Local and Remote Memory: Memory in a NUMA System

by
Christoph Lameter
christoph@lameter.com
Revision: April 25th, 2006

A draft of a corresponding paper may be found at
<http://ftp.kernel.org/pub/linux/kernel/people/christoph/pmig/numamemory.pdf>

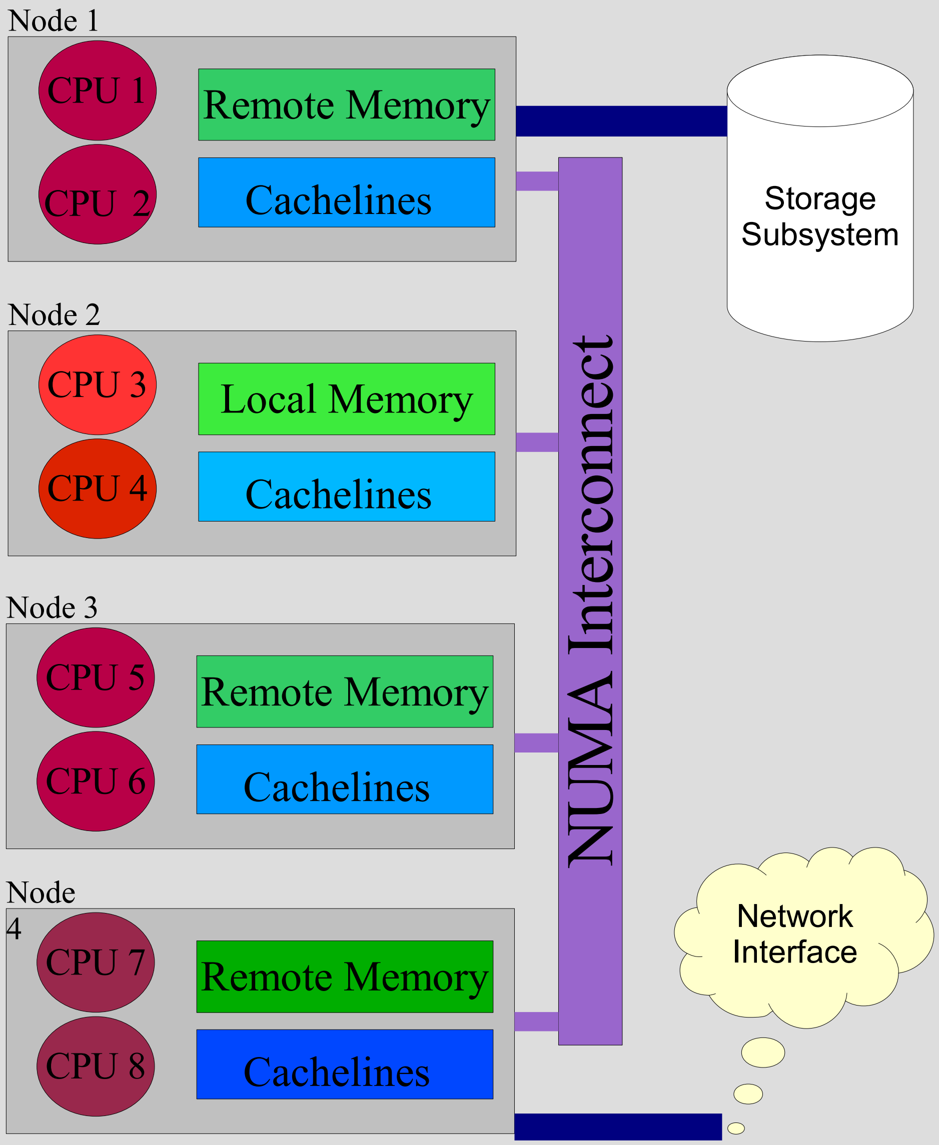
Memory seems to be so simple. If you need it then just get some and use it. That works just fine on most Linux architectures. On a NUMA system the distance of the memory to the executing process matters. Performance can sink dramatically if memory references too frequently are to memory on remote nodes. Local memory is special to the execution context because it has minimal latency and optimal bandwidth characteristics. Under NUMA the operating system must figure out how to assign memory to a process in an optimal way so that the process can utilitze all memory resources and run with the highest performance.

