Leader-us系列

大型分布式系统案例实战 第13周

DATAGURU专业数据分析社区

法律声明

【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

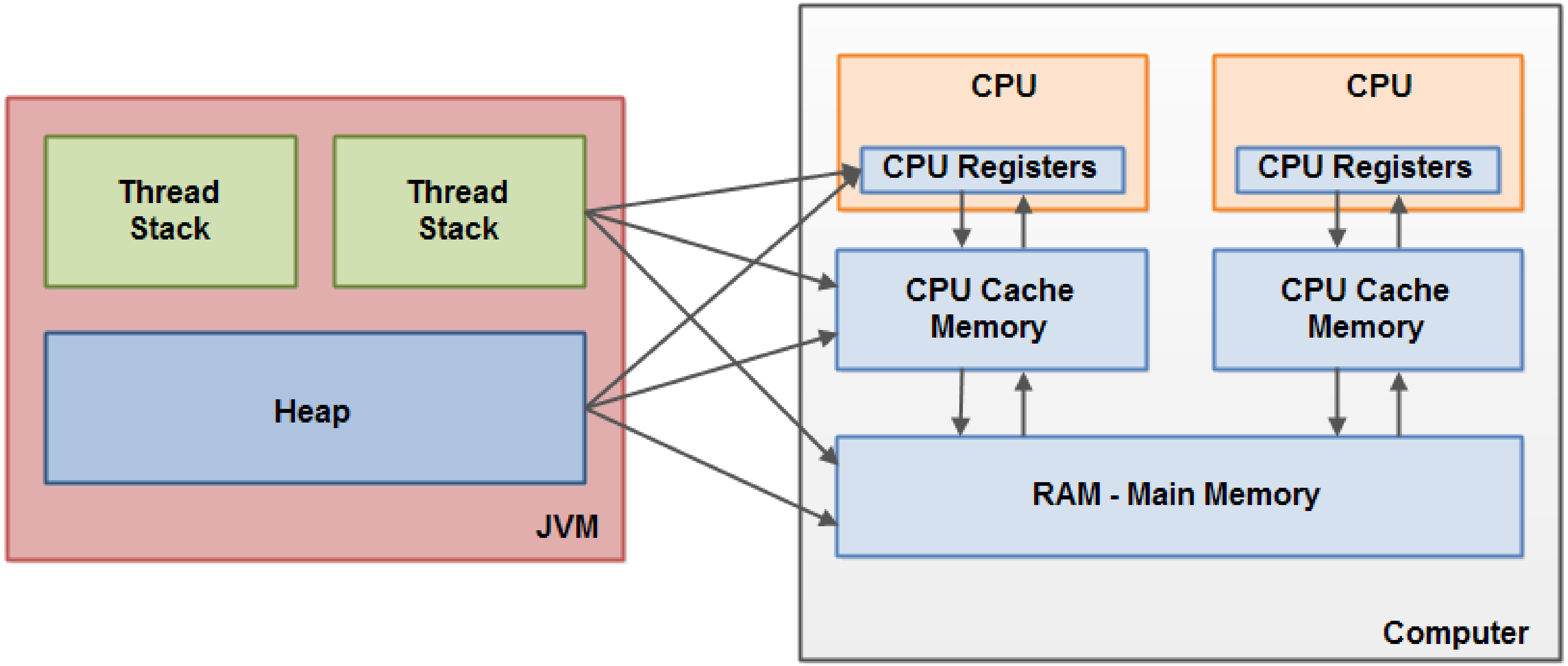
http://edu.dataguru.cn

Java分布式系统中的高性能难题

- 内存和并发编程领域的难题
- 网络通信框架的挑战
- 高性能事件派发机制探讨

内存和并发编程领域的难题

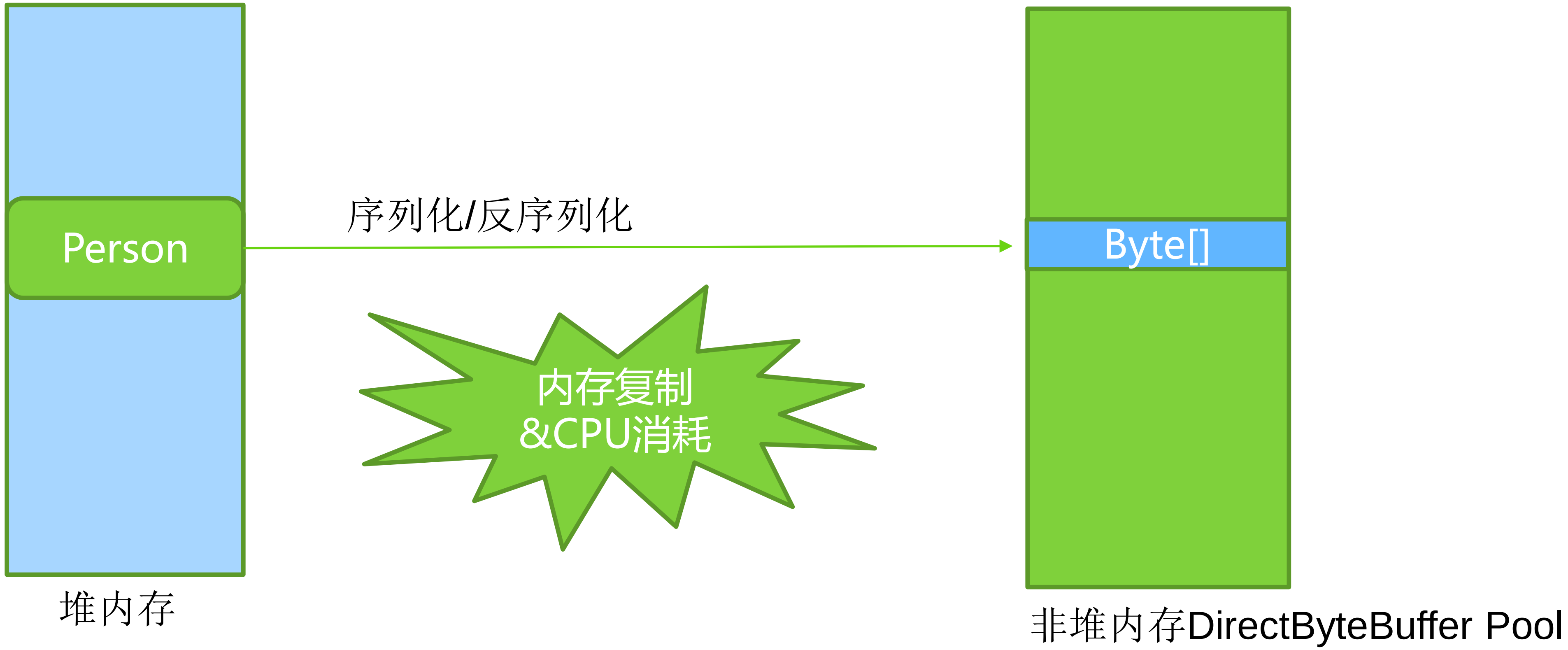
成也萧何，败也萧何



内存和并发编程领域的难题

Java程序无法自由操作物理内存

DirectByteBuffer则目前还没有很好的内存释放办法



内存和并发编程领域的难题

The sun.misc.Unsafe API, which is used by many libraries and frameworks, is being removed as default from Java 9

OpenJDK的非堆JDK增强提议（JDK Enhancement-Proposal，JEP）试图标准化一项基础设施，它从Java6开始，只能在HotSpot和OpenJDK内部使用。这种设施能够像管理堆内存那样管理非堆内存，同时避免了使用堆内存所带来的一些限制。

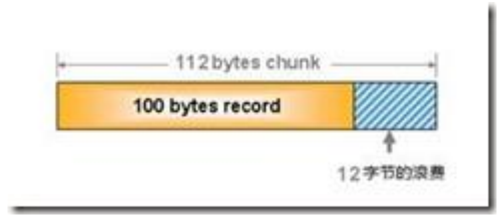
Off Heap JEP 1 - replacement for off heap memory access under sun.* esp Unsafe, but also FileChannelImpl and FileDispatcherImpl (including thread safety)
Off Heap JEP 2 - allocation of objects on the stack, improving EscapeAnalysis or adding developer annotation hints
Off Heap JEP 3 - on heap structs
Off Heap JEP 4 - off the heap structs

Jillegal is a library including unknown tricks of Java objects already lives on offheap

