A High-Performance Sensor for Cluster Monitoring and Adaptation

V. Vishwanath[*], W. Feng[†], M. Gardner[§], and J. Leigh[*]

[*]University of Illinois at Chicago, [†]Virginia Tech, [§]Los Alamos National Laboratory

Abstract

As Beowulf clusters have grown in size and complexity, the task of monitoring the performance, status, and health of such clusters has become increasingly more difficult but also more important. Consequently, tools such as Ganglia and Supermon have emerged in recent years to provide the robust support needed for scalable cluster monitoring. However, the scalability comes at the expense of accuracy in that the tools only obtain data samples through an entry in the `/proc` *filesystem and only at the granularity of a kernel tick, i.e., 10 milliseconds. As an alternative to using* `/proc` *as a sensor for Ganglia and Supermon, we propose a dynamic, high-fidelity, event-based sensor called MAGNET (Monitoring Apparatus for General kerNel-Event Tracing). Unlike our previous incarnation of MAGNET, this incarnation allows for the dynamic insertion and deletion of instrumentation points and improves performance by approximately 100% over our previously low-overhead MAGNET and approximately 25% over the Linux Trace Toolkit (LTT) while providing superior functionality and robustness over LTT. Furthermore, our latest MAGNET is flexible enough to morph itself into other tools such as* `tcpdump` *and yet still high performance enough to perform over 250% better than* `tcpdump`*. It can also be used as a diagnostic (or debugging) tool, a performance-tuning tool, or a reflective tool to enable self-adapting applications in clusters or grids.*

Keywords: dynamic instrumentation, system monitoring, performance analysis, `tcpdump`, self-adapting applications, tools.

1 Introduction

Monitoring the performance and health of a cluster can be a daunting challenge for any system administrator, let alone an application scientist. For example, with large-scale clusters having mean time between failures (MTBFs) that are measured in hours, as shown in Table 1, hardware problems can cause nodes to crash or to become inaccessible, and chasing down these

problematic nodes in even a modest-sized cluster (e.g., 32 nodes) without any monitoring tools is akin to finding a "needle in a haystack."

As a result, cluster monitoring tools such as Ganglia [SKMC03,MCC04] and Supermon [SM02] have emerged and rapidly matured to address this problem in a scalable manner. However, the scalability of these tools comes at the expense of accuracy as they use the `/proc` filesystem as their "monitoring sensor." That is, these cluster monitoring tools are only sample-based and can only provide measurements at the granularity of a kernel tick, i.e., 10 ms. As an alternative to using the `/proc` filesystem as a sensor for Ganglia and Supermon, we propose a *dynamic, high-fidelity, event-based sensor* called MAGNET (Monitoring Apparatus for General kerNel-Event Tracing).

In addition to enabling on-line cluster monitoring, MAGNET can also be morphed into a higher-performance, subsystem-specific monitoring tool, e.g., "`tcpdump` on steroids," or used as a diagnostic tool, a performance-tuning tool, or a reflective tool to enable self-adapting applications in clusters or grids. Concrete examples of using MAGNET in many of the aforementioned scenarios can be found in [GFBE03].

System	CPUs	Reliability
ASC Q	8,192	MTBI: 6.5 hrs.
ASC White	8,192	MTBF: 5 hrs. (2001), 40 hrs. (2003)
PSC Lemieux	3,016	MTBI: 9.7 hrs.

MTBF: mean time between failures; MTBI: mean time between interrupts

Table 1: Reliability of Large-Scale Clusters

In summary, this paper presents the design, implementation, and evaluation of MAGNET, a high-performance sensor for cluster monitoring and adaptation 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 sampling-based), transformable (e.g., "`tcpdump` on steroids"), controllable (i.e., event stream can be filtered), and open source (i.e., Linux).

2 Background

In this section, we provide a brief overview of our previous work on MAGNET, hereafter referred to as MAGNETv2.0, and its accompanying MAGNET User-Space Environment (MUSE). We then describe the kprobes mechanism in the Linux 2.6 kernel that we leverage in our major overhaul of MAGNETv2.0.

MAGNETv2.0 was implemented as a kernel patch to the Linux kernel [GFBE03]. It creates a circular buffer of event records in kernel memory and provides a function magnet_add(), which inserts an event record into the buffer. A call to magnet_add() is placed at each point where kernel information is desired. To export kernel events to user space, MAGNETv2.0 provides magnet-read, a program that reads event records from the circular buffer in kernel memory and saves them to disk as a MAGNET trace for off-line post-processing.

However, monitoring and logging every single event with MAGNET can result in a "drinking from the fire hose" deluge of data for the aforementioned disk as well as for cluster monitors like Ganglia. As such, the ability to select an appropriate level of detail at which to monitor (or filter) events is critical. For example, it is sufficient to collect a periodic heartbeat from nodes in order to monitor the availability of computing resources in a cluster. However, if there is a problem with low performance due to poorly timed arrivals of messages, details concerning message arrival times are needed. In general, it is difficult to know a priori what level of detail is required. Furthermore, magnet-read from MAGNETv2.0 only provides the capability for off-line post-processing.

Consequently, we created MUSE – MAGNET User-Space Environment – to allow for the on-line monitoring of nodes in a cluster (or grid) [GBEF03]. The information available through MUSE can be tuned to provide just the right level of detail for the task at hand, whether that task be debugging, tuning, or status monitoring. Furthermore, MUSE can be used to support the development of self-adapting applications – applications that are aware of the environment in which they execute and can adapt their behavior based on that awareness.

MUSE provides an environment for user-space applications to make convenient use of the wealth of information that MAGNET exports from the operating system (OS) kernel. It consolidates the functions of event filtering and information synthesis into one component:

magnetd, a multithreaded daemon process with two main threads of execution, a data-collection thread and a master-server thread. The former extracts event records from the MAGNET kernel buffer and processes them. The latter listens for command connections from clients interested in obtaining information from magnetd and creating server threads to service their requests. The architecture of MAGNETv2.0 and MUSE is shown in Figure 1.

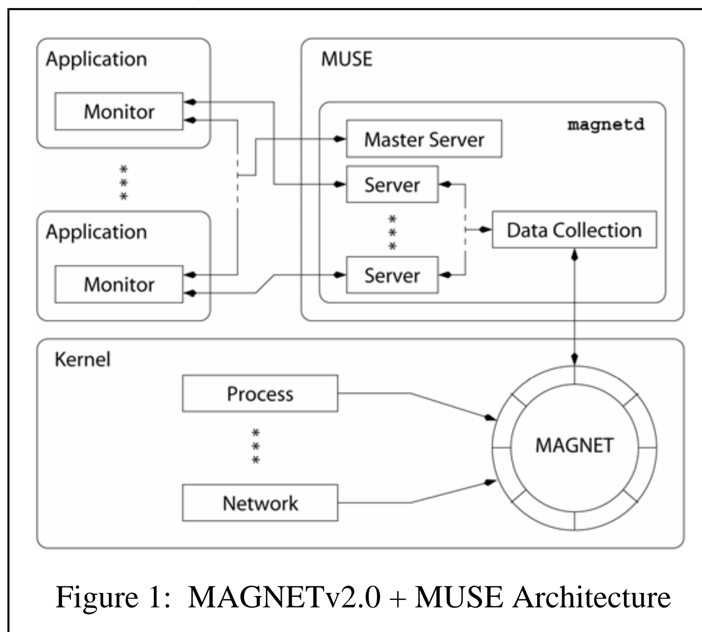

Figure 1: MAGNETv2.0 + MUSE Architecture

Although MAGNETv2.0 achieved high-fidelity, fine-grained monitoring by allowing any kernel event to be monitored and by timestamping each event with the highest-resolution time source available on most machines, the CPU cycle counter, it suffered from a pair of problems. First, the overhead was not as low as we had hoped. Specifically, the implementation used a pass-by-value mechanism resulting in an additional copy of the event stream. Consequently, monitoring the network stack with MAGNETv2.0 resulted in a 50% performance degradation in throughput. Second, because MAGNETv2.0 was implemented as a kernel patch, any change in the instrumentation monitoring points required a re-boot of the OS.

