CALIBRATING AN ACTIVE NETWORK NODE

Yannick Carlinet, Virginie Galtier, Kevin L. Mills, Stefan Leigh, Andrew Rukhin

Abstract Active Network technology envisions deployment of virtual execution environments within network elements, so that nonhomogeneous processing can be applied to network traffic. For management purposes, each node must have a meaningful understanding of resource requirements - in terms of bandwidth, memory, and processing. To express the processing requirements in a platform-independent manner, we are developing a model of CPU time usage, which comes in two parts: a node model and an application model. In order to generate instances of the model, one needs to gather some metrics relative to the platform, that is, to calibrate a node. We have investigated what factors this process of calibration should account for, and especially how background load on a node affects our ability to obtain accurate calibrations for the CPU time used by node operating system calls and by virtual execution environments. We have shown that a background load, either computation intensive or input/output intensive, has little influence on the calibration. On the contrary, a memory consuming background load introduces an overhead in some measurements. The paper draws the conclusion that the calibration of a node can be done whatever the background load, provided that the memory consuming loads can be suppressed if necessary.

1. INTRODUCTION

Active Network technology envisions deployment of virtual execution environments within network elements, such as switches and routers, so that nonhomogeneous processing can be applied to network traffic associated with services, flows, or even individual packets. To use such a technology safely and efficiently, individual nodes must provide mechanisms to manage resources associated with specific network traffic. In order to provide such management mechanisms, each node must have a meaningful understanding of resource requirements for each active application. In Active Network nodes, resource requirements typically come in three categories: bandwidth, memory, and processing. Well-accepted metrics exist for expressing bandwidth (bits per second) and memory (bytes) in units independent of the capabilities of particular nodes. Unfortunately, no well-accepted metric exists for expressing processing (i.e., CPU time) requirements in a platform-independent form. To address this problem, we are developing a model of CPU time usage for an active application. The model consists of two parts: a node model and an application model. The node model represents the capabilities of a specific node with respect to elements likely to affect the performance of applications. The application model represents CPU time requirements in terms of elements contained within the node model. This paper investigates a method to calibrate Active Network nodes in order to generate instances of the node model. Specifically, the paper assesses the degree to which background load on a node affects our ability to obtain accurate calibrations for the CPU time used by node operating system calls (e.g., Linux system calls) and by execution environments (e.g., Java Virtual Machines).

We have shown that the presence of a computationally intensive competing workload does not affect significantly the calibration of system calls or execution environments on active network nodes. Our paper also shows that the presence of a competing workload of input/output intensive processes does not affect significantly the calibration of system calls or execution environments. We show as well that a memory consuming background load can have a noticeable influence on the calibration of some system calls and the execution environment. In this paper we describe and quantify this influence.

The paper is organized into four main sections. We begin by describing the context of the study: in Section 2 we consider the sources of variability in CPU time usage for an Active Network node. To the degree feasible any model proposed must account for these sources of variability. In Section 3, we describe the general outlines of a model to represent CPU time usage on an Active Network node. The model is described only in sufficient detail to motivate the need for node calibration. Then in Section 4, we describe an approach to calibrate the system calls in a node operating system. We give some results from calibrating Linux system calls, and we specifically address the influence of background load on the calibration measurements. In Section 5, we describe an approach to calibrate virtual execution environments running on a node operating system. We give some results from calibrating ANTS (Java virtual environment), and we specifically address the influence of background load on the calibration measurements. Section 6 discusses the results we obtained.

2. SOURCES OF VARIABILITY IN CPU TIME USAGE IN AN ACTIVE NETWORK NODE

Any reasonable metric for an application's CPU time requirements must account for the major sources of variability affecting the application. In this Section we identify and discuss the major sources of variability likely to affect the CPU time requirements of an active application. A proposed architecture for an Active Network node (Calvert, 1998) identifies several components and the relationships among them. Figure 1 gives a conceptual overview of the major components and relationships.

