C1000K高性能服务器构建技术

余锋
http://yufeng.info
mryufeng@gmail.com
2010/10/16

C1000K面对的挑战

C10K问题：http://www.kegel.com/c10k.html 时间是2001年

现在是2010年，10年过去了，虽然软硬件技术也相应提高了，

挑战还在：

- 1M的tcp并发，即使每个链接按照16K内存算，需要至少24G内存。
- 1M的tcp链接中，有20%每秒活跃，那么200K每秒。
- 没有革命性的技术改进，算法和操作系统和库变化不大。
- 用户对服务响应时间和可靠性要求越来越高。

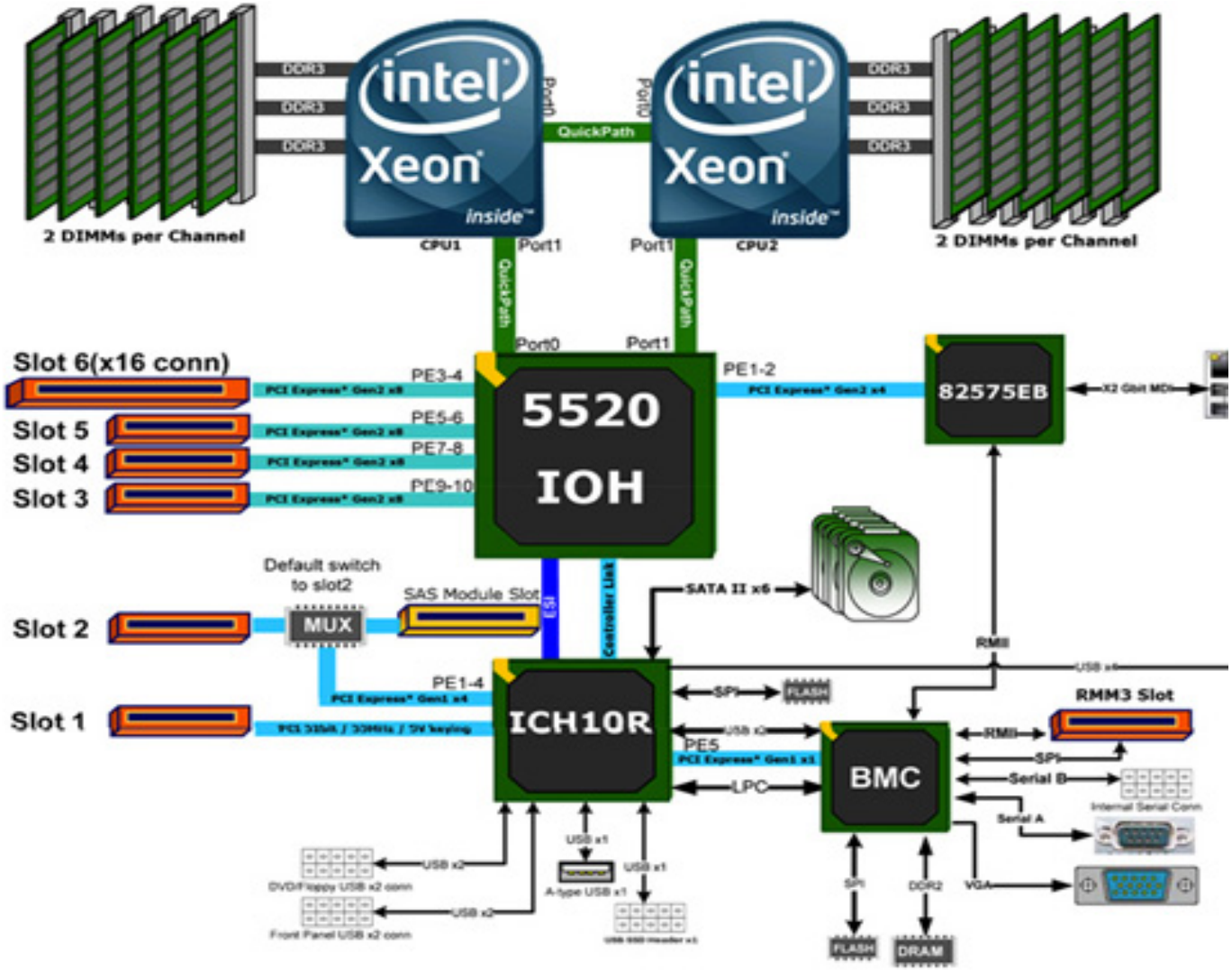
硬件约束：Dell R710， Intel E5520 *2， 24G内存， 640G SAS

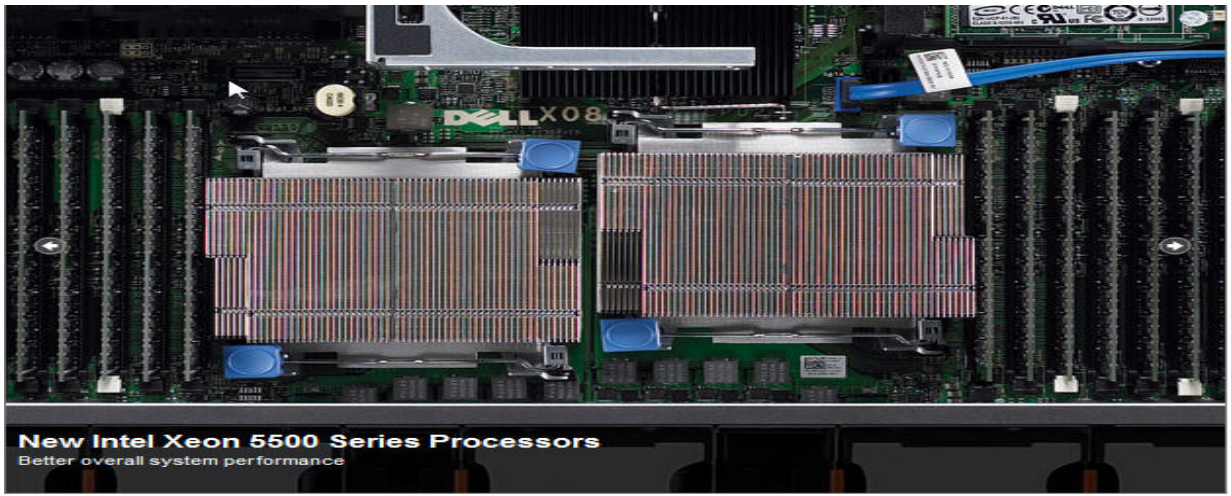
解決方案

TODO

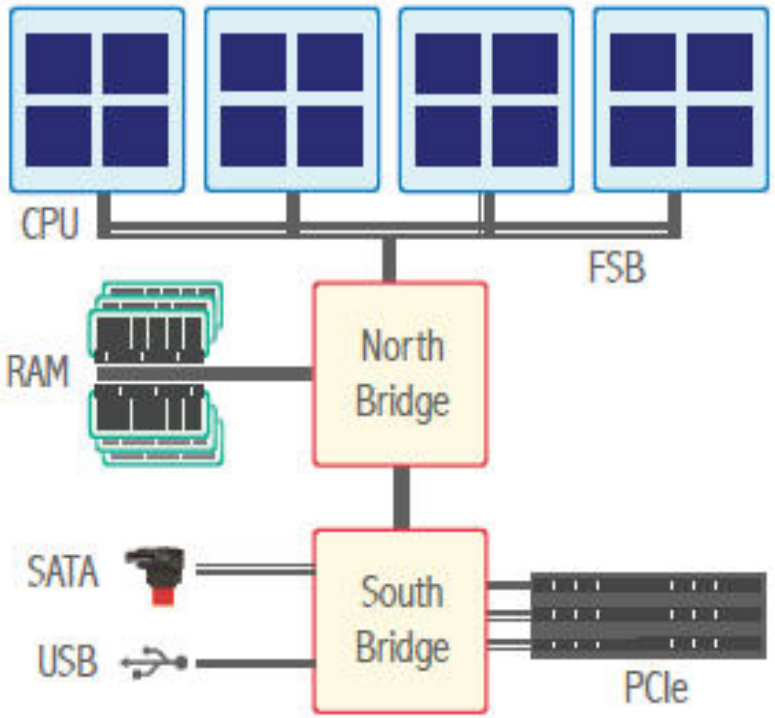
Agenda

- 硬件层面变化和思考
- 操作系统层面变化和思考
- 语言和库层面变化和思考
- Erlang平台层面变化和思考
- 调优工具

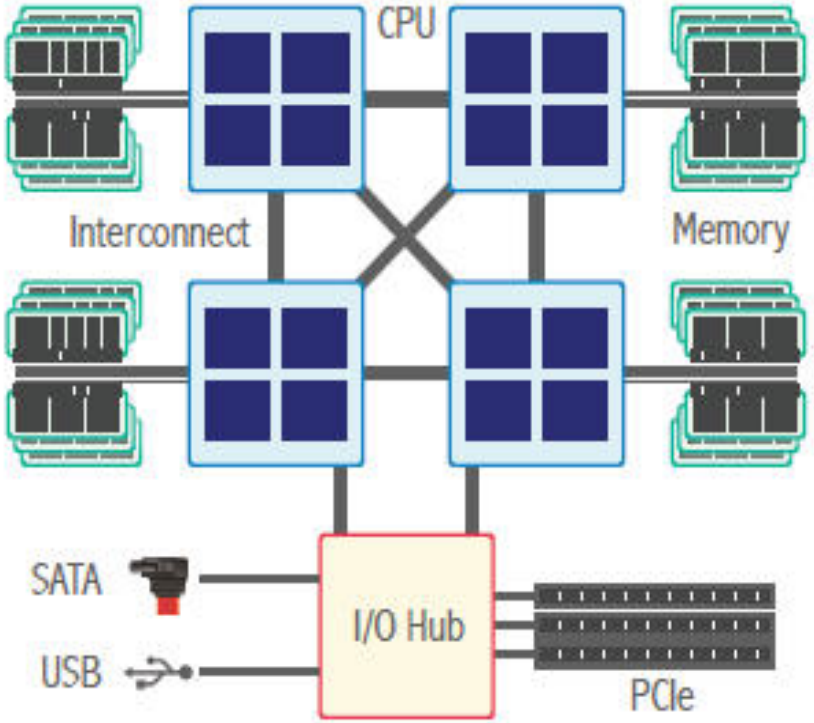

DDR3
DDR3
DDR3
intel
Xeon
inside
2 DIMMs per Channel
CPU1
Port0
Port1
QuickPath
CPU2
2 DIMMs per Channel
Slot 6(x16 conn)
Slot 5
Slot 4
Slot 3
Slot 2
Slot 1
PE3-4
PE5-6
PE7-8
PE9-10
5520
IOH
PE1-2
82575EB
Default switch
to slot2
MUX
SAS Module Slot
SATA II x6
ESI
Controller Link
RMII
PE1-4
ICH10R
SPI
FLASH
RMM3 Slot
PE5
BMC
LPC
Serial B
Serial A
USB x2
USB x1
A-type USB x1
DVD/Floppy USB x2 conn
Front Panel USB x2 conn
SPI
DDR2
VGA
FLASH
DRAM