Our major overhaul of MAGNET, as presented in this paper, addresses the above problems by using low-overhead kernel probes for monitoring. These monitoring probes can be dynamically inserted and deleted with minimal overhead, thus eliminating the need for OS re-booting. The probes have zero overhead when disabled.

Kernel Probes

Kernel Dynamic Probes (Kprobes) is a dynamic instrumentation mechanism that provides a lightweight interface to dynamically break into any kernel routine and collect debugging and performance information non-disruptively without re-compiling or re-booting the OS kernel.

Kprobes is based on the work done on Dprobes from IBM. Kprobes is part of the mainstream kernel and available on the following architectures in Linux 2.6: ppc64, x86_64, sparc64, i386, and IA64. Kprobes enables locations in the kernel to be instrumented with code, instrumentation code that runs when the processor encounters that probe point. Once the instrumentation code completes execution, the kernel resumes normal execution.

A probe is an automated breakpoint that is implanted dynamically in an executing (kernel-space) module without the need to modify its underlying source. With each probe, a corresponding probe event handler address is specified. Probe event handlers run as extensions to the system breakpoint interrupt handler and are expected to have little or no dependence on system facilities. Because of this design feature, probes may be implanted in the most hostile of environments – interrupt-time path, task-time path, disabled path, inter-context-switch path, and SMP-enabled code path – without adversely skewing system performance.

There are currently three types of probes in the mainstream Linux 2.6 kernel: kprobe, jprobe, and kretprobe (also called return probes). All three types of probes allow instrumentation points to be dynamically inserted and deleted. However, they differ in the location where they can be inserted. A kprobe can be inserted on virtually any instruction in the kernel and provides access to the register arguments; a jprobe is inserted at the entry of a kernel function and provides convenient access to the function's arguments; and a kretprobe (or return probe) is inserted at the return address of a kernel function, fires when a specified function returns, and provides access to the return value of a function call. A fourth probe, currently under development, is the Direct Jump Probe. The direct jump probe (djprobe) from Hitachi uses a jump mechanism instead of the expensive software-interrupt mechanism. A djprobe can be inserted on virtually any instruction in the kernel and provides access to the register arguments.

With respect to their implementation, 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 a CPU hits the breakpoint instruction, a trap occurs, the CPU's registers are saved, and control passes to the registered kprobe. A kprobe executes the "pre_handler" associated with the kprobe and then single-steps its copy of the probed instruction. After the instruction is single-stepped, kprobes executes the "post_handler,"

if any, that is associated with the kprobe. Execution then resumes with the instruction following the probe point. A jprobe is implemented using a kprobe that is placed on a function's entry point. It employs a simple mirroring principle to allow seamless access to the probed function's arguments. kretprobe is implemented taking advantage of the trampoline instruction. djprobe uses a jump mechanism instead of the expensive software-interrupt mechanism. It thus has an extremely low overhead. djprobe is relatively new work in progress and is not part of the mainstream Linux kernel currently. Its inclusion is under discussion. Table 1 summarizes the differences between the various probe mechanisms that are currently available.

ProbeType	Linux 2.6?	Mechanism	ProbeType Accessibility	Performance Overhead
Kprobes	yes	Int3	Registers, global variables, and data structures	~800 ns
Jprobes	yes	Int3 (built on top of Kprobe)	Arguments of a function call, registers, global variables, and data structures	~1000 ns
Kretprobes	yes	Trampoline	Return value of a function call, registers, global variables, and data structures	~1100 ns
Djprobe	no	Jump	Registers, global variables, and data structures	~10 ns

Table 2: Comparison of the Kernel Probe Mechanisms in the Linux 2.6 Kernel

3 Design and Implementation of MAGNET

The interactions and feedback from the end users of previous versions of MAGNET played a key role in the new design. Its design and implementation are motivated with the goal of acceptance into the Linux kernel in mind. We believe that acceptance into the mainstream kernel, while not necessary, is important as it would more naturally enable the adoption of MAGNET into production clusters. This necessitates re-using existing code and subsystems in the Linux kernel rather than re-inventing the wheel.

Below, we present a brief overview of the MAGNET framework. We then proceed to elucidate the various subsystems in the MAGNET framework and how they work cohesively as a sensor for high-performance scalable cluster monitoring.

3.1 Overview

As shown in Figure 2, the MAGNET framework consists of a global *Control Subsystem* that interacts and oversees four other subsystems: *Monitoring Subsystem (MoniSub)*, *Kernel-Space Event-Filtering Subsystem (KernEveSub)*, *magnetfs,* and *Event-Stream Clients (EC).*

The MoniSub monitors events and produces an event stream that needs to be logged. This event stream is then passed through the KernEveSub. . The filtered event-stream is logged into the magnetfs logging subsystem. magnetfs consists of a set of circular buffers where the data can be logged. magnetfs exposes the circular buffers in the logging subsystem as regular files to the user-space clients that can be read by user-space clients using regular file operations. The circular buffers can also be read by kernel-space clients directly. The Control Subsystem interacts with the other subsystems and is responsible for the dynamic runtime configuration and management of the subsystems in the MAGNET framework.

