优秀博文汇总

#学习

后端工程

知识扫盲系列

- GitHub - doocs/advanced-java: 😮 互联网 Java 工程师进阶知识完全扫盲

并发编程篇

伪共享

- Java8使用@sun.misc.Contended避免伪共享 | Idea Buffer

CAS

- 与大部分并发队列使用的Lock相比，CAS显然要快很多。CAS是CPU级别的指令，更加轻量，不需要像Lock一样需要OS的支持，所以每次调用不需要kernel entry，也不需要context switch
- AtomicStampedReference 解决**ABA**问题
- cas 机制实现
- 彻底理解synchronized - 掘金
- Compare-and-swap - Wikipedia

并发优秀文章

- 从 LongAdder 中窥见并发组件的设计思路 | 犀利豆的博客

并发框架源码解析系列文章

- akka 系列
  - AKKA 2.3.6 Scala 文档
- 线程池系列
  - FutureTask源码解析 | Idea Buffer
  - 深入理解Java线程池：ThreadPoolExecutor | Idea Buffer
  - 深入理解Java线程池：ScheduledThreadPoolExecutor | Idea Buffer
- AbstractQueuedSynchronizer系列
  - 深入理解AbstractQueuedSynchronizer（一） | Idea Buffer
  - 深入理解AbstractQueuedSynchronizer（二） | Idea Buffer
  - 深入理解AbstractQueuedSynchronizer（三） | Idea Buffer
- Tomcat系列
  - Tomcat源码：类加载器 | Idea Buffer
- 并发容器剖析系列
  - 并发容器之ConcurrentLinkedQueue - 掘金
  - concurrent hashmap https://blog.csdn.net/lsgqjh/article/details/54867107
- 锁系列

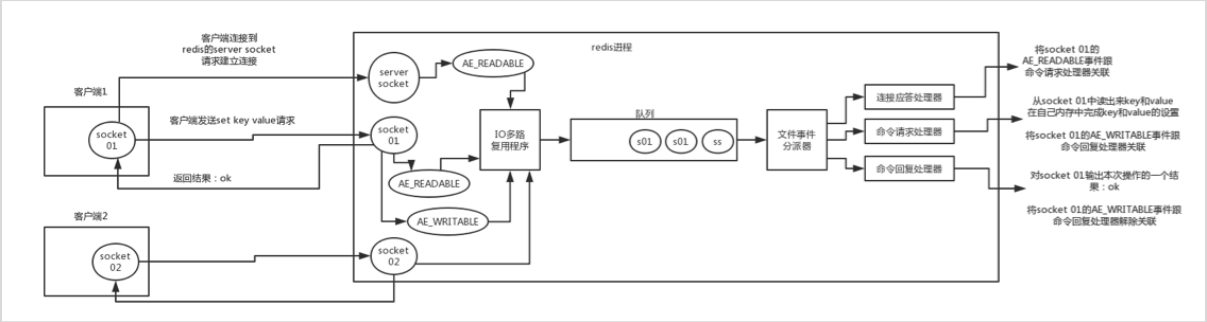
    - java 中的锁 – 偏向锁、轻量级锁、自旋锁、重量级锁 | Idea Buffer
    - 七张图彻底讲清楚ZooKeeper分布式锁的实现原理【石杉的架构笔记】 - 掘金
- Netty系列
    - 工作原理

> - 一旦用户端连接成功，将新建一个channel同该用户端进行绑定
> - channel从EventLoopGroup获得一个EventLoop，并注册到该EventLoop，channel生命周期内都和该EventLoop在一起（注册时获得selectionKey）
> - channel同用户端进行网络连接、关闭和读写，生成相对应的event（改变selectinKey信息），触发eventloop调度线程进行执行
> - 如果是读事件，执行线程调度pipeline来处理用户业务逻辑

    - 基础概览
        - 阻塞IO，非阻塞 IO 和异步 IO概念
        - NIO技术概览 | Idea Buffer
        - Netty4源码分析-Bootstrap | Idea Buffer
        - Netty4源码分析-ServerBootstrap | Idea Buffer
        - Netty4源码分析-Channel | Idea Buffer
        - AIO写文件的OutOfMemoryError | Idea Buffer
        - 自顶向下深入分析Netty（七）--ChannelPipeline和ChannelHandler总述 - 简书
        - **netty 零拷贝实现**【转】认真分析mmap：是什么 为什么 怎么用 | Idea Buffer
        - 从内核文件系统看文件读写过程 - 胡潇 - 博客园
        - Netty零拷贝三个方面（Netty权威指南第二版487页）
            - 接收和发送bytebuffer
            - compositeByteBuf，将多个bytebuf封装成一个
            - 文件传输
    - 性能分析
        - Netty 长连接服务性能分析
        - 好文：Netty精粹之基于EventLoop机制的高效线程模型 - 天空之家 - 博客园
    - Reactor线程模型（Netty权威指南第二版483页）
        - 单线程模型
        - 多线程模型
        - 主从Reactor模型

- *Kafka* 系列
    - low api就是低成本，不需要自己手动管理提交等，相反high level api就是高成本
    - auto commit机制
        - autocommit机制
        - Kafka消息消费一致性 - 乱世浮生
    - *kafka*源码分享博主技术 | Matt's Blog

    - Spark streaming 基础--使用低阶API消费Kafka数据(手动更新offset) - 程序园
- *Redis*系列
    - 深入学习Redis系列文章



