High-Fidelity Monitoring in Virtual Computing Environments

W. Feng* V. Vishwanath† J. Leigh† M. Gardner*

*Department of Computer Science
Virginia Tech
Blacksburg, VA 24060
feng@cs.vt.edu, mkg@vt.edu

† Department of Computer Science
Univ. of Illinois at Chicago
Chicago, IL 60607
venkat@evl.uic.edu, spiff@uic.edu

ABSTRACT

We present a high-fidelity monitoring infrastructure that enables real-time analysis and self-adaptation at both the systems and applications level in virtual computing environments. We believe that such an infrastructure is needed as each paradigm shift (in this case to virtual computing environments) brings new challenges along with new capabilities.

Monitoring the performance and health of large-scale computing environments can be a daunting challenge. Without appropriate tools, tracking down problems, whether caused by hardware or software, is akin to finding a "needle in a haystack."

In a clustered computing environment, monitoring tools such as Ganglia and Supermon have emerged and rapidly matured to address this problem in a scalable manner. However, these tools sacrifice information to achieve scalability, e.g., they use the /proc filesystem as their data source. Consequently, such tools are sample-based and can only provide measurements at the granularity of an OS time slice. Furthermore, these tools are designed to operate in the context of a physical environment rather than a virtual one. Therefore, as an alternative, we present a dynamic, high-fidelity, event-based infrastructure for both physical and virtual computing environments called MAGNET.

Keywords

Virtual computing, monitoring, probes, filters, event-based, framework

1. INTRODUCTION

With the current trend in virtualization of resources, computing is returning to the seeds initially sown by the IBM System/370 architecture and VM/370 operating system (OS) in the 1970s. Virtual computing offers a unique opportunity to dramatically improve resource utilization, to consolidate resources in order to simplify system administration or reduce cost, and to dynamically re-purpose physical resources for the task at hand. However, *efficiently* accomplishing the above requires a run-time understanding of the dynamic behavior of the environment within which the virtual environments execute. Fine-grained monitoring of the physical system facilitates the identification and elimination of scalability roadblocks for virtual computing environments. In addition, the monitoring information enables the virtual environment to automatically adapt its behavior to make efficient use of the physical resources allocated to it. Thus, we propose a monitoring framework called MAGNET that is

- dynamic (i.e., monitoring points can be inserted and deleted with very low overhead),
- high fidelity (i.e., events can be logged at CPU cycle-counter granularity, typically nanoseconds),
- event-based (i.e., versus sample-based or count-based),[1]
- transformable (e.g., "tcpdump on steroids"), and
- controllable (i.e., event stream can be filtered and sampled).

The principles that have guided the design of our MAGNET prototypes (both probe-based and SystemTap-based) will ultimately be incorporated as part of our larger *Scalable, Extensible, and Reliable Virtual Computing Environment* (SERViCE). Of particular interest is the opportunity to investigate scalable and high-fidelity monitoring in virtual environments to better support automated hybrid physical/virtual resource management, power awareness, and virtual application development and deployment in a scalable manner.

Ensuring scalability in SERViCE requires a global understanding of the dynamic behavior of the environment within which the virtual services execute. This implies that MAGNET must have efficient introspective capabilities for both the physical and virtual worlds. Though there exists a substantial body of work that addresses the monitoring of software directly running on a physical system, it remains to be

[1]Event-based monitoring subsumes sample- or count-based approaches albeit care must be taken to limit overhead.

International Conference on the Virtual Computing Initiative, May 7-8, 2007, Research Triangle Park, North Carolina, USA.

seen how much of the work translates directly to monitoring resources supporting a virtual environment.

It is the dual nature of the physical/virtual computing environment that provides additional challenges, such as how to map the physical results from monitoring to the virtual environments such that applications can reason about their behavior without needing to know about applications running in other unrelated environments. We envision that the physical run-time system and the applications in the virtual environments may wish to adapt their behavior simultaneously. Clearly, the behaviors of the two are coupled, and hence, a theory that allows the two adaptations to proceed stably and predictably is needed.

Ubiquitous, low-overhead monitoring also provides the information needed to make better resource management decisions in order to reduce power consumption. For example, reading/writing a block of data from/to storage takes a significant amount of time; many CPU cycles can be wasted waiting for the I/O to complete. Because operating system power-management daemons operate in user space, they cannot predict when a block will be read or written, and hence, must use heuristics to decide when the power (or energy) consumption of the processor or other resources can be reduced. Clearly there is a potential for increased power efficiency if more precise information can be obtained. With fine-grained monitoring, the power management system can be directly informed when such events occur leading to better optimization of the system behavior.

Virtual monitoring tools for SERViCE will also benefit the development and deployment of applications in virtual environments. Traditional tools developed for the physical world and used in a virtual environment may provide misleading results. For example, CPU performance counters or cycle registers are often used to diagnose performance problems in the physical world. Using those counters in the virtual world may mislead developers who think they are measuring their application's performance but instead see the interposed effects of unrelated computations running in other virtual environments. These and other issues can be addressed by tools that provide an appropriate virtualization of the performance of the physical machine obtained by fine-grained monitoring. The key will be a flexible system for developing monitoring tools.

