

Dynamic Large Pages

Dave Hansen IBM Linux Technology Center



Why Large Pages?

- Fewer objects to manage
- Fit more objects in CPU caches
- Per-page operations become cheaper
- Per-page structures become "smaller"
- Any cache miss is increasingly expensive
- >They are "special" on Linux



"Old" Workloads

Performance is critical Grew out of HPC/DB space Large Memory Footprint Page-level handling (faults, etc...) Willing to work around usability >mlock() tolerance "Custom" Applications



State of the Art

Interfaces: fs / SHM / libhugetlbfs Faulting replaces preallocation COW support for private at fork() Reservations Quota Support NUMA Policy Awareness Lumpy Reclaim Gigantic Pages



Linux VM Support

>NUMA

- Delayed allocation- better locality
- Round-robin pool population
- Deterministic COW support
 - Needed for MAP_PRIVATE support
 - MAP_PRIVATE needed for transparent replacement
- Lumpy Reclaim



Admin Interfaces

Display

>/proc/meminfo (incomplete)

>/sys/kernel/mm/hugepages

Configuration

hugepages=

- >/proc/sys/vm/nr_hugepages (r/w)
 - Not 100% dependable

kernelcore=

hugeadm – wrapper for all of these



Multiple HW Sizes

>ppc64: <u>4k</u>, <u>64k</u>, 16M, 16GB* >x86: 4k, 2M/4M, 1GB* ia64: everything >parisc: 4M ≻s390: 4k, 1M >sh: 64k, 256k, 1M, 4M, 64M, 512M >sparc: 64k, 512k, 4M

> * Gigantic Page size Base page size option



Gigantic Pages

>amd64: 1GB >powerpc: 16GB Early allocation is required before power on – ppc boot-time – x86 Separate pools from regular page allocation and other huge pages



Multi-size support

- Compile time selection?
- Gigantic mean no one-size-fits-all approach can possibly work
- >sysfs interfaces
- >enumerate/allocate
- Permit multiple mounts
- Separate allocation pools



Virtualization - KVM

Large memory use mlock() Custom app, willing to modify Performance concerns... TLB miss 5x cost, with new h/w Perfect huge page application!





Fragmentation Locked into memory – no reclaim Hardware must be dedicated >Separate, discrete interfaces Permissions >Amplification of bad NUMA placement decisions >Architecture TLB weakness



Candidate Users

- Large, contiguous memory users
- Poor temporal or spatial locality
- Bottlenecks on fault speed
- Pagetable size overhead
 - Large shared mapping
- Pagetable cache footprint



Application Work

- Using SHM? Add SHM_HUGETLB
- libhugetlbfs
 - >Drop-in replacement for malloc()/shmget()
 - >Works for complex apps like firefox!
 - Link normally or use LD_PRELOAD
 - Executables in huge pages
 - >Administraton with hugeadm



Future Work

- User Stacks
- Transparent promotion/demotion
- Continuing improvements in page reclamation
- Power management / Memory Hotplug
- libhugetlbfs

>documentation/usability

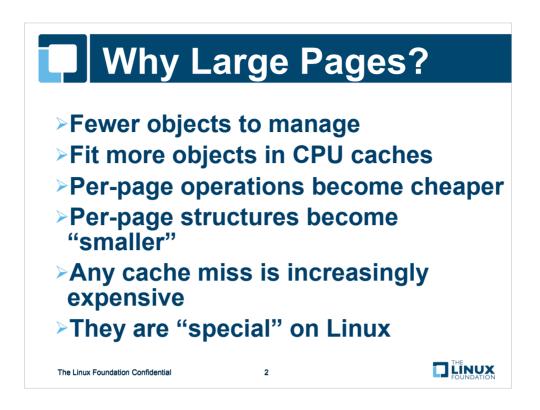


Further Reading

- Cost of Pagetable lookups in virtual machines:
- http://www.amd64.org/fileadmin/user_upload/pub/p26-bhargava.pdf
- http://sourceforge.net/projects/libhugetlbfs/
- http://www.ibm.com/developerworks/wikis/display/LinuxP/libhugetlbfs+FAQs