    - 微博分布式缓存构建 微博到底有多重视分布式缓存 - - ITeye博客
- *Mysql*系列
    - mysql 幻读的详解、实例及解决办法
        - 行锁锁定的是索引，而不是记录
        - RR 手动加 select for update，即便当前记录不存在，当前事务也会获得一把记录锁（因为InnoDB的行锁锁定的是索引，故记录实体存在与否没关系，存在就加 行X锁，不存在就加 next-key lock间隙X锁），其他事务则无法插入此索引的记录，故杜绝了幻读。
        - 所以 mysql 的幻读并非什么读取两次返回结果集不同，而是事务在插入事先检测不存在的记录时，惊奇的发现这些数据已经存在了，之前的检测读获取到的数据如同鬼影一般
        - 不可重复读侧重表达 读-读，幻读则是说 读-写，用写来证实读的是鬼影
    - MVCC
        - Innodb 默认RR的事务隔离级别导致读取的数据永远是事务开始的时候的版本，不管mvcc情况下某一行数据是否被更新了；而RC隔离级别就能读取到最新的状态
        - 事务以排他锁的形式修改原始数据；把修改前的数据存放于undo log，通过回滚指针与主数据关联

Disruptor

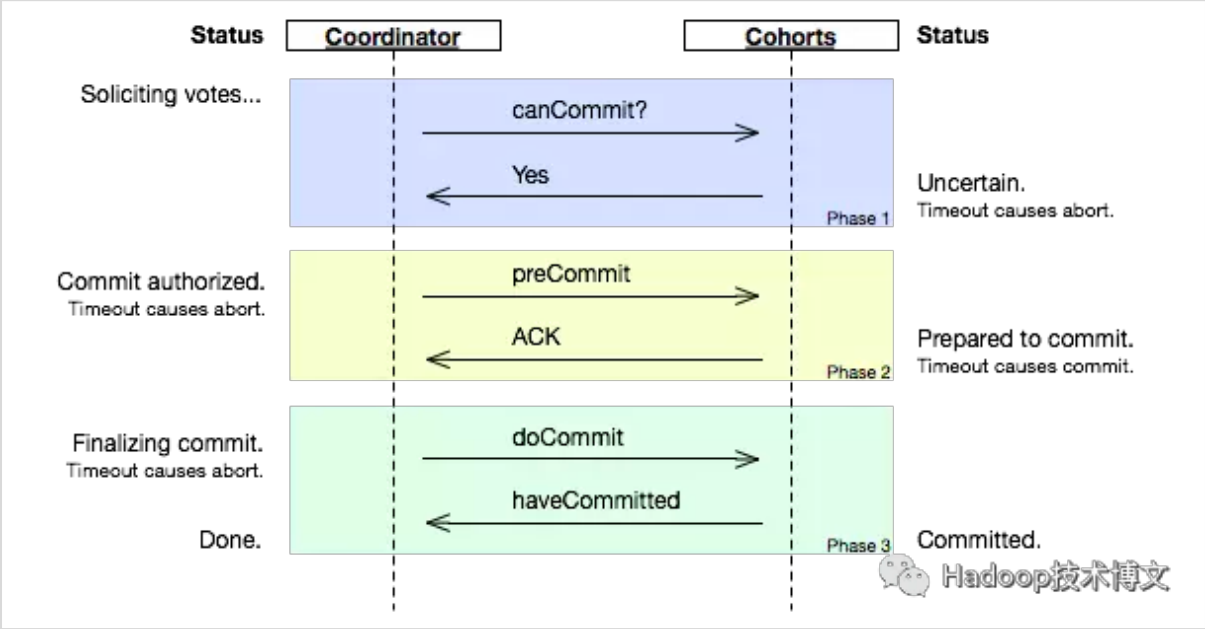
- 相关博文
    - Disruptor使用指南
    - log4j2使用了disruptor: log4j2性能剖析 - 白中墨的个人空间 - 开源中国
    - 美团点评：高性能队列——Disruptor -

并发编程优秀博主

- 归档 | Idea Buffer

分布式

- 分布式事务的实现原理
- *tcc*
  - FESCAR：阿里重磅开源分布式事务解决方案
  - 一个tcc例子
  - 终于有人把"TCC分布式事务"实现原理讲明白了！ - 51CTO.COM
- 2phase 3phase提交
  - 对分布式事务及两阶段提交、三阶段提交的理解 - Andy奥 - 博客园
  - CSDN-专业IT技术社区
  - 关于分布式事务、两阶段提交协议、三阶段提交协议 - bcombetter - 博客园



- Raft
  - Raft协议详解 - 知乎
  - Raft动态示意各个流程
- Paxos
  - 分布式一致性算法——paxos-HollisChuang's Blog
  - 如何浅显易懂地解说 Paxos 的算法？ - 知乎
  - Zookeeper之分布式系统的一致性算法 - 知乎
- 布隆过滤器 海量数据处理算法- CSDN博客

分布式优秀博主

- Martin Kleppmann's blog

JVM

- JAVA ASM字节码框架
  - JAVA ASM字节码框架 | BruceFan's Blog

垃圾收集器

- G1
  - 可能是最全面的G1学习笔记 - 简书
  - R大关于G1的帖子
  - 美团： Java Hotspot G1 GC的一些关键技术 -
- CMS
  - 不可错过的CMS学习笔记 | 并发编程网 – ifeve.com