The rest of this paper is organized as follows: Section 2 provides background information on the probe mechanisms that serve as the basis for MAGNET's functionality. Section 3 describes our design and implementation of the MAGNET framework. Section 4 quantifies the effect that MAGNET has on the system being monitored. Section 5 presents related work, followed by the conclusion in Section 6.

2. BACKGROUND

MAGNET relies on kernel probes, a dynamic instrumentation mechanism that is available in the Linux 2.6 kernel (see Figure 1) . They are based on the work done on dprobes by IBM. Kernel probes can be dynamically inserted and deleted with minimal overhead, thus eliminating the need to recompile the kernel and reboot the system each time instrumentation is added or removed, a characteristic of the earlier statically configured versions of MAGNET [3, 4].

Kernel probes provide a lightweight mechanism to dynamically "break into" any kernel routine and collect debugging or performance information. More specifically, a dynamic

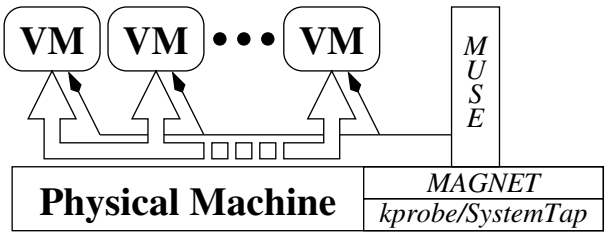

Figure 1: Architecture of MAGNET in a Virtual Environment

probe is an automated breakpoint that is dynamically inserted into executing kernel-space code without the need to modify its underlying source. Associated with each probe is a corresponding event-handler that runs as an extension to the system breakpoint's interrupt handler with little to no dependence on system facilities. Consequently, probes may be seamlessly implanted into many "hostile" environments including the context-switch path, the interrupt-time path, and SMP-enabled code paths.

There are three types of probes available in the Linux 2.6 kernel: kprobes, jprobes, and kretprobes. An additional, experimental probe type called djprobes is also being considered for inclusion in the kernel.

2.1 kprobe

A kprobe can be inserted on vitually any instruction in the kernel and provides access to the register arguments. When a kprobe is registered, it makes a copy of the probed instruction and replaces the first byte(s) of the probed instruction with a breakpoint instruction (e.g., 'int3' on i386 and x86_64). When the a processor encounters the breakpoint instruction, a trap occurs, the processor's registers are saved, and control passes to the kprobe code. The kprobe executes the associated pre-handler and single steps through its copy of the probed instruction. Upon completion, the kprobe executes any post-handler that is associated with the kprobe.

2.2 jprobe

A jprobe provides convenient access to the arguments of a kernel function. It is implemented by placing a kprobe at a function's entry point and uses a mirroring technique to allow seamless access to the function's arguments. A jprobe can only be inserted at the function's entry.

2.3 kretprobe

A kretprobe, also known as a return probe, is inserted at the return address of a kernel function, thus providing access to the return value of a function call. It is implemented via a trampoline instruction.

2.4 djprobe

In contrast to the three aforementioned probes, the djprobe — also known as the *direct jump probe* — uses a jump instruction instead of a breakpoint to hook into the instruction stream. This approach reduces the overhead associated with probe invocation. On the x86 platform, the djprobe is implemented using the relative 'jmp' opcode instead of 'int3' break opcode that is used by the other probes, thus providing the ability to dynamically hook into any kernel function entry point with dramatically lower overhead.

3. THE MAGNET FRAMEWORK

The MAGNET framework consists of a global control subsystem that interacts with and manages four other subsystems: monitoring, event filtering and synthesis, performance logging, and event-streaming. It is also responsible for the dynamic run-time configuration and management of the four subsystems within the MAGNET framework. The monitoring subsystem produces an event stream that is passed through an event filter and synthesis engine before being logged to *magnetfs*, a high-performance logging subsystem exporting data as regular files. This allows user-space event-stream clients to read the data using regular file operations.

3.1 Control Subsystem

Th control subsystem provides dynamic run-time configuration and management of the subystems within the MAGNET framework. It consists of *magnetk* — a kernel thread that is responsible for communication with user-space applications and other kernel modules. In addition, the control subsystem contains an instrumentation database which contains a list of probes registered with the control subsystem, their associated filters, and the files in use by the logging subsystem.

MAGNET uses netlink sockets [6] for communication between *magnetk* and user-space processes or kernel modules. Typical *magnetk* messages include (1) enabling, disabling, adding, or removing instrumentation points on-the-fly, (2) creating or deleting log files in *magnetfs*, and (3) adding or deleting event filters and synthesizers to winnow and format the data.

