Vertical Perimeter Based Enhancement of Streaming Application

P.Manikandan
Department of Computer Science
Pondicherry University
Puducherry, India
aravindkarthi10@gmail.com

R.Kathiresan
Department of computer science,
Kalasalingam University,
Madurai,India
kathirbykathir@gmail.com

Marie Stanislas Ashok
Head and System manager,
Computer centre,
Pondicherry University,
Puducherry, India
ashokms@yahoo.com

***Abstract*-The explosion of web and increase in processing power meets a large number of short lived connections making connection setup time equally important. With Fire Engine, the networking stack went through one more transition where the core pieces (i.e. socket layer, TCP, UPD, IP, and device driver) used an IP Classifier and serialization queue to improve the connection setup time, scalability, and packet processing cost. The "Fire Engine" approach is to merge all protocol layers into one STREAMs module which is fully multi threaded. Inside the merged module, instead of using per data structure locks, use a per CPU synchronization mechanism called "vertical perimeter". The "vertical perimeter" is implemented using a serialization queue abstraction called "squeue"**

***Keywords*- threading, scheduling, dispatching, STREAM, Multicore, Fire Engine, Vertical Perimeter, CPU scheduling, task queue, IP Multithreading.**

I. INTRODUCTION

In recent years, since the clock rate of single processor cannot be increased without overheating, to increase performance, manufacturers develop multicore systems instead. In order to run the applications on multicore systems, there are many parallel algorithms developed, e.g. parallel H.264 applications. By exploiting parallelism, multicore systems compute more effectively. Multiple threads and processes are common useful approaches to speed up user a task with one thread in some operating systems, e.g. Linux.

In our observation, threads sometimes are not dispatched reasonably on processors. We redefine these anomalies formally from multiprocessing timing anomalies and focus on thread manipulation on multicore systems. For example, even if some cores are idle, the operating system does not dispatch any thread to the idle ones. Furthermore, even if users find out this situation, they still cannot directly dispatch these threads accordingly if the operation system does not provide related system calls. Although there are some existing mechanisms, like procfs, to access system information and system calls, like sched_setaffinity (), to dispatch threads to certain processors, dispatch wraps these into a complete API. It not only provides generic interface for future extension and high portability without kernel modification, but also performs much better than procfs.

A. *Fire Engine*

The Fire Engine [10] networking stack for the Solaris Operating System (OS) is currently under development by Sun. Enhanced network performance and a flexible architecture to meet future customer networking needs are twin goals of Fire Engine development. Addressing existing requirements, including increased performance and scalability, Disaster Recovery (DR), Secure Internet Protocol (IPSec), and IP Multiprocessing (IPMP), as well as future requirements—such as 10-gigabits per second (Gbps) networking, 100-Gbps networking, and TCP/IP Offload Engine (TOE)—are given equal priority.

Implemented in three phases, Fire Engine's development stages are structured to provide increased flexibility and a significant performance boost to overall network throughput. Phase 1 has already been completed and these goals have been realized in services using TCP/IP. Web-based benchmarks show a 30- to 45-percent improvement on both SPARC® and Intel x86 architectures, while bulk data transfer benchmarks show improvements in the range of 20 to 40 percent. Phases 2 and 3 should deliver similar overall performance improvements. With increased flexibility and performance boosts of this magnitude, FireEngine is well on its way to reinforcing Sun's Solaris OS as the commercial standard for networking infrastructure.

B. *Performance Barriers*

The existing TCP/IP stack uses STREAMS perimeters and kernel adaptive mutex for multithreading. As the current STREAMS perimeter provides per module, per protocol stack layer, or horizontal perimeters. This can, and often does, lead to a packet being processed on more than one CPU and by more than one thread, leading to excessive context switching and poor CPU data locality.