内存和并发编程领域的难题

内存存储的通用性难题

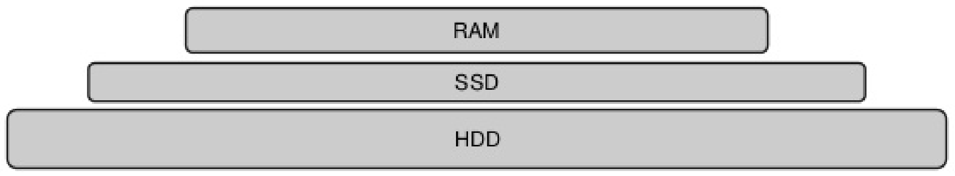
内存使用效率问题
内存不足的情况下磁盘溢出的问题
内存回收的问题

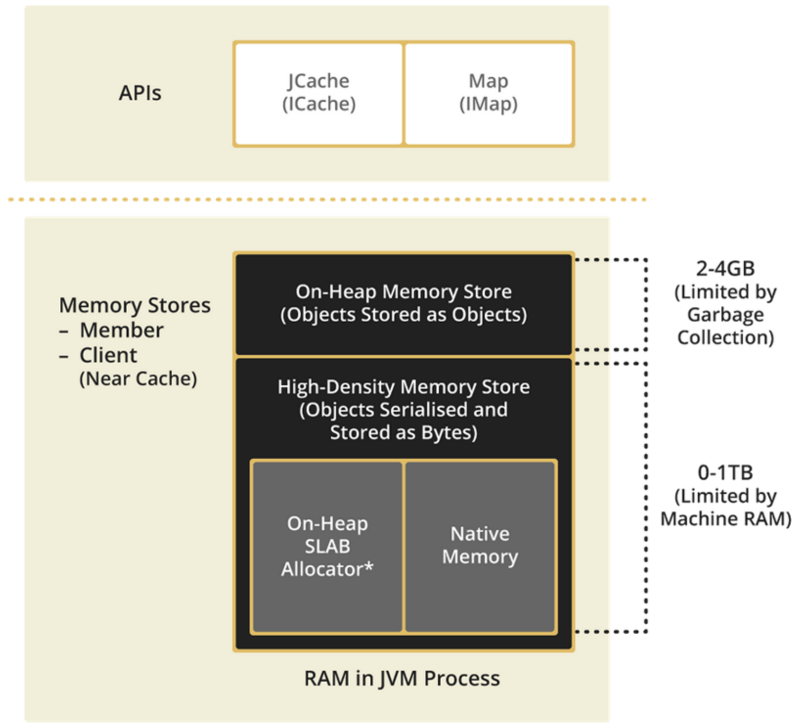


内存利用率难以保障

Tachyon as tiered storage

The capability to work with storage hierarchy allows to utilize available ram and SSD in more efficient way.

