ISSN:0975-9646

ANALYSIS OF MESSAGE PASSING IN FOS: An Operating System for Clouds and Multicores

Akansha Gautam , Soma Sircar , Pankaj Gupta

School of Computer and Engineering,
Birla Institute of technology,Mesra

Abstract- Factored Operating System basically focuses on advancements in the operating system for clouds and cores with number of other cores. It is a microkernel based design that helps in making efficient use of all the available architectural resources and also keen to make the performance better. The main job of factored operating system is to divide the operating system services in group of distributed system processes that is used for passing the message instead of sharing memory. There are number of cores and each and every address space is particularly attached to them individually. This assignment of cores to entire system increases the abundance of cores at the data centre. Hence no application will rest on hold competing for the on-core cache resources. But for using these resources the message passing system be wasted in transfers of requests between cores.

INTRODUCTION

The two main reasons behind the development of factored operating system are-

1. difficulty in maintaining the scalability of an operating system.
2. each cloud fragment processing their isolated interfaces

To overcome these problems fos was created and it communicates through message passing. In message passing an uniform interface is provided making an efficient usage of available architectural resources.

Messages are passed through a mailbox which can even act as an endpoint for one way channel for communicating with other process. In this process the mailbox is build on a different machine then the one sending the message and the message is directed by a proxy server. Hence, we can conclude that a single message processing machine can act as a multiple machine in a cloud data centre.

There are basically two ways of message transportation in cache coherent multiprocessors:

1. Kernel messaging: kernels are used to transfers message to data structures by receiving process's address space
2. User messaging: works on the basis shared memory channel processing, hence takes more time as it proceses more number of cycles but provides better throughput and lower latency.

Main Contributions:

- Cache performance is properly studied
- Fast messaging is done
- Transportation of messages is done dynamically
- Messaging through multiple machines
- Analysis of message performance

DESIGN AND IMPLEMENTATION

There are few basic aims:

1. The number of cores per die has been steadily increasing to unprecedented levels
2. System as a service cloud providers use virtualization to multiple access to physical machine

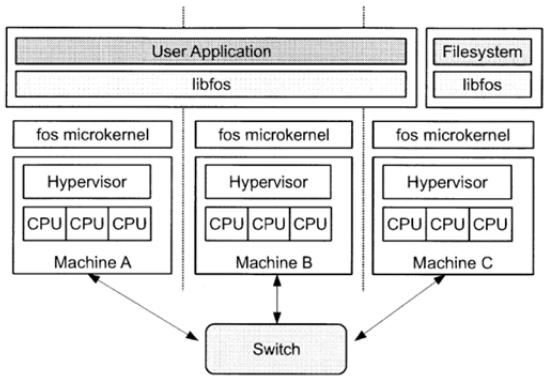

Figure 1: A view of fos running on three separate physical machines in a cloud data centre. In this, the application requires more cores than a single machine can provide. As a result, a single image, powered by libfos, maintains the illusion that the application is running on one large machine. The application relies on a file system service that is provided by the fos OS. Communication with the file system is seemless and uniform, regardless of the physical machine that initiated the request

The most importation issue with this single system message processing was memory cache coherence was not supported.

System Architecture

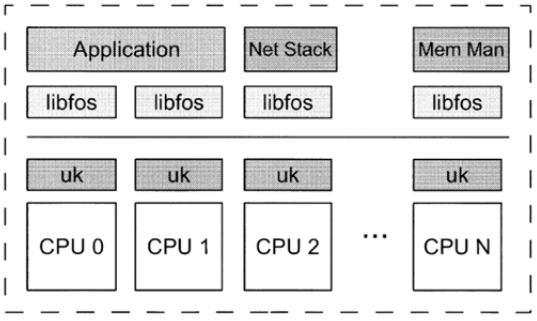

Figure 2. Each core runs a microkernel in privileged mode and libfos in unprivileged mode. Higher-level OS services

run in separate processes and are accessed through message passing, there are a network stack and a memory manager too

- ❖ Fleet Services

Designed for:
1. maximizing scalability
2. maintaining elasticity

Service:

- Naming: provides connections between the file names and their addresses
- File system: provides external support
- Networking: provides support in transportation.
- manages access to physical memory
- servicing page faults

MESSAGEPASSING OPTIMIZATIONS

Basically there were performance issues in case of micro kernels because they make context switching costs whenever an OS runs in a separate domain[1].Hence here we discuss few techniques for efficient message sharing

1. Kernel message passing optimization: L4 was one of the first operating systems that offered fast message passing performance through the kernel[7].This was accomplished through a variety of optimization techniques. First, L4 stores small messages directly in processor registers, avoiding the need to read such messages from memory. This optimization is not possible when messaging between different cores, because registers cannot be shared between cores directly. Second, the L4 microkernel sets up temporary virtual memory mappings (exposed only to the kernel) that allow it to copy the message payload directly into the receiver's address space.
2. User-level message passing is an alternate approach where the kernel only performs context switching (for cases that require multiplexing a single core) and memory setup. All other aspects of message passing are performed at user-level. This notion was pioneered in URPC [8]. URPC combines an efficient user-level message passing scheme with user-level thread scheduling. Messages are sent and received by the application directly over a shared memory mapping established before the first message is sent.