C. *Network Performance*

FireEngine introduces a new highly scalable, packet classification architecture called **Firehose**. Each incoming packet is classified early on, then proceeds through an

optimized list of functions—the Event List —that makes it easy to add protocols without impacting the network stack's complexity, performance, or scalability. FireEngine concentrates on improving the performance of key server workloads that have a significant networking component. The impact of network performance on these workloads, as well as benchmarks that describe overall workload performance

D. *Performance metrics*

Applications often use networking in two distinct ways: To perform transactions over the network, or to stream data over the network. Transactions are short-lived connections transferring a small amount of application data, while streaming data is a transfer of large amounts of data during long-lived connections. In the transaction case, performance is determined by a combination of the time it takes to get the first byte (first-byte latency), connection set up/tear down, plus network throughput (bits per second or bps). In the streaming case, performance is dominated by overall network throughput. These parameters impact performance in various ways, depending on the amount of data transferred. For instance, when transferring one byte of data, only first-byte latency and connection set up/tear down count. When transferring very large amounts of data, only network throughput is relevant. Finally, there is the ability to sustain performance as the number of active simultaneous connections increases. This is often a requirement for Web servers. A networking stack must take into account the host system's hardware characteristics. For low-end systems, it is important to make efficient use of the available hardware resources, such as memory and CPU. For higher-end systems, the stack must take into account the high variability in memory access time, as well as system resources that offload some functions to specialized hardware.

Fire Engine focuses on these network performance metrics [10]:

• Network throughput
• Connection set up/tear down
• First-byte latency
• Connection and CPU scalability
• Efficient resource usage.

E. *Vertical Perimeters*

The Solaris 10 FireEngine project introduces the abstraction of a vertical perimeter, which is composed of a new kernel data structure, the squeue_t (serialization queue type), and a worker thread owned by the squeue_t , which is bound to a CPU. Vertical perimeters or squeues by themselves provide packet serialization and mutual exclusion for the data structures. FireEngine uses a per-CPU perimeter, which is a single instance per connection. For each CPU instance the packet is queued for processing, and a pointer to the connection structure is stored inside the packet. The thread entering squeue may either process the packet immediately, or queue it for later processing. The choice depends on the squeue's entry point and its state. Immediate processing is possible only when no other thread has entered the same squeue. A connection instance is assigned to a single squeue_t so it is processed only within the vertical perimeter. As a squeue_t is processed by a single thread at a time, all data structures used to process a given connection from within the perimeter can be accessed without additional locking. This improves both CPU and thread context data locality of access for the connection metadata, packet metadata, and packet payload data, improving overall network performance.

This approach also allows:

• The removal of per-device driver worker thread schemes, which are often problematic in solving system-wide resource issues.
• Additional strategic algorithms to be implemented to best handle a given network interface, based on network interface throughput and system throughput (such as fanning out per-connection packet processing to a group (of CPUs).

II. RELATED WORKS

Many techniques have been developed to exploit parallelism. OpenMP [5] is a tool where looped tasks can be partitioned into multiple independent tasks automatically. Affine partition [6], [7], is another method that can find the optimal partition, which maximizes the parallelism with the minimum synchronization. However, few works have been done to address on how to allocate resources to threads on multicore systems.

André C. Neto, Filippo Sartori, [1] proposed MARTe is a framework built over a multiplatform library that allows the execution of the same code in different operating systems. Drawback is latency. François Trahay, Élisabeth Brunet Alexander Denis [2], the author present a thread safety while processing by locking mechanism. Drawback is of Deadlock. Fengguang Song, Shirley Moore [3], This paper proposes an analytical model to estimate the cost of running an affinity-based thread schedule on multicore systems. Tang-Hsun Tu, Chih-Wen Hsueh [4], Decomposing of thread by User Dispatching Mechanism (UDispatch) that provides controllability in user space to improve application performance. Ana Sonia Leon [8], proposed Chip Multi-Threading (CMT) architecture which maximizes overall throughput performance for commercial workloads. Drawback is low performance. Sunay Tripathi, Nicolas Droux, Thirumalai Srinivasan, presented a paper that is a new architecture which addresses Quality of Service (QoS) by creating unique flows for applications and Services.