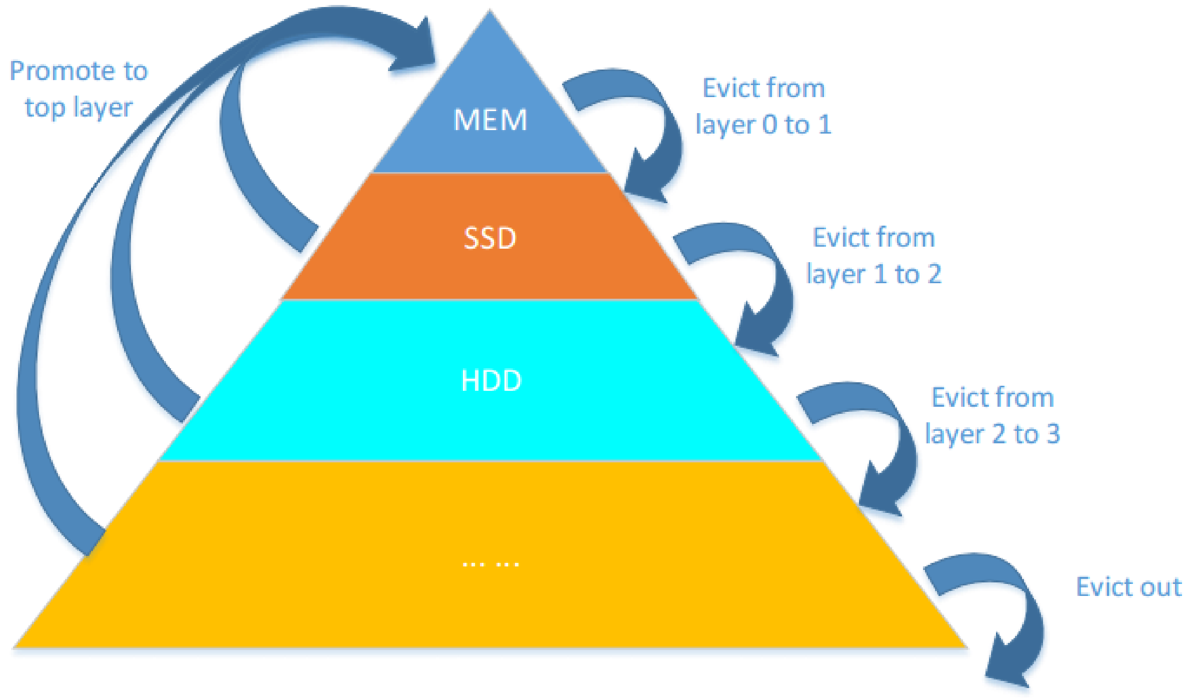


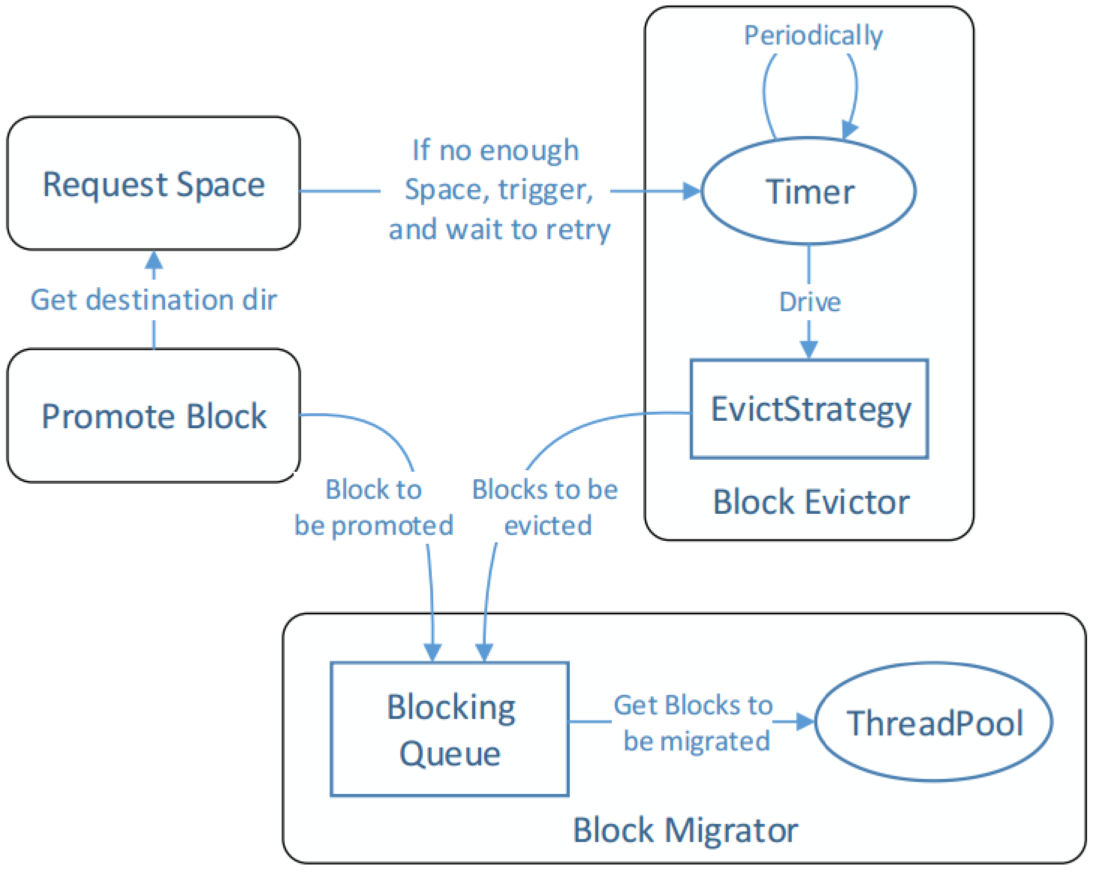

APIs
JCache (ICache)
Map (IMap)
Memory Stores
– Member
– Client (Near Cache)
On-Heap Memory Store (Objects Stored as Objects)
High-Density Memory Store (Objects Serialised and Stored as Bytes)
On-Heap SLAB Allocator*
Native Memory
RAM in JVM Process
2-4GB (Limited by Garbage Collection)
0-1TB (Limited by Machine RAM)

默认512M OS的内存池，分为默认4M大小的管理单元，默认采用ThreadLocal方式管理内存单元。
存在用来存放Index、偏移量等Metadata信息的内存单元，MetaData的占用空间默认是12%