DELL X08
New Intel Xeon 5500 Series Processors
Better overall system performance

Dell R710机器



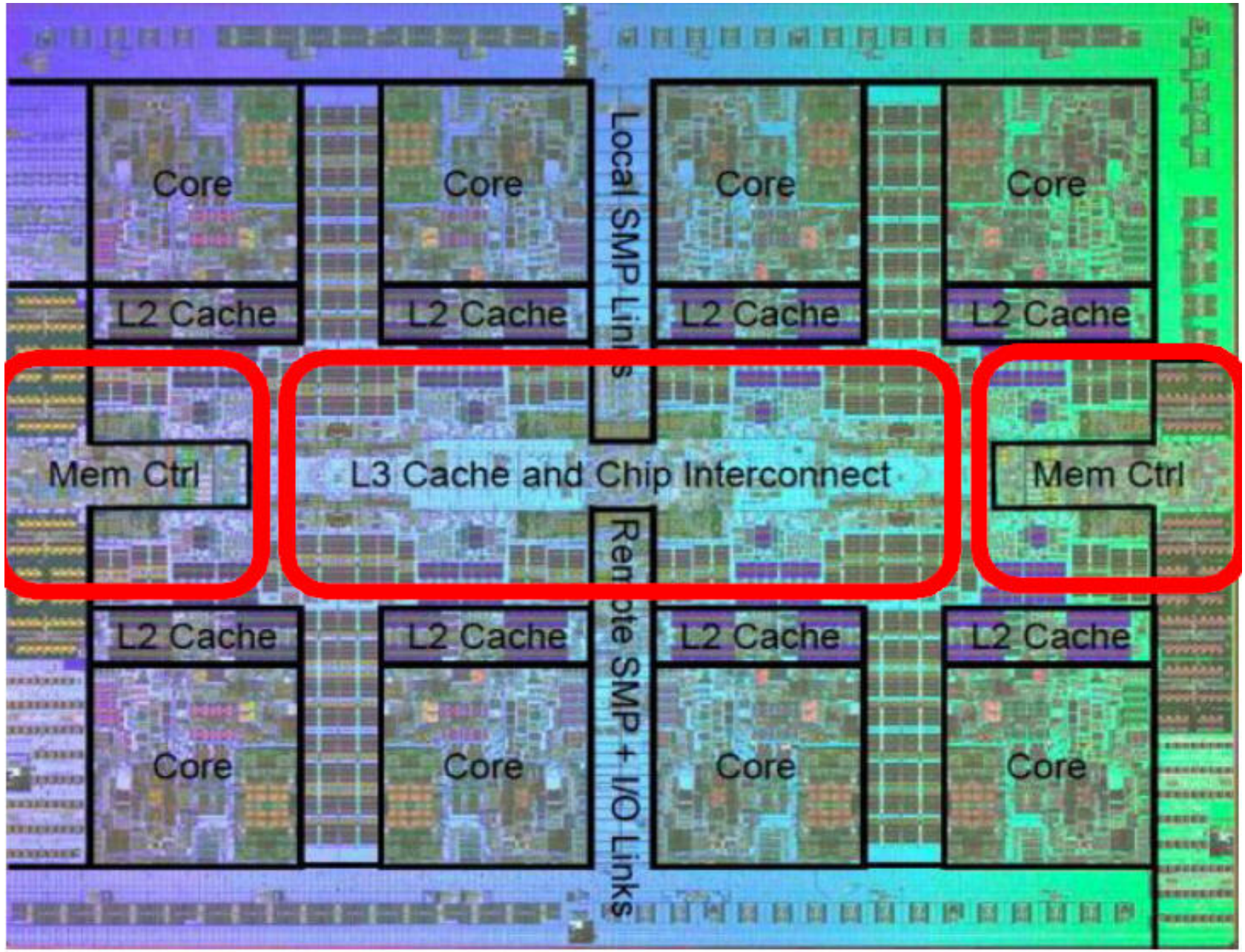
过去



现在

北桥慢慢成为过去！

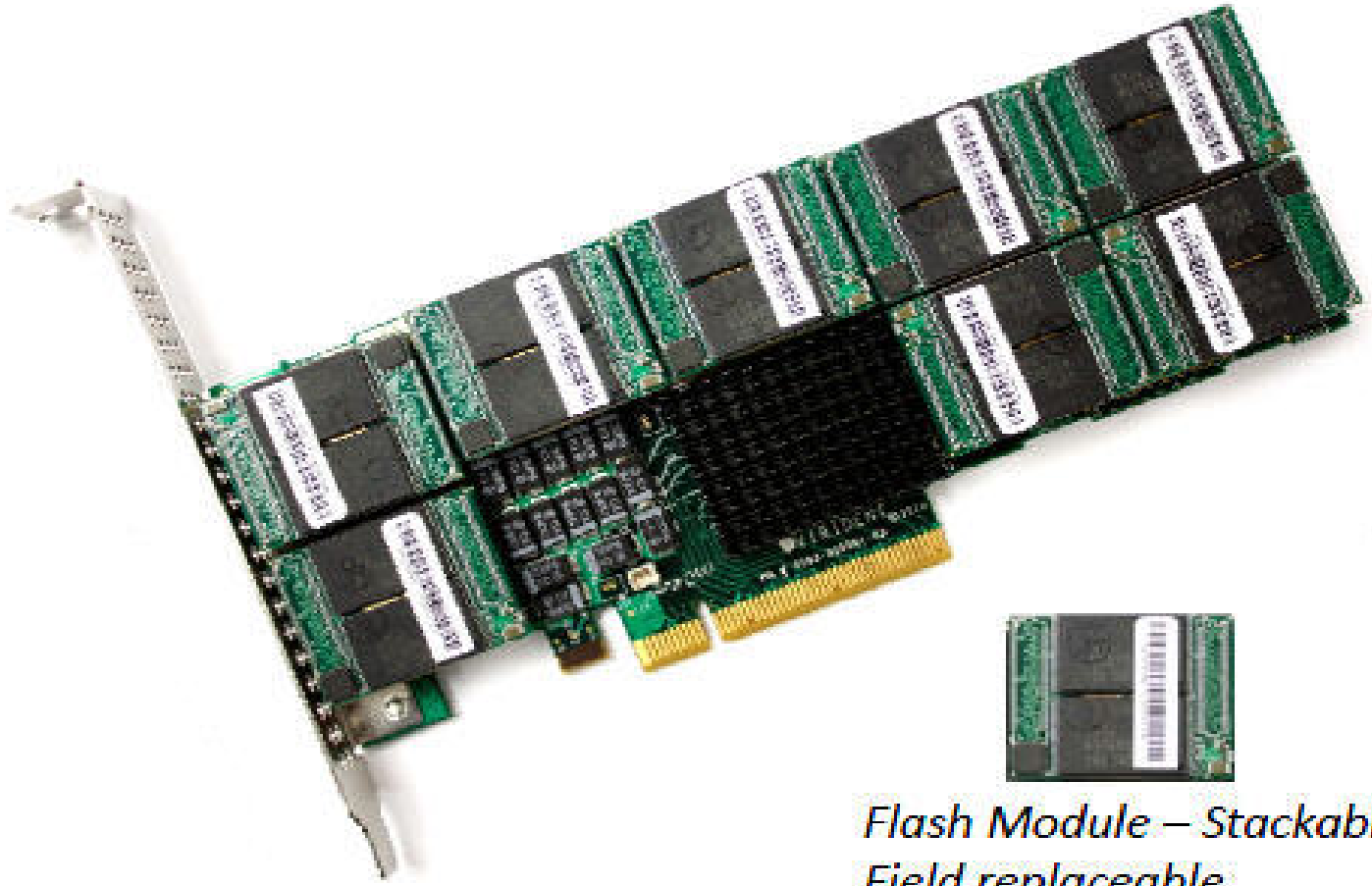
硬件体系巨大变化



Cache在CPU硬件上的版面，也充分说明了cache的重要性

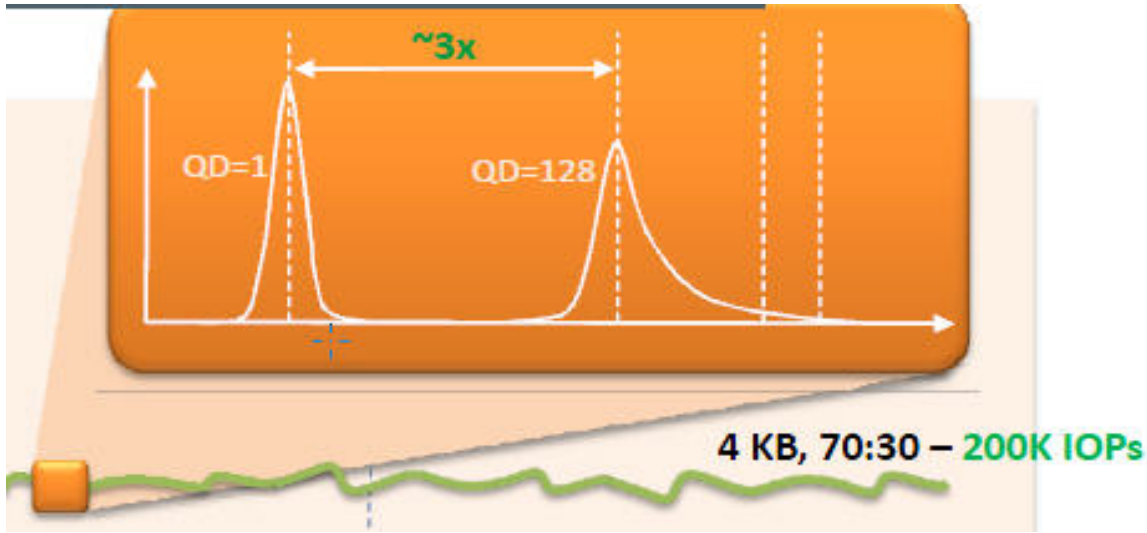

RISER 2
RISER 1
Integrated NICs
Four integrated NIC ports for more I/O bandwidth

内置四张网卡如何用？



Flash Module – Stackable,
Field replaceable

IOPS高达200K，带宽800M



Virident pci-e卡

小结

- 硬件变得和过去很不一样，性能越来越高。
- 硬件从CPU，内存，网卡都在试图scale， 我们要配合硬件的并行化趋势。
- 硬件在cache方面下了很多血本，提高数据的locality。
- 采用合适的硬件，比如说ssd盘代替sas盘。