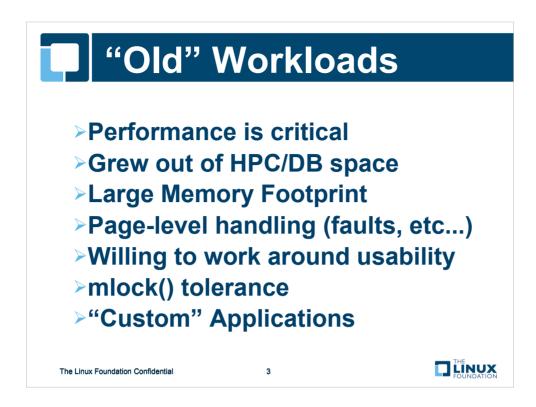




It costs the same number of cpu cycles more or less to do a large page minor fault or a small page one. But, the benefits of a large page fault are much higher.

smaller in terms of percentage. A fixed N-byte object becomes relatively much smaller when the M-byte page it represents gets larger

'expensive' in terms of performance. CPUs are bottlenecked on memory bandwidth and caches are continuing to increase in their importance.



There are classic workloads that have used large pages not necessarily the ones where they best fit



Interfaces: fs / SHM / libhugetlbfsFaulting replaces preallocation

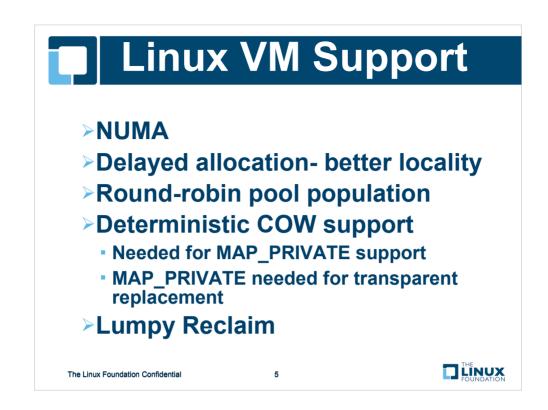
- >COW support for private at fork()
- Reservations
- >Quota Support

NUMA Policy Awareness

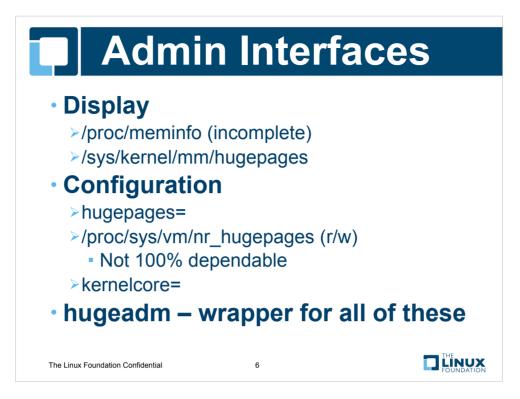
- >Lumpy Reclaim
- Gigantic Pages

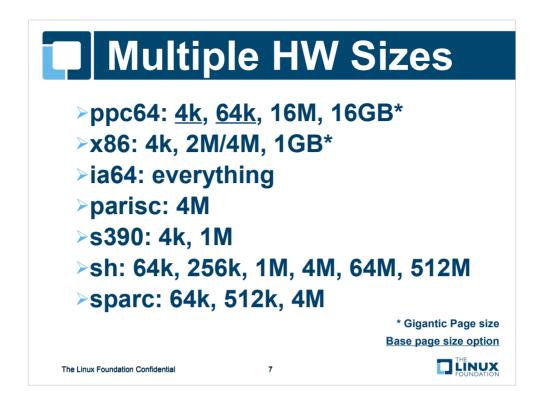
The Linux Foundation Confidential

4

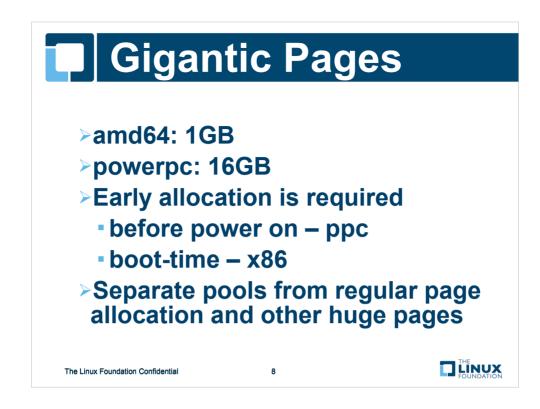


COW usage used to give random app behavior. Now we can at least guarantee that parents will keep their huge pages and children have an opportunity to to get their own copies, too.