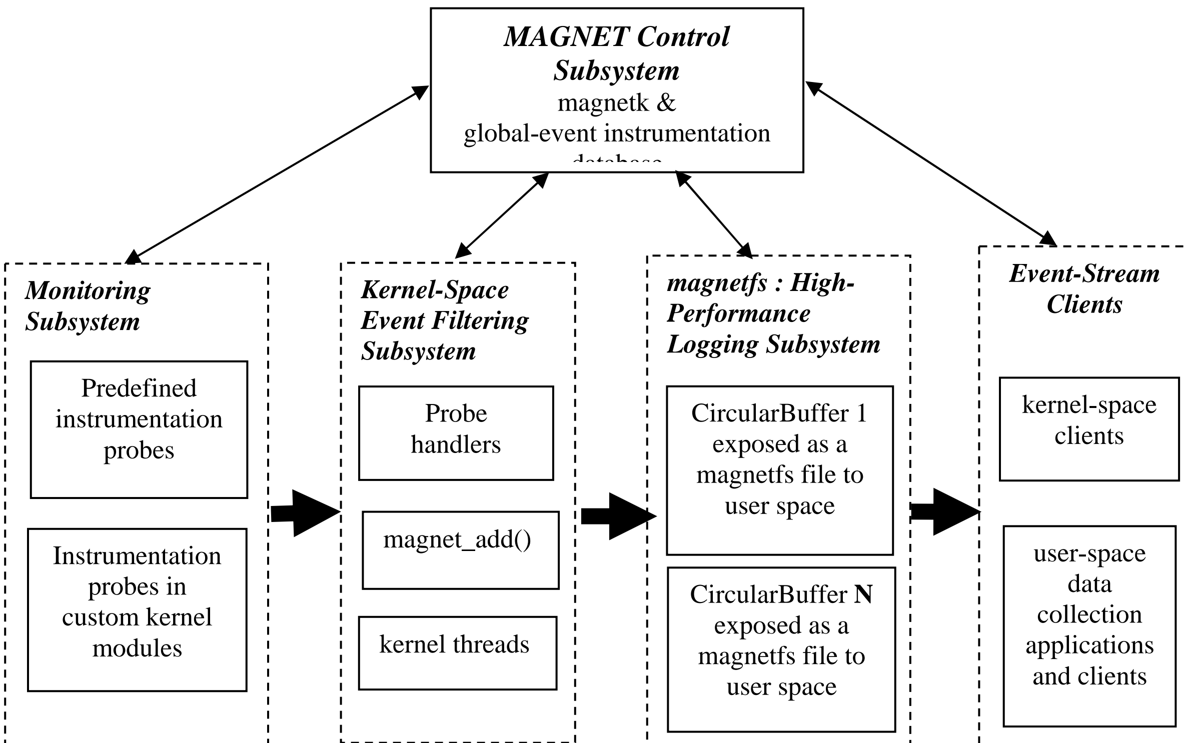

Figure 2: Architecture of the MAGNET Framework

3.2 Monitoring Subsystem (MoniSub)

The current design of MAGNET specifies a monitoring subsystem (MoniSub) that leverages the dynamic probes mechanism in the 2.6 kernel. This overcomes the need to patch a kernel with a static instrumentation point, as in MAGNETv2.0, and consequently, eliminates the need to re-boot the OS every time an instrumentation point is inserted or deleted. The new design and implementation of the MoniSub in MAGNET leverages the kprobes and jprobes mechanisms to instrument addresses and function entry points in the kernel. (We plan to incorporate the djprobes mechanism when it is accepted in the mainstream kernel.) The MoniSub is composed of predefined probes and predefined handlers. Further, the MoniSub is extendable. For instance, one can design a custom kernel module with custom probes and handlers. These can be registered with the MAGNET control subsystem to allow records to insert themselves into the magnetfs logging system.

```
struct magnetEvent {

   unsigned int _header;
/* bits 0-5 – CPU ID */
/* Bits 6-15 -  EVENT ID */
/* Bits 16-31 – Event size in Bytes */
/* Configurable at module loading */

   PTR_TYPE _abstractID;
/* PTR_TYPE is 4 or 8 bytes depending on the
architecture. The value depends on the event type */

   unsigned long long _timer;
/* high precision timing based on TSC */

  unsigned int _pid;
/* usually the tgid of the process but depends on the
Event Type and can be a event specific value */

 unsigned char* _eventRecord;
}
```

Figure 4: MAGNET Event-Record Structure

The MAGNET instrumentation record, as shown in Figure 4, currently consists of the following fields. It is configurable during the loading of the module. The instrumentation record format overcomes many of the limitations of MAGNETv2.0 and supports SMP configurations, 32-bit and 64-bit architectures, and variable-size records. The variable-size records enable efficient usage of the circular buffers in kernel space and provides the added functionality for recording the most pertinent and relevant information. The implementation of MAGNETv2.0 ultimately used a fixed length and fixed format instrumentation record.

3.3 Kernel-Space Event-Filtering Subsystem (KernEveSub)

Kernel-space event filtering (KernEveSub) is a critical service to classify and efficiently disseminate events in HPC systems. Event filtering greatly reduces the large volume of data that is transferred between kernel and user space that would otherwise need to be processed by user space. We implement event filtering in MAGNET with the help of a data structure of pertinent MAGNET event headers for each instrumentation point. Events can be filtered based on the processor ID, TGID, and abstract ID. A user–space application or a kernel client can register events of interest through magnetk. Three mechanisms of event filtering in kernel space are currently supported:

1. Events can be filtered with the help of a custom probe handler associated with the instrumentation point. This probe handler is typically loaded with a custom kernel module and registers with the MAGNET framework.
2. Events can be filtered in the `magnet_add()` call. Every probe handler calls `magnet_add` to insert a record into an appropriate buffer in the `magnetfs` logging subsystem. Before the event is logged, the header is parsed and matched with the list of predicates associated with the event. If the header satisfies the predicates, then it is inserted.
3. Events can be filtered with the help of a kernel thread that parses through the event stream record by record. If the header of the record satisfies the predicate, it is added to the `magnetfs` logging subsystem.

One would use filtering in probe handlers and `magnet_add` for filtering high-frequency streams to reduce the volume of data transferred to user space. The probe handlers typically must be small, simple, and consume as few CPU cycles as possible. Filtering using a kernel thread (kthread) is useful 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 time window. Filtering can be also carried out in user space with the support of user-space analysis and synthesis tools such as MUSE.

3.4 magnetfs : High-Performance Logging Subsystem

magnetfs is an efficient mechanism for logging event streams and transferring data at high-bandwidth rates from kernel space to user space. However, with network speeds outstripping the ability of end-host systems to keep up [BHHJ05,SCDL03], end hosts are increasingly becoming the bottleneck in high-performance computing (HPC). In addition, there is also an ever-increasing number of applications that try to adapt based on the conditions of the end host. These applications need in-depth information of what is going on in the kernel in order to adapt in real time. Thus, we need a logging mechanism that supports high data-transfer rates between kernel space to user space and that consumes as few CPU cycles as possible. For example, in order to monitor 10-Gbps streams, we would need to transfer data at a rate of 33.3 MBps from kernel to user space (assuming an MTU of 1500 bytes for packets and 40 bytes of event-record logging per packet). In addition, end-user applications that consume the monitored data need to be informed of critical events as soon as possible. Consequently, the above requirements dictate the need for an event-delivery mechanism that provides very low-latency notification of critical events from kernel space to user-space applications. With the above design goals in mind, we propose magnetfs – a high-performance logging subsystem for event streams and for efficient transfer of the event streams from kernel space to user space.

With an implementation mantra of reusing existing kernel subsystems to facilitate acceptance into the mainstream Linux kernel, we leveraged the debugfs file system [K-H04] in order to design magnetfs. The debugfs is an in-kernel file system, merged into the mainstream Linux 2.6 kernel and enabled by default in most Linux distributions. It enables the export of data from kernel space to user space. Because debugfs is extendable and allows default file-system operations to be overloaded with custom operations for each file in the debugfs file system, we leveraged it in the design of magnetfs to support high-performance

```
int file = open("/debugfs/MAGNET/data/file",O_RDONLY);

int nread;
while(1){
lseek(file, option, val); // Configure the read operations
nread = read(file, buf, sizeof(buf));
if (nread < 0){
            close(file);
            break;
}
```

Figure 3: Typical Pseudocode of a User-Space Client Reading a magnetfs File

transfers between kernel and user space. A circular buffer, created in the kernel for logging data, is exported to the user space as a file under the debugfs mount point. This file can be read by user-space applications via regular file operations. Probe handlers write logging data into the buffer; this data can then be consumed by user space when the file is read. The circular buffer can also be read by other kernel-space clients directly. Multiple buffers can be created in the magnetfs, and each buffer is then exposed as a file to user space. Alternatively, one can use a single global buffer to log all the events; this helps to maintain the global ordering of events. However, as the number of events to be monitored increases, the locking mechanism of a single buffer becomes a potential bottleneck. Hence, for scalable design, we also support the creation of multiple buffers wherein for a set of related events, one could create a new buffer. This reduces the locking overhead of a single global buffer.