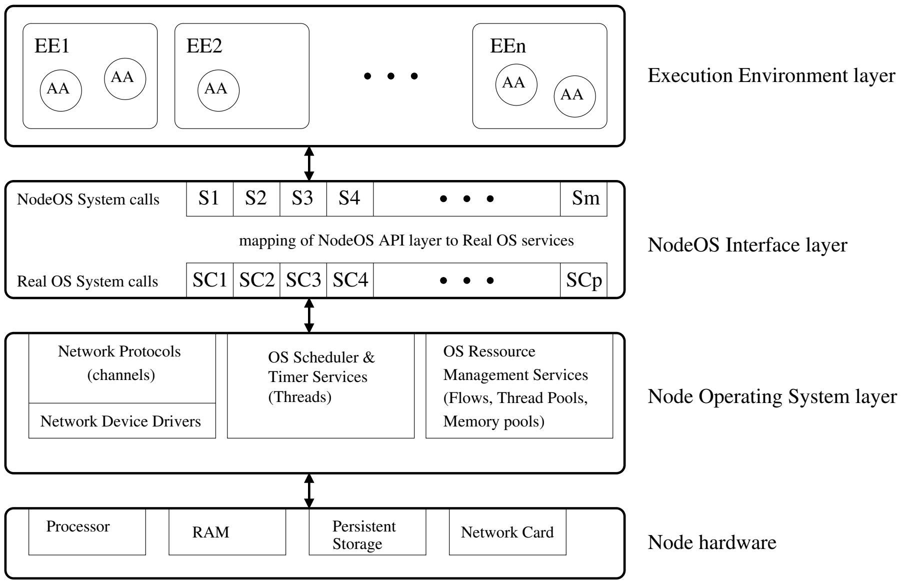

Figure 1 Architecture of a Active Network Node

The components can be viewed as four layers: the hardware, the node operating system (Node OS) and interface, the execution environment (EE), and the active application (AA). Each EE provides a virtual execution environment (similar for example to a Java Virtual Machine, Lindholm and Yelling, 1997) in which AAs can execute. Several EEs have been defined and implemented within the Active Networks research community (e.g., see Wetherall et al., 1999, Alexander et al., 1997, Schwartz et al., 1999, Bhattacharjee et al., 1997, and Mosberger and Perterson, 1997). In addition, several implementations of a Node OS are being developed (e.g., see Kaashoek and et al., 1997, Decasper et al., 1999, and Ford et al., 1997). To enable any EE to run over any implementation of a Node OS, a standard application-programming interface, in the form of system calls, is defined within a separate Node OS specification, in Peterson (ed.), 1999.

Our analysis of this model and of real systems, revealed the main sources of variability affecting the CPU time requirements of an AA. In the hardware layer, the main factors that influence the execution time of an application include: the frequency of the processor, the type of processor (e.g., Pentium, Pentium II, K6, Sparc, and so on), the amount of memory available on the host, the speed of the different buses (e.g., memory, I/O, and system), the technology of the persistent storage (SCSI or IDE hard drive, for instance) - but only if the application makes I/O accesses, and the type of network card (e.g., 10 or 100 Mbps Ethernet). Within the node operating system (OS) and node OS interface layer, the main sources of variability include: the performance of the device drivers, the performance in managing processes and memory, and the nature and performance of the system calls provided by the operating system. In the case of networking system calls, performance of reads and writes also vary based on the specific protocol stacks that are buried beneath the system calls, as well as the implementation of those protocols. Within the EE layer, performance can be affected by the mapping between the EE system calls and the OS system calls (a mapping usually defined by libraries), as well as the compiler and options used to compile the EE. For example for a Linux system, if the EE uses the Java Virtual Machine (JVM), then we must consider the performance of the C library and the results from the C compiler used to compile the JVM. Finally, the execution of a specific AA can go through many paths in the code of the program. The path of execution taken can depend on many things, such as the state of the node (e.g., whether data is cached or not), the data carried (e.g., the length of the data to be processed), and even the state of other nodes (e.g., in an active multicast application, an intermediate node creates and sends as many new packets as the number of subscribed nodes). This paper does not address the modeling of an AA (which is described in Galtier et al., 2000), but rather focuses on estimating the performance of the lower layers (hardware, Node OS, and EE).