Java框架系列

Spring

- Jpa
  - 介绍了jpa防踩坑姿势，也说明反对jpa的理由

Tomcat

- Servlet 3
  - servlet3异步原理与实践 - 简书
- tomcat
  - Tomcat源码剖析 - rhwayfun专栏

数据结构系列

- 以后有面试官问你「跳跃表」，你就把这篇文章扔给他

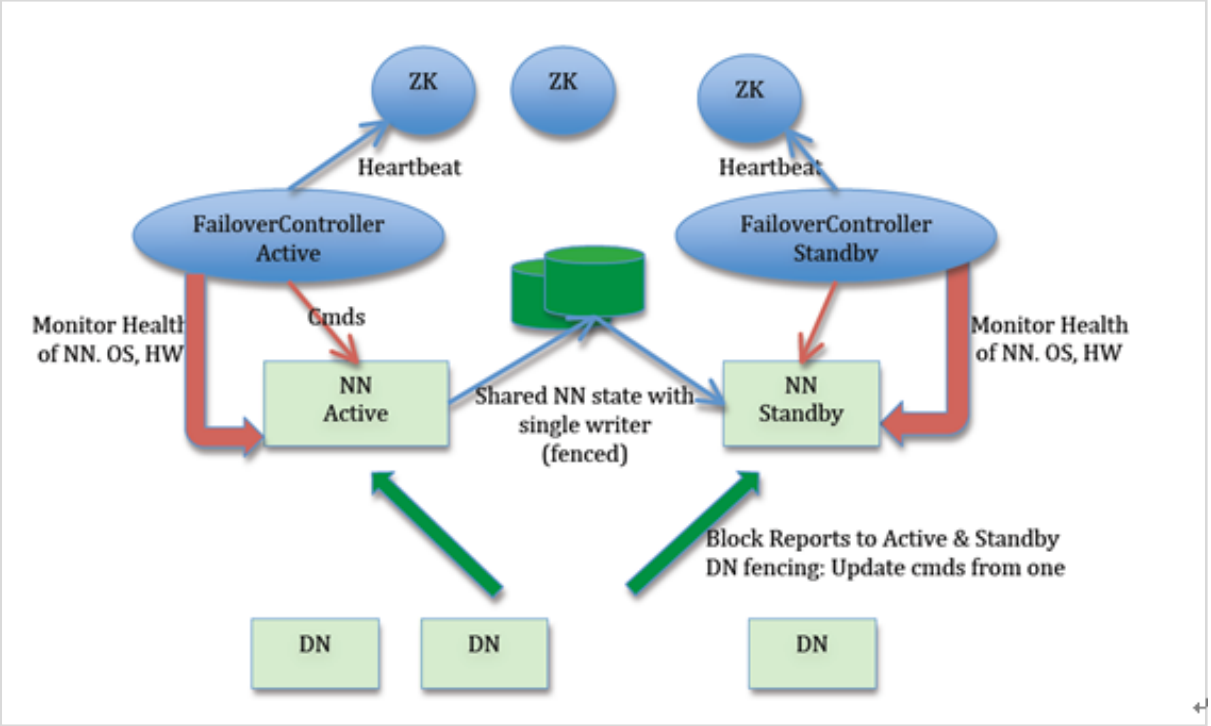
大数据

Hadoop相关

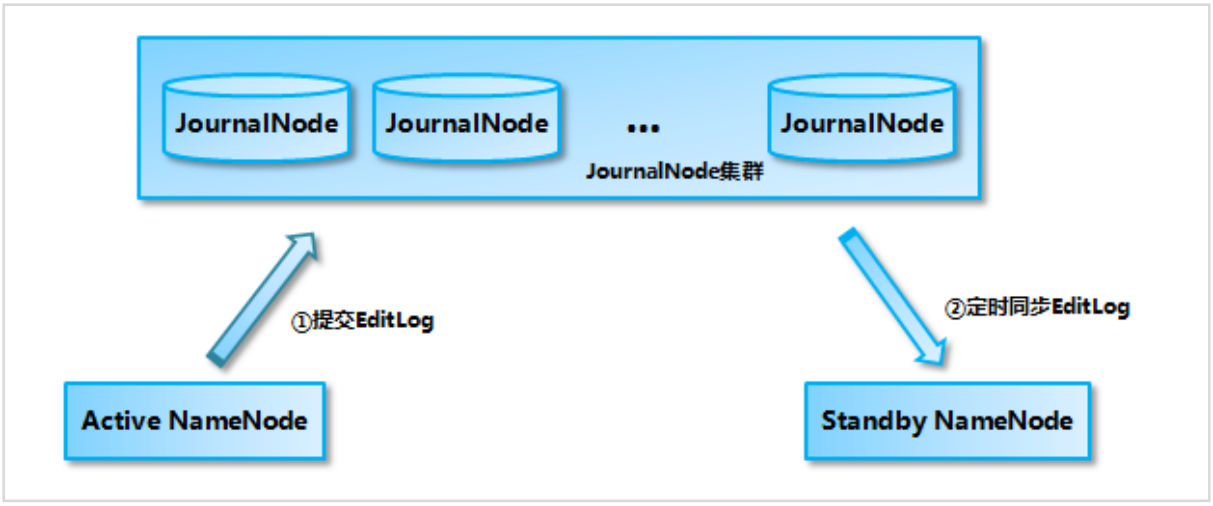
- Hadoop 3.0
  - Hadoop 3.x 新特性剖析系列2 - 掃文資訊
  - *HDFS Erasure Coding*
    - 冷热数据备份方式的改变再聊HDFS Erasure Coding - 走在前往架构师的路上 - CSDN博客
    - Hadoop 3.0 Erasure Coding 纠删码功能预分析 - 走在前往架构师的路上 - CSDN博客
    - hdfs 3.0 纠删码运维手册
    - HDFS EC：将纠删码技术融入HDFS

> 纠删码则可以在同等可用性的情况下，节省更多的空间，以`rs-6-3-1024K`这种纠删码策略为例子，6份原始数据，编码后生成3份校验数据，一共9份数据，只要最终有6份数据存在，就可以得到原始数据，它可以容忍任意3份数据不可用，而冗余的空间只有原始空间的`0.5`倍，只有副本方式的`1/4`