The types of events being monitored can be extended on the fly by inserting user-defined modules into the running kernel. The instrumentation points dynamically join the monitoring framework by registering their probes with the control subsystem. Upon a registration request, *magnetk* returns a unique event identifier to the kernel module. This identifier is used to annotate related event records before logging them to *magnetfs*. Once registered, instrumentation points begin communicating with *magnetk* over netlink sockets.

3.2 Monitoring Subsystem

As mentioned in Section 2, the MAGNET monitoring subsystem uses kprobes to instrument function in the kernel. In one MAGNET prototype, we use the kprobes directly; in our other prototype, we utilize them indirectly through SystemTap. For the purposes of this paper, we focus on the former.

A wide variety of probes can easily be made available by compiling and loading additional kernel modules. We have predefined probes, filters, and synthesizers to instrument the network stack, the process scheduler, and the memory management subsystem. New instrumentation points are registered with the MAGNET control subsystem when the module is loaded. When the instrumentation are enabled, event records are logged to files in the *magnetfs* system, as described in Section 3.4 below.

Figure 2 shows a typical MAGNET event record. The actual record format is configurable during the loading of the MAGNET kernel module. The record format is flexible enough to support SMP configurations, 32- and 64-bit architectures, and variable-sized records. Variable-sized records were added to the latest version of MAGNET to enable more efficient use of the circular buffers in kernel space and to further reduce the amount of data transferred to user space.

```
struct magnetEvent {

  unsigned int _header;
  // _header[5:0]: CPU/Core ID
  // _header[15:6]: Event ID
  // _header[31:16]: Event size in bytes
  // Configurable at module loading

  PTR_TYPE _abstractID;
  // PTR_TYPE is 4 or 8 bytes, architecture-dependent.
  // The value depends on the event type.

  unsigned long long _timer;
  // high precision timing based on the timestamp counter

  unsigned int _pid;
  // usually the tgid of the process

  unsigned char* _eventRecord;
};
```

Figure 2: MAGNET Instrumentation Record

3.3 Event-Filtering and Synthesis Subsystem

Event filtering and synthesis is a critical service that classifies, analyzes, and efficiently disseminates events. Filtering has the benefit of dramatically reducing the "drinking from the firehose" effect. It is particularly important to do *kernel-space* event filtering as it reduces the amount of data that is transferred between the kernel and user space, thereby reducing overhead. Synthesis also reduces the amount of data being transferred while increasing the utility of the information. (An example of synthesis is converting an event stream into periodic averages or descriptive statistics. In this way, sample- and statistics-based monitoring can also be inexpensively provided.)

Filtering and synthesis functionality is extremely important in virtual computing environments because it allows the event stream to be separated into views tailored for the virtual machines receiving the data. It is through this mechanism that isolation is achieved in spite of the fact that virtual machines share the physical resources of the host. Individualized data streams allow virtual environments to adapt without being distracted by the behavior of the other environments.

Currently, kernel-space event filtering is implemented with the help of a hash table. Events may be filtered based on the processor ID, the event type, or the abstract ID returned from the control subsystem at registration. Three mechanisms of event filtering in kernel space are currently supported:

1. Via a probe handler associated with the instrumentation point.
2. Via an explicit call to the `magnet_add()` function.
3. Via a kernel thread that traverses the event stream selecting records to be logged.

Probe handlers can decide at the time an event occurs whether or not to record the event. Alternatively, `magnet_add()` can be called explicitly to record an event. Finally, kernel threads traverse the event stream to select

records that satisfy the specified predicates, and optionally, to synthesize higher-level information from low-level events.

Typically, one of the first two mechanisms is used to process streams containing high-frequency events in order to reduce the volume of data that needs to be transferred from kernel to user space. The probe handlers generally must be small, simple, and consume as few CPU cycles as possible to reduce measurement perturbation. The third mechanism, filtering using a kernel thread (*kthread*), is better suited for low-frequency events. Filtering with the aid of a *kthread* facilitates the composition of complex filters that can correlate between different events occurring within a given time window. Additional filtering can also be carried out in user space with the support of user-space analysis and synthesis tools such as Autopilot [11], MUSE [4], and Prophesy [13].

3.4 High-Performance Logging System

The MAGNET filesystem, *magnetfs*, efficiently logs event streams and transfers data at high rates from kernel to user space. *Magnetfs* is built upon *debugfs*, an in-kernel filesystem that was added to Linux 2.6 and enabled by default in most Linux distributions. *Debugfs* was designed to transfer debugging messages from kernel modules to user space in order to facilitate driver debugging. It provides an interface for export the values of variables and counters to user-space applications and was designed to replace the ubiquitous printk often used for debugging.

However, *debugfs* has an extensible design allowing the default filesystem operations to be overloaded on a per file basis. We refer to the debug filesystem overloaded with filesystem operations to support high-performance transfers of MAGNET event streams between kernel and user space as *magnetfs*. *Magnetfs* supports a rich set of high-performance interfaces that include blocking I/O, polling-based I/O, asynchronous I/O, and signal-driven I/O. This mechanism enables the user-space data-collection application to efficiently process data. Experience with earlier versions of MAGNET that only supported polling I/O showed that the other modes are important for reducing the CPU usage of user-space applications that consume the event stream.