Agenda

- 硬件层面变化和思考
- 操作系统层面变化和思考
- 语言和库层面变化和思考
- Erlang平台层面变化和思考
- 调优工具

```
Package 1 Cache and Thread details

Box Description:
Cache  is cache level designator
Size   is cache size
OScpu# is cpu # as seen by OS
Core   is core#[_thread# if > 1 thread/core] inside socket
AffMsk is AffinityMask(extended hex) for core and thread
CmbMsk is Combined AffinityMask(extended hex) for hw threads sha
       CmbMsk will differ from AffMsk if > 1 hw_thread/cache
Extended Hex replaces trailing zeroes with 'z#'
       where # is number of zeroes (so '8z5' is '0x800000')
      +-----------+-----------+-----------+-----------+
Cache |   L1D     |   L1D     |   L1D     |   L1D     |
Size  |   32K     |   32K     |   32K     |   32K     |
OScpu#|    1    9 |    3   11 |    5   13 |    7   15 |
Core  |c0_t0 c0_t1|c1_t0 c1_t1|c2_t0 c2_t1|c3_t0 c3_t1|
AffMsk|     2  200|     8  800|    20  2z3|    80  8z3|
CmbMsk|   202     |   808     |  2020     |  8080     |
      +-----------+-----------+-----------+-----------+

Cache |   L1I     |   L1I     |   L1I     |   L1I     |
Size  |   32K     |   32K     |   32K     |   32K     |
      +-----------+-----------+-----------+-----------+

Cache |   L2      |   L2      |   L2      |   L2      |
Size  | 256K      | 256K      | 256K      | 256K      |
      +-----------+-----------+-----------+-----------+

Cache |   L3                                          |
Size  |   8M                                          |
CmbMsk| aaaa                                          |
      +-----------------------------------------------+
```

```
Handle 0x110B, DMI type 17, 28 bytes
Memory Device
        Array Handle: 0x1000
        Error Information Handle: Not Provi
        Total Width: 72 bits
        Data Width: 64 bits
        Size: 2048 MB
        Form Factor: DIMM
        Set: 6
        Locator: DIMM_B3
        Bank Locator: Not Specified
        Type: <OUT OF SPEC>
        Type Detail: Synchronous
        Speed: 1066 MHz (0.9 ns)
        Manufacturer: 00CE04B380CE
        Serial Number: 86A5C153
        Asset Tag: 02094503
        Part Number: M393B5673EH1-CF8

Handle 0x110C, DMI type 17, 28 bytes
Memory Device
        Array Handle: 0x1000
        Error Information Handle: Not Provided
        Total Width: 72 bits
        Data Width: 64 bits
        Size: No Module Installed
        Form Factor: DIMM
        Set: 4
        Locator: DIMM_B4
        Bank Locator: Not Specified
        Type: <OUT OF SPEC>
        Type Detail: Synchronous
        Speed: Unknown
        Manufacturer:
        Serial Number:
        Asset Tag:
        Part Number:
```

```
[admin@my174 ~]$ lspci
00:00.0 Host bridge: Intel Corporation 5520 I/O Hub to ESI Port (rev 13)
00:01.0 PCI bridge: Intel Corporation 5520/5500/X58 I/O Hub PCI Express Root Port 1 (rev 13)
00:03.0 PCI bridge: Intel Corporation 5520/5500/X58 I/O Hub PCI Express Root Port 3 (rev 13)
00:04.0 PCI bridge: Intel Corporation 5520/X58 I/O Hub PCI Express Root Port 4 (rev 13)
00:05.0 PCI bridge: Intel Corporation 5520/X58 I/O Hub PCI Express Root Port 5 (rev 13)
00:06.0 PCI bridge: Intel Corporation 5520/X58 I/O Hub PCI Express Root Port 6 (rev 13)
00:07.0 PCI bridge: Intel Corporation 5520/5500/X58 I/O Hub PCI Express Root Port 7 (rev 13)
00:09.0 PCI bridge: Intel Corporation 5520/5500/X58 I/O Hub PCI Express Root Port 9 (rev 13)
00:14.0 PIC: Intel Corporation 5520/5500/X58 I/O Hub System Management Registers (rev 13)
00:14.1 PIC: Intel Corporation 5520/5500/X58 I/O Hub GPIO and Scratch Pad Registers (rev 13)
00:14.2 PIC: Intel Corporation 5520/5500/X58 I/O Hub Control Status and RAS Registers (rev 13)
00:1a.0 USB Controller: Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #4 (rev 02)
00:1a.
00:1a.
00:1d.
00:1d.
00:1d.
00:1e.
00:1f.
01:00.
01:00.
02:00.
02:00.
03:00.
07:00.
08:04.
08:05.
08:06.
0a:00.
0b:00.
0c:03.
[admin
```

```
                L M B E N C H  3 . 0   S U M M A R Y
                ------------------------------------
                (Alpha software, do not distribute)

Basic system parameters
----------------------------------------------------------------------------
Host                 OS Description                Mhz  tlb  cache  mem   scal
                                                       pages line   par   load
                                                             bytes
--------- ------------- ----------------------- ---- ----- ------ ------ ----
my174.cm4 Linux 2.6.18-        x86_64-linux-gnu 1593                       1

Processor, Processes - times in microseconds - smaller is better
----------------------------------------------------------------------------
Host                 OS  Mhz null null      open slct sig  sig  fork exec sh
                           call  I/O stat clos TCP  inst hndl proc proc proc
--------- ------------- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
my174.cm4 Linux 2.6.18- 1593 0.05 0.17 0.63 1.65 4.79 0.22 1.53 125. 564. 1677

Basic integer operations - times in nanoseconds - smaller is better
-------------------------------------------------------------------
Host                 OS  intgr intgr  intgr  intgr  intgr
                           bit   add    mul    div    mod
--------- ------------- ------ ------ ------ ------ ------
my174.cm4 Linux 2.6.18- 0.6400 0.3200 0.2200   15.1   14.5

Basic uint64 operations - times in nanoseconds - smaller is better
-------------------------------------------------------------------
Host                 OS int64  int64  int64  int64  int64
                        bit    add    mul    div    mod
--------- ------------- ------ ------ ------ ------ ------
my174.cm4 Linux 2.6.18-  0.630        0.2200   28.8   18.4

Basic float operations - times in nanoseconds - smaller is better
-------------------------------------------------------------------
Host                 OS  float  float  float  float
                         add    mul    div    bogo
--------- ------------- ------ ------ ------ ------
my174.cm4 Linux 2.6.18- 1.9100 1.7000 9.4400 8.8400

Basic double operations - times in nanoseconds - smaller is better
-------------------------------------------------------------------
Host                 OS  double double double double
                         add    mul    div    bogo
--------- ------------- ------ ------ ------ ------
```