内存和并发编程领域的难题

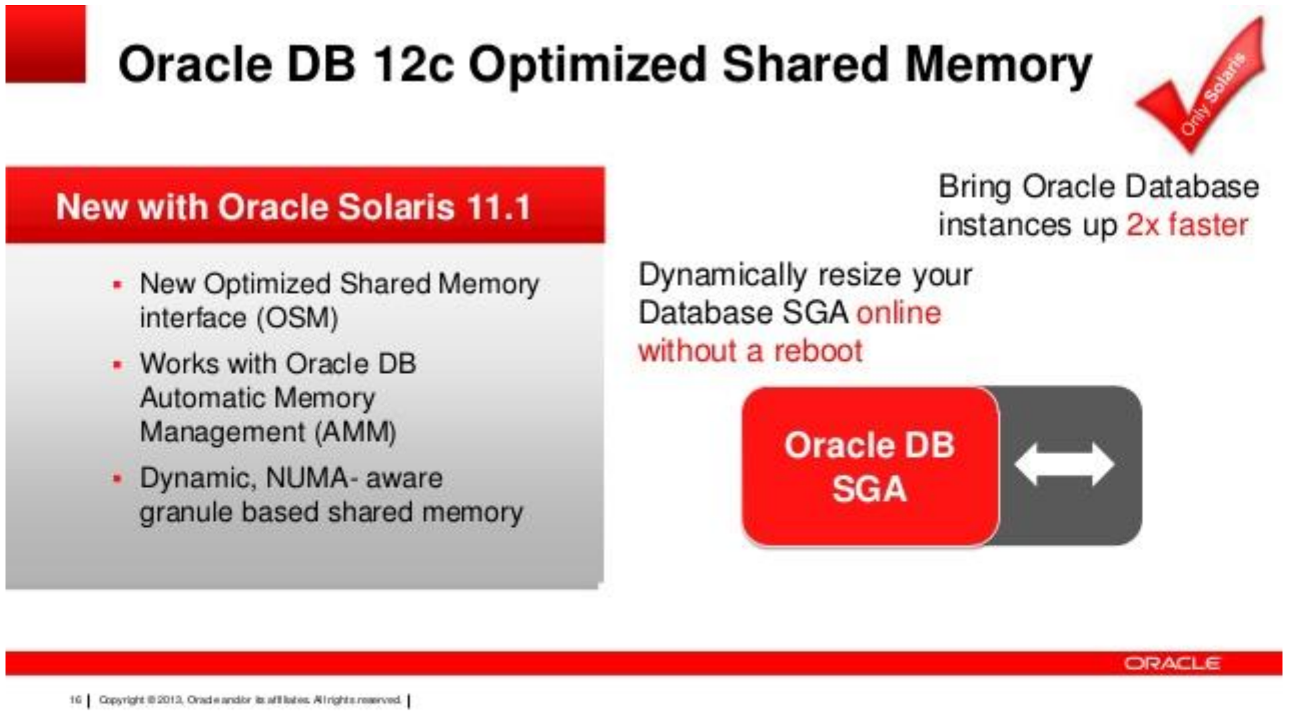
TACHYON的多层存储问题





内存和并发编程领域的难题

Java线程亲和性问题



Oracle DB 12c Optimized Shared Memory

Only Solaris

New with Oracle Solaris 11.1

- New Optimized Shared Memory interface (OSM)
- Works with Oracle DB Automatic Memory Management (AMM)
- Dynamic, NUMA- aware granule based shared memory

Bring Oracle Database instances up 2x faster

Dynamically resize your Database SGA online without a reboot

Oracle DB SGA

ORACLE

16 | Copyright © 2013, Oracle and/or its affiliates. All rights reserved. |

NUMA下内存共享的设计是个难点

https://github.com/OpenHFT/Java-Thread-Affinity实现了将线程绑定到CPU上

内存和并发编程领域的难题

缓存友好的数据结构

•Data structures that are contained within a single cache-line are more efficient.
•Organize your data to avoid alignment holes
•Don't neglect the cache in data structure and algorithm design
•Use smaller data types
•Make sure all adjacent data is actually used in the hot loops. Otherwise, consider breaking up data structures into hot and cold components, so that the hot loops use hot data.
•Avoid algorithms and datastructures that exhibit irregular access patterns, and favor linear datastructures

Spark支持缓存计算，提高了数据处理的速度通过更有效地利用CPU上L1，L2和L3的缓存，因为它们比物理内存在速度上快多个数量级。剖析Spark用户应用程序时，我们发现大部分的CPU时间是从物理内存等待获取数据。我们正在设计缓存友好的算法和数据结构，所以应用程序将花费更少的时间等待从内存获取数据，从而用更多的时间做有用的工作。

内存和并发编程领域的难题

缓存友好的数据结构

二分检索是查找有序数组最简单然而最有效的算法之一, 由于二分检索的随机跳跃性，该算法并非缓存友好的

CATree 是一个基于B-树的数据结构，它使用数组存储数据，由于没有指针，所以 Cache 利用率更高.使用 CATree 可以降低查找算法的 DRAM 访问次数，改进后的算法整体性能有很大提高，即在 600 规则的性能评价实验中，改进算法比聚合位向量算法快 30%，比位向量算法快 94%.

HAT-trie: A Cache-conscious Trie-based Data Structure for Strings

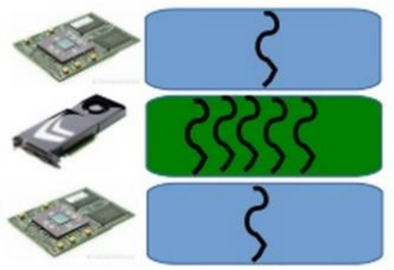
The Need to Rewrite Established Algorithms——Dr. Dobb's

内存和并发编程领域的难题

GPU并行编程

IBM Power 8 – now with GPU acceleration

- GPU devices plug into the host PCIe bus to provide massive arrays of co-processors



- Typical scenario for heterogeneous programming:
  - Host computer with CPU(s) and GPU(s) installed on PCIe bus
  - Programmer identifies parallelizable, compute intensive routine, and codes to GPU
  - Flow of data and control passes between CPU host and GPU device under control of host device

Particularly suited to scientific and numerical analysis problems (e.g. linear algebra). We have focused on Nvidia CUDA as the programming model for exploiting GPUs.