3. GENERAL OUTLINE OF A MODEL FOR ACTIVE NETWORK NODES AND APPLICATIONS

We have defined a model that represents CPU time usage of AAs as a function of the CPU time used in Node OS system calls and in a specific EE between Node OS system calls. An AA transaction enters and exits a state-transition graph at an idle state (S_0). Between entry and exit, the AA executes a series of system calls (states S_1 through S_m), also executing within an EE between each system call. The fundamental model views the execution of each AA transaction as a set of transitions in the graph, where the probability of making a specific transition is driven by the AA logic, and where the time taken during each transition is a function of the time spent in the source system call, and a function of the time spent in the EE during the transition between system calls. The projected CPU time used by a transaction is then the sum of all the transition times.

The AA model can be represented as a vector of system calls and a matrix of transitions. By representing an AA as a function of Node OS system call performance and EE performance on a node, we expect to scale AA models into terms meaningful for specific nodes and for a specific EE on the node. We define two transformations: (1) node-to-reference (NR) and (2) reference-to-node (RN). Prior to transferring an AA model between two nodes, the model is subjected to an NR transform. The model, with its CPU time requirements expressed in terms of a reference node, is then transmitted across the network. Upon arrival at the next node, the model is subjected to an RN transform. The combination of these two transforms will scale the CPU times within an application model from a form understood on one node into a form understood on another. This paper investigates how the calibration of system calls and EEs might be accomplished, and considers how uncontrollable background loads might affect calibration measurements.

4. CALIBRATING A NODE OPERATING SYSTEM

We reports the results from calibrating the CPU time used by 13 system calls on two different nodes, *node A* and *node B*, whose characteristics are given in Table 1 below. We selected these 13 system calls because they are the set used by the active application ping. They are representative of the calls that any Active Application (AA) would make, because the active ping application performs the very basic operations of an AA. In ping, the active packet goes through all the network layers of the node, accesses the information available at the node, accesses the routing table of the node, sometimes changes itself (when going back toward the source) or forwards itself to the next hop. Every AA is likely to perform these operations, and therefore to invoke the same system calls we have measured.

	node A	*node B*
CPU	pentium II - 333 Mhz	pentium II - 450 Mhz
RAM	128 Mbytes	128 Mbytes
storage	SCSI hard drive	IDE hard drive
OS	linux 2.2.7	linux 2.0.36
EE	ANTS 1.2 on jdk1.1.6	ANTS 1.2 on jdk1.1.7B

Table 1 Characteristics for the Platforms Used in the Experiments

To assess the effects of background load on our measurements, we define three forms of background load: computation intensive, input/output (i/o) intensive, and memory consuming. The computation intensive workload consists of a process that continuously compresses 5,000 bytes of data, using the Huffman coding algorithm. The main purpose of this process is to fill the cache memory of the processor to uncover any increase in measured CPU time use related to switching processing contexts. The i/o intensive workload consists of a process that continuously copies a 1 Mbyte file, using 100 byte blocks. The main purpose of this process is to uncover any increase in measured CPU time use related to processing competing i/o traffic. The memory consuming load repeats the task of generating randomly an array of 10,000,000 integers and sorting it. To calibrate the system calls, we execute a workload that makes each call 200 times, using strace (Akkerman et al., 2000) to compute the average CPU time taken for each system call.

We performed measurements of the system calls made by the workload first without any load at all on each machine. Then we made the same measurements while varying the number of processes (from 1 to 5, and 10) generating computation load. We repeated that same procedure replacing the computation load with the two other kinds of background load (i/o and memory loads). The platforms used are unable to run more than 5 memory consuming load processes, due to a lack of memory. Therefore we varied the number of processes generating such a load only from 1 to 5. For a given type of load and a given number of competing processes of that load, we performed several measurements to uncover any possible variability in the results. When measuring a particular system call, we perform enough replications to ensure that the true mean lies within 5 % of the computed mean with 95 % confidence.