A magnetfs file, i.e., the circular buffer, can be configured for event-based delivery of data or bulk delivery of data from kernel-space to user-space applications. As magnetfs files are implemented as circular buffers in kernel space, they are consumed when read. In other words, the files are not seekable like traditional files. This also limits the file to just one reader. As a user application cannot use lseek() for random access of magnetfs files, we modified the lseek() function to set the attributes of the magnetfs file, i.e., the circular buffer. This is analogous to the setsockopt() operation on a socket. An application can use lseek to set the buffer attributes such as READSIZE_WATER_MARK – the minimum amount of buffer that must be copied from kernel space to user space before a read operation can complete, MIN_READ_TIMEOUT – the minimum amount of time before the read operation can return, READCOUNT_WATER_MARK – the minimum number of events that must be copied over before a read can complete. Relative to the way that data is transferred, READSIZE_WATER_MARK is used for bulk transfer of data from kernel space to user space, and READCOUNT_WATER_MARK is used for the event-based delivery of data. The size of each buffer is a power of 2 to take advantage of the cheap comparison operation to test for the end of the buffer.

The 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 program to efficiently process data. Support for asynchronous I/O is very

critical for reduced CPU usage of user-space consumer applications in HPC systems. This helps in mitigating the CPU usage as seen in the polling-based reads in MAGNET 2.0. This overcomes a key performance limiting factor of MAGNETv2.0 where the client has to constantly poll for data thus consuming additional CPU cycles.

3.5 MAGNET Control Subsystem

The MAGNET control subsystem is responsible for the dynamic runtime configuration and management of the various subsystems in the MAGNET framework. It consists of magnetk – a kernel thread that is created when the magnet module is loaded and that is destroyed on the unloading of the module. magnetk is responsible for communication with the user-space applications and other kernel modules. The MAGNET control subsystem also consists of an event-instrumentation database. This contains a list of probes registered with the MAGNET control subsystem and their associated filters and files in use in the logging subsystem.

The communication between magnetk and user-space processes and the kernel modules is achieved with the help of *netlink sockets* [IETF3549]. Netlink sockets facilitate communication within the kernel and between kernel and user space. This method of communication is preferred over the `ioctl` interface and the `/proc` file-system interface by Linux kernel developers and maintainers. The communication between magnetk and the other subsystems is achieved via a reliable delivery mode over the unreliable netlink sockets with the help of acknowledgements and message sequencing. The message formats use well-defined headers for control and data messages. Typical control messages that magnetk receives from user space and the kernel modules include (i) requests to enable or disable instrumentation points during runtime, (ii) requests to create a new file to log events of interest in `magnetfs`, and (iii) requests to add or delete filters for event filtering.

Custom kernel modules with instrumentation points can join the MAGNET framework at runtime by registering their probes with the MAGNET control subsystem. As above, this is enabled by communicating with magnetk over netlink sockets. On a registration request, a unique event identifier is returned by magnetk to the kernel module. This identifier is used while inserting the related event records into the MAGNET logging subsystem, i.e., `magnetfs`.

The MAGNET control subsystem is also responsible for creating the circular buffers and managing them. The MAGNET control subsystem exposes the current state and attributes of the various subsystems via the magnetfs logging subsystem via control files that are read-only. These control files can then be read using normal file operations and are not consumed like the magnetfs data files.

4 Performance Evaluation

In this section, we quantify the performance impact of the MAGNET implementation. First, we compare the maximum achievable bandwidth of MAGNET to the same system running tcpdump. We then analyze and evaluate the performance of MAGNET's 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, thus making MAGNET an ideal sensor for cluster monitoring tools such as Ganglia and Supermon.

4.1 Experimental Method

The testbed used for the evaluation consists of two identical nodes equipped with dual Opteron 2.4-GHz processors, 1 MB of L2-cache and 4 GB of 200-MHz DDR SDRAM. The Linux distribution used was SUSE 9.3 with the 2.6.12.6-SMP kernel.org kernel patched with the Chelsio TCP offload engine (TOE) kernel patch. The motherboard used was a Tyan K8W Thunder, and the chipset was an AMD-8131. Each node had a Chelsio T210 TOE-based 10-Gigabit Ethernet (10GigE) network adapter plugged into a 133 MHz/64-bit PCI-X slot; the nodes were connected back-to-back. The driver version used for the network adapters was 2.1.4.

The offload capabilities of the network adapter were deliberately disabled by specifying the "toe_disable" parameter at boot time. This ensured that the protocol processing was done on the end host and not offloaded to the network adapter. The 10GigE Chelsio network adapters were configured with an MTU of 9000 bytes. The two nodes also had a 1-GigE E1000 Intel LX network adapter, each on a 100 Mhz / 64bit PCI-X slot. The GigE network adapters were also connected back-to-back with an MTU of 1500 bytes. For a workload, we used Iperf 2.0.2 [Iperf] on the sender side in order to saturate the network. We minimize the amount of interference in

our measurements by minimizing the number of processes running on the test machines to iperf and a few essential services. Between the two test nodes, the maximum unidirectional throughput achieved by a single Iperf UDP stream was 7.22 Gbps. This throughput limitation is mainly due to the PCI-X theoretical maximum bandwidth of 8.4 Gbps

4.2 MAGNET as tcpdump

Gigabit Ethernet (GigE) is already the most widely used interconnect in HPC environments. In the November 2005 listing of the TOP500 supercomputer list, nearly 50 percent of these supercomputers used GigE interconnects. As the 10GigE cost per port continues to drop exponentially, 10GigE will soon make inroads as a cluster interconnect of choice. As such, commodity performance-monitoring tools for 10GigE environments will be extremely important.

Traditionally, network performance has been monitored via tcpdump [tcpdump], a ubiquitous tool that allows one to monitor and capture network packets in Ethernet networks. It uses the libpcap library to intercept network packets and usually requires super-user permissions. libpcap is implemented as a library working in user space under a variety of operating systems. It involves a system call or similar facility that causes a switch into kernel mode and a copy of memory from the kernel to the user-level library. This call-and-copy is repeated for every packet traveling across the interface being monitored.

To mimic a *superset* of tcpdump behavior, we added instrumentation probes in MAGNET to monitor the journey of an UDP packet through the network stack. We added a probe each at the following four functions that a UDP packet traverses at the sender.

1. inet_sendmsg: This event indicates the handoff of the packet from the application layer to the socket layer.
2. udp_sendmsg: This event captures the UDP processing of the packet.
3. ip_push_pending_frames: This captures the packet at the IP layer.
4. dev_queue_xmit: This indicates the handoff of the packet to the Ethernet device.

