Performance Analysis of Virtual Environments

Introduction

Virtual machines are becoming an integral part of both industrial data center facilities and large academic experimentation environments. It's important to measure the overheads of virtualization and identify performance bottlenecks in the virtual setups. This project aspires to provide tools to perform fine-grained analysis of resource utilization, virtualization overheads, and performance bottlenecks, in a virtualized software stack.

Our project consists of two steps. First, we instrument the virtual machine monitor to collect a variety of fine-grained performance events, for example, VM scheduling, transitions to and from the hypervisor, exception, page faults, interrupts, cross-domain communication channels and so on. The subset of the collected events is broad enough to answer various questions about performance of individual VMs and the system overall. To answer questions about performance of virtual device drivers (disk, network), we extend our tracing infrastructure to collect events from the device driver VMs, which provide these services.

Second, we implement an off-line trace analysis framework, which allows development of various performance analysis algorithms in a convenient and reusable way. Each analysis algorithm consists of a collection of probes, which observe specific performance events. Probes are clean, general and composable.

We implement our performance analysis framework for the Xen virtual machine monitor, which is a full feature, open source hypervisor, and a de-facto industry standard for industrial data-centers.

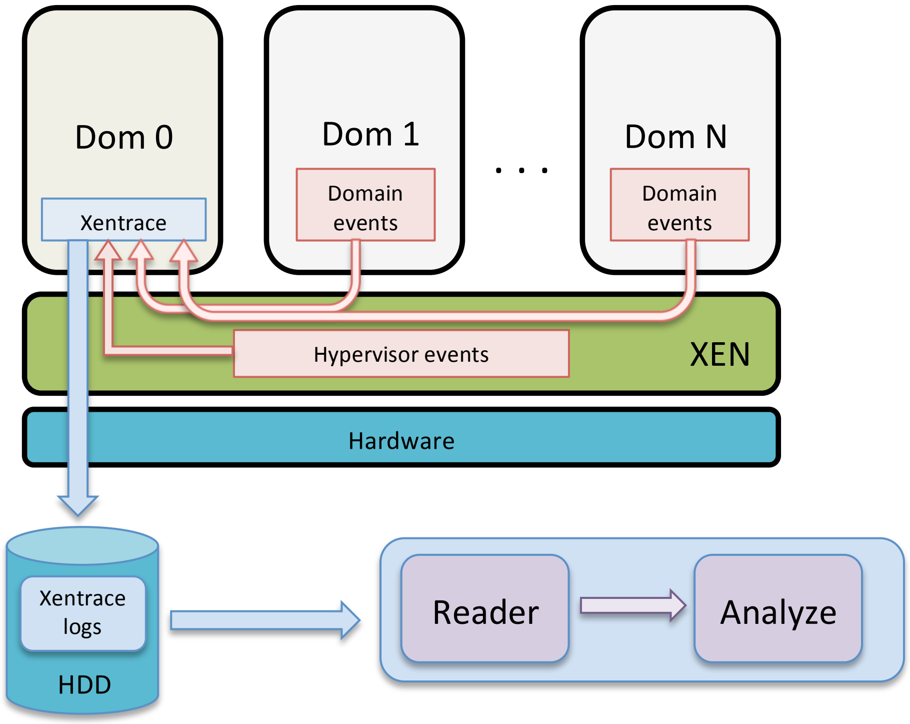

Fig 1: Xen and Xentrace Architecture

1. Xen Overview

The Xen hypervisor allows multiple guest operating systems to share hardware resources of a single physical host, as shown in Fig 1. These guest operating systems or virtual machines are interchangeably called as "*Domains*" in Xen. Dom 0 is the administrative domain with elevated privileges.

2. Data Collection

Xentrace is a lightweight tracing utility in Xen that collects hypervisor level events. Xentrace consists of 3 parts: tracing macros, fast communication channel and user level daemon application. Xentrace is typically run from Dom 0(control VM) and a daemon periodically dumps them to the disk (Fig 1). The amount of data collected by Xentrace can be enormous. Although this raw data can be quite useful during debugging, it does a poor job of providing a high level view of performance problems. For example, data collected during a disk intensive operation for 1 minute easily exceeds 700 MB or about 20 million lines in the log. To extract meaningful information from such a log can be time consuming and prone to errors.

Due to enormous information collected by Xentrace, these trace logs are chosen to be the data source. To collect additional performance events not provided by Xentrace, one could insert appropriate trace macros inside the Xen or domain (Linux) source code. Trace macros are trivially easy to write.

3. Analysis Framework

The goal of this project is to not just build a specific set of performance analysis tools, but to also give the developer a generic lightweight framework to write any type of performance analyses he or she desires.

The implementation is divided into 2 main components: *Reader* and *Analyses* (Fig 1). The Reader component parses the data inside the logs and neatly constructs C style data structures that are eventually passed to the Analyses component. The Analyses consists of a group of handlers for different types of events. These handlers are the only things that need to be written by the developer to build custom analysis. Some of the analyses implemented for this project are discussed in section 4.

4. Analysis Algorithms

Some of the analysis tools being implemented for this project are listed below.

- *CPU Utilization*: Apart from the Physical CPU core usage, this analysis also shows CPU usage by each VM and their respective Virtual CPU (VCPU) utilization. Some scenarios where such granularity is necessary are:
  - Check if hypervisor configuration adheres to Service Level Agreements (SLA) between hosting providers and clients.
  - Detecting unbalanced mapping between physical CPUs and VMs.
- *Time in Hypervisor*: During an I/O intensive operation, a VM may spend much of its time being idle while the hypervisor does the work on its behalf. Hence this may give a wrong picture to someone who just relies on CPU usage data, as it doesn't accurately represent the time spent in hypervisor. It can also help identify bottlenecks in the execution. (Ex: high latency I/O)
- *CPU Scheduling Latency*: Due to overscheduling of VCPUs or physical CPU being busy, VM context switches can get delayed. This analysis measures how much time a VM had to wait to get scheduled since the request to context switch was sent out.
- *Disk I/O*: A block request from a VM navigates through many different queues like the VM elevator queue, Xen Shared ring buffers, queues inside Dom 0 etc., before the request actually reaches the HDD. The response also traverses the same path. Hence monitoring the state of these queues can help identify overheads of virtualization and identify bottlenecks in the disk I/O pipeline.