In *magnetfs*, circular buffers are created in the kernel for logging data. Probe handlers record the event records into one or more of the circular buffers. Double buffering allows events to be transferred to user-space without hindering data collection. Multiple buffers are also useful to reduce mutual exclusion effects of SMP or multicore systems. Multiple buffers can also be used during filtering to separate the event streams in preparation for delivery to specific virtual environments. Alternatively, a single buffer for all events can be used to maintain the global ordering of events without the need for an on-the-fly merge. Buffers are exported to user space as files under *debugfs* mount points. The files can be read by user-space applications via regular file operations. Circular buffers can also be read directly by kernel-space clients, such as *magnetk* threads.

Buffering delivers high-bandwidth for bulk delivery of data to user space. However, it also increases the average delay from when an event occurs until it is available in user space. Therefore, *magnetfs* can also be configured to be event-triggered. User-space applications use an `ioctl` to set a buffer's read-size watermark (the minimum amount of data that must be available before a read operation can succeed), the read timeout (the maximum delay before returning data), and the minimum read count (the minimum number of events before a read can succeed). Using these parameters, the performance of individual *magnetfs* files can be tuned as appropriate.

4. PERFORMANCE EVALUATION

We evaluate the performance of the MAGNET implementation by measuring the impact of monitoring with MAGNET on user-space performance. We then analyze the performance benefits of MAGNET's kernel-event filtering and sampling capabilities. We use the above to demonstrate the scalability of MAGNET to monitor at extremely high event rates and log data at extremely high rates with negligible event-loss rates and low CPU utilization as a substrate upon which to build a monitoring, management, and adaptation infrastructure for virtual environments.

4.1 Monitoring the Network Subsystem

The network performance of virtual environments is of great concern for several up-and-coming application areas, e.g., high-performance or grid computing, where virtualization is starting to be used as a packaging and delivery mechanism for scientific computations [5], or application service providers that are using virtualization for server consolidation to reduce cost. In this section, we evaluate the impact of MAGNET on the performance of 10-Gigabit Ethernet as it monitors the journey of packets through the network stack.

4.1.1 Testbed

The testbed consists of two identical dual 2.4-GHz Opteron single-core systems with 1 MB of L2-cache and 4 GB of 200-MHz DDR SDRAM running a Linux 2.6.12.6-SMP patched with the Chelsio TCP offload engine (TOE) support (driver version 2.1.4). The Tyan K8W Thunder motherboards feature an AMD-8131 chipset. Each node has a Chelsio T210 TOE-based 10-Gigabit Ethernet (10GigE) network adapter plugged into a 133 MHz/64-bit PCI-X slot. The systems are connected back-to-back.

The offload capabilities of the network adapters are deliberately disabled at boot time to ensure that all the protocol processing is done on the end hosts and not offloaded to the network adapters. The Chelsio network adapters are also configured with a MTU of 9000 bytes, as is common in data centers. The two nodes also have an Intel E1000 LX 1-Gigabit Ethernet network adapter, each on a 100-MHz / 64-bit PCI-X slot, also connected back-to-back, with a MTU of 1500 bytes.

4.1.2 Methodology

The impact of the monitoring mechanism is estimated by measuring the performance degradation caused by running MAGNET while maximizing the transfer rate between the two test hosts and compare the results to the same test without MAGNET enabled. As a further point of comparison, the impact of running a kernel-based *tcpdump*,[2] the traditional tool for capturing network packet traces, is also measured.

[2]Tcpdump was configured with PCAP_FRAMES=max and PCAP_TO_MS=0. The kernel was compiled with the PF_PACKET and PACKET_MMAP options to improve the packet capture performance.

The source of network traffic is IPerf 2.0.2, an application-level, end-to-end bandwidth measurement tool that both sends/receives packets and computes the transfer rate. Any overhead caused by monitoring should show up as a reduction in the observed "goodput"[3] and as an increase in CPU utilization over the baseline in which no monitoring is performed. To minimize interference, we restrict the processes running on the end hosts to IPerf, MAGNET (or tcpdump), and a few essential services.

MAGNET's dynamic instrumentation points are used to monitor the journey of UDP packets through four different layers of the networking stack on both sender and receiver.

- Socket layer: `inet_sendmsg` and `inet_recvmsg` kernel functions,
- Transport layer: `udp_sendmsg` and `udp_recvmsg` kernel functions,
- Internetwork layer: `ip_push_pending_frames` and `ip_rcv_finish` kernel functions,
- Access layer: `dev_queue_xmit` and `net_rx_action` kernel functions.