- 架构
  - **跨机房** 美团点评数据平台融合实践
- HDFS
  - 再议HDFS写流程之pipeline | big data decode club
  - 向量化执行引擎简介
  - **nn/rm切主** HDFS 和 YARN 的 HA 故障切换
  - Namenode内存分析 | Haihua's blog
  - *namenode* 调优 NameNode Garbage Collection Configuration: Best Practices and Rationale - Hortonworks
  - Hadoop NameNode 高可用 (High Availability) 实现解析



  - QJM共享存储，基于paxos算法来保存editslog，采用多个JournalNode



- YARN
  - 别被官方文档迷惑了！这篇文章帮你详解yarn公平调度 - 腾讯云+社区 - 博客园
  - Yarn原理系列文章
  - Yarn资源请求处理和资源分配原理解析 - 小昌昌的博客 - CSDN博客
  - 董的博客 RMContainer状态机分析
  - Yarn FairScheduler 的资源预留机制导致的一次宕机事故分析 - 小昌昌的博客 - CSDN博客
    - Yarn对预留的处理，只是一种标记，即，将某个服务器标记为被某个container预留。任何一个被某个container预留的服务器，在取消预留或者被预留的container被成功分配之前，这个节点不会为其它container分配资源，只会一次次尝试为这个预留的container分配资源。而正常的没有被预留的服务器节点，则是根据我们定义的优先级(一个资源队列节点的孩子节点之间的优先级、同一个队列节点中应用的优先级、应用内container的优先级)选择出一个资源请求，在这个节点上分配一个container运行这个资源请求。
    - YARN 一个JDK的bug导致RM无法分配Container - 张伟的专栏 - CSDN博客
    - YARN-Fair Scheduler allocation 参数含义
  - *RM*重启高可用Apache Hadoop 2.9.2 – ResourceManger Restart
    - NodeManagers and clients during the down-time of RM will keep polling RM until RM comes up
  - Capacity Scheduler VS Fair Scheduler
    - Capacity Scheduler - vs - Fair Scheduler - 简书
- 任务监控工具
  - **Dr.Elephant** Dr.Elephant用户指南
  - Dr. Elephant 介绍 | Legendtkl

Spark 相关

文档

- dataset 相关操作

- Spark动态资源分配 Dynamic Resource Allocation – lxw的大数据田地
- Spark 动态资源分配(Dynamic Resource Allocation) 解析 - 云+社区 - 腾讯云
- *kafka producer one instance per jvm*Spark and Kafka integration patterns · allegro.tech

checkpoint 机制

- RDD之七：Spark容错机制 - duanxz - 博客园
  - 局限：application 重新编译之后，再去反序列化checkpoint数据就会失效；spark streaming + kafka中可以通过自行维护消费的offsets来保证消费的进度

shuffle 机制

- Spark Shuffle的技术演进 - 简书
  - Hash Based Shuffle
  - Sort Based Shuffle

> shuffle共有三种，别人讨论的是hash shuffle，这是最原始的实现，曾经有两个版本，第一版是每个map产生r个文件，一共产生m*r*个文件，*由于产生的中间文件太大影响扩展性，社区提出了第二个优化版本，让一个core上map共用文件，减少文件数目，这样共产生core*r个文件，好多了，但中间文件数目仍随任务数线性增加，仍难以应对大作业，但hash shuffle已经优化到头了。为了解决hash shuffle性能差的问题，又引入sort shuffle，完全借鉴mapreduce实现，每个map产生一个文件，彻底解决了扩展性问题

- Spark Sort Based Shuffle内存分析 - 简书
- Spark基础以及Shuffle实现分析

> 窄依赖和宽依赖的分类是Spark中很重要的特性，不同依赖在实现，任务调度机制，容错恢复上都有不同的机制。
>
> - 实现上：对于窄依赖，rdd之间的转换可以直接pipe化，而宽依赖需要采用shuffle过程来实现。
> - 任务调度上：窄依赖意味着可以在某一个计算节点上直接通过父RDD的某几块数据（通常是一块）计算得到子RDD某一块的数据； 而相对的，宽依赖意味着子RDD某一块数据的计算必须等到它的父RDD所有数据都计算完成之后才可以进行，而且需要对父RDD的计算结果需要经过shuffle才能被下一个rdd所操作。
> - 容错恢复上：窄依赖的错误恢复会比宽依赖的错误恢复要快很多，因为对于窄依赖来说，只有丢失的那一块数据需要被重新计算， 而宽依赖意味着所有的祖先RDD中所有的数据块都需要被重新计算一遍，这也是我们建议在长"血统"链条特别是有宽依赖的时候， 需要在适当的时机设置一个数据检查点以避免过长的容错恢复。

