.Net并行编程优化与实战

主要内容

一、防止CPU高速缓存的失效

二、选择合适的线程同步机制

1. 常用锁机制介绍

2 实现无锁的数据操作

3. 缓存模块的设计演变

三、 .Net 4.0 并发数据结构--ConcurrentQueue<T> 的设计

四、关于线程池

五、 NUMA架构的机器上的优化

六、选择合适的并行编程模型

七、案例1--某集团全球数据中心数据处理模块的设计。

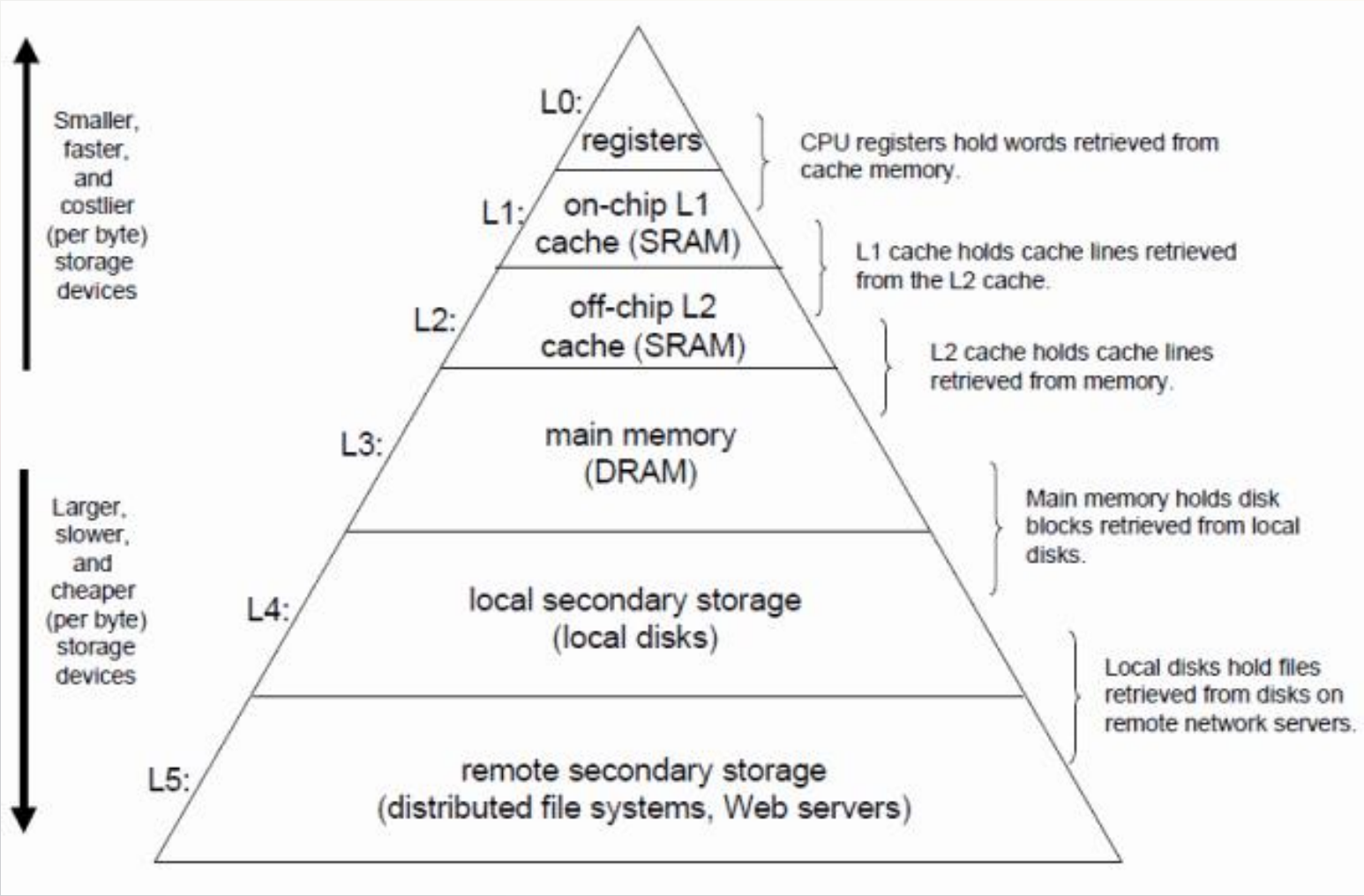