just an indicator of why we need hstates so badly



just an indicator of why we need hstates so badly

Multi-size support

Compile time selection?
Gigantic mean no one-size-fits-all approach can possibly work
sysfs interfaces
enumerate/allocate
Permit multiple mounts
Separate allocation pools

The Linux Foundation Confidential

9

Virtualization - KVM

Large memory use

≻mlock()

Custom app, willing to modify

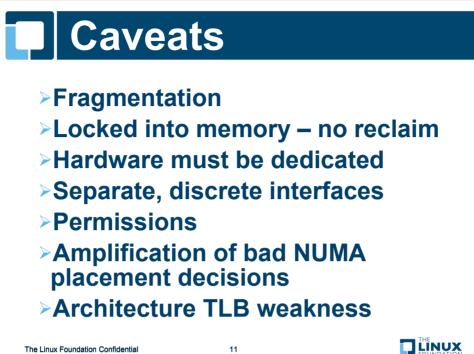
>Performance concerns...

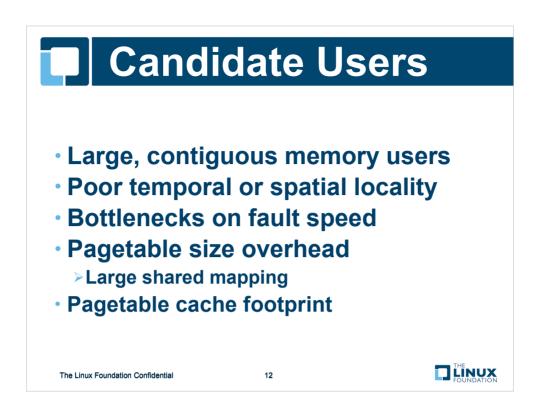
>TLB miss 5x cost, with new h/w

Perfect huge page application!

The Linux Foundation Confidential

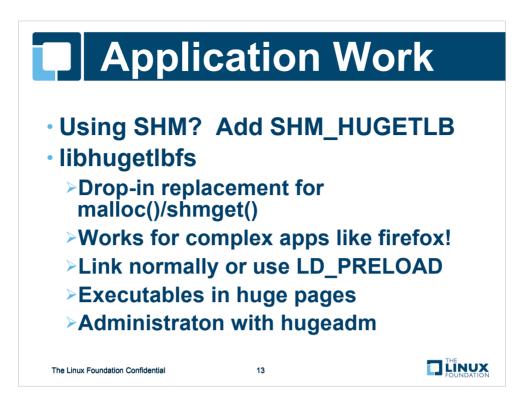
10

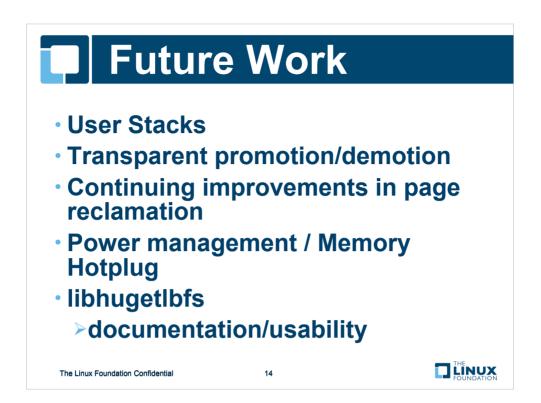




Temporal locality

- Tendency to re-reference memory
- Sparse accesses imply low temporal locality
- Use-once (e.g. STREAM) has low locality
- Tree elimination solves have higher locality
- Spacial locality
- Tendency to reference nearby memory
- Random access low locality
- Cache blocking, higher spacial locality





Further Reading		
Cost of Pagetable lookups	in virtual machines:	:
http://www.amd64.org/filea		oub/p26-bhargava.pdf
http://sourceforge.net/proje	•	
http://www.ibm.com/develc	operworks/wikis/dis	play/LinuxP/libhugetlbfs+FAQs
The Linux Foundation Confidential	15	