III. FIRE ENGINE ARCHITECTURE

The Solaris FireEngine networking performance improvement project adheres to these design principles [10]:

• *Data locality*: Ensures that a connection is always processed by the same CPU whenever possible
• *CPU modelling*: Efficient use of available CPUs and interrupt/worker thread model. Allows use of multiple CPUs for protocol processing
• *Code path locality*: Improves performance and efficiency of TCP/IP interactions
• *TCP/IP interaction:* Switches from a message passing-based interface to a function call-based interface.

Because of the large number and dependent nature of changes required to achieve FireEngine goals, the development program is split into three phases:

Solaris 10 Fire Engine phase 1 [10]: Fundamental infrastructure implemented and a large performance boost realized. Application and STREAMS module developers see no changes other than better performance and scalability.

Solaris 10U and SX Fire Engine Phase 2 [10]: Feature scalability, offloading, and the new Event List framework implemented.

1. Solaris 10 Fire Engine phase 1 architecture

A. IP Classifier-Based Fan Out

When the Solaris IP receives a packet from a NIC, it classifies the packet and determines the connection structure and vertical perimeter instance that will process that packet. New incoming connections are assigned to the vertical perimeter instance attached to the interrupted CPU. Or, to avoid saturating an individual CPU, a fan-out across all CPUs is performed. A NIC always sends a packet to IP in interrupt context, so IP can optimize between interrupt and noninterrupt processing, avoiding CPU saturation by a fast NIC.

There are multiple advantages with this approach:

• The NIC does minimal work, and complexity is hidden from independent NIC manufacturers.
• IP can decide whether the packet needs to be processed on the interrupted CPU or via a fan out across all CPUs.
Processing a packet on the interrupted CPU in interrupt context saves the context switch, compared to queuing
the packet and letting a worker thread process it.
• IP can also control the amount of work done by the interrupt without incurring extra cost. On low loads,
processing is done in interrupt context. With higher loads, IP dynamically changes between interrupt and
polling while employing interrupt and worker threads for the most efficient processing. In the case of a single
high bandwidth NIC (such as 10Gbps), IP also fans out the connection to multiple CPUs.
• If multiple CPUs are applied, the connection is bound to one of the available CPUs servicing the NIC. Worker threads, their management, and special fan-out schemes can be coupled to the vertical perimeter with little code complexity. Since these functions reside in IP, this architecture benefits all NICs.
• The DR issues arising from binding a worker thread to a CPU can be effectively handled in IP.

The Solaris 10 FireEngine project introduces the abstraction of a vertical perimeter, which is composed of a new kernel data structure, the **squeue_t** (serialization queue type), and a worker thread owned by the squeue_t, which is bound to a CPU. Vertical perimeters or squeues by themselves provide packet serialization and mutual exclusion for the data structures. FireEngine uses a per-CPU perimeter, which is a single instance per connection. For each CPU instance the packet is queued for processing, and a pointer to the connection structure is stored inside the packet.

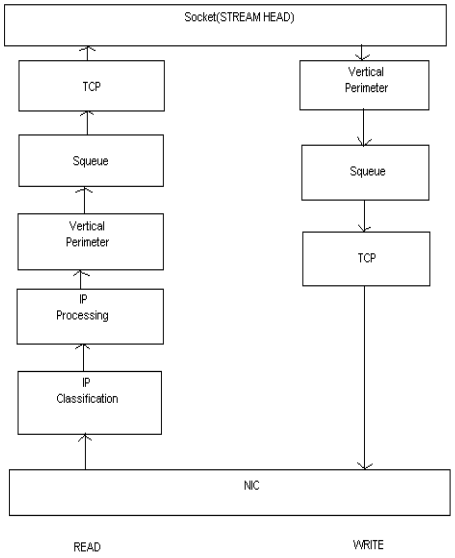


Figure 2: Packets flowing in TCP through vertical perimeter

tcp_input - All inbound data packets and control messages
tcp_output - All outbound data packets and control messages
tcp_close_output - On user close
tcp_timewait_output - timewait expiry
tcp_rsrv_input - Flow control relief on read side.
tcp_timer - All tcp timers