For each event, we record information pertinent to the instrumentation point. MAGNET not only records information similar to tcpdump for a UDP packet, but it also provides a rich set of *additional* information of the events that occur as a UDP packet traverses the network stack. In

addition to the headers, MAGNET logs the timing information available through the timestamp counter (TSC) register when each probe is hit, the processor ID that handles the packet, the thread ID for the packet, among others. This is logged into a global buffer in a magnetfs file in the debugfs file system. The global buffer has a size of 2 MB to store the event records. magnet-read, a user-space program reads the event records from the magnetfs, consumes the data. In the case of tcpdump, the monitored data is logged onto the tmpfs, an in-memory file system. Thus the performance of tcpdump is not affected by the speed of the disks. magnet-read also stores the event streams onto the tmpfs and is not affected by the disk speeds.

4.2.1 MAGNET vs. tcpdump

In Figure 5, we compare the performance impact of monitoring with MAGNET and

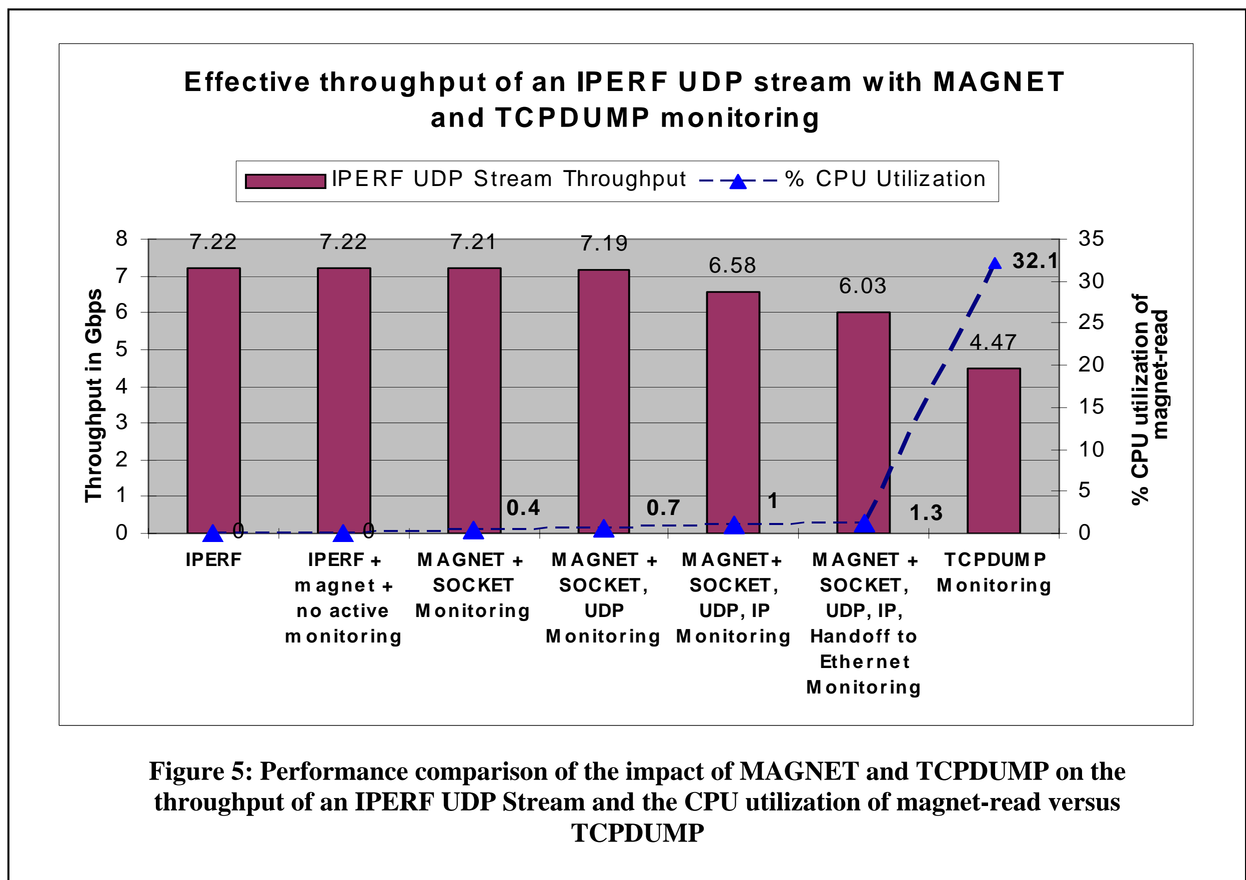

Figure 5: Performance comparison of the impact of MAGNET and TCPDUMP on the throughput of an IPERF UDP Stream and the CPU utilization of magnet-read versus TCPDUMP

tcpdump on an Iperf UDP stream. An Iperf UDP stream sustains 7.22 Gbps between the two test nodes over the 10GigE Chelsio network adapters. When the magnet kernel module is loaded

and probes are disabled, there is no loss in throughput for the UDP stream. Thus, there is no impact on the throughput of the UDP stream when MAGNET is loaded and the probe points are disabled. The impact of MAGNET on the throughput of the UDP stream when we monitor the socket layer (one event per packet) is 0.1% and the socket + UDP layer (two events per packet) is 0.4%. As the number of events to be monitored for a packet increases, the throughput of the UDP stream drops. This is due to the fact that when a probe is hit, an int3 software interrupt handler is invoked by the probe and results in a context switch with control passing to the probe handler. Then, there is additional computation done in the probe handler by MAGNET to save the necessary state and header information. (If, or when, djprobes is accepted in the mainstream Linux kernel, the overhead of the probe mechanism is projected to drop by two orders of magnitude, thus making the overhead of multi-probing in MAGNET on the throughput of the UDP stream to be very minimal.) In stark contrast, just monitoring a single event per packet with tcpdump results in a throughput drop of nearly 40% on the UDP stream!

The CPU utilized by magnet-read to read the information from the magnetfs and store it onto the tmpfs, is between 0.4 % to 1.3%. Thus, *the CPU overhead of MAGNET is about 32 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 hosts. In addition, there are no events lost by MAGNET while 0.01% events are lost by tcpdump in this test.

4.2.2 Scalability Analysis of magnetfs

In this experiment, we compare the performance of the magnetfs to transfer data efficiently from kernel space to user space. MAGNET was configured with four instrumentation points for the UDP stream. A UDP iperf stream between the two 10GigE network adapters was used as a traffic generator. In the case of MAGNET, the total amount of data logged for the combined four events; i.e. the data logged per packet; is given on the X-axis in Figure 6. For tcpdump, the total amount of data to be captured was restricted with the help of the –s switch and is given bythe X axis as well. We measured the goodput i.e., the effective throughput of transferring data from kernel space to user space. As we can see from Figure 6, magnetfs achieves a throughput