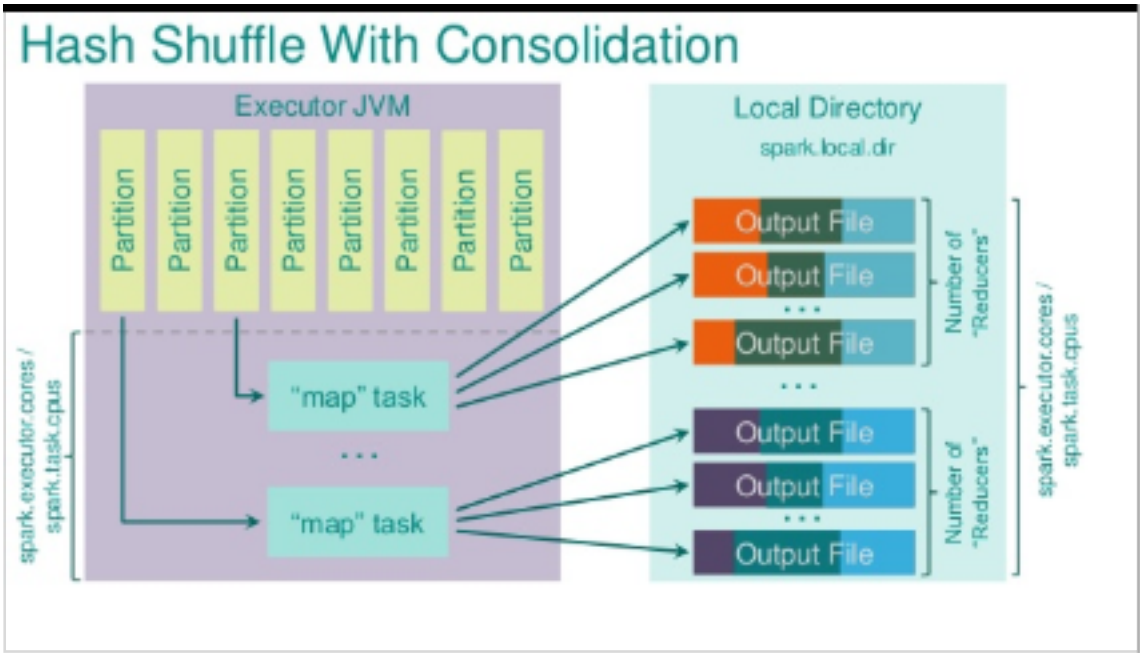
底层原理

- 大数据底层原理剖析*Spark/Hadoop* GitHub - ColZer/DigAndBuried: 挖坑与填坑
- Spark RDD容错性 RDD之七：Spark容错机制 - duanxz - 博客园
- *spark* 内存管理Tuning Java Garbage Collection for Apache Spark Applications - The Databricks Blog
- 深入Spark底层设计SparkInternals
- 深入理解SparkSql底层 (Legacy)
- Tuning Java Garbage Collection for Apache Spark Applications - The Databricks Blog
- Apache Spark 统一内存管理模型详解
- Spark Streaming 与 kafka 0.8 的版本结合不支持动态分区检测 与Kafka 0.10 的版本支持; flink

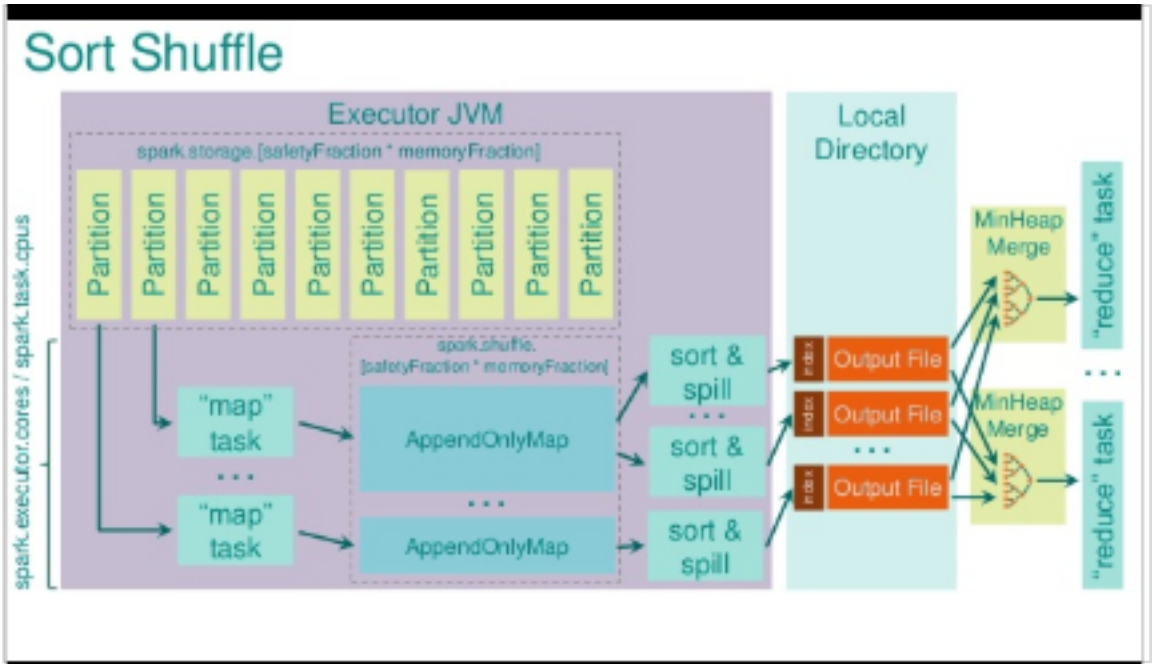
支持动态检测新增的分区

```
复制代码
/**
* Returns the latest (highest) available offsets, taking new partitions into account.
 protected def latestOffsets(): Map[TopicPartition, Long] = {    val c = consumer
   paranoidPoll(c)    // 获取所有的分区信息
   val parts = c.assignment().asScala    // make sure new partitions are reflected in cu
   // 做差获取新增的分区信息
   val newPartitions = parts.diff(currentOffsets.keySet)    // position for new partitio
   // 新分区消费位置，没有记录的化是由auto.offset.reset决定
   currentOffsets = currentOffsets ++ newPartitions.map(tp => tp -> c.position(tp)).toMa
   c.pause(newPartitions.asJava)    // find latest available offsets
   c.seekToEnd(currentOffsets.keySet.asJava)
   parts.map(tp => tp -> c.position(tp)).toMap
 }
```

- Spark中的join实现原理
  - HashJoin 和 Sort Join
- *spark shuffle*设计原理 Spark Architecture: Shuffle | Distributed Systems Architecture
  - 对各种参数讲的很细Spark技术内幕：Shuffle的性能调优
  - 彻底搞懂spark的shuffle过程（shuffle write） - 大葱拌豆腐 - 博客园
  - hash shuffle with consolidation