Node	Call	no load	comp 10	I/O 10	mem 5
node A	close	7	8	8	9
	link	22	23	23	26
	open	11	13	12	13
node B	close	9	9	10	9
	link	32	32	33	35
	open	32	34	34	61

Table 2 Time in μs spent in system calls, while varying the background load

Table 2 shows the results we obtained for a selection of three system calls. In this table, “comp 10” should be read as “10 competing computation intensive processes”. Similarly “I/O 10” refers to a background load of 10 competing input/output intensive processes. “mem 5” is a load of 5 processes of the memory consuming load. The precision of the numbers given in the table is not better than plus or minus 1 μs, the resolution of the measurement tool we used.

On both machines and for each system call, there is no significant difference between the measurements without load and with the computation load (whatever the number of competing processes). Indeed, the overhead measured, if any, does not exceed the precision of the measurement tool (there was no overhead larger than 2 μs). Similarly, on both machines and for each system call, there is no significant difference between the measurements without load and with the input/output intensive load (whatever the number of competing processes).

On *node B*, when running the memory consuming loads, we have noticed an increase only in the processing time of the calls `open`, `link`, and `stat`. That increase is 29 μs (91 %) for `open`, 3 μs (9 %) for `link`, and 9 μs (75 %) for `stat`. On *node A*, with the same kind of load, three system calls - `read`, `link`, and `socket` - show a significant difference with measurements taken under no load. They all three have a 4 μs overhead. We discuss these results in Section 6.

5. CALIBRATING AN EXECUTION ENVIRONMENT

We report here the results from calibrating ANTS (Wetherall et al., 1999), a Java-based execution environment for active networks. We have calibrated ANTS using two different calibration workloads, active ping and active multicast, on two different platforms (*node A* and *node B* - see Table 1). We have executed each calibration workload at least 200 times to generate several CPU time usage measures, including mean, variance, 95^{th} and 99^{th} percentiles, as well as minimum and maximum. We used the `proc` filesystem in Linux to make the measurements (i.e., to get the CPU time consumption of the active application). The resolution is 1 centisecond. Unlike the measurements in Section 4 (where we measure only kernel time), here we measure the total CPU time - i.e., the sum of kernel and user time. First we measured the two calibration workloads without any background load on the two platforms. Then in a second experiment, we performed calibrations while varying the number of processes (2 and 10) generating computation load. In a third experiment, we performed calibrations while varying the number of processes (2 and 10) generating i/o load. We also performed calibrations while varying the number of processes (2 and 5) generating memory consuming load. We employed the same load generators used previously to calibrate system calls. The two active applications we used for the calibration can be found in the standard distribution of ANTS.

To measure the multicast application we used a network of four nodes, all running on the same machine. The topology of the nodes is as follows: the server node is connected to an intermediate node, which in turn is connected to the two client nodes. We recorded the CPU usage of the intermediate node. The two client nodes regularly send messages subscribing to the multicast list. From the moment we start recording the CPU time used by the intermediate node, the server node sends 30 data messages (one every second) to the multicast list. We used unchanged the data messages given as examples in ANTS (version 1.2) distribution, as well as all the parameters of the application. The intermediate node forwards these messages to the clients, since they have both subscribed.

When the server is done sending the 30 messages, we stop recording the CPU usage of the intermediate node and we store the result. Then we stop the four nodes and start them again for the next replication of the measurement. For the active ping application, we used a testbed network consisting of three logical nodes in series, all three running on the same machine. We recorded the CPU usage of the intermediate node.

Table 3 below reports a summary of the results we obtained. The table reports the difference in the measurements with a background load compared to the same measurement without any load, for both applications (ping and multicast) and both platforms.