Introduction

- A toy Linux/NUMA System
- Linux Memory Management
- Linux NUMA Memory Management
- Efficient placement of memory
- Memory Reclamation
- Memory Migration
- Memory Policy
- Cpusets
- Potential future NUMA work

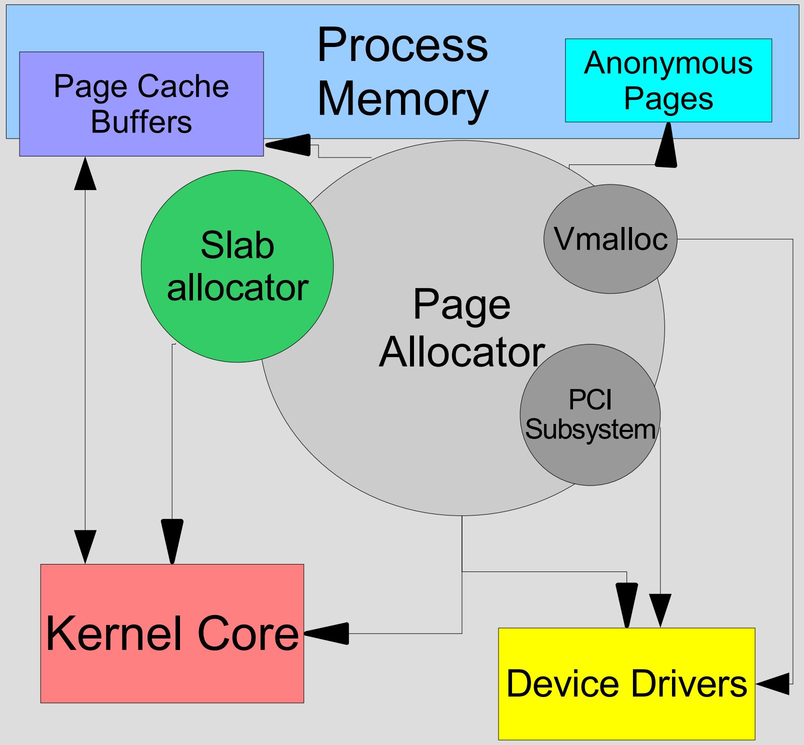
A sample NUMA System

- UP, SMP, NUMA
- Nodes
- Interconnect
- Numa distance
  - Off node
- SLIT tables
  - Local Distance
  - Remote Distance
- Node Local
- Device Local
- Placement: Where to allocate from



Linux Memory Management

- Memory in Pages
- Page allocator
- Anonymous vs. File backed memory
- Slab allocator
- Device allocator
- Page Cache
- read() / write() vs. mmapped I/O.

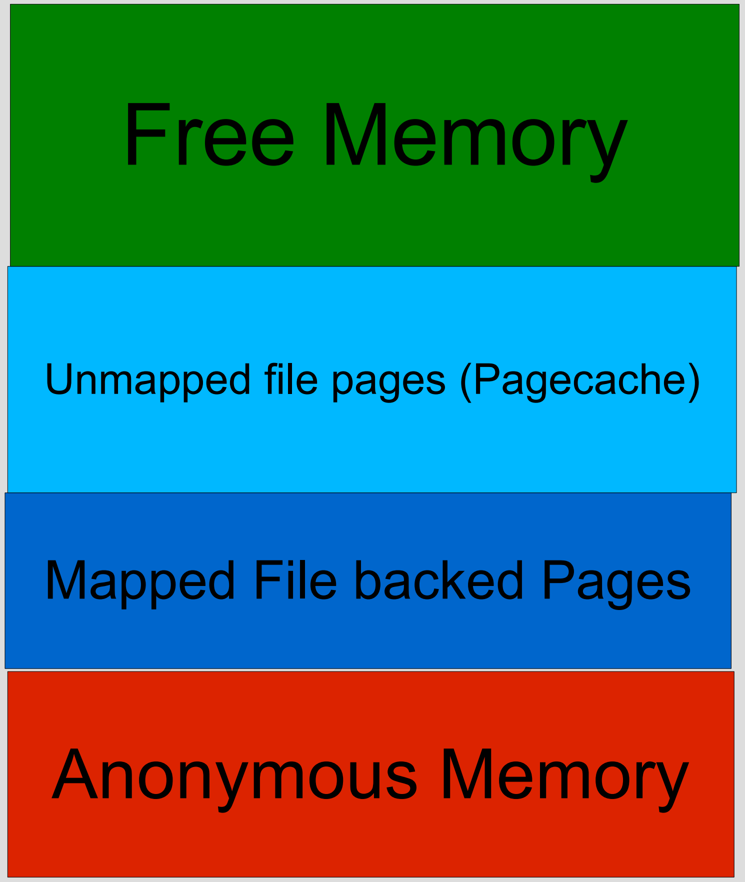


UMA Linux Memory Allocators

- Page allocator:
  alloc_pages(flags, order)
- Slab allocator:
  kmalloc(size, flags)
  kmem_cache_alloc(cache, flags)
- Virtual kernel memory:
  vmalloc(size)
- Device memory:
  dma_alloc_coherent(device, size, &addr, flags)