1.根据业务设计自己的异步队列

2.多线程写日志文件的问题与解决方案

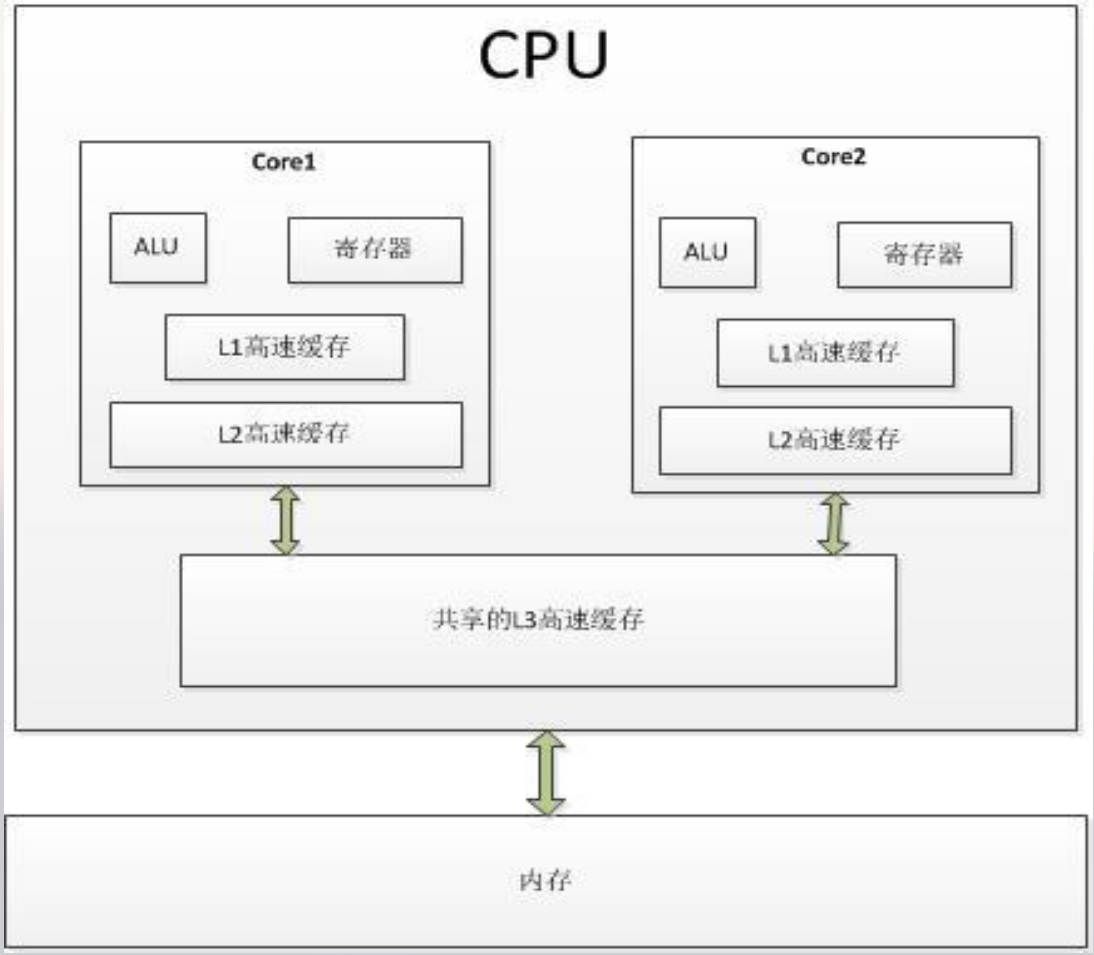
3.减少IO操作将监控日志输出到DebugView

八、案例2--某实时采集计算平台的研发

一、防止CPU高速缓存的失效



一、防止CPU高速缓存的失效



二、选择合适的线程同步机制

(1) volatile关键字

(2) Interlock（原子操作）

Interlocked.Increment(ref m_high)

