Jericho: Achieving Scalability through Optimal Data Placement on Multicore Systems

- Stelios Mavridis, Yannis Sfakianakis, Anastasios Papagiannis, Manolis Marazakis, and Angelos Bilas

Institute of Computer Science (ICS)
Foundation for Research and Technology – Hellas (FORTH)
Outline of this talk

- Does thread/data affinity matter for I/O performance on NUMA multicore servers?
- Jericho Design
- Evaluation
- Conclusions
Thread/Data affinity matters ... a lot!

GB/sec with Different Thread Placements (memory, read)

- Average
- Best Placement
- Worst Placement

out-of-the-box performance level

[ fio, independent files, sequential reads, 4KB requests ]
Impact of thread/data affinity [alt. view: IOPS/core]

IOPS (1000s) per core with Different Thread Placements (memory, read)

- Average
- Best Placement
- Worst Placement

IOPS/core deteriorates with more cores!
% Local vs Remote memory references

% Local Memory References with Different I/O Stacks

- Baseline 2.6.32, Default Placement
- Baseline 2.6.32, Proper Placement

0.3-1.2% with Worst Placement!
What workloads are affected?

- Application threads accessing independent file sets
- Batch-style processing of independent files
  - e.g. Map-Reduce on Hadoop
- Virtual machines
  - running from (private) device images
Contributions

- A simple locality management scheme for proper affinity with negligible overhead
- Page-cache capable of controlling page placement
  - Cache organized as slices, mapped to NUMA memory nodes
- NUMA-aware filesystem to enforce task placement
  - Based on the location of file buffers
Why a custom I/O stack?

- Local access by a simple and fast check (upon I/O access)
- ... via pre-arranged division of buffers
- ... via placement rule implemented in the filesystem
- No need to track the location of individual buffers
Jericho: Custom I/O stack in the Linux kernel

[User-space]

Application Threads

[Kernel-space]

Inode Cache
Dentry Cache
Page Cache

XFS
Extents Allocator
Journal

VFS

Buffers/Caching

Filesystem

I/O Scheduler

To I/O Devices

[Native Stack]

Common I/O Path

Inode Cache
Dentry Cache
JeriCache

Rq Queue
Pipeline
Hash Table
Evictor

Slices

JeriFS

Task Affinity Control

Container Allocator
Journal

Slices

[Jericho Stack]
11.9 GBytes/sec per memory channel
23.8 GBytes/sec per memory controller
47.6 GBytes/sec per socket

1-2 hops on system interconnect for remote memory accesses
... over paths capable of <= 12.8 GBytes/sec

Estimation of local vs remote memory accesses with the likwid tool (via H/W performance counters)
Thread/Data Affinity: Local Access Case

No crossing of the system interconnect

Local Access

Threads

File Blocks
Thread/Data Affinity: Remote Access Case
Jericho := JeriCache + JeriFS

- JeriCache: Mapping of storage block ranges to NUMA nodes
  - Implement policies based on the actual data set of each application
  - Simpler than tracking I/O requests & I/O buffers in Linux kernel
  - **Slice**: a cache instance limited to memory of single NUMA node
  - Caches a specified range of blocks from underlying storage device

- JeriFS filesystem: Manage set of block ranges
  - Determines on the fly, for each I/O, location of issuing task & data
  - Space is divided in block ranges, each mapped to a JeriCache slice
  - Allows files to use specific locations on devices and NUMA nodes
    - + allocator (per-slice)
    - + atomic updates (one journal per-slice + cross-slice coordination)
Locality Management

- Exploit the fact that filesystem code runs in same context (albeit in kernel space) as the user
- Migrate user context without modifying the OS scheduler
- At every IO access, FS checks affinity of issuing thread
  - Migrate threads, not buffers
  - At remote buffer access, FS alters thread’s affinity
  - Run only on CPU core of NUMA node where buffer is placed
  - Force a context switch, effectively migrating the process
- When process resumes execution, I/O is guaranteed to only cause local memory references in JerichoCache
Design Highlights: JeriCache

Contexts:
- I/O issuer
- Pipeline thread (one per cache)
- Evictor (one per cache)

Inlining optimization:
No context switch “Issuer → Pipeline” needed in the case of a cache hit

Eviction proceeds concurrently with request processing (LRU approximation)
Design Highlights: JeriFS

Files are mapped to slices – inheritance from parent dir.