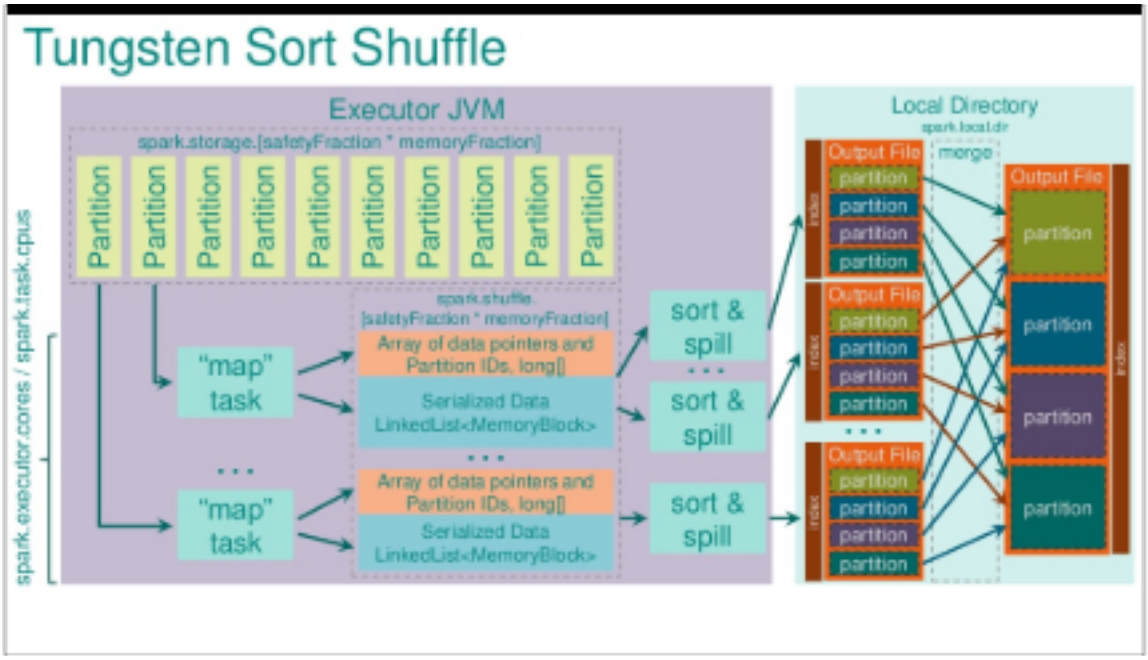


  - sort shuffle



- 每个Spill File是单独的写入disk，不会像Hadoop Mapper那样会on disk merge，只是在数据被reducer请求的时候通过MinHeap实时的merge

- Tungsten Sort Shuffle



- Spark 1.6 Tungsten-sort并入Sort Based Shuffle

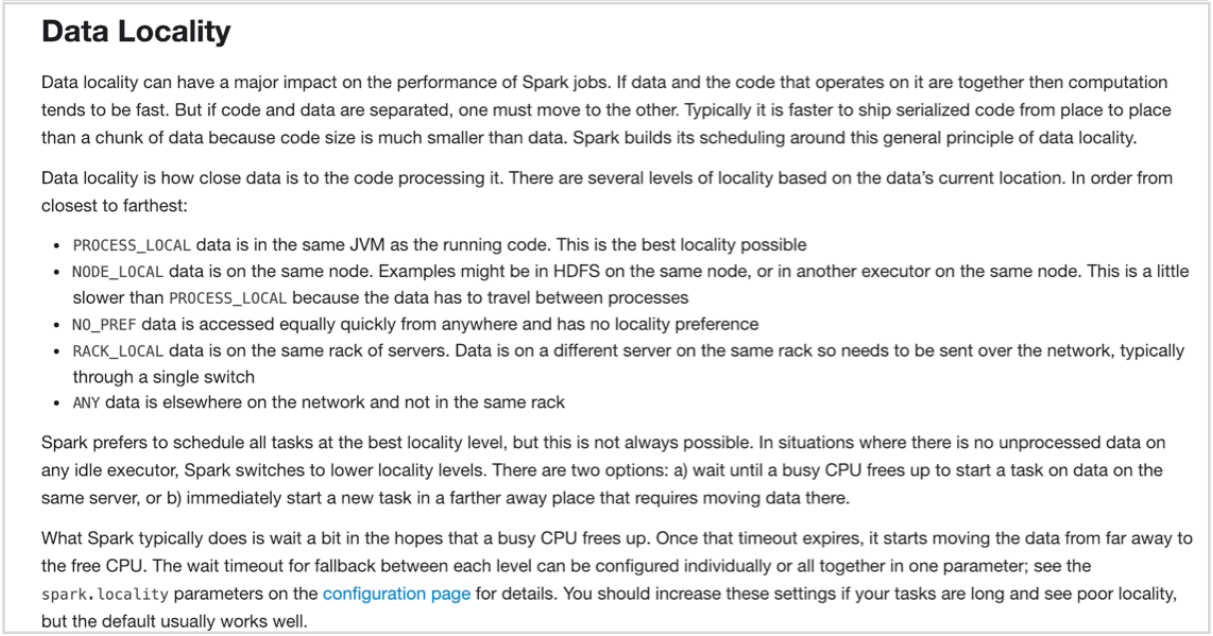
由SortShuffleManager自动判断选择最佳Shuffle方式，如果检测到满足Tungsten-sort条件会自动采用Tungsten-sort Based Shuffle，否则采用Sort Based Shuffle