For each event, MAGNET logs the packet header, the time the event occurs using the CPU timestamp counter register (TSC), and the IDs of the CPU and thread that did the processing. Events are logged to an 8-MB *magnetfs* buffer. We point out that tcpdump only generates a single record for each packet while MAGNET was configured to record four events per packet. In addition, the MAGNET event records also contain CPU and thread IDs while tcpdump records do not.

4.1.3 Results

As seen in Figure 3, the goodput of the IPerf-generated UDP stream running on an unmonitored system is 7.22 Gbps between the two test nodes over the 10GigE Chelsio network adapters. There were no packets dropped. With the MAGNET kernel module loaded and instrumentation probes disabled, there is still no impact on the data rate of the IPerf UDP stream, as is expected since no instrumentation points have been inserted.

The decrease in goodput at the socket layer (one event per packet) when MAGNET is enabled is 0.1%, while the decrease is 0.4% for both the socket and UDP layers (two events per packet). As the number of events per packet increases, the goodput decreases. This is due the execution of a kernel probe interrupt (x86 int3 instruction) resulting in a context switch for each instrumentation point, along with the overhead of saving the event record and packet headers.[4] In contrast, monitoring a single event per packet with tcpdump results in a goodput decrease of nearly 40%. In addition, there are no events lost by MAGNET while 0.01% events are lost by tcpdump. At 10 Gbps speeds, tcpdump struggles to keep up while MAGNET can monitor multiple layers in the network stack with far lower impact.

[3]Goodput is the effective rate at which valid data is sent. It is typically lower than the throughput since the latter also includes any retransmitted packets.

[4]If djprobes are accepted into the Linux kernel, the overhead of the probe mechanism is projected to drop by a factor of 5–6 as djprobe uses a "jmp" mechanism instead of an int3 software interrupt. This would definitely make the overhead of multiple probes in MAGNET negligible.

Also shown in Figure 3, the CPU utilization for MAGNET is between 0.4% to 1.3% while the utilization of tcpdump is 32.1%. Thus, the CPU overhead of MAGNET is approximately 25 times lower than tcpdump. The low CPU utilization achieved by MAGNET is extremely important as it has minimal impact on the application running on the end host

4.1.4 Scalability Analysis

As applications and environments become more distributed and concurrent, particularly with the rapid adoption of multi-core processors for commodity computing, understanding performance issues requires understanding the interaction of the application with the underlying kernel hardware and software to identify performance bottlenecks. The challenge is exacerbated by the push for server consolidation fueled by rapid adoption of virtualization technologies. Applications that were once hosted separately must now share hardware causing subtle performance anomalies. Understanding the issues is often an iterative process and necessitates the logging of a large amount of data for both online and offline analysis. This requires monitoring and logging mechanisms capable of handling large volumes of data Additionally, the monitoring and logging mechanism must have low impact. We have shown that the overhead of MAGNET is very low. In this section, we investigate the ability of MAGNET to handle large amounts of data.

We start by evaluating the ability of *magnetfs* to transfer data efficiently from kernel space to user space using asynchronous I/O for reading the logged data. As before, MAGNET is configured to monitor an IPerf UDP stream between the two test nodes. The *magnetfs* buffer is configured for bulk data delivery with a maximum capacity of 8 MB and with a read watermark of 256 KB.

The total amount of data logged per packet to the tmpfs file system is given by the X-axis in Figure 4 while the transfer rate from kernel to user space is shown on the Y-axis. The data transfer rate is up to 306.11 MBps (2.448 Gbps) with no lost events. Also shown in Figure 4 is the CPU utilization of the user-space application that reads the data. As the amount of data to be transferred increases, the CPU utilization of the transfer application increases as is expected.

4.2 Performance of Statistical Sampling

Statistical sampling is beneficial for applications that need to adapt based on a statistical average over a period of time rather than due to a specific event. This is particularly useful for self-adaptive, reflective applications, e.g., application-level, rate-based network transport protocols such as RAPID [1]. Statistical sampling in support of virtual environments can provide needed information to make better resource allocation decisions for the various virtual machines running on a host. For example, the prior execution behavior of a payroll application obtained through statistical sampling is likely to indicate that resources demands increase dramatically two business days before payday, but are low the remainder of the week. Armed with this information, fewer virtual computations should be scheduled on the physical server during that time. While it may be feasible to manually accommodate such behavior when then degree of virtual machine co-scheduling is low, it quickly becomes intractable as the number of cores increase, as is envisioned now that the exponential increase in transistors

