View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by KITopen

First published in: Poster/WiP session of the EurosSys'09, Nuremberg, Germany, April 2009

NUMA-Aware User-Level Memory Management for Microkernel-Based Operating Systems

Philipp Kupferschmied, Jan Stoess, Frank Bellosa

System Architecture Group
University of Karlsruhe

In the recent past there has been a growing interest from research and practitioners in operating system (OS) design with a minimal kernel base. While minimal (or micro) kernel based systems are a long-standing idea from a standpoint of OS research, such systems are now successfully employed in real-world scenarios, typically either as true microkernel systems or as hypervisors [5, 7]. At the same time, tremendous advances in multi-processing and multi-core technology have lead to computer platforms where the typical memory demands are so high that memory bandwidth is becoming a crucial bottleneck. The answer to that problem is a non-uniform memory architecture (NUMA), where processors or subsets of processors are connected to their own portion of main memory via a dedicated bus, thus alleviating overall load on each of the buses. NUMA architectures typically imply that the local portion of memory can be accessed fast, whereas remote accesses are significantly slower (as an example, an AMD Opteron NUMA platform in our lab shows a remote-to-local read memory access ratio of about 1.4). From a software standpoint, data locality and placement are thus of paramout importance in such systems. Ideally, most of a processor's working set should remain in its local memory. Common techniques to work toward that goal are data replication and migration.

In this work, we explore the requirements for a microkernel-based, NUMA-aware operating system. On the one hand, designing a system with regard to NUMA architectures requires to improve the locality of all data items where possible. On the other, the goal of maximum flexibility for a microkernel requires to keep the changes to the microkernel itself to the minimum. We thus strive for an OS architecture which allows for flexible memory management at user-level, but also improves the locality of kernel data structures without requiring policies within the kernel.

Our approach stays in contrast to virtual machine systems such as Disco [2], since it enables explicit application-controlled NUMA memory management rather than to encapsulate applications in virtual machines and to hide memory-management issues from them. It is also different to the Tornado [4] and K42 [1] approaches in that it restricts the kernel to providing simple and low-level primitives such as per-node address spaces, rather than using a sophisticated clustered object system, which allows multiple-component objects to appear like a single object.

Our core kernel design principle is that the kernel should provide only low-level and localized memory-management abstractions, leaving most of the NUMA address space construction to user-level. Particularly, the kernel offers support for *local* address space construction only: An address space always belongs to the NUMA node on which it is created, and the kernel allocates all address space management information on that node. This leads to more efficiency and better predictability of the base address space construction primitives. However, an address space can contain memory mappings both to local and to remote memory. Applications can thus construct cross-node NUMA address spaces by hand, that is, by creating logically coherent per-node address spaces with identical virtual-to-physical mappings, either to node-local or to remote memory. Synchronization of address spaces is left up to the applications; the kernel

EVA-STAR (Elektronisches Volltextarchiv – Scientific Articles Repository)
http://digbib.ubka.uni-karlsruhe.de/volltexte/1000026891

offers a cross-node communication primitive to allow signaling and synchronization on address space updates. Node-local regions (virtual addresses that are mapped to different physical addresses on each node, a useful property e.g. for transparent code replication) can easily be constructed by not synchronizing all mappings between related address spaces. When a thread migrates from one node to another, it is also migrated into another address space. The bookkeeping of which address spaces belong to the same application is performed at user-level.

For evaluation, we have built a NUMA-aware OS prototype based on the L4 microkernel [6]. L4 offers address spaces as a first class abstraction. They are populated by user-level pagers, which can map pages of memory into an address space when a pagefault occurs, and, later on, unmap them asynchronously if needed. The kernel represents address spaces by means of page tables; the mapping dependencies are represented by means of a mapping database (MDB) that keeps track of how physical frames are mapped to address spaces. We enhanced L4's in-kernel representation of address spaces by assigning a *home node* to each address space, set to the NUMA node, on which the address space is created. The kernel allocates memory for page tables and corresponding MDB entries on that home node. When a user-level application is parallely active on multiple nodes, we use one L4 address space on each node instead of using a single, node-spanning address space. By setting the home node appropriately, this solution ensures that the corresponding kernel data is allocated on the NUMA node, on which the address space is active. We propose to use a per-node pagefault handler, each of which is responsible for handling pagefaults that occur on its node. For all mappings that shall be global, that is, equally visible in all per-node address spaces, synchronization is performed between these pagers, and entirely at user-level: From the viewpoint of the kernel, the per-node address spaces are completely unrelated. This eliminates the need for an additional synchronization primitive within the kernel, the choice of which would depend on hardware characteristics such as the remote-to-local latency ratio [3]. The user-level pagers can implement arbitrary synchronization mechanisms, based on shared memory, IPC, or combinations of both, and tailored towards the hardware and application characteristics.

Synchronization between per-node address spaces is performed lazily, by means of pagefaults. Consequently, n pagefaults will occur until a global mapping is established on n nodes. However, if parallel threads work mainly on local data, the actual number of pagefaults is comparable to using a single address space only.
A pager can only unmap mappings it has established. If a pager on one node wants to globally unmap a page, it must notify all other pagers to do so as well, making the unmap-operation more expensive. Further investigations must show if this overhead is acceptable in realistic scenarios, and if there are more efficient solutions.