Node	Application	comp 10	I/O 10	mem 5
node A	ping	+ 3.2 % (+ 1.2 cs)	- 3.2 % (- 1.2 cs)	+ 1.3 % (+ 0.5 cs)
	multicast	+ 2.0 % (+ 2.3 cs)	+ 0.9 % (+ 1.0 cs)	- 1.3 % (- 1.5 cs)
node B	ping	0 % (0 cs)	- 0.6 % (- 0.2 cs)	+ 67 % (+ 21.3 cs)
	multicast	+ 5.2 % (+ 5.3 cs)	+ 6.1 % (+ 6.1 cs)	+ 20.7 % (+ 20.9 cs)

Table 3 Relative and absolute (measured in centiseconds) increase of the measurements with a background load, compared to the case without any load

The results indicate that EE calibration is largely unaffected when competing with a load of computation intensive processes. When competing with an input/output intensive load, the calibration is still little influenced. Note that for ping, on both platforms, it takes actually less time for the workload to execute when there is an I/O intensive load. While this decrease may seem surprising, our explanation is given in the discussion section. When competing with a memory consuming load, the results differ. On *node B*, the CPU time needed to run both active applications increases significantly. While the increase expressed as a percentage seems very different for ping and multicast, it is interesting to notice that the absolute increase is the same (approximatively 21 cs). On *node A*, a memory consuming load does not influence the measurements at all.

6. DISCUSSION

In this section we discuss the results obtained, and presented in the previous two sections. The following explanations (that have not been verified but which we believe to be true) refer to mechanisms present in the Linux kernel. For more details about these mechanisms, see Rusling, 1999. The following could hold for other operating systems as well.

A computational background load fills the cache memory of the processor. Therefore when the process we are measuring gains the processor, the latter first empties its cache memory and fills it with the data that the process will need to use. That operation requires CPU processing and the CPU time needed is taken on behalf of the process we are measuring. That explains the (relatively small) increase of the total CPU time required to run an active application. However, for system call calibration, there is no increase in the measured time. That is because we are only measuring in that case the time spent in kernel mode. When the process enters into kernel mode, it has already gained the processor before, therefore the processor has already replaced the data in its cache memory. The processing time needed to empty and fill the cache memory cannot be measured in the CPU system time (also called kernel time).

One of the main effects of an input/output intensive background load is that many processes want to access kernel ressources. Indeed, a process generating I/O load spends most of its time in kernel mode. When the process we are measuring wants to enter into kernel mode, it has to wait first until the kernel is released by the process that had a lock on it. When detecting that the kernel is locked by another process, the process adds itself in the kernel semaphore's wait queue. Then it enters in a loop (spin lock) checking if the lock on the kernel has been freed. All the processing time required to perform the spin lock is taken on behalf of the process waiting, and it happens in user mode. This explains why there is a very small overhead in the measurements of multicast (on both machines). For the ping application, the small decrease is not significant because it is only 1.2 μs (at most), so it is small compared to the precision we can achieve. There is no reason why the system calls should see an overhead in the time they require, because when the system call starts, the process has already entered the kernel, therefore there is no CPU processing required to wait for the kernel lock to be released.

The memory consuming load fills the random access memory (RAM), and partly the virtual disk memory (swap memory). As a consequence, a process competing with that kind of background load can hardly keep its data in RAM. Because the operating system needs all the RAM to execute the background loads, it flushes the memory address space of the process we are measuring and stores it on the swap filesystem. Every time the process we measure gains the processor, it has to load all of its data into RAM from the swap space. This loading consumes significant CPU processing time, thus, explaining the increase of the active application time on *node B*. On *node A*, since the swap partition is on a SCSI disk, the page swapping is performed by the SCSI controller and there is much less additional CPU processing required for that task. Therefore there is little overhead measured in the time taken by the active applications. For the system calls, similarly to the computation intensive load case, when the process enters the kernel, it has already gained the processor, therefore it has already performed the memory swapping. That is why the memory swapping time is not accounted for in the system time, and consequently neither in the node OS calibration. However, if the kernel (on behalf of a process making a system call) needs to allocate additional memory, a page swap will be triggered, and all the CPU time needed for that page swap will be counted in the time used by that system call. That is why some system calls see their execution time increase because of the memory consuming load. Which system calls will see such an overhead is determined by the particular implementations of the calls.