NVIDIA. CUDA

Three-tiers of exploitation in Java:

- CUDA4J : a low-level interface to the GPU for applications that want direct control from Java, enabling reuse of kernels from Java, faster time to market
- Java SE library exploitation : backing standard Java APIs with a GPU implementation for improved performance (sort, etc)
- Dynamic workload off-loading : identifying patterns in application code that will benefit from parallelisation directly in the JIT

GPU-enabling standard Java SE APIs

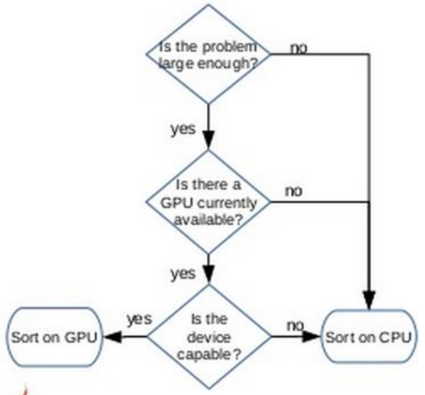
- Natural question after seeing the good speed-ups using explicit programming …
- What areas of the standard Java API implementation are suitable for off-loading onto GPU?

- We picked

  `java.util.Arrays.sort(int[] a)` and friends
  - GPU modules exist that do efficient sorting

We employ heuristics that determine if the work should be off-loaded to the GPU.

- Overhead of moving data to GPU, invoking kernel, and returning results means small sorts (<~20k elements) are faster on the CPU.
- Host may have multiple GPUs. Are any available for the task?
- Is there space for conducting the sort on the device?

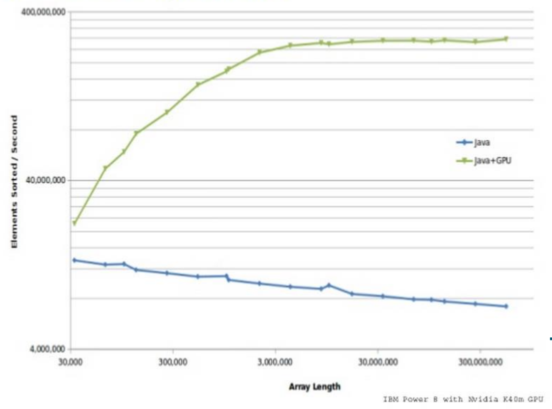


Is the problem large enough?
no
yes
Is there a GPU currently available?
no
yes
Sort on GPU
yes
Is the device capable?
no
Sort on CPU

Java
IBM Developer Kits for Java
ibm.com/java/jdk

30

GPU-enabled array sort method



网络通信框架的挑战

Netty是性能最好的NIO框架么

CoralReactor is a powerful, easy to use and ultra-low-latency Java library for network communication with zero garbage creation and minimal variance. Moreover, **what stands out about CoralReactor is its simplicity**

CoralReactor vs Netty   **10 times faster** and produces zero garbage.

We have developed a NIO networking library that performs under 2 microseconds over loopback without producing any garbage for the GC. As Peter Lawrey mentioned, the native JDK selector produces a lot of garbage but we have fixed all these *garbage leaks* by implementing our own epoll selector. Busy waiting the selector thread is great for latency but there must be a balance not to burn the chip or consume a lot of energy. Our selector implementation use low-level tricks to implement a kind of *energy saving mode* that takes care of that balance.

网络通信框架的挑战

看看大牛怎么说高性能网络通信的编程问题

Peter Lawrey

If you want to save micro-seconds, you will want to use busy waiting non-blocking NIO for threads on dedicated cpus. This doesn't scale well as you need to have plenty of CPU but does minimise the latency for handling IO. I suggest you also bind the isolated CPUs to minimise jitter.

You will want to avoid using Selectors as they block and/or create quite a bit of garbage adding to GC pauses.

Also to minimise latency you will want to use a low latency, kernel bypass network adapter such as Solarflare.

You will want to use a push parser so long messages can be decoded/parsed as they download. i.e. you won't want to wait until the whole messages is recieved before starting.

Using these tricks in combination can save 10 - 30 micro-seconds off every request or inbound event.

Netty is a better solution for scalability ie, higher net throughput, but at a small cost to latency, as do most frameworks which are based on support web services where milli-seconds delays are tolerable.

网络通信框架的挑战

OpenOnload® is a high performance network stack from Solarflare that dramatically reduces latency and cpu utilisation, and increases message rate and bandwidth

- Application-to-application latency below 1.7us.
- Send or receive millions of messages per second per CPU core.
- Binary compatible with existing applications.
- Includes innovative techniques that reduce interrupt overheads and improve scalability.
- Open Source (GPLv2).
- Runs on many Linux distributions and most 2.6 and 3.x kernels, including realtime.
- Supports 32-bit and 64-bit user-level and kernels (including 32-bit user-level with 64-bit kernel).