调查系统

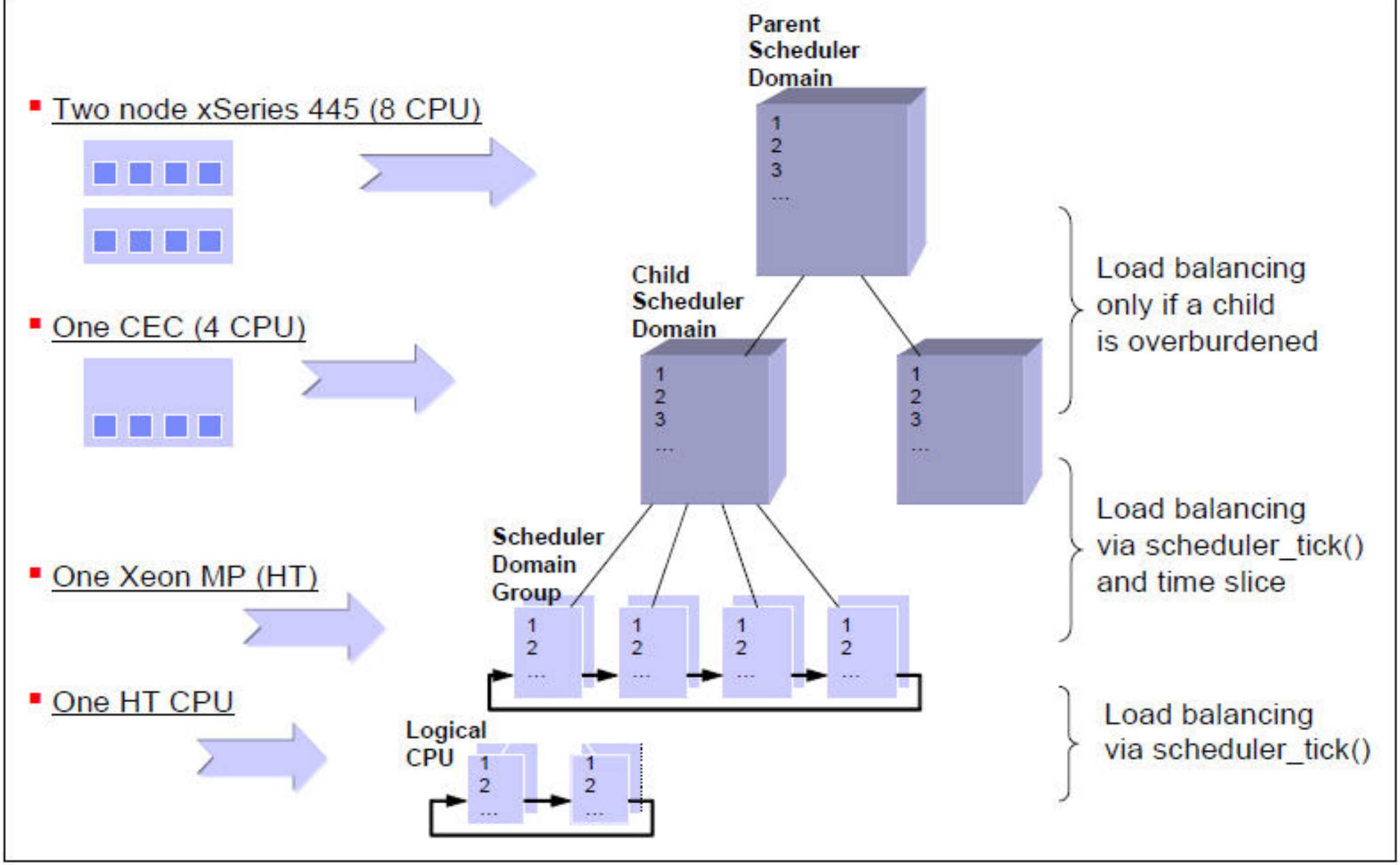


Figure 1-9 Architecture of the O(1) CPU scheduler on an 8-way NUMA based system with Hyper-Threading enabled

Numa架构下的调度器，CPU亲缘性

```
[admin@my174 ~]$ numactl --hardware
available: 2 nodes (0-1)
node 0 size: 12120 MB
node 0 free: 7543 MB
node 1 size: 12091 MB
node 1 free: 3920 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
[admin@my174 ~]$
```

```
[admin@my174 ~]$ numastat
                           node0           node1
numa_hit                689123506       718798292
numa_miss                25213368       109086988
numa_foreign            109086988        25213368
interleave_hit           33207777        26910439
local_node              659288361       688358204
other_node               55048513       139527076
[admin@my174 ~]$
```

```
[admin@my174 ~]$ cat /proc/`pgrep numademo`/numa_maps | perl numa-maps-summary.pl
N0        :      1633324 (  6.23 GB)
N1        :       988254 (  3.77 GB)
active    :      2621431 ( 10.00 GB)
anon      :      2621461 ( 10.00 GB)
dirty     :      2621461 ( 10.00 GB)
mapmax    :          115 (  0.00 GB)
mapped    :          119 (  0.00 GB)
[admin@my174 ~]$
```

google Tcmalloc numa aware 版本

- Numa不同的节点间访问代价不同。
- 不适当的设置，会导致有的节点的内存空闲，有的需要swap。
- libnuma改善内存分配的亲缘性，numactl 改变内存分配的策略。
- /proc/pid/numa_maps了解你的进程内存在节点间的分布情况。

Numa matters

Largepage

TLB miss的代价

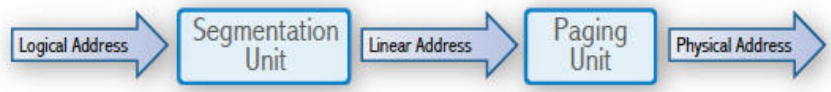
过去4K一页
现在通过HugeTLBfs实现
2M一页

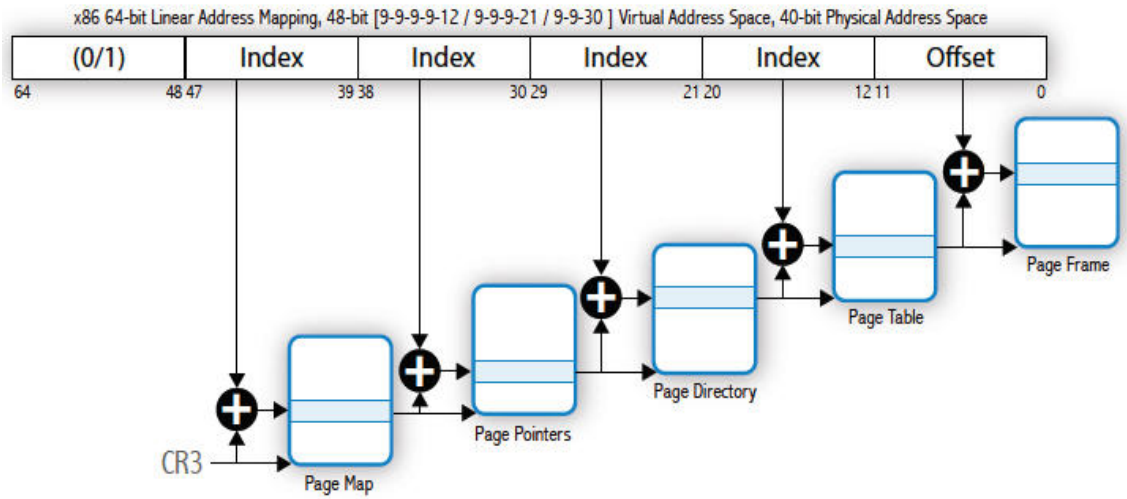
大大减少TLB miss

oprofile可以告诉我们tlb的miss率

```
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
Hugepagesize:       2048 kB
```

Virtual Address Translation





The Memory Wall

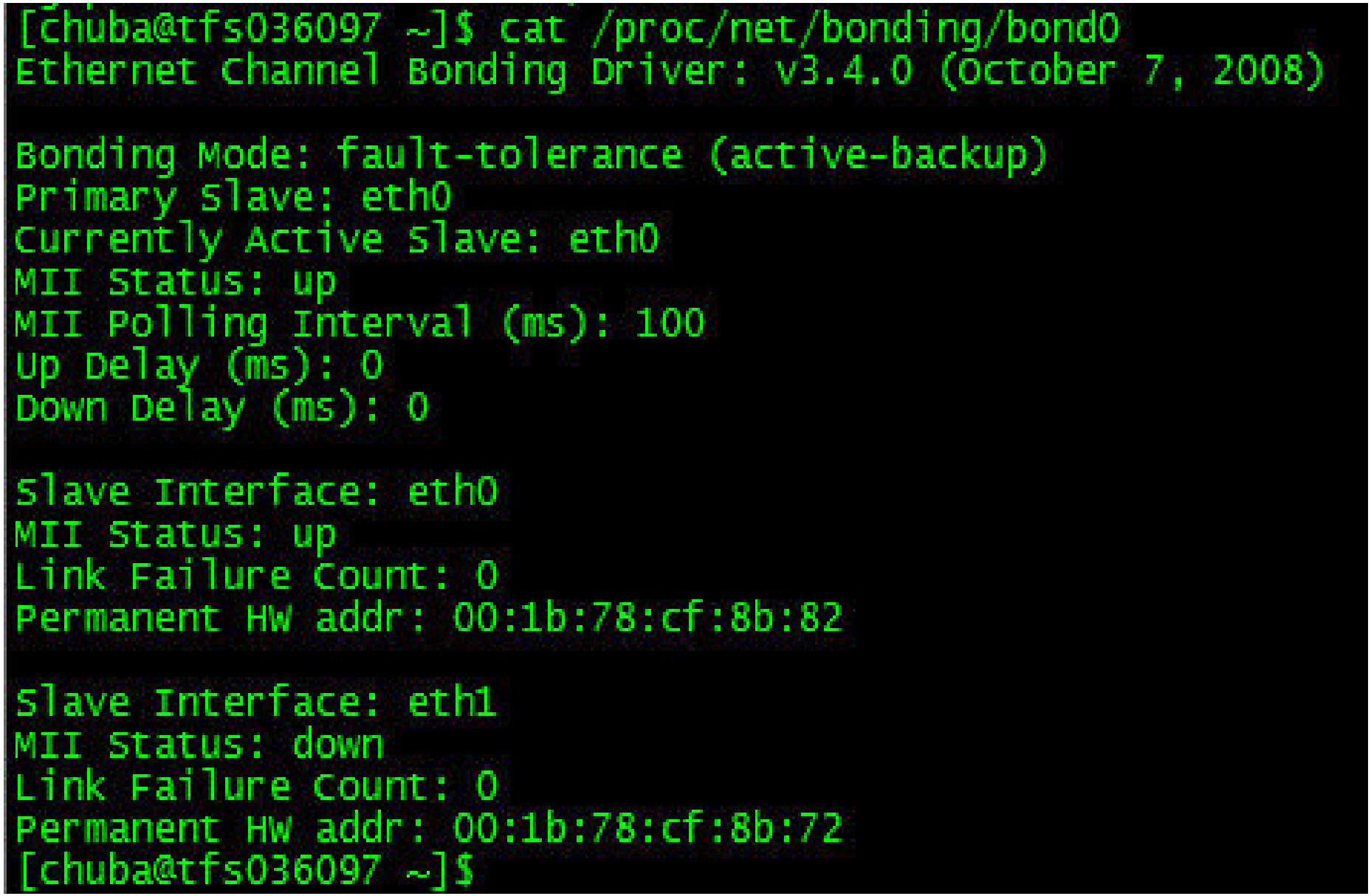
Average memory access time
= Hit time + Miss rate × Miss penalty.

I/D$: L1 hit = 2-3 clock cycles.

I/D$: L1 miss, L2 hit =~ 10-15 cycles.

TLB: L1 miss, L2 hit =~ 8-10 cycles.

TLB: L1 miss, L2 miss =~ 30+ cycles.