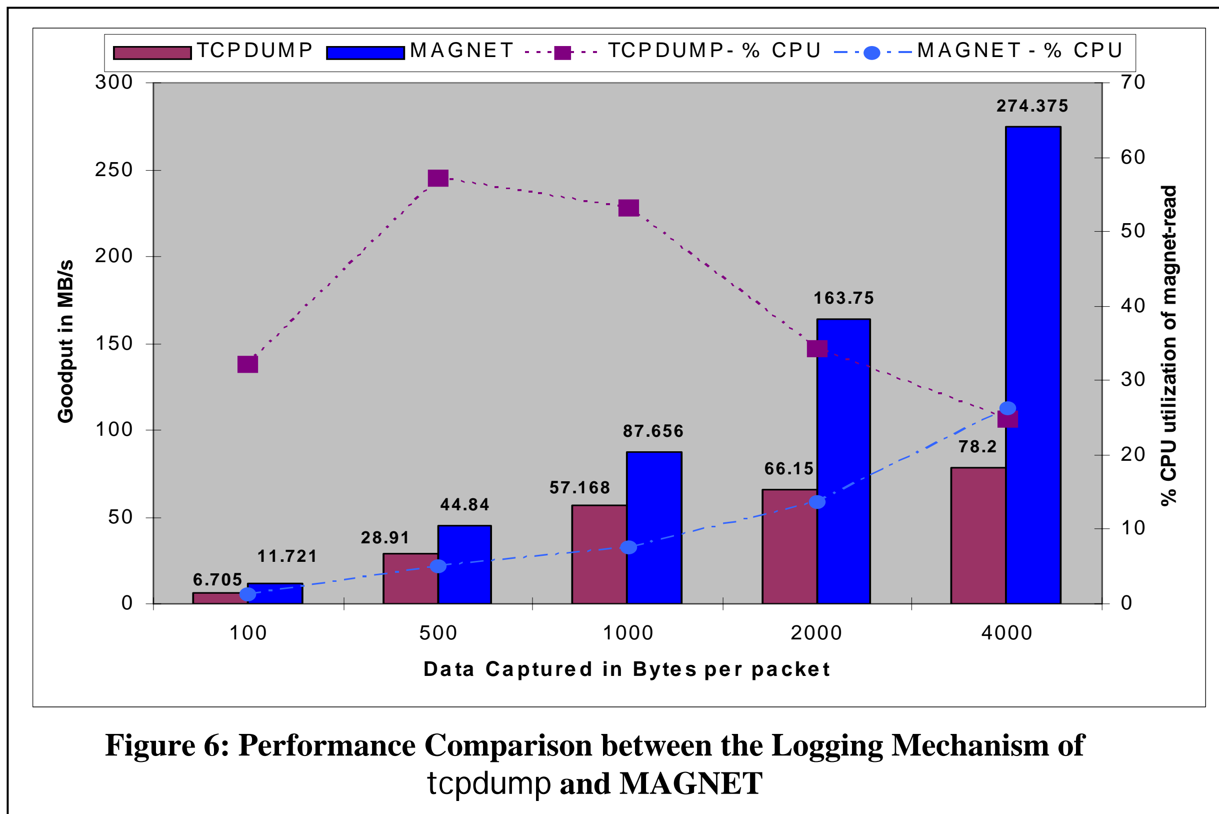

Figure 6: Performance Comparison between the Logging Mechanism of tcpdump and MAGNET

of 274.375 MBps (2.195 Gbps) from kernel space to user space. This high performance in large throughput transfers can be attributed to the high-performance interfaces supported by magnetfs for efficient transfer of data from kernel space to user space. The magnetfs buffer of 2 MB was configured for bulk delivery with MAX_READ_WATERMARK of 256K. magnet-read was configured to asynchronously read from magnetfs. The CPU utilization of magnet-read, the user-space application reading the data, versus tcpdump is also plotted. The CPU utilization of tcpdump drops as the data captured per packet is increased beyond 500 bytes. This is due to the fact that the percentage of packets dropped by tcpdump increases with an increase in the amount of data logged. As shown in Table 3, at 4000 bytes per packet, *tcpdump drops 80% of the packets,* and hence, the CPU utilization drops as it does not process the packets. *In contrast, magnetfs with a 0% event loss rate efficiently logs and transfers every event record to user space.*

Data logged in bytes per packet	100	500	1000	2000	4000
% of packet dropped by tcpdump	0	1.5	28.4	63.9	80.6
% of packet dropped by MAGNET	0	0	0	0	0

Table 3: Performance Comparison Between the Amount of Events (Packets) Dropped by tcpdump versus MAGNET

We can see that as the amount of data from kernel space to user space increases, the user-space application has to do more work to consume this data. This leads to higher CPU utilization at large transfer rates for magnet-read. Figure 6 clearly shows that magnetfs is able to transfer data at very high rates from kernel space to user space.

4.2.3 Performance Comparison of Statistical Kernel-Event Sampling in MAGNET

MAGNET supports event sampling wherein one can configure the sampling frequency in terms of number of events and time interval for logging the event stream. This is particularly important for very high frequency event streams such as monitoring 10GigE packet streams.

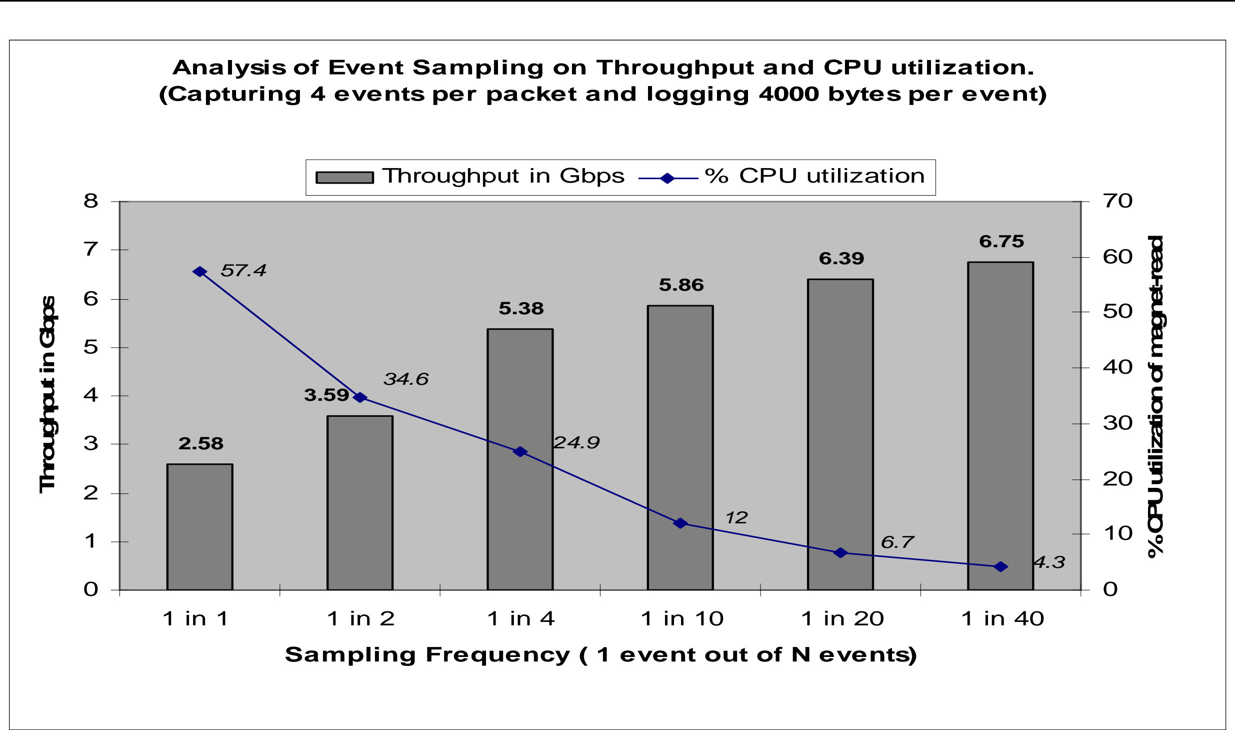

Figure 7: Performance Analysis of Statistical Kernel Event Sampling on the throughput of an IPERF UDP Stream and CPU utilization of magnet-read