线程切换的代价

•1、当前执行任务的时间片用完之后，系统CPU正常调度下一个任务。
•2、 当前执行任务碰到IO阻塞, 调度器将挂起此任务, 继续下一任务
•3、 多个任务抢占锁资源, 当前任务没有抢到,被调度器挂起, 继续下一任务
•4、 用户代码挂起当前任务, 让出CPU时间
•5、 中断。硬件中断：外设发送电信号给处理器，异步，IRQ（interrupt request）；软中断：异常情况 or 特殊指令集，软中断可以用来实现system call。

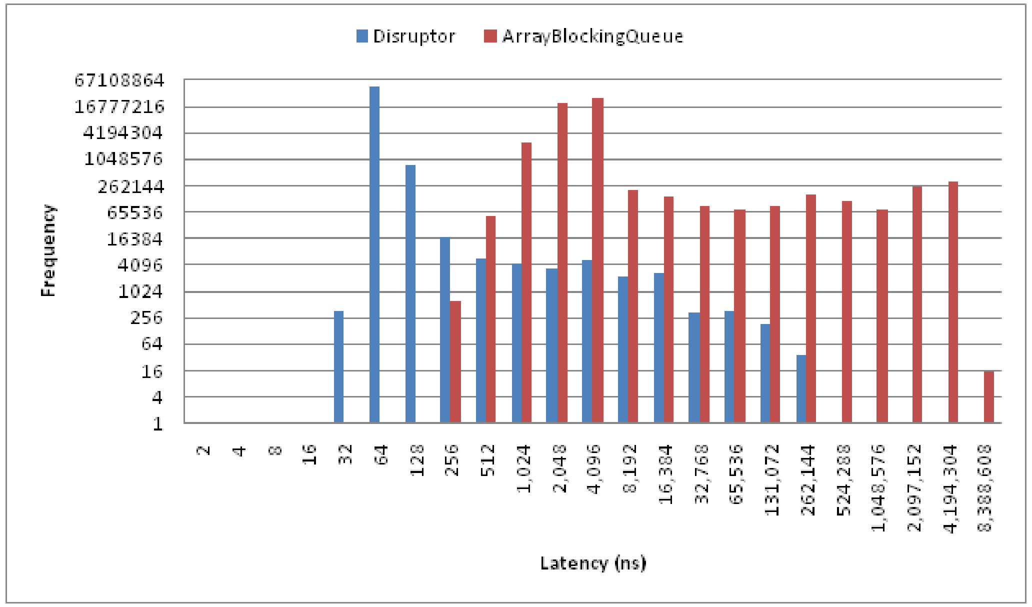
A context switch can mean aregister context switch, atask context switch, astack frame switch, athread context switch, or a process context switch.（表现形式：进程之间，线程之间，栈帧之间）

高性能事件派发机制探讨 Disruptor

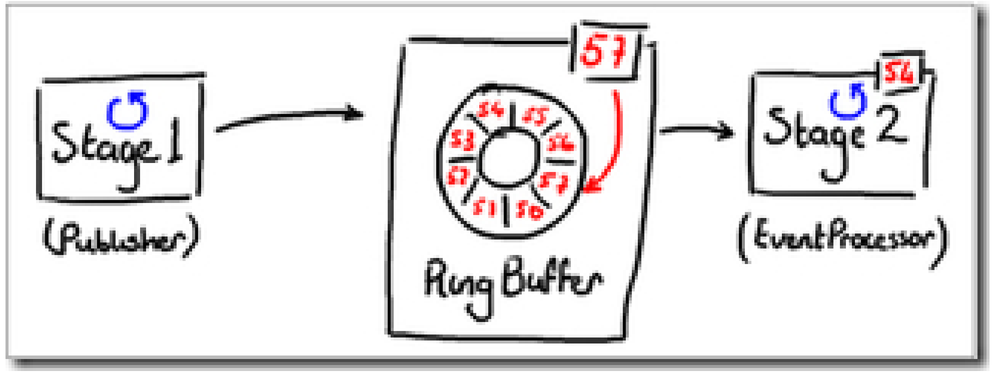
外汇、黄金交易顶级平台LMAX介绍

LMAX是在英国注册并受到FCA监管（监管号码为509778）的外汇黄金交易所。

LMAX aims to be the fastest trading platform in the world. Clearly, in order to achieve this we needed to do something special to achieve very low-latency and high-throughput with our Java platform. Performance testing showed that using queues to pass data between stages of the system was introducing latency, so we focused on optimising this area.