我们需要网卡的负载均衡模式（mode 0），需要交换机的支持

网卡bonding

中断平衡

```
[root@my174 beam]# dstat
----total-cpu-usage---- -dsk/total- -net/total- ---paging-- ---system--
usr sys idl wai hiq siq| read  writ| recv  send|  in   out | int   csw
  5   1  92   3   0   0|1417k 4532k|   0     0 |  24k   33k|1559    25k
  0   0 100   0   0   0|   0     0 | 420B 1092B|   0     0 |1002   106
  0   0 100   0   0   0|   0     0 | 140B  364B|   0     0 |1003   122
  0   0 100   0   0   0|   0     0 | 140B  364B|   0     0 |1003   108
```

硬中断:

```
yufeng@yufeng-laptop:~$ cat /proc/interrupts
           CPU0       CPU1
  0:    3757785    3774177   IO-APIC-edge      timer
  1:       4053       4013   IO-APIC-edge      i8042
  8:          0          1   IO-APIC-edge      rtc0
  9:       2198       2141   IO-APIC-fasteoi   acpi
```

- irqbalance 智能的均衡硬件中断。
- 手动 `[root@linux /]#echo ff > /proc/irq/19/smp_affinity`

软中断:

```
yufeng@yufeng-laptop:~$ cat /proc/softirqs
                    CPU0       CPU1
          HI:          0          0
       TIMER:    2821708   13270908
     NET_TX:          1          7
     NET_RX:      60385      57542
      BLOCK:      89454      87914
```

RPS/RFS 解决softirq平衡

RPS is not automatically switched on, you have to configure it.

```
echo ffff >/sys/class/net/eth0/queues/rx-0/rps_cpus
```

Same for RFS if you prefer to use RFS

```
echo 16384 >/sys/class/net/eth0/queues/rx-0/rps_flow_cn
```

显著提高软中断的均衡性，大大提高性能。

```
e1000e on 8 core Intel
   No RFS or RPS                104K tps at 30% CPU
   No RFS (best RPS config):    290K tps at 63% CPU
   RFS                          303K tps at 61% CPU

RPC test tps       CPU%     50/90/99% usec latency   Latency StdDev
  No RFS/RPS       103K     48%      757/900/3185                4472.35
  RPS only:        174K     73%      415/993/2468                491.66
  RFS              223K     73%      379/651/1382                315.61
```

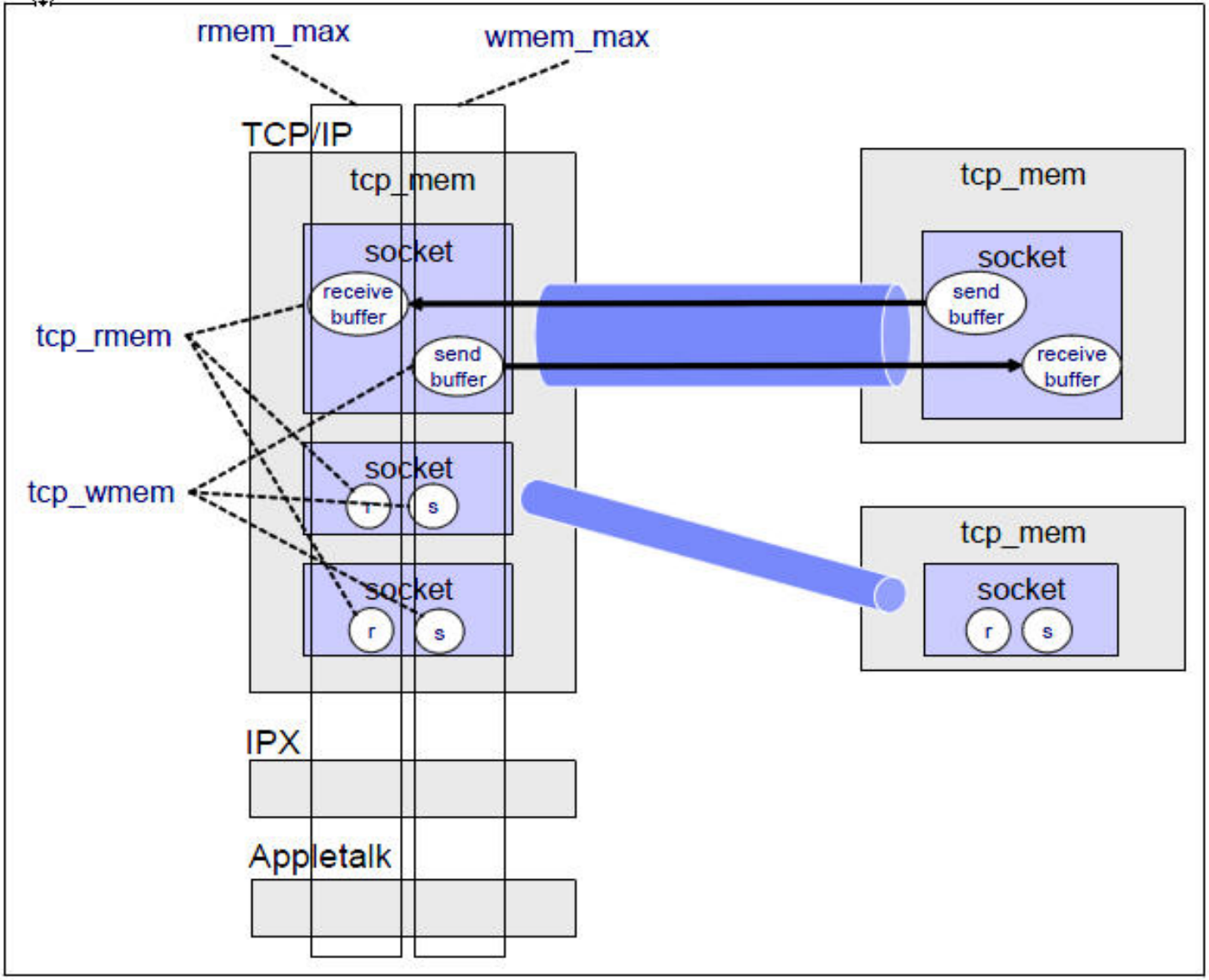


Figure 1-24 socket buffer memory allocation

```
net.ipv4.tcp_fin_timeout = 30
net.ipv4.tcp_keepalive_time = 1200
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.ip_local_port_range = 1024 65000
net.ipv4.tcp_max_syn_backlog = 8192
net.ipv4.tcp_max_tw_buckets = 5000
```

原则： dmesg可以观察到协议栈在抱怨什么，它抱怨什么我们解决什么！

TCP协议栈内存 不可交换物理内存

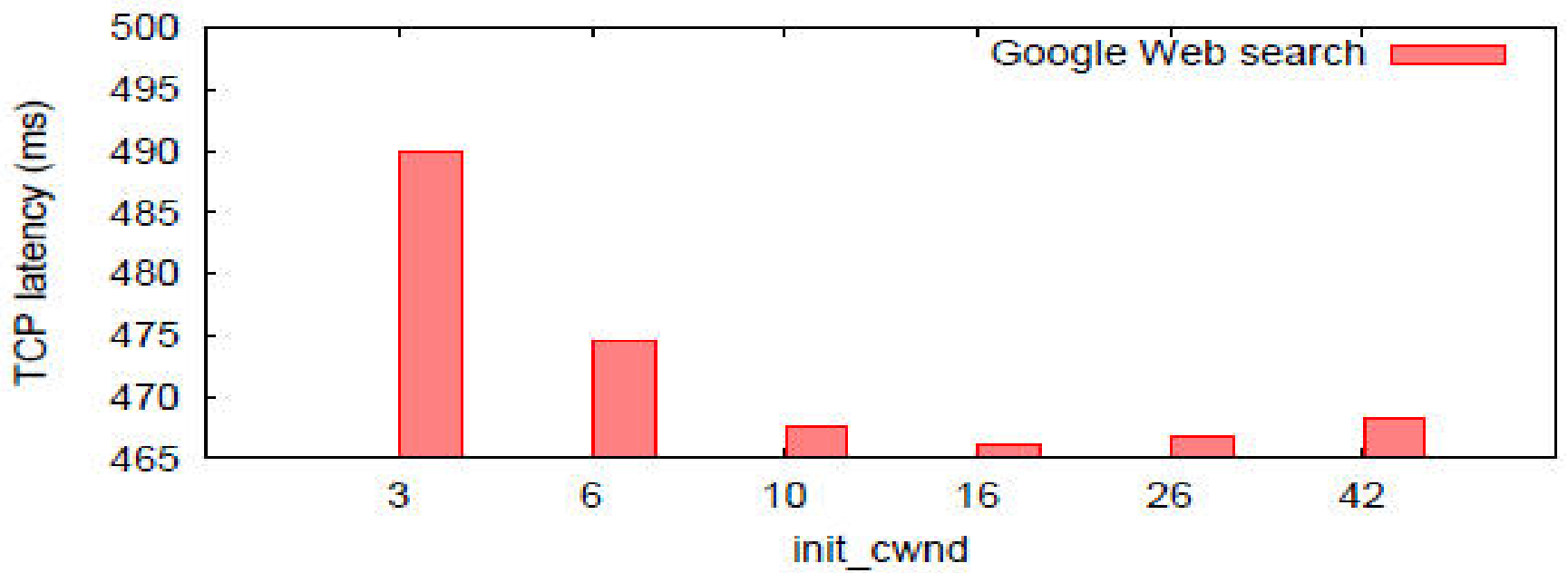
微调协议栈

来自google的initcwnd调优

通过提高初始拥塞窗口的大小(3)，大大减少短连接的响应时间.

make sure your Linux kernel is 2.6.30 or higher.

```
ip route change [default via a.b.c.d dev ethX ... ] initcwnd 10
```