IV. ANALYSIS

1. Scheduling algorithm

Good cluster schedulers attempt to minimize job wait time while maximizing cluster utilization. The maximization of utilization and minimization of wait time are subject to the policy set by the scheduler administrator.
Types of scheduling [11]:

- *Long-term scheduling* : the decision to add to pool of processes to be executed

- *Mid-term scheduling* : the decision to add to the number of processes that are partially or fully in memory
- *Short-term scheduling :* decision as to which available process will be executed
- *I/O scheduling :* decision as to which process's pending request shall be handled by an available I/O device

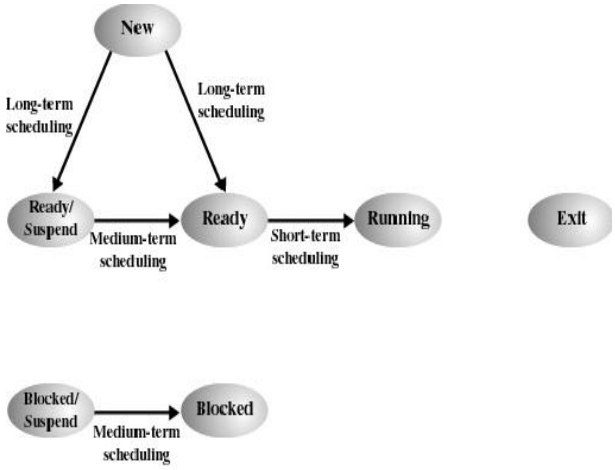


Figure 3 : Scheduling and process state transition

A. *Fair share scheduling*

The FS scheduler has two levels of scheduling: process and user. Process level scheduling same as in standard UNIX (priority and nice values act as bias to scheduler as it repositions processes in the run-queue). User level scheduler relationship can be seen in the following (simplified) pseudo-code. Whereas process-level scheduling still occurs 100 times a second, user-level scheduling adjustments (usage parameter) occur once every 4 seconds. Also, once second, process-level priority adjustments that were made in the previous second begin to be "forgotten". This is to avoid starving a process. FSS is all about making scheduling decisions based on process sets rather than on basis of individual processes.

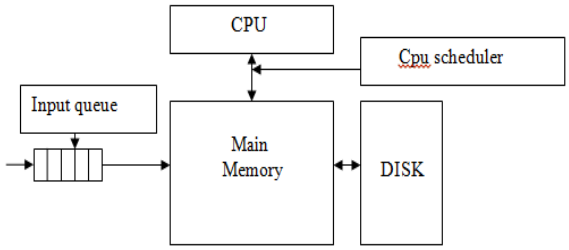


Figure 4: Scheduling process

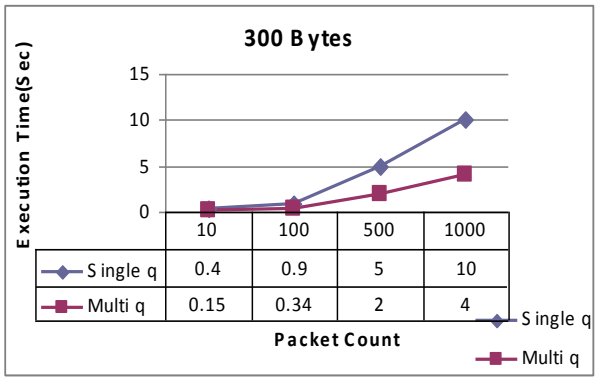
fs_interval (): Duration of each fairshare window.
fs_depth (): Number of fairshare windows factored into the current fairshare utilization calculation.
fs_decay (): Decay factor applied to weighting the contribution of each fairshare window in the past.