- Spark Shuffle之Tungsten-Sort 无法用于map端的aggregator
- Spark Tungsten揭秘

- spark 计算本地性
  - 谈谈spark 的计算本地性 | sunbiaobiao
  - 数据本地性调度原理

- taskscheduler会按照本地性原则发送task，如果executor没有响应，会选择降级成低一个级别的数据本地性，如process local降级为node local，以此类推

> 假如 一个 task 要处理的数据，在上一个 stage 中缓存下来了， 这个 task 期望的 就是以 PROCESS_LOCAL 来运行， 这个时候缓存数据的executor 不巧正在执行 其他的task， 那么我就等一会， 等多长时间呢， spark.locality.wait.process这么长时间， 如果时间超了， executor 还是没有空闲下来， 那么我没有办法， 我就以NODE_LOCAL 来运行 task， 这个时候我想到 同一台机器上其他 executor 上跨jvm 去拉取数据， 如果同一台机器上有其他空闲的 executor 可以满足， 就这么干， 如果没有， 等待 spark.locality.wait.node 时间， 还没有就以更低的 Locality Level 去执行这个 task



Data Locality

Data locality can have a major impact on the performance of Spark jobs. If data and the code that operates on it are together then computation tends to be fast. But if code and data are separated, one must move to the other. Typically it is faster to ship serialized code from place to place than a chunk of data because code size is much smaller than data. Spark builds its scheduling around this general principle of data locality.

Data locality is how close data is to the code processing it. There are several levels of locality based on the data's current location. In order from closest to farthest:

- `PROCESS_LOCAL` data is in the same JVM as the running code. This is the best locality possible
- `NODE_LOCAL` data is on the same node. Examples might be in HDFS on the same node, or in another executor on the same node. This is a little slower than `PROCESS_LOCAL` because the data has to travel between processes
- `NO_PREF` data is accessed equally quickly from anywhere and has no locality preference
- `RACK_LOCAL` data is on the same rack of servers. Data is on a different server on the same rack so needs to be sent over the network, typically through a single switch
- `ANY` data is elsewhere on the network and not in the same rack

Spark prefers to schedule all tasks at the best locality level, but this is not always possible. In situations where there is no unprocessed data on any idle executor, Spark switches to lower locality levels. There are two options: a) wait until a busy CPU frees up to start a task on data on the same server, or b) immediately start a new task in a farther away place that requires moving data there.

What Spark typically does is wait a bit in the hopes that a busy CPU frees up. Once that timeout expires, it starts moving the data from far away to the free CPU. The wait timeout for fallback between each level can be configured individually or all together in one parameter; see the `spark.locality` parameters on the configuration page for details. You should increase these settings if your tasks are long and see poor locality, but the default usually works well.

Streaming

- Spark Streaming 数据清理机制 , Dstream.cache与Rdd.cache的区别
- *spark kafka offset* 管理Spark & Kafka - Achieving zero data-loss
- Structured Streaming: 用户可以使用 Dataset/DataFrame 或者 SQL 来对这个动态数据源进行实时查询
  - 官方文档Structured Streaming Programming Guide - Spark 2.4.0 Documentation
  - Structured Streaming-infoq
  - Introducing Low-latency Continuous Processing Mode in Structured Streaming in Apache Spark 2.3 - The Databricks Blog
  - Processing Data in Apache Kafka with Structured Streaming

Spark && Kafka exactly once

- Spark Streaming 中如何实现 Exactly-Once 语义
- Spark Streaming消费kafka使用及原理 - 知乎
- Spark Streaming 的 Checkpoint 特性如果因为代码变更，checkpoint数据也无法使用，所以

还是要手动管理offsets

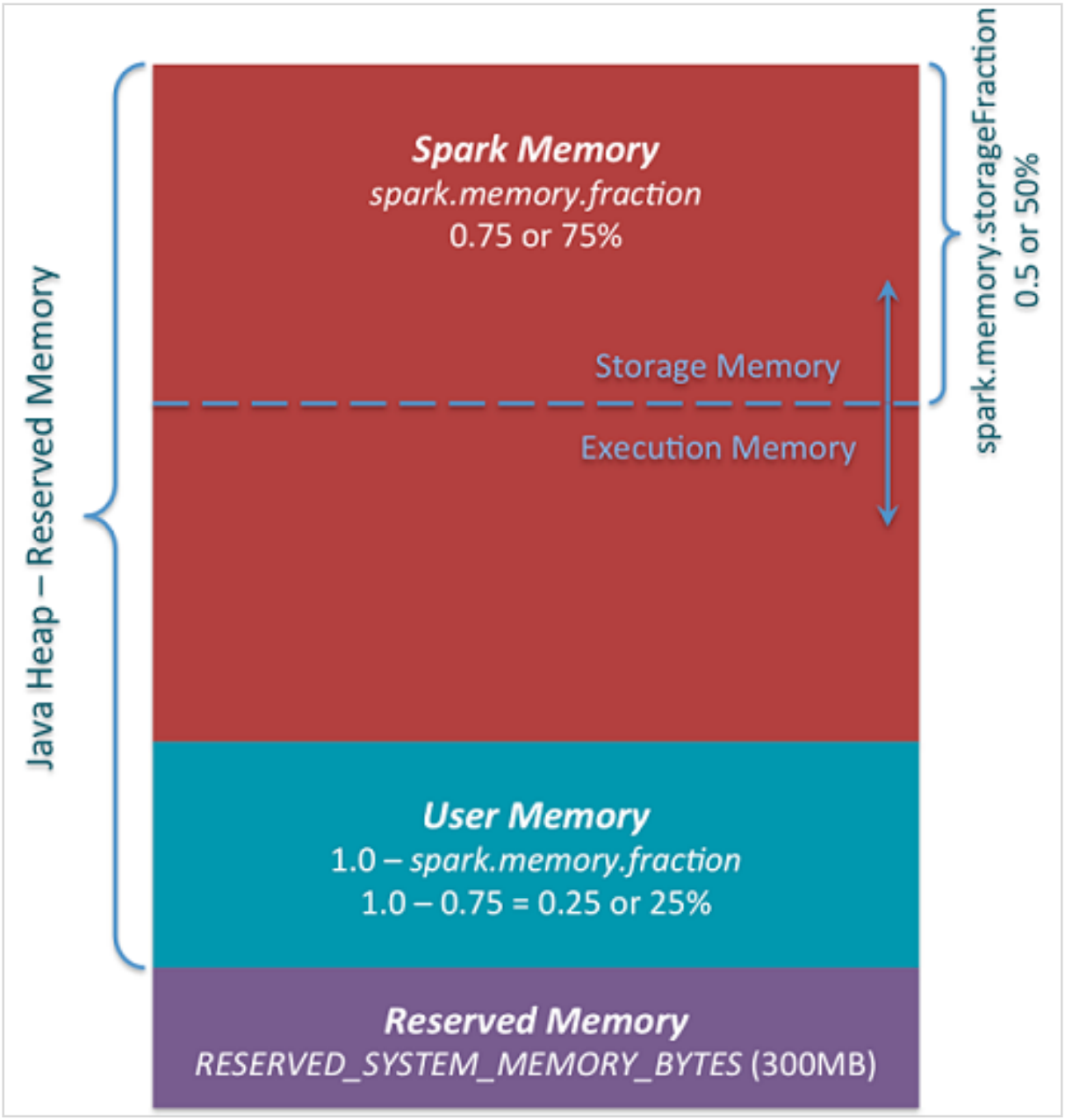
- 业务代码实现exactly once
  - 对于没有shuffle的情况： foreachpartition中每个partition生成一个全局唯一的ID，通过这个id关联这一批数据的执行结果，从而做到原子性
  - 有shuffle的情况：需要将处理结果拉到driver端进行事务的操作