I/O buffers are mapped to NUMA nodes

Storage device blocks are assigned to cache instances
**fio, 4KB Random Reads**

### GB/sec with Different I/O Stacks (memory, read)

- **Jericho**
- **Baseline, 2.6.32**
- **Baseline, 3.13**

**Baseline: Plateau at ~20 GB/sec**

**Jericho scales (max: ~38GB/sec)**

---

Jericho (presentation at MSST'14)
fio, 4KB Random Reads [alt. view: IOPS/core]

IOPS (1000s) per core with Different I/O Stacks (memory, read)

- Red: Jericho
- Green: Baseline, 2.6.32
- Yellow: Baseline, 3.13

Jericho: ~ 150 kIOPS/core
fio, 4KB Random Reads [ alt. view: IOPS/node ]

IOPS (1000s) per NUMA node with Different I/O Stacks (memory, read)

- Jericho
- Baseline, 2.6.32
- Baseline, 3.13

Jericho: ~ 1.2M IOPS/Node (2.4M IOPS/socket)
21.3 Gbytes/sec per memory controller
(42.6 Gbytes/sec per socket)

1-2 hops on system interconnect for remote memory access
... over paths capable of <= 12.8 GBytes/sec

PCIe path to devices: 10.4 GBytes/s
... comparable to a system interconnect link
Is affinity an issue when we have device I/O?

Affinity is already an issue, and fast devices are becoming more prevalent...

**MB/sec with Different Thread Placements**

(device, read)

- Bad Placement
- Proper Placement
Conclusions

- Scalability limitations of the Linux kernel due to NUMA effects, with 24+ cores
- Thread/data affinity is a major issue for workloads accessing data from memory and fast storage devices
- Improvements achieved with Jericho I/O stack (64 cores):
  - 2.9x for seq. reads, 3.4x for seq. writes
  - (similar results for random IOPS)
Questions?

Manolis Marazakis
Institute of Computer Science, FORTH – Heraklion, Greece
E-mail: maraz@ics.forth.gr
Web: http://www.ics.forth.gr/carv

Acknowledgements

EuroServer
(EU FP7-ICT-610456)

HiPEAC3
(EU FP7-ICT-287759)

IOLanes
(EU FP7-ICT-248615)
# Testbed specifications

<table>
<thead>
<tr>
<th>Specification</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor socket count</td>
<td>4</td>
</tr>
<tr>
<td>Cores/processor socket</td>
<td>16</td>
</tr>
<tr>
<td>Motherboard</td>
<td>Tyan S8812</td>
</tr>
<tr>
<td>Processor type</td>
<td>AMD Opteron 6272 (2.1GHz)</td>
</tr>
</tbody>
</table>
| Processor core caches         | L1: 2x32KB (code - per 2 cores), 16KB (data)  
L2: 4x2MB (per 2 cores)       
L3: 2x8MB (per 4 cores)       |
| DRAM (DDR3, # DIMMs)          | Up to 16 (up to 512 GB, currently 256 GB) |
| Interconnect type             | HyperTransport 3.1 |
| Interconnect topology         | Point-to-Point, asymmetric |
| Storage Devices               | 16 Solid State Disks (Samsung 830 Series) in RAID0 configuration |
| Storage Controllers           | 2x LSI MegaRAID SAS 9265-8i controllers, each with 8 Solid State Disks attached |
Is affinity an issue when we have device I/O?

[alt. view: IOPS / core]

![IOPS per core with Different Thread Placements](chart)

- Red: Bad Placement
- Green: Proper Placement
Scaling with the number of NUMA nodes

IOPS (1000s) per NUMA node
(memory, read)

Jericho

# NUMA nodes (8 cores per node)

Jericho (presentation at MSST'14)
Degradation of Query Execution Time

% degradation (wrt 1 VM)

Performance interference between independent VMs

[ TPC-H/Q5, 7GB database (PostgreSQL), kvm, Linux 2.6.32 ]
Independent Virtual Machines (no device I/O)

Query Execution Time

- Baseline (2.6.32, xfs)
- Jericho

[ UNPUBLISHED RESULT ]

Interference reduction with Jericho:
- Isolated cache instance, per VM
- Dedicated path to storage, per VM

seconds

# VMs
NUMA effects will become even more pronounced