V.RESULTS

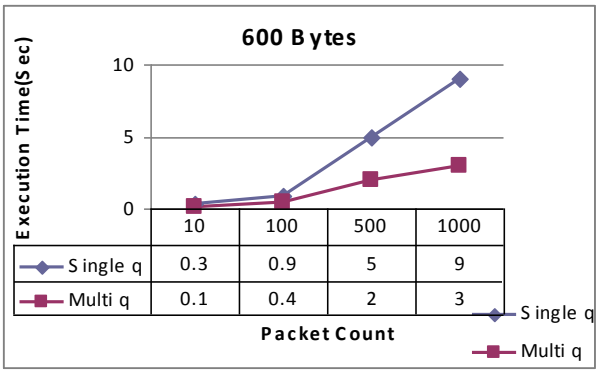
Comparison of Single Queue and Multiple Queues:

Here we are examine that execution time is less,using of multiple queues than single queue.comparison regarding to the paket count and length of bytes. Execution time in nanosecond but easy reference we converting into seconds.

Length of bytes 300:



Length of bytes 600:



VI.CONCLUSION

In this paper, we presented the vertical perimeter for enhancing the streaming application. In multicore environment, using multiple threads is a common useful approach to improve application performance. Nevertheless, even in many simple applications, the performance might

degrade when the number of threads increases. However, in our observation, the more significant effect is the dispatching of threads. In solaris 10 the fire engine has the concept of vertical perimeter. By using the vertical perimeter we can improve the streaming application and we mainly concentrated on the queue management, assigning the core, thread allocating and time profiling. For the allocation of process we need the scheduling algorithm. Here we used fair share scheduling (FSS). FSS will make scheduling process easier and efficient. So are going to analysis and implement the FSS in this paper.

VII. REFERENCES

[1] André C. Neto, Filippo Sartori, "MARTe: A Multiplatform Real-Time Framework", *IEEE transactions on nuclear science*, vol. 57, no. 2, april 2010.

[2] François Trahay, Élisabeth Brunet, "An analysis of the impact of multi-threading on communication performance", *IEEE transactions 2009*.

[3] Fengguang Song , Shirley Moore, "Analytical Modeling and Optimization for Affinity Based Thread Scheduling on Multicore Systems", *IEEE transactions 2009.*

[4] Tang-hsun Tu, Chih-wen hsueh, Rong-Guey Chang, "A portable and efficient User Dispatching Mechanism for multicore system", *IEEE International conference on Embedded and real-time computing systems and applications*, pp.427-436,2009

[5] L. Dagum and R. Menon, "OpenMP: An Industry-Standard API for Shared-Memory Programming," *IEEE Computational Science & Engineering*, vol. 5, no. 1, pp. 46–55, Jan. 1998.

[6] W. Lim, G. I. Cheong, and M. S. Lam, "An Affine Partitioning Algorithm to Maximize Parallelism and Minimize Communication," *Proceedings of the 13th international conference on Supercomputing*, pp. 228–237, 1999.

[7] W. Lim and M. S. Lam, "Maximizing Parallelism and Minimizing Synchronization with affine Transforms," *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 201–214, 1997.

[10] Simon Hauger, *A Novel Architecture for a High-Performance Network Processing Unit: Flexibility at Multiple Levels of Abstraction,* 978-1-4244- 5174-6/09/$26.00 ©2009 IEEE.

[8] Ana Sonia Leon*, Senior Member, "* A Power-Efficient High-Throughput 32-Thread SPARC Processor*", IEEE journal of solid-state circuits*, vol. 42, no. 1, jan. 2007.

[9] W.-Y. Cai and H.-B. Yang. *Cross-layer QoS optimization design for wireless sensor networks*. In *Wireless, Mobile and Sensor Networks, 2007.*

[11] Tong Li, Alvin R. Lebeck, "Spin Detection Hardware for Improved Management of Multithreaded Systems", *IEEE transactions on parallel and distributed systems*, vol. 17, no. 6, June 2006.

[12] Sunay Tripathi, "FireEngine - A New Networking Architecture for the Solaris Operating System" www.sun.com/bigadmin/content/networkperf/FireEngine_WP.pdf, Nov. 2004.

[13] Prof. Navneet Goyal, Department of Computer Science & Information System, BITS, Pilani,"operating system".