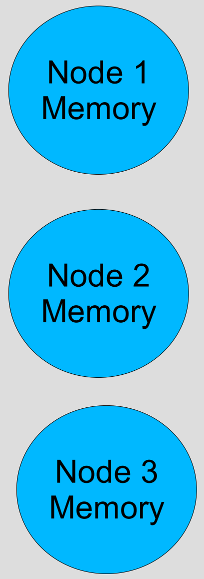
UMA Memory Reclaim

- Anonymous memory freed when process terminates
- Mapped file backed pages become unmapped but are not freed. So unmapped file pages accumulate.
- If memory runs low the swapper begins reclaim of memory.
- Light reclaim just frees unmapped file pages.
- If memory stays tight then memory may be unmapped which will allow the freeing of mapped file backed pages and the swapping out of anonymous pages.



NUMA Memory Management

- Memory management per node
- Memory state and possibilities of allocation
- Traversal of the zonelist (or nodelist)
- Process location vs. memory allocation
- Scheduler interactions
- Predicting memory use?
- Memory load balancing
- Support to shift the memory load



New NUMA Memory Allocators

- New semantics for UMA allocators and additional functions.
- Page Allocator:
  ```
  alloc_pages_node(node, flags, order)
  ```
- Slab Allocator:
  ```
  kmalloc_node(size, flags, node)
  kmem_cache_alloc(size, flags, node)
  ```
- Virtual Kernel Memory:
  ```
  vmalloc_node(size, node)
  ```
- dma_alloc_coherent(device, size, &addr, flags)

Efficient Allocation

- Minimal Latency = Node Local
- Only possible if enough memory and processors on one node (SMP like).
- Efficient allocation are mostly possible for small Unix like processes
- For device drivers stay on the device node
- Resource limitations on NUMA interlink
- Node saturation with remote requests.
- Scheduler interference
- Need to have the ability to direct allocations

Complications when placing Memory

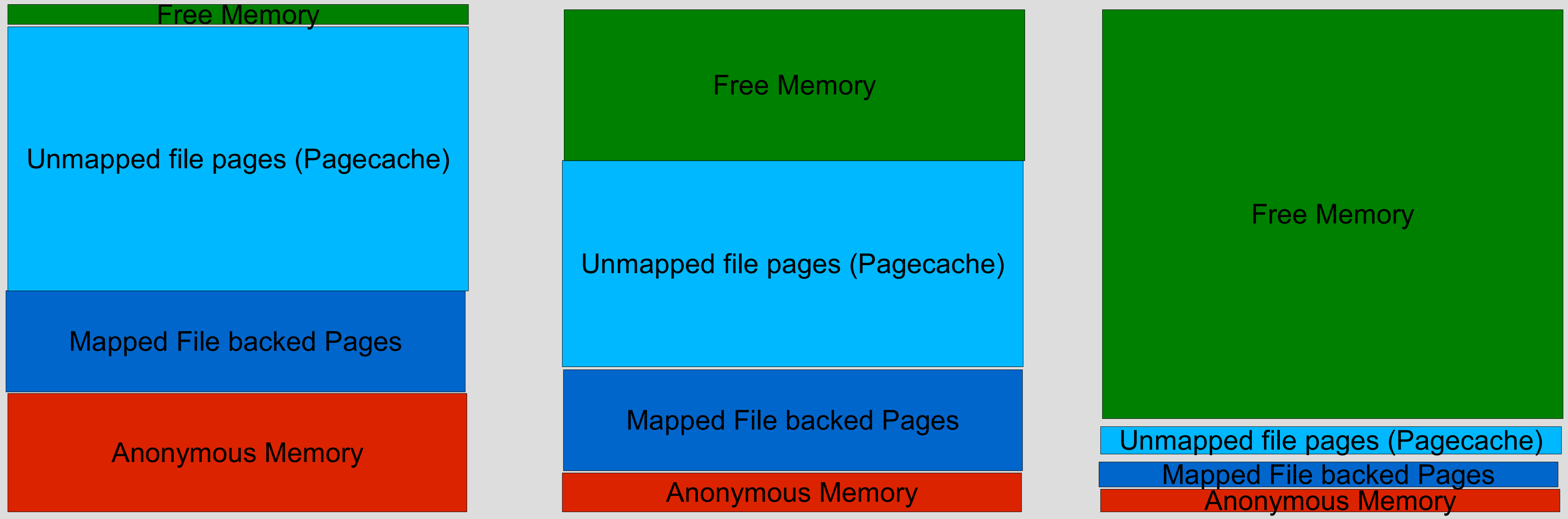
- Problems develop when applications require the resources of multiple nodes
- Memory Categorization for allocation
  - Thread specific -> Node local
  - Shared read/write -> Spreading
  - Shared read/only -> Replication
- Need to balance memory between partici-pating nodes.
- Page cache overflowing single nodes
- Swapper not active for overflowing nodes.