Figure 7 shows the effect of statistical sampling of the event stream on the performance of the Iperf UDP Stream. In this experiment, we capture the socket, UDP layer, IP layer, and the handoff to the Ethernet device for each packet and record 4,000 bytes per event, i.e., 16,000 bytes per packet. The filters for each event were configured to sample based on the occurrence of the events. We can see that as the sampling frequency decreases, the throughput achieved by the Iperf UDP streams increases. Even though each packet hits the probe, the probe handler first checks if the event needs to be filtered before logging the event. Thus, the overhead of computation in the probes handler encountered by each packet is lower as the sampling frequency decreases. The CPU utilization also decreases as the sampling frequency decreases. This is primarily due to the smaller amount of data that is logged. Also, as mentioned earlier, the overhead of the probe mechanism is projected to reduce substantially as djprobes is incorporated into the mainstream kernel.

Obviously, there exists a tradeoff between capturing every event and statistically sampling the events. Capturing every event results in perfect information although at the expense of increased CPU utilization and higher impact on the UDP stream throughput. Statistical sampling of the event stream reduces the impact on the throughput of the UDP stream and also reduces CPU utilization. If perfect information is desired, then with clusters adopting multi-core processors, we can envision dedicating a processor, "a monitoring co-processor," for processing the event stream and thus reduce the impact on the application.

4.2.4 Filtering in Kernel Space with MAGNET vs. Filtering with `tcpdump`

We compare the performance of filtering in kernel space with MAGNET with filtering with `tcpdump`. In this experiment, we first ran an Iperf UDP stream between the two machines' Gigabit Ethernet (GigE) network adapters. The throughput achieved by this UDP stream was 956 Mbps. We then started a second simultaneous Iperf stream between the two machines' 10GigE network adapters. The throughput achieved by the second stream was 5.97 Gbps. The drop in performance of the second Iperf stream is due to the first UDP stream in the background. 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, UDP, IP, and the handoff

to the Ethernet device. An event filter with the thread ID of the second IPERF stream was added to the four instrumentation points to MAGNET. tcpdump was run with the Ethernet interface of the second UDP stream specified as a filter with the -i option.

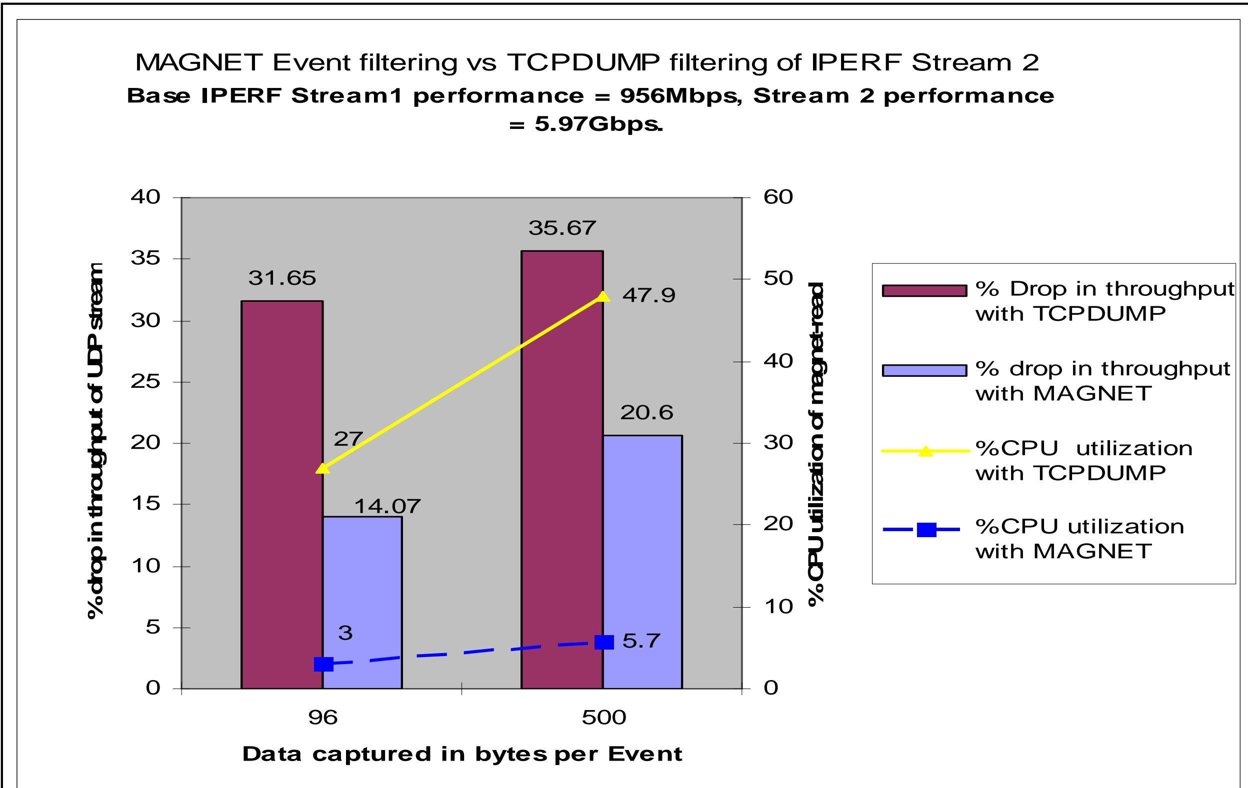

Figure 8: Performance analysis of MAGNET Event Filtering and TCPDUMP filtering on IPERF UDP Streams

From Figure 8, it can be seen that the percentage drop 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 drop in performance with MAGNET is due to the amount of data being logged per event. In case of the 96 bytes per event, MAGNET recorded a total of 96 bytes/event x 4 events = 388 bytes per packet compared to the 96 bytes for tcpdump. The performance of the UDP streams would improve as the amount of data recorded is reduced. The CPU overhead of magnet-read was far lower than the CPU utilization of tcpdump. tcpdump was also unable to capture 2% of the packets in the "96 bytes of data" case and around 27% for the "500 bytes of data" case. MAGNET captured and filtered the event streams (which were four times larger than the event streams for tcpdump) without any event loss.

5 Related Work

A great deal of prior work has been done in the areas of kernel monitoring. This has traditionally been achieved by static kernel instrumentation and dynamic kernel instrumentation. Static kernel instrumentation is used by tools such as Linux Trace Toolkit (LTT) [YD00], K42 [K42], our earlier version of MAGNET, and other tools that enhance LTT [SBM05] and LTT-ng [Ltt-ng]. Dynamic kernel instrumentation is used by tools such as Paradyn [MCCH95], Dtrace [CSL04] in Solaris, and SystemTap [SystemTap] which also uses the kprobes mechanism.