高性能事件派发机制探讨 Disruptor

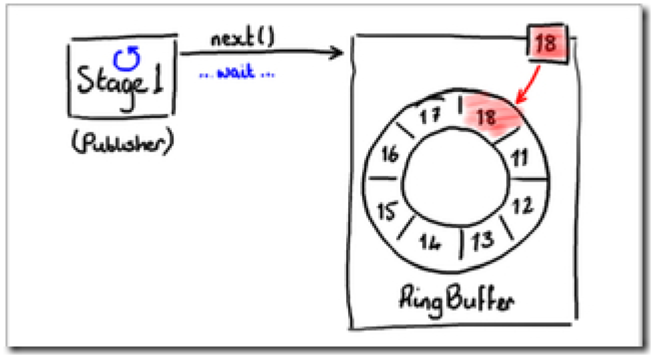


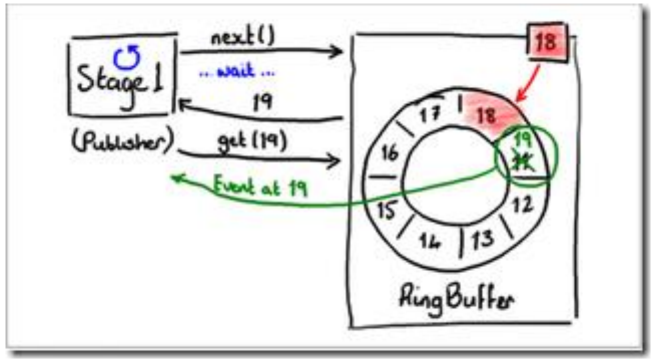
RingBuffer里的每一个元素都有一个序列号（sequence number）来索引，RingBuffer维护当前最新放置的元素的序列号，这个序列号一直递增

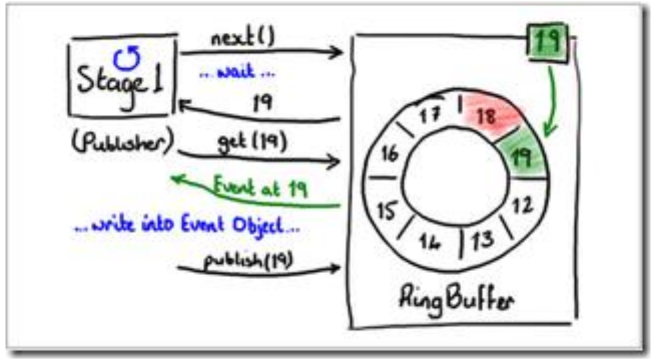
Disruptor的关键特性是无锁编程，这个是通过单一写线程的方式实现的 - 即一块数据永远只有一个线程写入。

高性能事件派发机制探讨 Disruptor

写消息过程





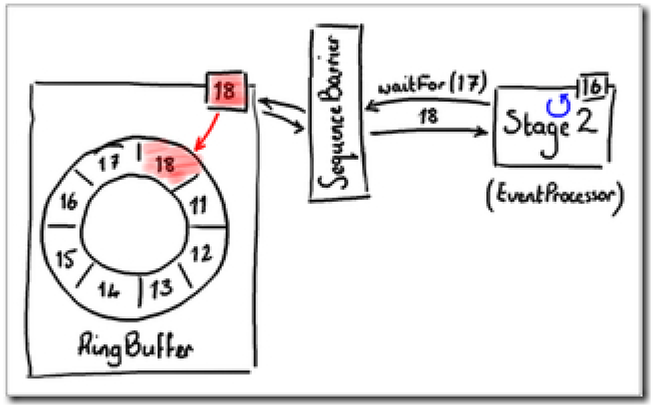


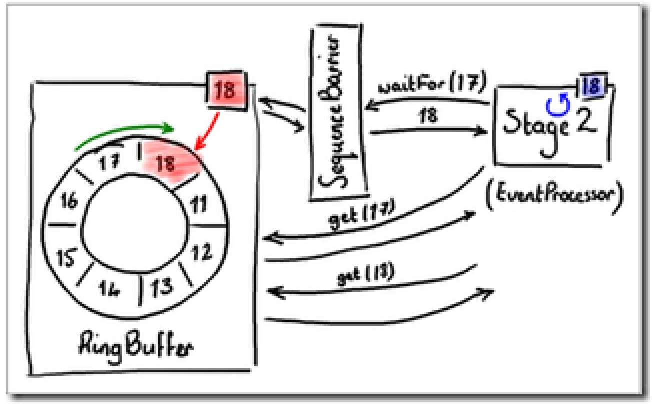
RingBuffer维护了最后一次写入的序列号（图3里的18号），因此就可以推知下一个空闲的槽号。RingBuffer通过检查所有从RingBuffer读取消息的EventProcessor的序列号，以判别下一个槽号是否空闲。

当生产线程拿到了下一个序利号之后，它从RingBuffer里拿到槽里保存的对象并执行任何操作。这个过程中，因为RingBuffer的最新序列号依然是18，因此其它线程无法读取19号槽里面的事件 - 生产线程还在处理它。

高性能事件派发机制探讨 Disruptor

读消息过程





RingBuffer里最新可用数据已经到18号槽了，因此waitFor返回18，即告诉EventProcessor可以一直读到第18号的所有数据，批处理

Disruptor框架里提供了一个叫做EventProcessor来从RingBuffer里读取数据。当生产线程向RingBuffer要求下一个可写入的空闲槽的序列号时，同时一个EventProcessor（类似消费者，但其并消费RingBuffer里的元素 - 即不从RingBuffer里移除任何元素）也会维护其最后所处理的数据的序列号，并要求下一个可处理的数据的序列号。

总结： Java分布式系统中的高性能难题

- 新硬件、数据规模、实时性所带来的挑战：
  新的数据结构和新的算法
- 分布式和云计算发展所带来的挑战
  网络性能和编程模型改进

Thanks

FAQ时间