CACHE PERFORMANCE STUDY

A new open source memory trace generator and cache simulator called CachEMU was created to evaluate the inherent advantages and disadvantages of each design in terms of memory access costs. It is crucial for future operating systems to use memory resources efficiently in order to avoid higher latencies and to conserve off-chip bandwidth [6]. Otherwise, poor cache hit rates could cause cores to loose excessive cycles waiting for memory references. The results in this study indicate that a fos-style OS accesses memory more efficiently through improved cache locality. Thus, it has the potential to be a superior design for future multicores.

PROPERTIES OF A MESSAGING SYSTEM

A variety of decisions and tradeoffs can be made in a message passing system. fos's
message passing system adheres to the following properties:
" All communication is connectionless. When stateful connections are required, they can be implemented at the application level.
* Several processes can enqueue messages to a single mailbox, but messages can only be dequeued by the specific process that registered the mailbox. This allows for both one-to-one and many-to-one communication patterns.
" Each mailbox has a fixed size buffer allocated to it. If the buffer is full, new messages are rejected until enough space becomes available.
* The receiver is guaranteed to receive messages in the same order that the sender enqueues them. However, the order of messages across multiple senders is unspecified.
" All messaging communication is asynchronous. In other words, a process can enqueue a new message before a previous message is dequeued.
A connectionless, many-to-one design was chosen over traditional channel based messaging (e.g. TCP/IP) for two reasons. First, it is expected to map more closely to hardware interfaces that support direct access to on-chip-network resources, resulting in a more raw and high-performance interface. Second, the reduction in messaging state could make it easier to support live process migration features in the future.

CONCLUSION

Hence, we conclude that there are two ways for passing message in factored operating system...In kernel based message passing the message possess high latency but low cost of setup infact it is completely opposite in User based message passing. The other point of discussion was the analysis of performance of operating system on single core and multiple cores. It also focuses on proper usage of cache and make it available.

BIBLIOGRAPHY

[1] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schupbach, and Akhilesh Singhania.The inultikernel: a new OS architecture for scalable inulticore systems.In SOSP '09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pages 29-44, 2009.

[2] Nathan (Nathan Zachary) Beckmann. Distributed naming in a factored operating system. Master's thesis, Massachusetts Institute of Technology. Dept. Of Electrical Engineering and Computer Science., 2010.

[3] Silas Boyd-Wickizer, Austin T. Clements, Yandong Mao, Aleksey Pesterev, M. Frans Kaashoek, Robert Morris, and Nickolai Zeldovich. An Analysis of Linux Scalability to Many Cores. In OSDI 2012: Proceedings of the 9th USENIX conference on Operating Systems Design and Implementation.

[4] David Wentzlaff, III Gruenwald, Charles, Nathan Beckmann, Adam Belay, Harshad IKasture, Kevin Modzelewski, Lamia Youseff, Jason E. Miller, and Anant Agarwal. Fleets: Scalable services in a factored operating system. Technical Report MIT-CSAIL-TR-2011-012, MIT CSAIL, March 2011.

[5] Donald Yeung, John Kubiatowicz, and Anant Agarwal. Multigrain shared memory.ACM Trans. Comput. Syst., 18:154-196, May 2000.
[6] Doug Burger, James R. Goodman, and Alain Kigi. Memory bandwidth limitations of future microprocessors. In Proceedings of the 23rd annual international symposium on Computer architecture, ISCA '96, pages 78-89, New York, NY, USA, 1996. ACM.
[7] Brian N. Bershad, Thomas E. Anderson, Edward D. Lazowska, and Henry M. Levy. User-level interprocess communication for shared memory multiprocessors. ACM Trans. Comput. Syst., 9:175-198, May 1991.
[8] Jochen Liedtke. Improving ipc by kernel design. In Proceedings of the fourteenth ACM symposium on Operating systems principles, SOSP '93, pages 175-188, New York, NY, USA, 1993. ACM.