Interlocked.CompareExchange(ref actual, newValue, oldValue)

容易出现的问题：ABA问题，循环过多问题

(3) Spinlock（自旋锁）

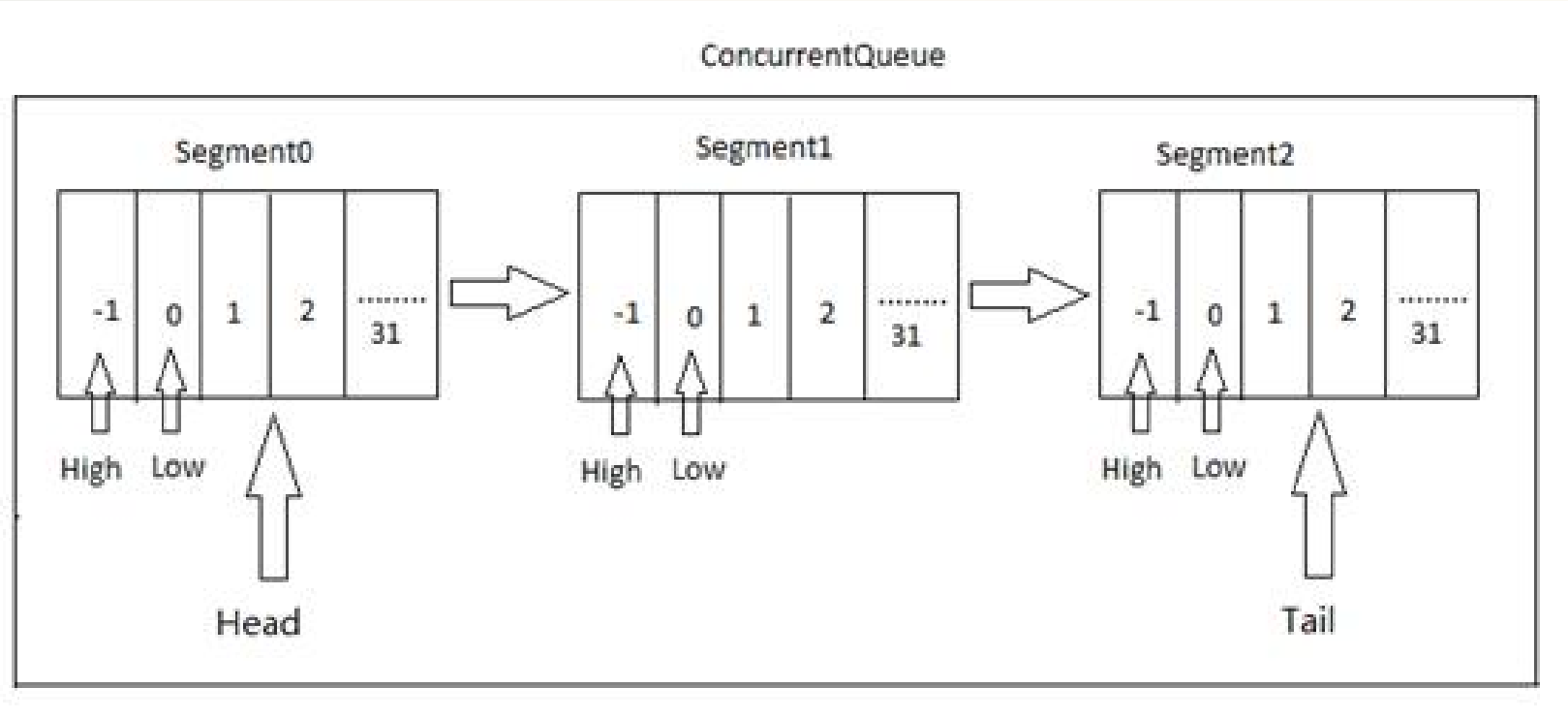
(4) Monitor(Lock)