7. CONCLUSION

This paper has investigated a method to calibrate active network nodes in order to generate a model for CPU time usage by active applications. Specifically, the paper assesses the degree to which background load on a node affects our ability to obtain accurate calibrations for the CPU time used by node operating system calls (e.g., Linux system calls) and by execution environments (e.g., Java Virtual Machines). Our paper has shown that the presence of a computationally intensive competing workload does not affect significantly the calibration of system calls or execution environments on active network nodes. This same conclusion holds when the background load we use is input/output intensive. We also show in this paper that the presence of a memory consuming background load increases the CPU time used by the execution environment, but only if it is the CPU that has to perform the memory page swapping. In the latter case, the overhead introduced by the load is very noticeable. If the processor can rely on another device - such as a SCSI controller - to perform the swapping, then there is little overhead due to the memory swapping. These conclusions imply that when performing the node OS calibration (that is, the measurement of the calls at the node OS interface, see Peterson (ed.), 1999), as well as when performing the execution environment calibration, the background load does not influence significantly the measurements, as long as any memory consuming background load can be suppressed. Nonetheless, it might not be necessary to actually suppress the memory consuming load in one of the following cases: there is enough memory on the host to avoid disk swapping, or the CPU does not bear the main burden of page swapping.

Acknowledgments

The work discussed in this paper was conducted under joint funding from the Defense Advanced Research Projects Agency (DARPA) and the National Institute for Standards and Technology (NIST). The authors thank Jon Turner for his suggestion that we investigate the effects of various background loads on our techniques for measuring CPU usage.

References

Akkerman, W., Lankester, B., Kranenburg, P., and Sladkey, R. (1991-2000). *Strace*. Gnu Public License, www.wi.leidenuniv.ml/~wichert/strace.

Alexander, D. S., Arbaugh, W. A., Hicks, M. W., Kakkar, P., Keromytis, A. D., Moore, J. T., Gunter, C. A., Nettles, S. M., and Smith, J. M. (1997). The switchware active network architecture. *IEEE Network Special Issue on Active and Controllable Networks*, vol. 12(no. 3):pages 29–36.

Bhattacharjee, S., Calvert, K. L., and Zegura, E. W. (1997). An architecture for active networking. *Proceedings High Performance Networking (HPN 97)*, White Plains, NY.

Calvert, K. (1998). Architectural framework for active networks. *Active Networks Working Group*.

Decasper, Parulkar, Choi, DeHart, Wolf, and Plattner (1999). A scalable, high performance active network node. *IEEE Network*.

Ford, Back, Benson, Lepreau, Lin, and Shivers (1997). The flux oskit: A substrate for os and language research. *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, ACM Press.

Galtier, V., Mills, K. L., Carlinet, Y., Leigh, S., and Rukhin, A. (2000). Expressing meaningful processing requirements among heterogenous nodes in an active network. *Proceedings of the Second International Workshop on Software Performance*.

Kaashoek, F. and et al. (1997). Application performance and flexibility on exokernel systems. In *16 th Symposium on Operating System Principles*, pages 52–65. ACM Press, New York.

Lindholm, T. and Yelling, F. (1997). *The Java Virtual Machine Specification*. Addison-Wesley, Reading, Mass.

Mosberger, D. and Perterson, L. L. (1997). Making paths explicit in the scout os. In *Proceedings of the Second Symposium on Operating System Design and Implementation*, pages 153–168. ACM Press, New York.

Peterson (ed.), L. (1999). Nodeos interface specification. *Active Networks Node OS Working Group*.

Rusling, D. A. (1996-1999). *The linux kernel*. Gnu Public License, www.-linuxdoc.org/LDP/tlk/.

Schwartz, Jackson, Strayer, Zhou, Rockwell, and Partridge (1999). Smart packets for active networks. *Proceedings of OpenArch 99*.

Wetherall, D., Guttag, J., and Tennenhouse, D. (1999). Ants: Network services without the red tape. *IEEE Computer*, pages 42–48.