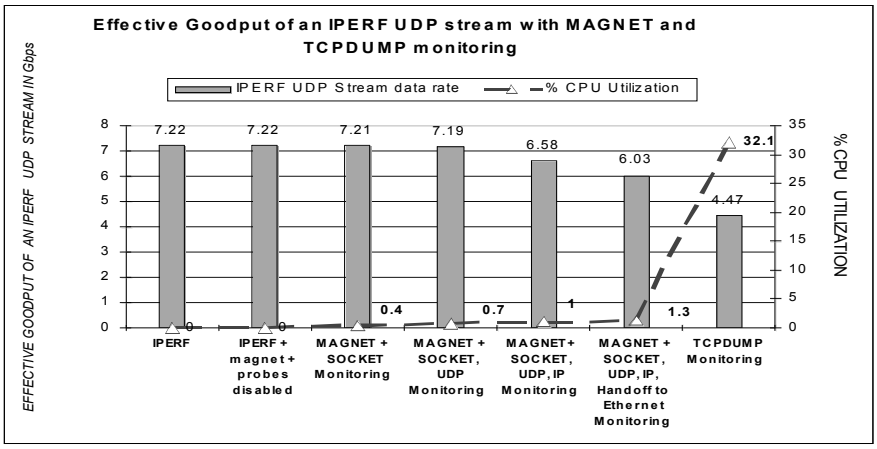

Figure 3: Impact of MAGNET and tcpdump on the Goodput and CPU Utilization

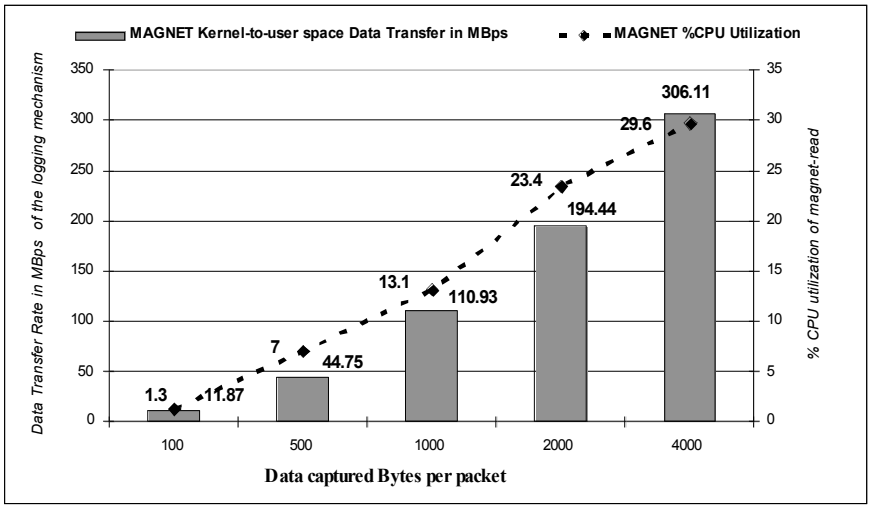

Figure 4: Performance of the Logging Mechanism in MAGNET

afforded by Moore's Law has turned to replication instead of higher clock speeds.

MAGNET supports event sampling wherein sampling is triggered by a timer or is based on the number of events logged. Figure 5 shows the effect of statistical sampling of the event stream on the goodput of the IPerf UDP Stream. In this experiment, we capture the arrival of a packet at the socket layer, UDP layer, IP layer, and the Ethernet layer for each packet (i.e., four events per packet) and record 4,000 bytes per event, i.e., 16,000 bytes per packet. The filters for each event are configured to sample based on the occurrence of the events.[5] We observe that as the sampling frequency decreases, the goodput achieved by the IPerf UDP streams increases and results in a reduction in the CPU utilization. A relatively modest one-in-40 sampling ratio allows nearly 94% of the peak bandwidth to be delivered while using less than 5% of the CPU in spite of the high event rate (211,250 events per second) being handled internally by the MAGNET implementation.

Even though the probe notes each packet, the probe handler first checks if the event should be filtered before logging the event. Thus, the average overhead of computation in the probe handler is lower as the sampling frequency decreases. The CPU utilization of user space applications also decreases as the sampling frequency decreases. This is primarily due to the lower amount of data logged. There exists a tradeoff between capturing every event and statistically sampling the events. Capturing every event results in complete information, though at the expense of increased CPU utilization and a greater impact on the system. Because it lowers overhead further, statistical sampling is highly useful in productions systems and for self-reflective applications.

4.2.1 Filtering in Kernel Space

We compare the performance of filtering in kernel space using MAGNET with filtering in kernel space using tcpdump. In this experiment, we first run an IPEF UDP stream between the two machines' Gigabit Ethernet (GigE) network adapters. The goodput achieved by this IPerf UDP stream

[5]It is reasonable to ask why so many bytes are logged when a statistical sample is required. The size of the record must be large enough to satisfy the most demanding of the simultaneous clients. Although one client may desire a statistical sample, another may require substantial data.