性能优化

- *spark* 官方优化指南 Tuning - Spark 2.4.0 Documentation
- Optimizing Spark jobs for maximum performance
- 解决Spark数据倾斜（Data Skew）的N种姿势
- **whole stage code gen**
  - Spark-WholeStageCodeGen源码学习笔记
  - **背景（重要）**Apache Spark as a Compiler: Joining a Billion Rows per Second on a Laptop - The Databricks Blog
  - Spark 2.x - 2nd generation Tungsten Engine | spark-notes
  - 【Spark】Spark性能优化之Whole-stage code generation - Supernova的博客 - CSDN博客

内存管理

- Apache Spark 统一内存管理模型详解 – 过往记忆
- Spark内存管理



Flink相关

- Doc
  - *Flink Forward 2018*大会资料下载 - 过往记忆
  - Apache Flink 1.7 Documentation
  - **好文** Spark Streaming VS Flink - 掘金
  - **详解分区** Flink 的 HDFS Connector - 简书
  - **checkpoint 优化** Apache Flink 1.7 Documentation: Tuning Checkpoints and Large State
- DataSet
  - 不得不会的Flink Dataset的Deltal 迭代操作
- checkpoint 实现机制：Chandy-Lamport算法实现

    - Apache Flink状态管理和容错机制介绍 – 过往记忆
    - Flink 状态管理与checkPoint数据容错机制深入剖析
    - checkpoint barrier 对齐机制：借助input buffer
- Flink with kafka
    - Flink 如何管理 Kafka 消费位点
    - **kafka in flink exactly once详解** FlinkKafkaConsumer使用详解
    - **flink on yarn 详解**Flink1.6 - flink on yarn分布式部署架构
    - **如何做到Flink与Kafka exactly once** An Overview of End-to-End Exactly-Once Processing in Apache Flink (with Apache Kafka, too!)
        - 中文翻译版Apache Flink 端到端（end-to-end）Exactly-Once特性概览 （翻译） - moyiguke的个人空间 - 开源中国
- 内核 &内存管理
    - Flink 原理与实现：内存管理 | Jark's Blog
    - Flink 原理与实现：内存管理 - 简书
    - 深入理解Flink核心技术 - 知乎
    - 深入理解Flink内存实现机制
    - Flink序列化框架分析_慕课手记
- Table SQL
    - Flink Table—— Dynamic Table
    - 浅析 Flink Table/SQL API - 掘金
- Broadcast state机制
    - A Practical Guide to Broadcast State in Apache Flink
    - Flink Broadcast 广播变量应用案例实战-Flink牛刀小试 - 掘金

HBase相关:

- HBase是CP的，2.0以后由于region replica，对a的容忍度也变高了
    - 深入解读HBase2.0新功能之高可用读Region Replica

Palo相关

- 设计理念 浅谈从Google Mesa到百度PALO |
- PALO支持分区剪枝（Partition pruning），支持bloomfilter做某列的索引，同时Index中会存储MIN/MAX等基本信息，方便做Predicate pushdown谓词下推

优秀博主

- 专注大数据领域博客
- 专注Flink原理实现
- Flink优化系列
- 各大大数据框架原理: 挖坑与填坑
- Flink学习博客

- 艾叔编程|大数据学习网|大数据学习的精选知识仓库

架构

- 亿级流量架构系列专栏总结【石杉的架构笔记】 - 掘金

Devops

Docker

- Docker 异常总结

操作系统

- *CPU load*问题 Linux系统下CPU使用(load average)梳理

编程概念

函数式思想

- 理解monad
  - 我所理解的monad(0) | 写点什么
  - Scala日记——Monad的概念简介 - 饥渴计科极客杰铿的博客 - CSDN博客
  - 图解 Monad - 阮一峰的网络日志