- 磁盘硬件的选择
- 文件系统的选择
- IO调动算法的选择
- page cache的设置
- 不同类型的I0系统调用

对I0的性能都有很多的影响！

让FIO工具告诉你答案！

IO子系统

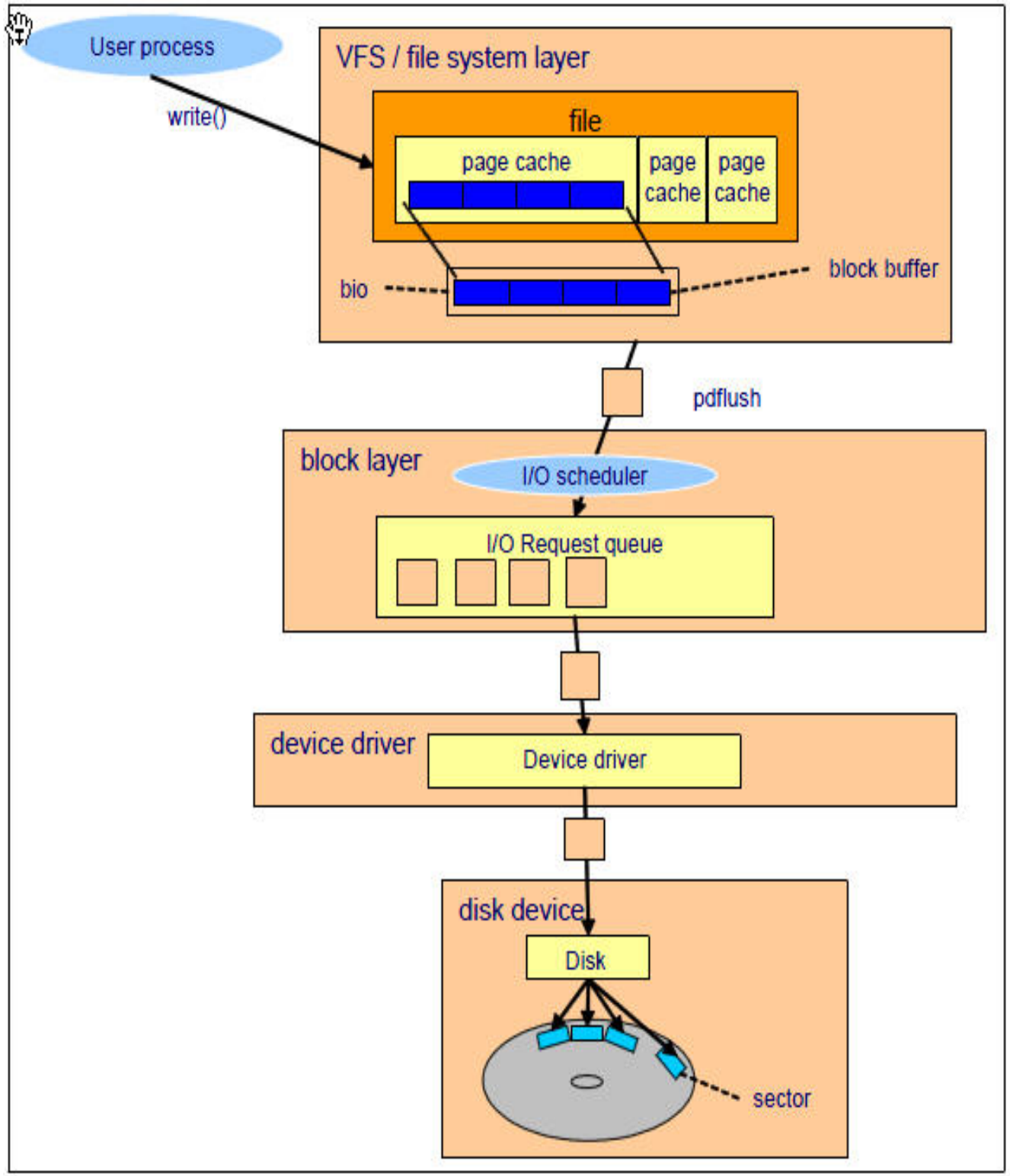


Figure 1-18 I/O subsystem architecture

采用异步IO

异步IO的好处， 应用批量提交请求，方便I0调动器合并请求，减少磁盘寻道和访问次数.

libaio: Linux native aio的封装, 在使用上可以用Linux eventfd做事件通知, 融入到Erlang的IO check机制。

glibc aio是用线程+同步调用模拟的, 完全不可用！

注意要保持设备队列的请求depth.

小结

- 采用64位操作系统
- 充分利用负载均衡技术，提高CPU和cache的利用率。
- 尽量用最新的linux内核，用降低稳定性，保持高性能。比如说Oracle的unbreakable Linux号称比RHEL5快85%。
- 尽量用新的能够提高性能的syscall.
- 常态监测你的系统， 找出性能减低的点，加以解决。

Agenda

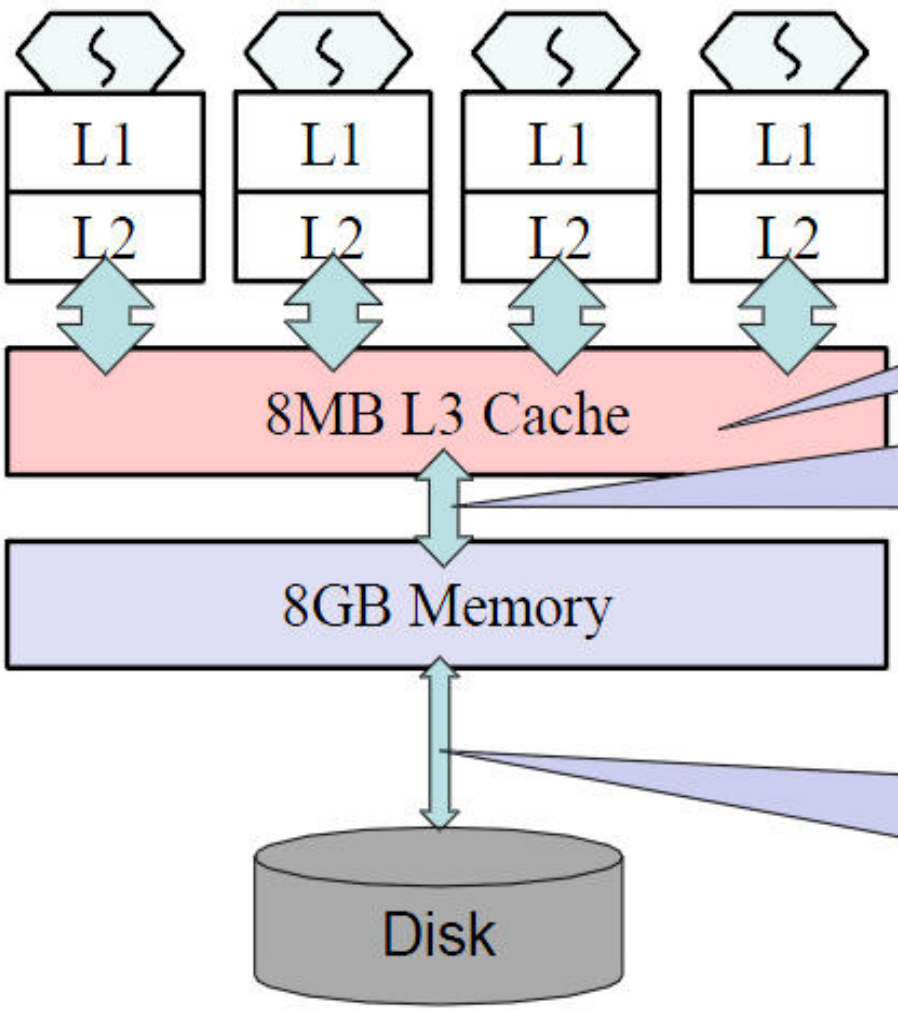
- 硬件层面变化和思考
- 操作系统层面变化和思考
- 语言和库层面变化和思考
- Erlang平台层面变化和思考
- 调优工具

CPU L1 cache reference	0.5 ns
CPU Branch mispredict	5 ns
CPU L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek (7200 rpm)	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet US->NL->US	150,000,000 ns

你需要知道的访问延迟数字

Performance Issues w/ the Multicore Architecture

Performance of a multicore system is significantly limited by slow data accesses to memory and disks



Cache Contention and Pollution:
Uncontrolled contention significantly increases cache misses

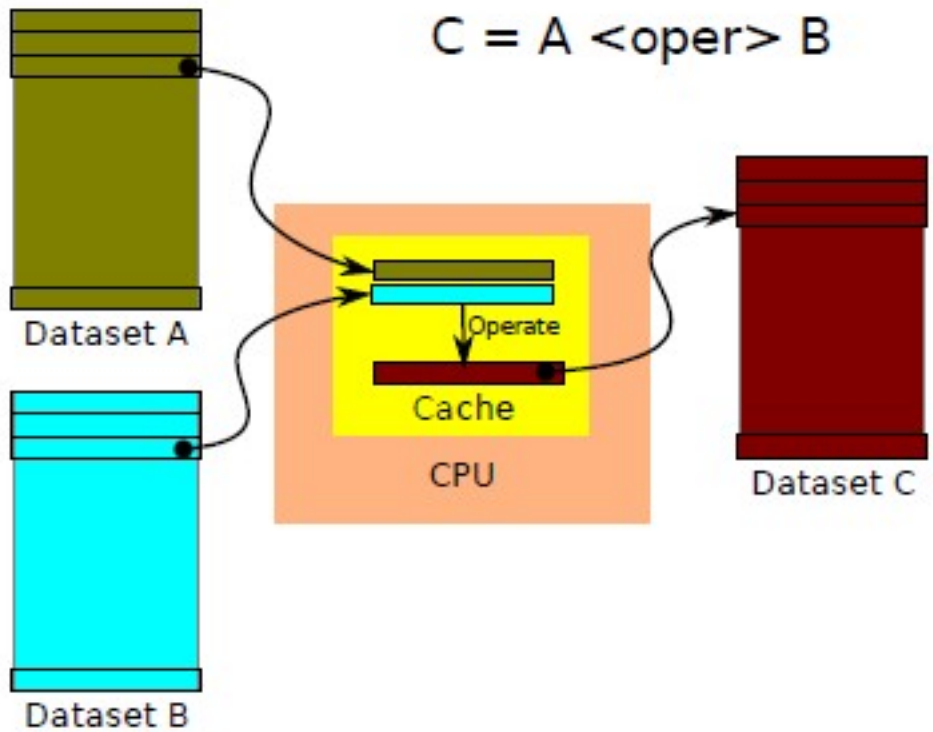
Memory Bus Congestion:
Limited memory bandwidth is competed for by multiple cores

"Disk Wall":
Limited storage throughput is shared by multiple cores

多核心架构下性能问题

The Blocking Technique I

When you have to access memory, get a contiguous block that fits in the CPU cache, operate upon it or reuse it as much as possible, then write the block back to memory:



Intel Xeon 7400 CPU: 96 KB *L1* cache (Data) and 16 MB of L3 cache

如何利用好我们的cache和空余CPU计算力？

压缩数据集

主存的访问速度很慢：8G/s， L1:300G/s。

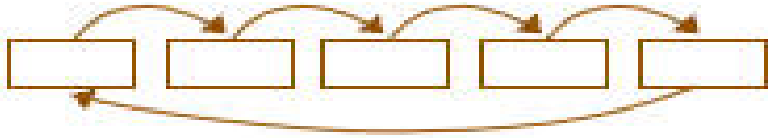
压缩我们的数据在传送，到目的地后解压，比直接传送要快。

Lzo 解压速度巨快，接近于memcpy， 压缩率大概在50%.