(5) ReadWriterLock(读写锁)

2.实现无锁的数据操作

3.缓存模块的设计演变

三、 .Net 4.0 并发数据结构 ConcurrentQueue<T>



四、正确的使用.Net线程池

（1）对于执行时间较短的任务都应当交给线程池去处理而不是开启一个新线程，对于需要长时间处理的任务交给单独的线程去处理。

（2）对于读写文件的任务不要交给线程池处理，因为线程池内的线程属于后台线程，应用程序意外关闭后线程也关闭从而丢失数据。

(3) 不要阻塞线程池里的线程

(4) asp.net 大量请求 最小线程数问题

（5）永远不要自己开发线程池，真正能用到产品级别线程池需要几个月的时间而且需要大量的测试，否则遇到问题悔之晚矣

五、NUMA架构的机器上的优化

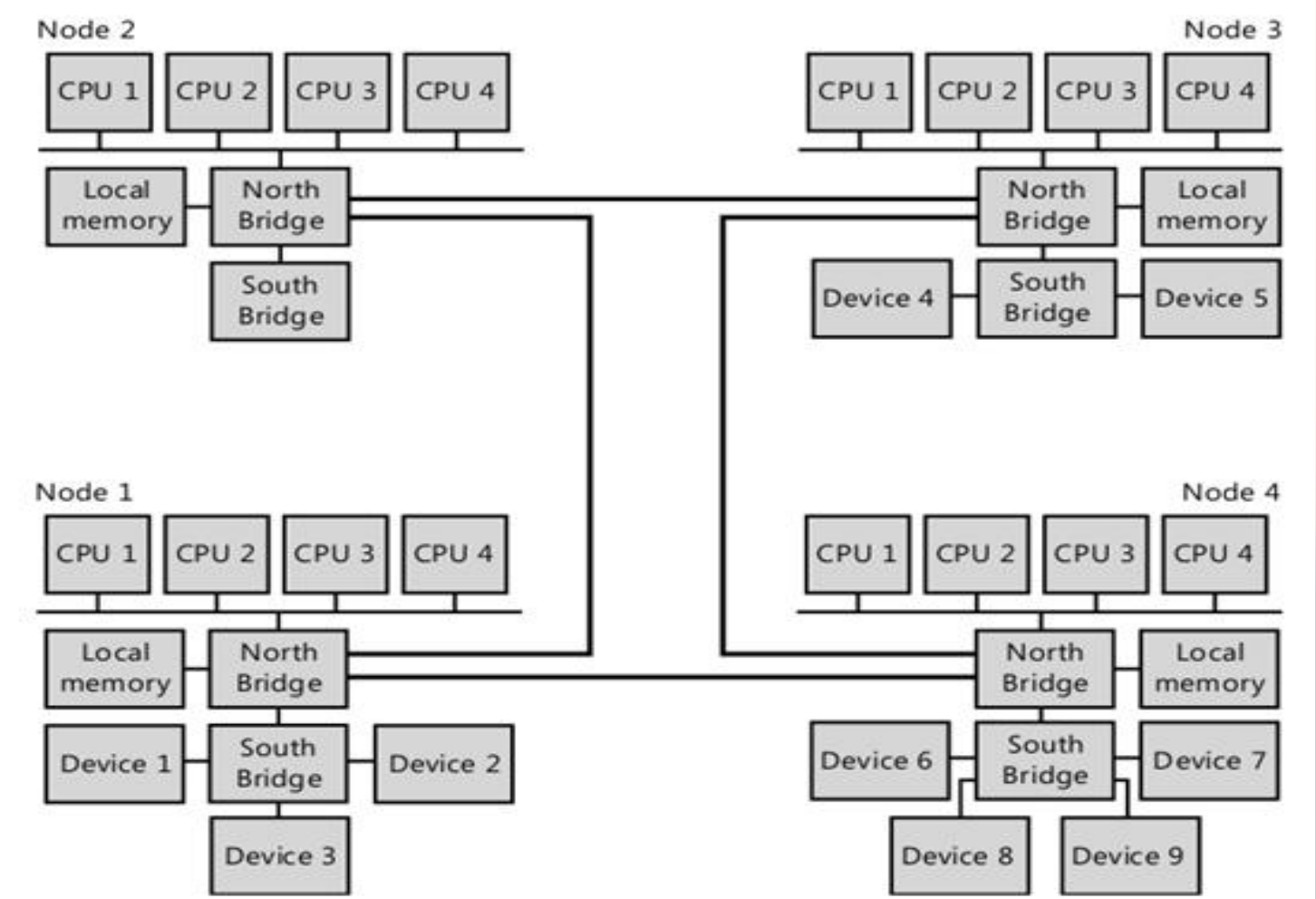


FIGURE 25-3 The architecture of a NUMA computer

五、NUMA架构的机器上的优化

（1）现代服务器基本都是NUMA架构，在这些机器上我们编写程序时最好开启.net的服务端垃圾回收模式，这样我们分配对象时就能在最近使用CPU对应的内存上去分配。

（2）不要在.net程序中使用绑定线程到指定内核或提升线程优先级的做法，因为如果绑定的线程正好与垃圾回收线程进行竞争那么性能会更慢。

六、选择合适的编程模型

1.数据并行

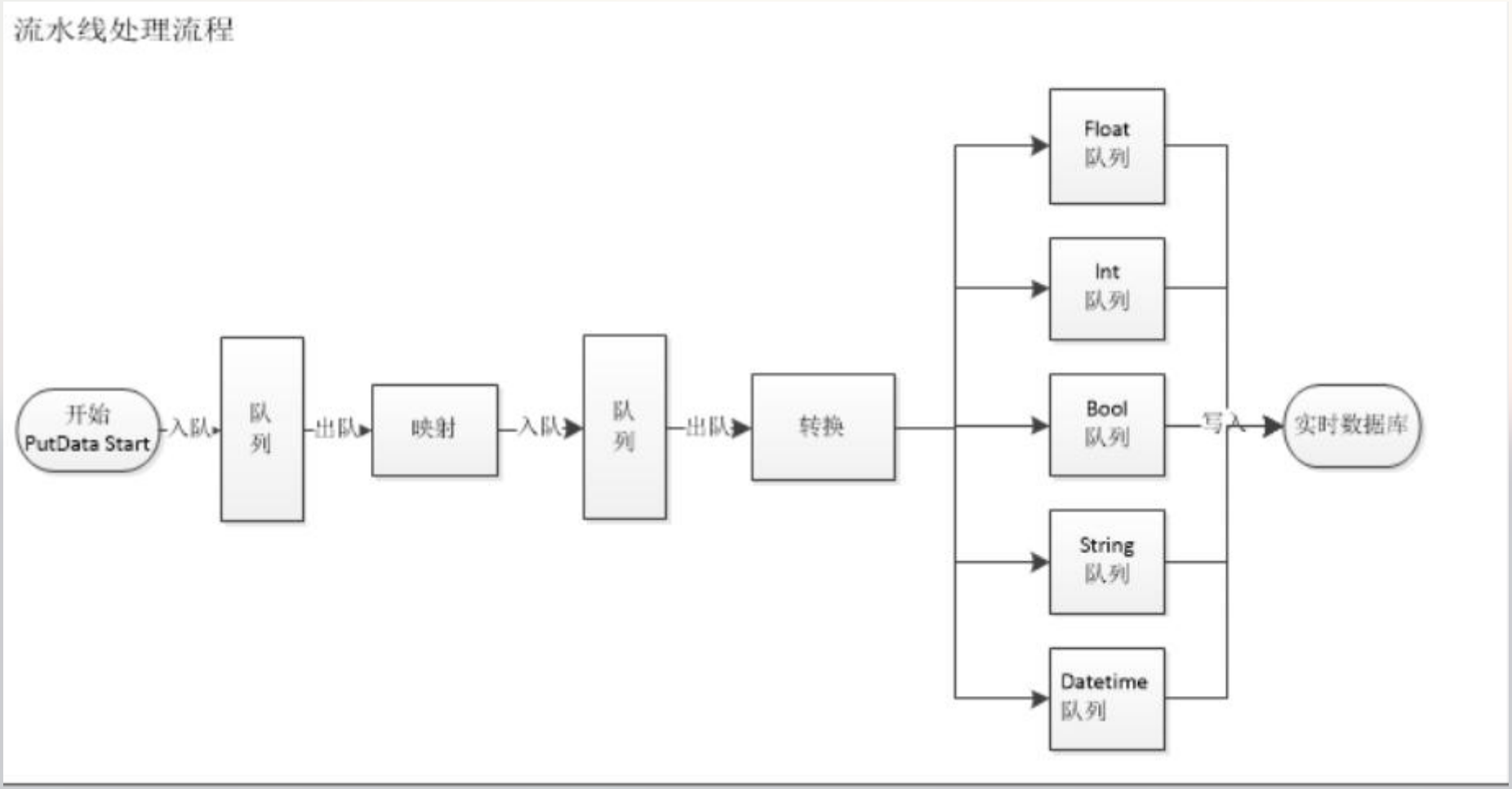
2.任务并行

3.流水线并行

七、数据处理模块设计