Memory Balancing

- Strategies on how to place data
- First place most critical structures then move on to secondary data.
- Limited availability of memory may lead to more compromises for less important data.
- Balancing the number of NUMA interlink requests per node.
- Consider device locality to I/O bound processes.

NUMA standard reclaim

- Reclaim is a global reclaim.
- Reclaim only active if total free memory becomes low
- No local reclaim results in lots of node allocations
- Off node allocations occur until all the nodes are out of memory. On a large NUMA system this can seem to never occur.
- The distances to the remote nodes become excessive.
- One technique used in the past is to manually drop the pagecache.



Memory Reclamation (zone reclaim)

- Swapper (global reclamation) not regularly running under NUMA
- Need for local reclamation. This has been implemented as zone_reclaim.
- Local reclaim from inactive page cache before going off line.
- Off node timeout (default 30 seconds)
- Advanced modes: Write and swap during zone reclaim.
- Included in 2.6.16.

Zone reclaim details

- /proc/sys/vm/zone_reclaim_mode
  - If != 0 then local memory reclaim is on.
  - Switched on based on largest NUMA distance encountered during boot. If the NUMA distance is more than 15 then its on.
- /proc/sys/vm/zone_reclaim_interval
  - Time to allow off node accesses if local reclaim failed to free local memory.
  - Heuristic necessary since we have no easy way to figure out how many pages are reclaimable. Once we have better NUMA VM statistics this could go.

Memory Migration

- Arose from the need to rebalance the load of running applications.
- Interface designed for system administrators and batch scheduler.
- 3 APIs: Policy based, commandline driven and cpuset controlled.
- Page is migrated by removing references.
- 2.6.16 has basic migration that still requires swap to be defined. Later version will be independent of swap and preserve more references.

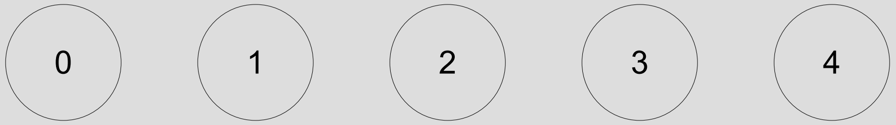
Memory Migration Methods

- cat /proc/*pid*/numa_maps
  - Shows current usage of pages on various nodes
- migratepages *pid from-nodes to-nodes*
  - Commandline tool to move pages of a process between nodes while preserving relative locations of pages within each node set.
- sys_migrate_pages
  - A new system call
- MPOL_MF_MOVE(_ALL) flag for sys_mbind().
- /dev/cpuset/*cpuset*/memory_migrate flag

Memory Policies

- Ability to direct allocations.
- Dynamically set allocation policies for a task.
- Set allocation policies for address ranges.
- Issues of file backed pages.

- MPOL_DEFAULT
- MPOL_PREFER
- MPOL_BIND
- MPOL_INTERLEAVE



Cpusets

- Convenient means of boxing in allocation and Processor use for large applications.
- Ability to control swap and various other things in a cpusets.
- Automigrate processes when nodes of a cpuset are changed or when moving a task between cpusets.
- Cpuset wide interleaving policy (memory spreading) for slab and page allocation for pathological cases of apps not prepared for NUMA.

Future Plans

- Preserve mappings during migration.
- More File system support for migration.
- Unify allocation constraints and policies
- Kernel text replication
- Replication of shared read only pages.
- Unify allocation constraints and policies to provide something that can provide the functionality of both cpusets and memory policies.
- Better Memory Balancing through per node statistics
- Remove configurations for memory reclaim
- Remove zone reclaim?

Conclusion

- Gradual Maturation of Linux / NUMA support
- NUMA is going mainstream with multi core configurations for x64, PowerPC and Sparc.
- Seems that x86 will be a major NUMA platform.