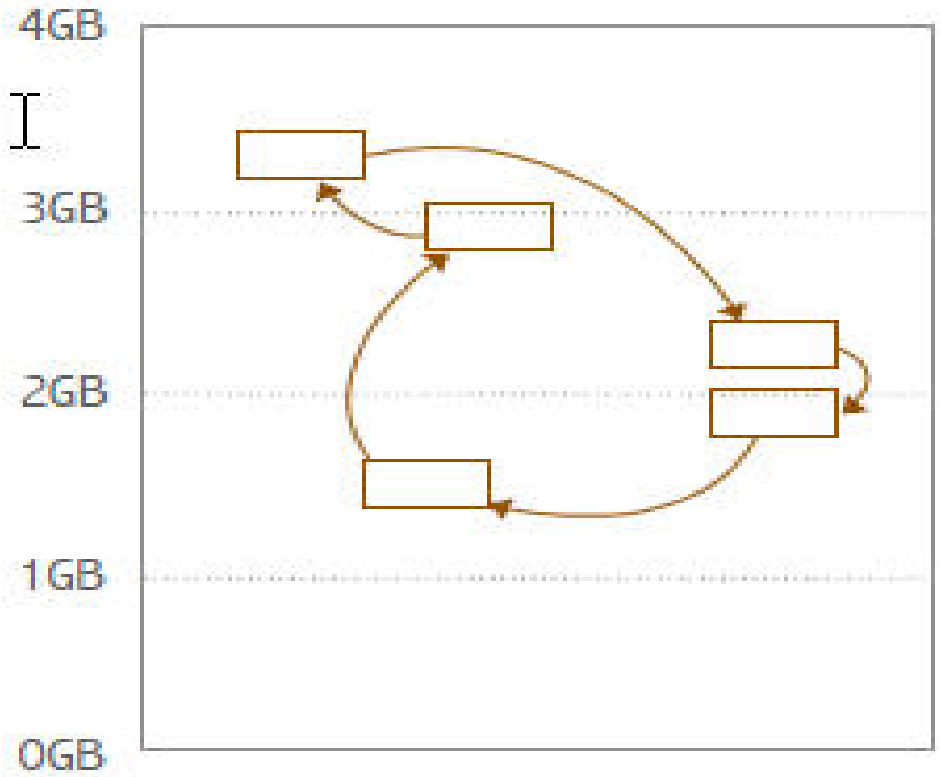
压缩速度比解压慢2-3倍， 对于读多写少的情况比较适合。

ramzswap显示压缩squid内存索引的压缩率：

OrigDataSize: 1968516 kB
ComprDataSize: 862015 kB

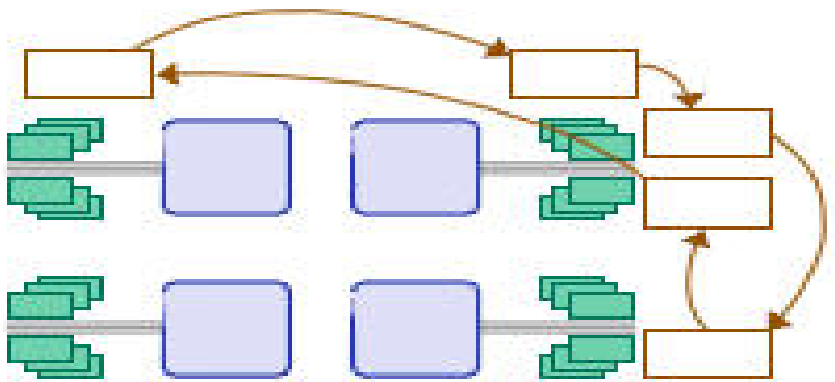


Erlang的 注意事项：



1. 单链表，只能表头访问，数据分散， 特别是数据被GC过后，中间的洞变大，对cache很不友好。

2. Erlang的IO支持 iolist, 底层会用writev发送数据，尽量用iolist.



列表 数据结构

避免数据搬动

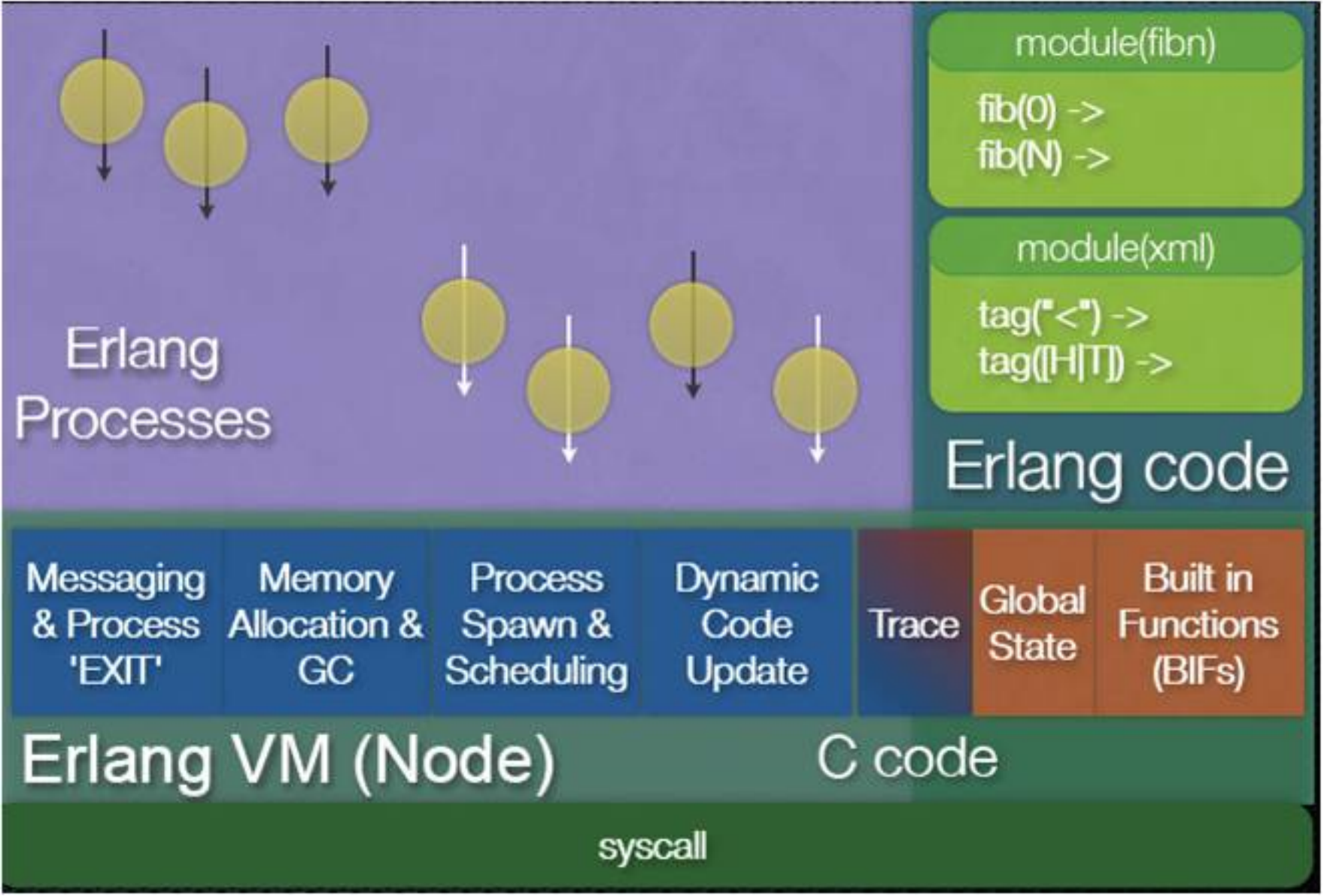
- (vm)splice，sendfile: 减少内核和用户空间的数据搬动 。
- readv/writev：gather read, scatter write。
- 多使用iolist。

小结

- 利用好cache 提高数据的locality。
- 采用更高效的算法。
- CPU大部分时间在空闲，等数据， 可以用时间换空间。
- 减少内存的搬动。
- 采用更快的编译器编译应用，比如说Intel ICC, 通常能快百分小几十。
- numa aware的内存分配池