提供数据服写入务供上层应用调用，数据写入服务处理的吞吐量要达到60w/s,也就是用户每秒发送60w(18M)的数据然后通过数据写入服务写到数据库中（数据库为公司自主研发的实时数据库）。

七、数据处理模块设计


流水线处理流程
开始
PutData Start
入队
队列
出队
映射
入队
队列
出队
转换
Float
队列
Int
队列
Bool
队列
String
队列
Datetime
队列
写入
实时数据库

七、数据处理模块设计

1.队列的设计

(1) 出队时数据处理的顺序要保证和入队时是一致的

(2) 支持多线程入队出队，尽量简化多线程编程的复杂度。

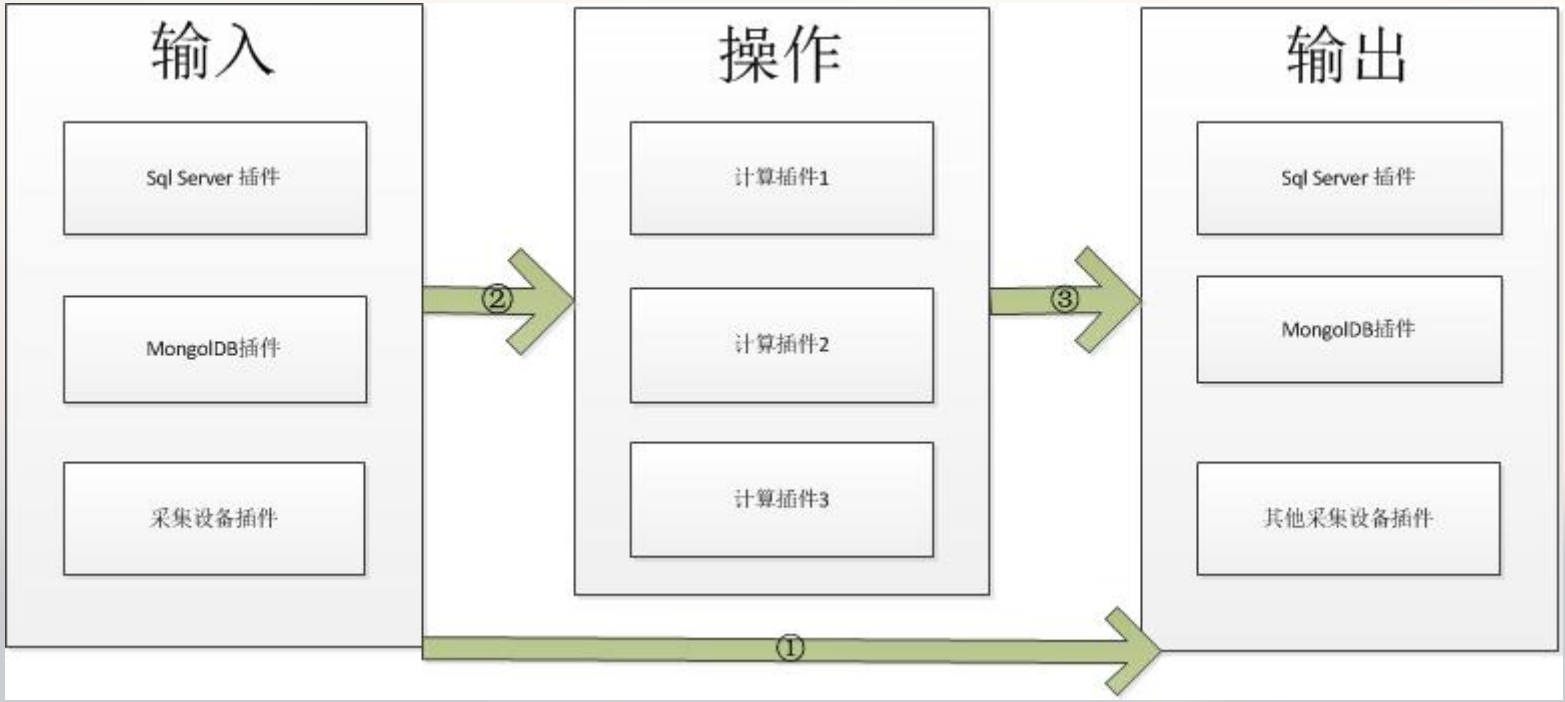
(3)支持事件触发机制，数据入队时才进行处理而不是使用定时处理机制, 而且内部能阻塞消费者线程。