Statistical sampling-based tools such as OPROFILE [Oprofile], PAPI [MW03], PERFMON [Perfmon] have been also used to monitor systems. These tools take advantage of the hardware counter of the machines and sample them at regular intervals. MAGNET was designed for event-based sampling but can also be morphed into a statistical sampling-based tool. Relayfs [Relayfs04] is an efficient mechanism to transfer large volumes of data from kernel space to user space. Relayfs is used by LTT to transfer data to user space. magnetfs also provides the same features of Relayfs such as multiple buffers for logging data, bulk delivery of data, and so on, but magnetfs also provides a richer set of interfaces such as asynchronous I/O and signal-based I/O for high performance. magnetfs buffers are also configurable by user-space applications to provide event-based delivery of data. Additionally, magnetfs leverages the debugfs, thus reusing the existing kernel subsystems. 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 Discussion and Future Work

A few of the more important features of the current design and implementation of MAGNET include the following:

- Dynamic instrumentation using the kernel probes mechanism
- Kernel-module implementation rather than a patch to enable easy installation.
- Variable-size records.
- Support for 64-bit architectures and SMP compliant.

- Dynamic kernel-event filtering and sampling.
- Asynchronous support for consuming monitored data.
- Asynchronous notification of "important" events.
- Configurable high-performance logging mechanism.
- Reduces the CPU usage of the user-space data-collection applications by providing high-performance data-access interfaces.
- Reuses existing code and subsystems in the Linux kernel.
- The current implementation can monitor the entire network stack and support for other kernel subsystems are being added.

The current logging system is a lock-based system and support for lockless logging such as the K42's lockless logging mechanism using the `cmpcxchg` call, available on most architectures, will be added. The current implementation copies the event instrumentation record from the probe handler into the circular buffer. It is possible to avoid this copy by reserving buffer space in the buffer via the handler and writing directly to the buffer. We plan to integrate our new version of MAGNET with MUSE and take advantage of the rich set of event handlers, event synthesis, and analysis capabilities of MUSE.

Event filtering in kernel space causes higher system perturbation than filtering at an application level. However, the filtering in the kernel takes place closer to the source of the event, and thus, results in a very low latency of response to the event. This is important for time critical events in HPC. We plan to leverage the kernel-event filtering capabilities of MAGNET and the application level filtering capabilities of MUSE to build a synergistic, adaptive, unified, runtime filtering mechanisms that takes advantage of both kernel-space and user-space event filtering. The current implementation supports a limited set of filter rules that are based on the event record headers or on custom kernel-filter modules. We plan to incorporate an extensive and robust filtering capability similar to the dynamic rule set used in netfilters [Netfilters].

A visual representation of the profiled data would be quite useful for performance analysis. We are currently investigating the use of IBM PerfExplorer to visualize the event streams.

Finally, MAGNET supports event-based monitoring, and it can be extended to capture a snapshot of the hardware performance counters and add it to the magnetfs buffer by leveraging complementary tools such as PAPI and Perfmon when a particular event of interest occurs.

7 Conclusions

In this paper, we presented a dynamic, high-fidelity, event-based sensor called MAGNET (Monitoring Apparatus for General kerNel-Event Tracing), an alternative to using /proc as a sensor for Ganglia and Supermon. MAGNET allows for dynamic insertion and deletion of instrumentation points. It provides a highly configurable, high-performance logging subsystem (magnetfs) that supports asynchronous notification of events and asynchronous transfer of event streams to user-space clients. MAGNET enables an extremely low-overhead mechanism for user-space clients to consume the monitored event stream. MAGNET also supports dynamic kernel-space event filtering and event sampling, an important service to classify and efficiently disseminate events in HPC systems. MAGNET improves performance by approximately 100% over our previously low-overhead MAGNETv2.0. MAGNET is flexible enough to morph itself into other tools such as tcpdump and yet still high performance enough to perform over 250% better than tcpdump.

8 References

[BHHJ05] A. Benner et al., "On the Feasibility of Optical Circuit-Switching for High-Performance Computing Systems," *SC|05*, Nov. 2005.

[CSL04] B. Cantrill, M. Shapiro, A. Leventhal, "Dynamic Instrumentation of Production Systems," *USENIX Annual Technical Conference*, 2004.

[Djprobes] The DJProbes Home page, http://lkst.sourceforge.net/djprobe.html

[GBEF03] M. Gardner, M. Broxton, A. Engelhart, W. Feng, "MUSE: A Software Oscilloscope for Clusters and Grids," IEEE IPDPS 2003.

[GFBE03] M. Gardner, W. Feng, M. Broxton, A. Engelhart, J. Hurwitz, "MAGNET: A Tool for Debugging, Analysis and Adaptation in Computing Systems," *CCGRID 2003.*

[IETF3549] IETF RFC 3549: http://rfc.net/rfc3549.html

[Iperf] The Iperf home page, http://dast.nlanr.net/Projects/Iperf/

[K42] The K42 Team. *K42 Performance Monitoring and Tracing*, 2001.

[K-H04] G. Kroah-Hartman, Linux Kernel Documentation, 2004.

[Kprobes] Linux kernel documentation.

[LTT-ng] The LTT-ng home page, http://ltt.polymtl.ca/

[MCCH95] B. Miller et al., "The Paradyn Parallel Performance Measurement Tool," *IEEE Computer*, 28(11):37–46, 1995.

[MCC04] M. Massie, B. Chun, D. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience," *Parallel Computing*, Jul. 2004.

[MW03] P. Mucci, F. Wolf, "An Introduction to PAPI and PAPI-based Tools," *SC2003.*

[YD00] K. Yaghmour, M. Dagenais, "Measuring and Characterizing System Behavior Using Kernel-Level Event Logging," *2000 USENIX* 2000.

[Netfilter] The Netfilter Homepage, www.netfilter.org

[Oprofile] The OProfile home page. http://oprofile.sourceforge.net

[Perfmon] The Perfmon homepage, http://www.hpl.external.hp.com/research/linux/perfmon/

[Relayfs03] T. Zanussi, K. Yaghmour, R. Wisniewski, R. Moore, M. Dagenais, "Relayfs: An Efficient Unified Approach for Transmitting Data from Kernel to User Space. *Linux Symposium*, July 2003.

[SBM05] S. Sharma, P. Bridges, A. Maccabe, "A Framework for Analyzing Linux System Overheads on HPC Applications," *6th LACSI Symp.*, Oct. 2005.

[SCDL03] L. Smarr, A. Chien, T. DeFanti, J. Lei, P. Papadopoulos, "The OptIPuter," *Communications of the ACM, Special Issue: Blueprint for the Future of High-Performance Networking*, 46(11):58-67, Nov. 2003.

[SKMC03] F. Sacerdoti, M. Katz, M. Massie, D. Culler, "Wide-Area Cluster Monitoring with Ganglia," *5th IEEE Int'l Conf. on Cluster Computing*, Dec. 2003.

[SM02] M. Sottile, R. Minnich, "Supermon: A High-Speed Cluster Monitoring System," *IEEE Int'l Conf. on Cluster Computing*, Sept. 2002.

[SystemTap] The SystemTap Homepage, http://sourceware.org/systemtap/

[Tcpdump] The tcpdump home page, http://www.tcpdump.org