**一：“钨丝计划”产生的本质原因**

1. Spark作为一个一体化多元化的（大）数据处理通用平台，性能一直是其根本性的追求之一，Spark基于内存迭代（部分基于磁盘迭代）的模型极大的满足了人们对分布式系统处理性能的渴望，但是有Spark是采用Scala+ Java语言编写的所以运行在了JVM平台，当然JVM是一个绝对伟大的平台，因为JVM让整个离散的主机融为了一体（网络即OS），但是JVM的死穴GC反过来限制了Spark（也就是说平台限制了Spark），所以Tungsten聚焦于CPU和Memory使用，以达到对分布式硬件潜能的终极压榨！
2. 对Memory的使用，Tungsten使用了Off-Heap，也就是在JVM之外的内存空间（这就好像C语言对内存的分配、使用和销毁），此时Spark实现了自己的独立的内存管理，就避免了JVM的GC引发的性能问题，其实也避免了序列化和反序列化。
3. 对Memory的管理，Tungsten提出了Cache-aware computation，也就是说使用对缓存友好的算法和数据结构来完成数据的存储和复用；
4. 对CPU的使用，Tungsten提出了Code Generation，其首先在Spark SQL使用，通过Tungsten要把该功能普及到Spark的所有功能中；（这里的CG就类似于Android的art模式）。

总结：JAVA操作的数据一般是自己的对象，但像C,C++语言可以操作内存中的二进制数据一样，JAVA同样也可以。**Tungsten的内存管理机制独立于JVM，所以Spark操作数据的时候具体操作的是Binary Data，而不是JVM Object！！！而且还免去了序列化和反序列化的过程！！！！！**

**二：“钨丝计划”内幕详解**

Tungsten sort-based shuffle是对sort-based shuffle的升级，对内存，cpu和cache方面做了很多优化。

1. 内存管理方面：钨丝计划使用了堆外内存，像C和C++的malloc函数一样从os分配内存，不受jvm管理，也就是说大小不受jvm内存大小的限制，销毁的话也不受JVM的管理（自己分配了内存而不销毁的话是os层面的内存泄漏），这就与C和C++语言完全一样了。通过这种机制分配的binary data,不是java的object级别的，故不是jvm内存中的东西，故没有gc问题，也就避免了jvm的死穴：内存管理方面的问题。

* Spark对off-heap进行了封装，Spark使用sun.misc.Unsafe来进行Off-heap级别的内存分配、指针使用及内存释放；如果是On-heap,所有的java object都是由一个64位的object的引用和一个64位的在object中的内容的offset来指定的，在gc时会导致堆内结构的重新组织，若在运行时分配了java object对象，由于其地址不可以改变，不考虑gc的问题，其内存管理的负担也远远大于off-heap。而off-heap给定的地址其实就是c的指针，直接指向数据结构，它的寻址效率跟高。
* Spark的应用开发者访问数据时不关心底层数据是在onheap还是在Offheap，故Spark做了个管理器，管理器可以根据数据是在堆内还是堆外进行具体的寻址。不管在堆内还是堆外，该管理器会完成具体地址寻址到具体数据的映射过程，这就像适配器，适配上面运行的代码和底层的数据，一旦适配，到底是那种方式就无所谓，是透明的，从程序运行时框架的角度来看，你是看不见是在堆内还是在堆外的。spark为了统一管理Off-Heap和On-heap，进行了封装提出了Page的概念。
* 寻址包含2部分，即所谓的page到底是哪个page,以及所谓的offset是多少。堆内和堆外只是算法不同，适配一下而已。Offheap下，内存直接是64位的Long指针来指向具体的数据，如果是onheap,有两部分，即64为的object的引用，以及64位的offset来表示我们的数据。Gc会导致堆的重组，重组要确保我们的object的地址是不变的，故做了page,page其实是个table,会有很多page,通过page table来对内存进行管理，把内存分成很多页，这类似于os的page table。内存寻址分为2部分，第一部分确定在那一页，第二部分确定在page内的具体偏移。
* 将逻辑地址映射到物理地址是钨丝计划管理的内部机制。64位前面的13位标识第几页，后面的51位表示页内部的偏移。Offheap 通过64位bit直接表示绝对的物理地址，onheap则首先确定jvm对象，然后才确定数据具体的偏移量。

1. CacheAwarecomputation机制主要是对基于内存的计算进行优化的方式，cpu cache的L1,L2,L3速度不同, 为了提高cache的命中率，spark设计了自己的数据机构，以前是遍历的方式，现在加了个指针指向了具体的cache的位置，查询的方式变成了KV对的方式，这就是CacheAwarecomputation的基本特征，它极大地提升了aggregation等操作的性能。
2. CG: 即Code Generation，将已有的代码变成本地的字节，不需要很多抽象和匹配。首先在Spark SQL中使用，通过Tungsten要把该功能普及到Spark的所有功能中；（这里的CG就类似于Android的art模式）后续在spark sql中具体讲解。