(4)容错性强，可以不间断运行。

2.多线程写日志文件的问题与解决方案

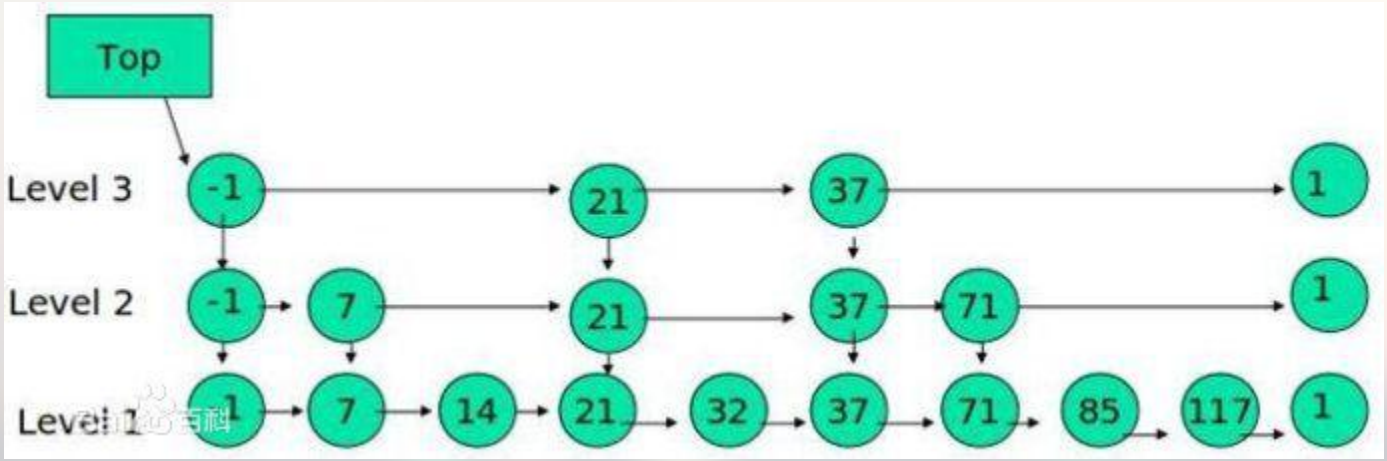
3.减少IO将操作将监控日志输出到DebugView

八、实时采集计算平台的研发



八、实时采集计算平台的研发

1 数据结构的优化：由并行跳跃表改为哈希加双向链表



2 多线程的优化：避免线程之间的数据共享去掉锁