References

[1] J. Appavoo et al. *Enabling scalable performance for general purpose workloads on shared memory multiprocessors.* Technical report, IBM Research, 2003.

[2] E. Bugnion et al. *Disco: Running commodity operating systems on scalable multiprocessors. ACM Transactions on Computer Systems*, 15(4), 1997.

[3] E. M. Chaves et al. *Kernel-kernel communication in a shared-memory multiprocessor. Concurrency: Practice and Experience*, 5(3), 1993.

[4] B. Gamsa et al. *Tornado: Maximizing locality and concurrency in a shared memory multiprocessor operating system.* In *Proceedings of the third Symposium on Operating Systems Design and Implementation.* 1999.

[5] G. Heiser. *Hypervisors for consumer electronics.* In *Proceedings of the 6th IEEE Consumer Communications and Networking Conference.* 2009.

[6] J. Liedtke. *On microkernel construction.* In *Proceedings of the 15th ACM Symposium on Operating System Principles.* 1995.

[7] VMWare. *Virtualization overview.* `http://www.vmware.com/pdf/virtualization.pdf`.

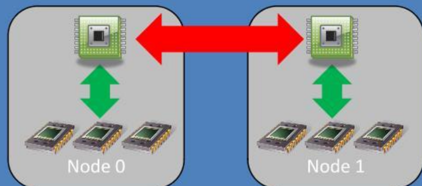
NUMA-Aware User-Level Memory Management for Microkernel-Based Operating Systems

Philipp Kupferschmied, Jan Stoess, Frank Bellosa

Problem Overview

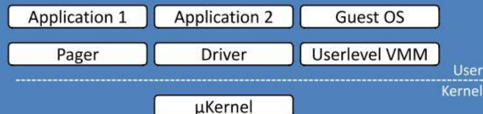
NUMA Systems

- Improved scalability compared to SMPs
- Expensive accesses to remote memory



Minimal Kernels

- Minimal set of abstractions & mechanisms
- No in-kernel policies
- OS services running at user-level



The NUMA-µK-Problem

How to exploit the NUMAness of the hardware while preserving a minimal, flexible, and policy-free microkernel?

Goals

Improved data locality

- Both for kernel and user data

Flexible UL memory management

- Including node-local data

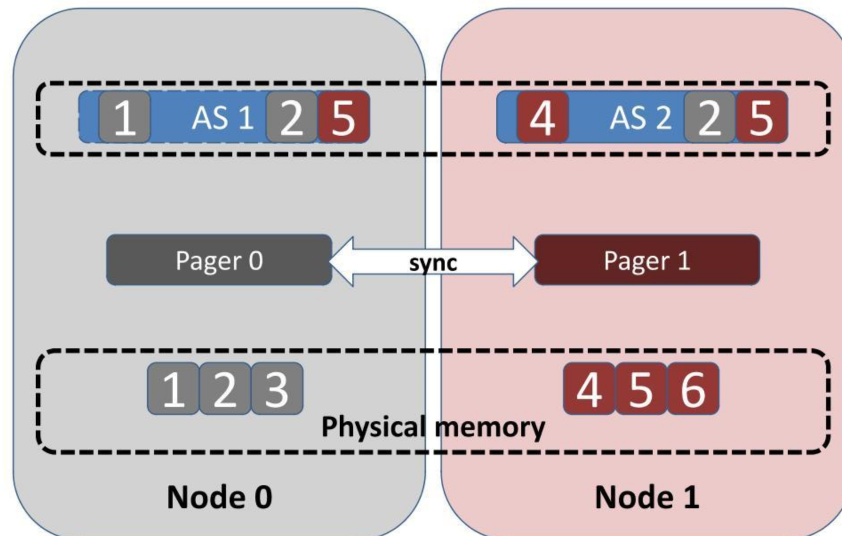
Minimal changes to the microkernel

- Avoid API-changes

Proposed Solution

For applications that span across multiple nodes, a different address space is used on each node. There is also a separate pager per node, which is responsible for handling pagefaults on that node. Synchronization between pagers is required to establish global mappings (i.e., mappings that are visible in all per-node address spaces).

Each address space is assigned a home node, set to that node on which the address space is created. The kernel allocates memory for address space management information (page tables and mapping database entries) on that home node, thus improving the locality of kernel data.



Benefits

Node-local AS management information

- Page tables and MDB entries are allocated on home-node of address space

Synchronization at user-level

- Per-node address spaces are unrelated from the viewpoint of the kernel (avoids in-kernel sync)

Different virt-to-phys mappings per node

- Allows for e.g., code replication and UTCB migration

Per-node access information

- Helpful for dynamic replication/migration

Drawbacks

Increased number of pagefaults

- Depends on the degree of sharing between threads on different nodes

More complex unmap

- Either by means of an IPC-based protocol or by the root pager

Outlook

Optimizations

How can we make synchronization between pagers more efficient? Can we improve the performance of unmap operations?

Evaluation

How does our solution compare to using one address space only? What is the impact on realistic workloads?

University of Karlsruhe, System Architecture Group