu* context to the secondary Kemari. After this synchronization, Kemari *unpauses* the guest domain and sends the captured event to the backend.

We used a guest domain whose pagetables are virtualized by *shadow pagetables* [3]. With this technique, we avoided depending on *suspend* used in the live migration [2], which might change the state of the guest. It also simplified the transfer mechanism by removing the canonicalization process of the pagetables.

## 3 Evaluation and future work

To evaluate Kemari, we ran I/O intensive benchmarks in our test environment, which consisted of two HP DL360G5 servers with two 3 GHz Xeon processors

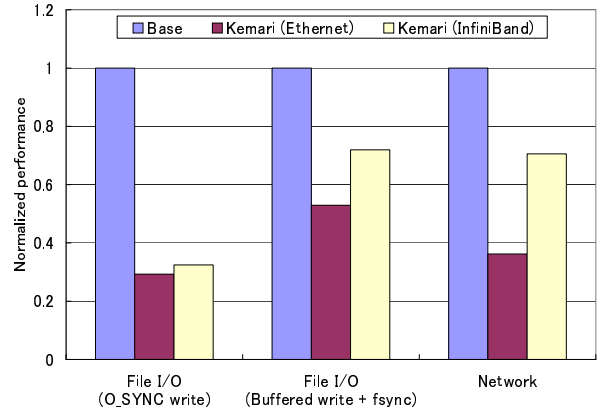


Figure 2: Performance of file I/O and network

in each, interconnected by Gigabit Ethernet and InfiniBand<sup>1</sup>. We used *iozone* to measure file I/O performance, and *netperf* for network performance. Figure 2 shows the results. The results demonstrated that Kemari connected by Ethernet achieves reasonable performance levels. InfiniBand boosted the performance of benchmarks, such as buffered write with fsync and network performance, both of which dirty many pages. On the other hand, it was not that effective for O\_SYNC write, which creates events more frequently compared to other benchmarks. We demonstrated that all benchmarks transparently continued when we shut downed the physical server running the primary Kemari from the HP ILO2 management console.

Our future work is to demonstrate the range of applications Kemari can manage to run transparently, to improve the performance of I/O intensive applications that send numbers of events, and to host fully virtualized domains.

## References

- [1] Brendan Cully et al. Remus: High availability via asynchronous virtual machine replication. In *NSDI'08*.
- [2] Christopher Clark et al. Live migration of virtual machines. In *NSDI '05*.
- [3] George Dunlap et al. Execution replay for multiprocessor virtual machines. In *VEE 2008*.
- [4] Paul Barham et al. Xen and the art of virtualization. In *SOSP '03*.
- [5] Thomas C. Bressoud et al. Hypervisor-based fault tolerance. *ACM Transactions on Computer Systems*, 4(1):80–107, February 1996.

<sup>1</sup>IP over IB is used for measurement.