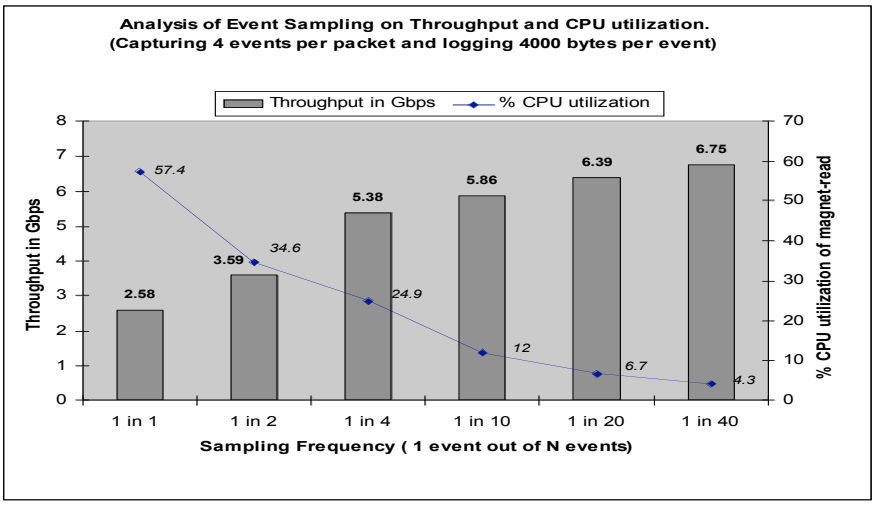

Figure 5: Impact of Statistical Sampling on the Goodput and CPU utilization

was 956 Mbps with 0% packet loss. A second IPerf application stream was started between the 10GigE network adapters of the machines. The goodput achieved by the second IPerf UDP stream is 5.97 Gbps. The aim of the experiment was to capture the event streams of the second IPerf UDP stream, i.e., the 10GigE streams. MAGNET was configured to instrument the socket layer. An event filter with the related abstract ID was added to the three instrumentation points to MAGNET via the control channel. tcpdump was run with the IP address and port of the second UDP stream specified as a filter. The goodput of the first IPerf stream did not drop while monitoring the second stream with either tcpdump or MAGNET.

From Figure 6, it can be seen that the percentage reduction in the throughput of the second UDP stream with tcpdump filtering is at least 150% to 225% more than the percentage drop with MAGNET event filtering. The CPU overhead of the application reading from MAGNET files was far lower than the CPU utilization of tcpdump. tcpdump was also unable to capture 1% of the packets in the "96 bytes of data" case and around 24% of the "500 bytes of data" case. MAGNET captured and filtered the event streams without any event loss. This is attributed mainly to the high-performance interfaces available in MAGNET.

5. RELATED WORK

Substantial related work exists for monitoring computing systems. Tools such as the Linux Trace Toolkit (LTT) [14] use static kernel instrumentation by way of patching and re-compiling the kernel. Clearly, this requirement makes it difficult to use in production systems. However, having to patch and recompile has also proved to be a significant barrier to adoption even in experimental systems, as we have found with previous versions of MAGNET.

A more flexible approach, taken by such tools as Dtrace [2], KernInst [12], and SystemTap [10] uses dynamic kernel instrumentation. Dtrace and SystemTap provide a powerful scripting language to probe and monitor the kernel and covers a large set of probe points. Dtrace predominantly runs on Solaris, but it was recently ported to the Mac OS X operating system. SystemTap leverages the kprobes mechanism, which has been ported to multiple architectures. In contrast, KernInst implements its own probing mechanism and supports only a few limited architectures; it also supports limited sampling and event-filtering capabilities. By building upon kprobes or SystemTap, MAGNET supports customize probe handlers and filters at run time via kernel modules, high-performance kernel-to-user space transfer of data, and kernel-space event filtering and sampling which are useful for production system monitoring and adaptation.

Statistical sampling-based tools such as OPROFILE [8], PAPI [7], and PERFMON [9] take advantage of the hardware performance counters of contemporary processors to sample certain events at periodic intervals. Hardware-related events, such as CPU cache misses that software-based tools such as MAGNET cannot observe, are easily captured. The opposite is also true. Software-based tools, such as MAGNET, are able to return information even when hardware performance counters are not available. As the tools compliment each other, both may be utilized to gain a complete perspective.

Other work related to MAGNET includes the ability to move large volumes of data between kernel space and user space. Linux Trace Toolkit – The Next Generation (LTTng) and SystemTap utilize the Linux kernel's relayfs [15] to transfer data to user space. *Magnetfs* provides features similar to relayfs such as multiple buffers for logging events and bulk delivery of data. In addition, however, *magnetfs* provides a richer set of interfaces such as asynchronous I/O and signal-based I/O for high performance transfer and low CPU utilization.

In summary, the MAGNET framework leverages work done in kernel instrumentation, event-based statistical sampling, event-based filters, and efficient kernel-to-user-space transfer of data to provide a high-performance, scalable, event-monitoring sensor mechanism.