Agenda

- 硬件层面变化和思考
- 操作系统层面变化和思考
- 语言和库层面变化和思考
- Erlang平台层面变化和思考
- 调优工具

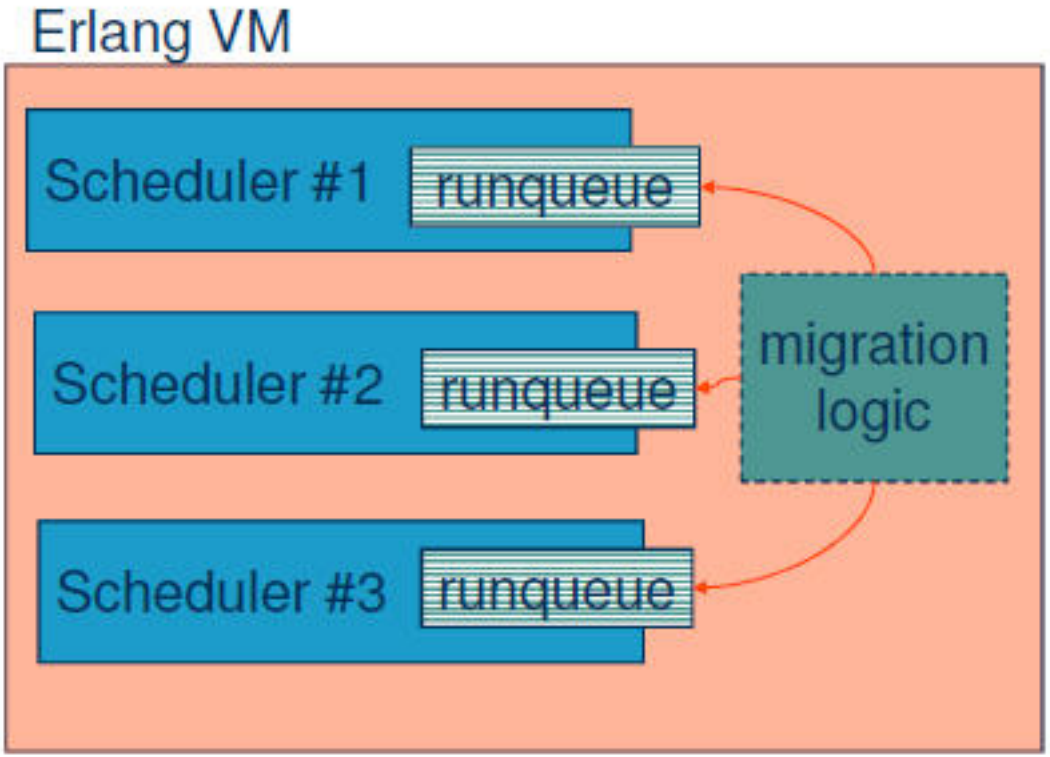


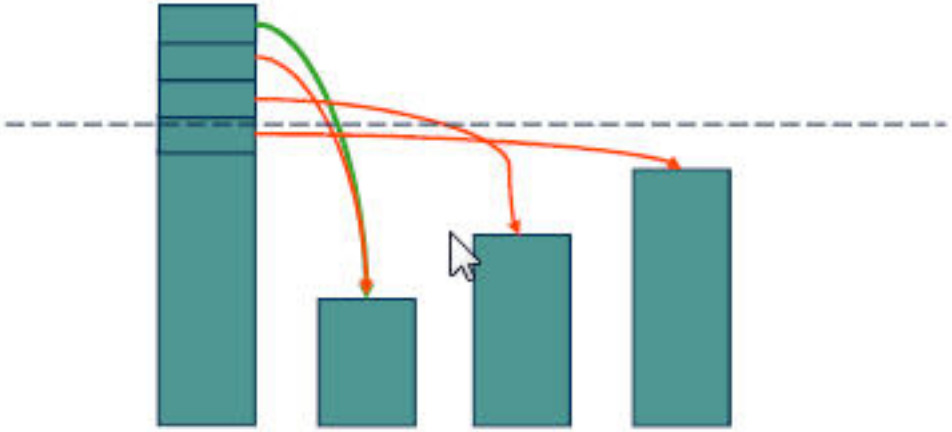
Erlang运行期内部结构图

- SMP版本和Plain版本，由erlexec动态选择根据参数选择。
- VM内部启用Hipe与否。
- 64位机器下halfword版本。

虚拟机的选择

Multiple runq-ueues
Erlang SMP VM (R13)





Running on full load or not!

调度器机制

```
[root@my174 results]# erl +sbt db
Erlang R14B (erts-5.8.1) [source] [64-bit] [smp:16:16] [rq:16] [async-threads

Eshell V5.8.1  (abort with ^G)
1> erlang:system_info(scheduler_bindings).
{1,3,5,7,0,2,4,6,9,11,13,15,8,10,12,14}
2> erlang:system_info(cpu_topology).
[{node,[{processor,[{core,[{thread,{logical,1}},
                           {thread,{logical,9}}]},
                    {core,[{thread,{logical,3}},{thread,{logical,11}}]},
                    {core,[{thread,{logical,5}},{thread,{logical,13}}]},
                    {core,[{thread,{logical,7}},{thread,{logical,15}}]}]}]},
 {node,[{processor,[{core,[{thread,{logical,0}},
                           {thread,{logical,8}}]},
                    {core,[{thread,{logical,2}},{thread,{logical,10}}]},
                    {core,[{thread,{logical,4}},{thread,{logical,12}}]},
                    {core,[{thread,{logical,6}},{thread,{logical,14}}]}]}]}]
```

spawn_opt 未公开参数 scheduler 用于绑定进程到指定调度器

```
  } else if (arg == am_scheduler && is_small(val)) {
      Sint scheduler = signed_val(val);
      if (erts_common_run_queue && erts_no_schedulers > 1)
          goto error;
      if (scheduler < 0 || erts_no_schedulers < scheduler)
          goto error;
      so.scheduler = (int) scheduler;
  } else {
      goto error;
```

绑定调度器

Erlang内存分配池

```
Eshell V5.8.1  (abort with ^G)
1>  erlang:system_info(alloc_util_allocators).
[temp_alloc,sl_alloc,std_alloc,ll_alloc,eheap_alloc,
 ets_alloc,binary_alloc,driver_alloc]
```

Numa aware 何时支持？ R14B？
largepage 何时支持？

内部有几百个分门别类的内存池。
mseg_alloc：通过mmap来向系统申请内存，批发给其他内存分配池。

Erlang 进程和Port

- 进程和现实世界1:1映射。
- 每个进程独立的堆和栈，独立的进行GC，消息通过拷贝的方式传递。
- 进程是根据时间片实现抢占式公平调度。
- Tcp port也是和现实世界1：1映射。
- 每个tcp port内部都有发送队列（高低水位线），以及接受缓冲区。
- Port通过Kernel Poll来实现事件监测，IO调动独立于进程调度，也是公平调度。

Erlang 进程单点问题

已知有单点的模块：

- x
- y

日志系统：

- 内置的太慢，推荐自己用shm来实现。

Erlang NIF

- R14新添加的，丰富的接口，容易用C库扩展Erlang的功能。
- NIF不像bif那样有trap机制，破坏Erlang调度器的抢占式调动。
- NIF千万不要调用阻塞函数， 比如调用mysql库。
- NIF有很大稳定性风险，错误会影响到整个VM。

EI (erlang C interface)

- 最近版本的EI修复了很多bug, 稳定了很多。
- 轻量，配合libev库做tcp链接接入服务器是非常好的选择。
- 丰富的RPC调用接口，直接访问后端Erlang服务器的模块。
- 支持多线程，多实例。

Mnesia

- In benchmarks done at Ericsson a few years ago
  - Mnesia tied the best commercially available cluster DBMS (Clustra) for transaction throughput and scalability
  - Two in-house products were faster - one became MySQL Cluster (NDB)
  - Mnesia beat them all on response times
- Linear scalability up to at least 50 nodes
  - If the data model is ideal for fragmentation
- A “dirty read” in Mnesia takes ~5-50 µsec (for relatively small objects)
  - Not possible to match when crossing memory protection boundaries

OTP最核心的部件Mnesia，是做分布式系统最关键的一环！

小结

- 并行化进程，按照1:1映射到现实世界。
- Erlang的调度器绑定，提高cach的利用率。
- halfword VM减少64位机器上的内存浪费。
- 开启Hipe Jit功能。
- 尽量使用binary，最贴近机器内存模型，cache友好。

Agenda

- 硬件层面变化和思考
- 操作系统层面变化和思考
- 语言和库层面变化和思考
- Erlang平台层面变化和思考
- 调优工具

推荐的性能调优工具

操作系统层面的：

- systemtap
- oprofile
- blktrace
- tsung

Erlang平台上的：

- lcnt

了解IO系统

性能测试工具：

- Fio 测试多种I0的效率（sync, mmap, libaio, posixaio...）
- Sysbench 简单易用的测试工具
- Iozone 侧重文件系统以及应用的数据访问模式

IO监视工具

- Blktrace
- btt 可视化你的I0活动
- seekwatcher 可视化你的磁头移动

```
[admin@my174 ~]$ ss -s
Total: 121 (kernel 147)
TCP:   6 (estab 3, closed 0, orphaned 0, synrecv 0, timewait 0/0), ports 3

Transport Total     IP        IPV6
*         147       -         -
RAW       0         0         0
UDP       9         6         3
TCP       6         2         4
INET      15        8         7
FRAG      0         0         0

[admin@my174 ~]$
```

netstat之类的工具对于大量的链接来讲实在太慢

```
-o, --options          show timer information
-e, --extended         show detailed socket information
-m, --memory           show socket memory usage
-p, --processes         show process using socket
-i, --info              show internal TCP information
-s, --summary           show socket usage summary
```

ss用于统计大量socket占用的资源情况

```
# stap examples/general/para-callgraph.stp
    'process("/usr/sbin/sendmail").function("*")'
      0 sendmail(4523):->doqueuerun
   1736 sendmail(4523):<-doqueuerun return=0x0
      0 sendmail(4523):->sm_blocksignal sig=0xe
     56 sendmail(4523):<-sm_blocksignal return=0x0
      0 sendmail(4523):->curtime
     22 sendmail(4523):<-curtime return=0x4c06fb34
      0 sendmail(4523):->refuseconnections name=0x93ad8b0
          e=0x343a80 d=0x0 active=0x0
     59 sendmail(4523): ->sm_getla
    109 sendmail(4523):  ->getla
    930 sendmail(4523):   ->sm_io_open type=0x3432c0
        timeout=0xfffffffffffffffe info=0x3231cd flags=0x2
        rpool=0x0
   1733 sendmail(4523):     ->sm_flags flags=0x2
   1771 sendmail(4523):     <-sm_flags return=0x10
   1876 sendmail(4523):     ->sm_fp t=0x3432c0 flags=0x10
       oldfp=0x0
  12409 sendmail(4523):     <-sm_fp return=0x372d7c
```

Systemtap帮助你了解你的程序

大部分的图片粘贴自Google搜索的文档，谢谢Google，谢谢这些可爱的作者。

谢谢大家！

Any question？