6. CONCLUSION

In this paper, we presented MAGNET, a high-fidelity monitoring infrastructure that enables real-time analysis and self-adaptation at both the systems level and applications level for virtual computing environments. Preliminary

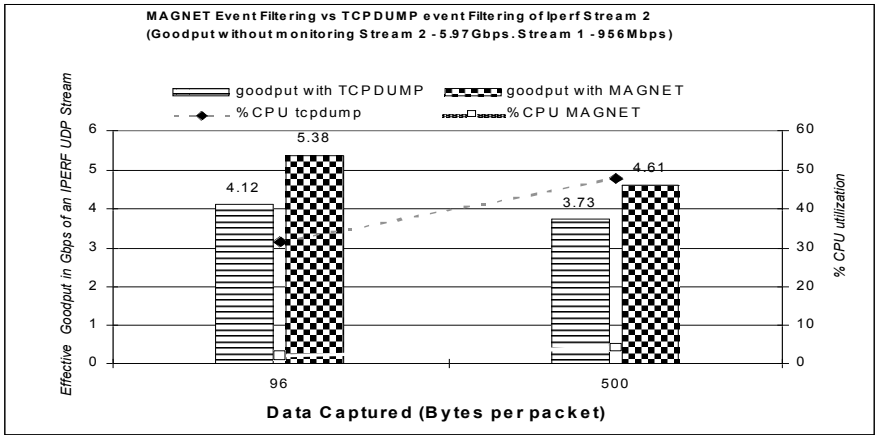

Figure 6: Impact of MAGNET and tcpdump Event Filtering on the Goodput and CPU Utilization

results have been favorable and point to further work in the area. We ultimately expect MAGNET to be incorporated as part of our larger *Scalable, Extensible, and Reliable Virtual Computing Environment* (SERViCE). Of particular interest is the opportunity to investigate scalable and high-fidelity monitoring in such virtual environments to better support automated hybrid physical/virtual resource management, power awareness, and virtual application development and deployment in a scalable manner.

Acknowledgment

This research is supported by IBM through an IBM Faculty Award VTF-873901, a NSF grant IIP-0804155, and the Department of Computer Science at Virginia Tech.

7. REFERENCES

[1] A. Banerjee, W. Feng, B. Mukherjee, and D. Ghosal. RAPID: An End-System Aware Protocol for Intelligent Data Transfer over Lambda Grids. *Proc. of the 20th International Parallel and distributed Processing Symposium (IPDPS 2006)*, Rhodes Island, Greece, April 2006.

[2] B. Cantrill, M. Shapiro, and A. Leventhal. Dynamic Instrumentation of Production Systems. *Proc. of the 2004 USENIX Annual Technical Conference*, Boston, Massachusetts, June-July 2004.

[3] M. Gardner, W. Feng, M. Broxton, A. Engelhart, and J. Hurwitz. MAGNET: A Tool for Debugging, Analysis and Adaptation in Computing Systems. *Proc. of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'2003)*, Tokyo, Japan, May 2003.

[4] M. Gardner, M. Broxton, A. Engelhart, and W. Feng. MUSE: A Software Oscilloscope for Clusters and Grids. *Proc. of the 17th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2003)*, Nice, France, April 2003.

[5] W. Huang, J. Liu, B. Abali, and D. Panda. A Case for High Performance Computing with Virtual Machines. *Proc.*
of Supercomputing 2007, Tampa, Florida, November 2007.

[6] IETF RFC 3549. http://rfc.net/rfc3549.html.

[7] P. Mucci and F. Wolf. An Introduction to PAPI and PAPI-based Tools. *Proc. of ACM/IEEE SC*, Phoenix, Arizona, November 2003.

[8] The OProfile Home Page. http://oprofile.sourceforge.net.

[9] The Perfmon2 Home Page. http://perfmon2.sourceforge.net/

[10] V. Prasad, W. Cohen, F. Eigler, M. Hunt, J. Keniston, and B. Chen. Locating System Problems Using Dynamic Instrumentation. *Proc. of the Ottawa Linux Symposium*, Ottawa, Canada, July 2005.

[11] R. Ribler, J. Vetter, H. Simitci, and D. Reed. Autopilot: Adaptive Control of Distributed Applications. *Proc. of the IEEE International Symposium on High-Performance Distributed Computing*, Chicago, Illinois, July 1998.

[12] A. Tamches and B. Miller. Fine-Grained Dynamic Instrumentation of Commodity Operating System Kernels. *Proc. of the USENIX Symposium on Operating Systems Design and Implementation*, New Orleans, Louisiana, February 1999.

[13] V. Taylor, X. Wu, J. Geisler, X. Li et al. Prophesy: An Infrastructure for Analyzing and Modeling the Performance of Parallel and Distributed Applications, *Proc. of the IEEE International Symposium on High-Performance Distributed Computing*, Pittsburgh, Pennsylavnia, August 2000.

[14] K. Yaghmour and M. Daenais. Measuring and Characterizing System Behavior Using Kernel-Level Event Logging. *Proc. of the 2000 USENIX Annual Technical Conference*, San Diego, Califronia, June 2000.

[15] T. Anussi, K. Yaghmour, R. Wisniewski, T. Moore, and M. Dagenais. Relayfs: An Efficient Unified Approach for Trasnmitting Data from Kernel to User Space, *Proc. of the Linux Symposium*, July 2003.