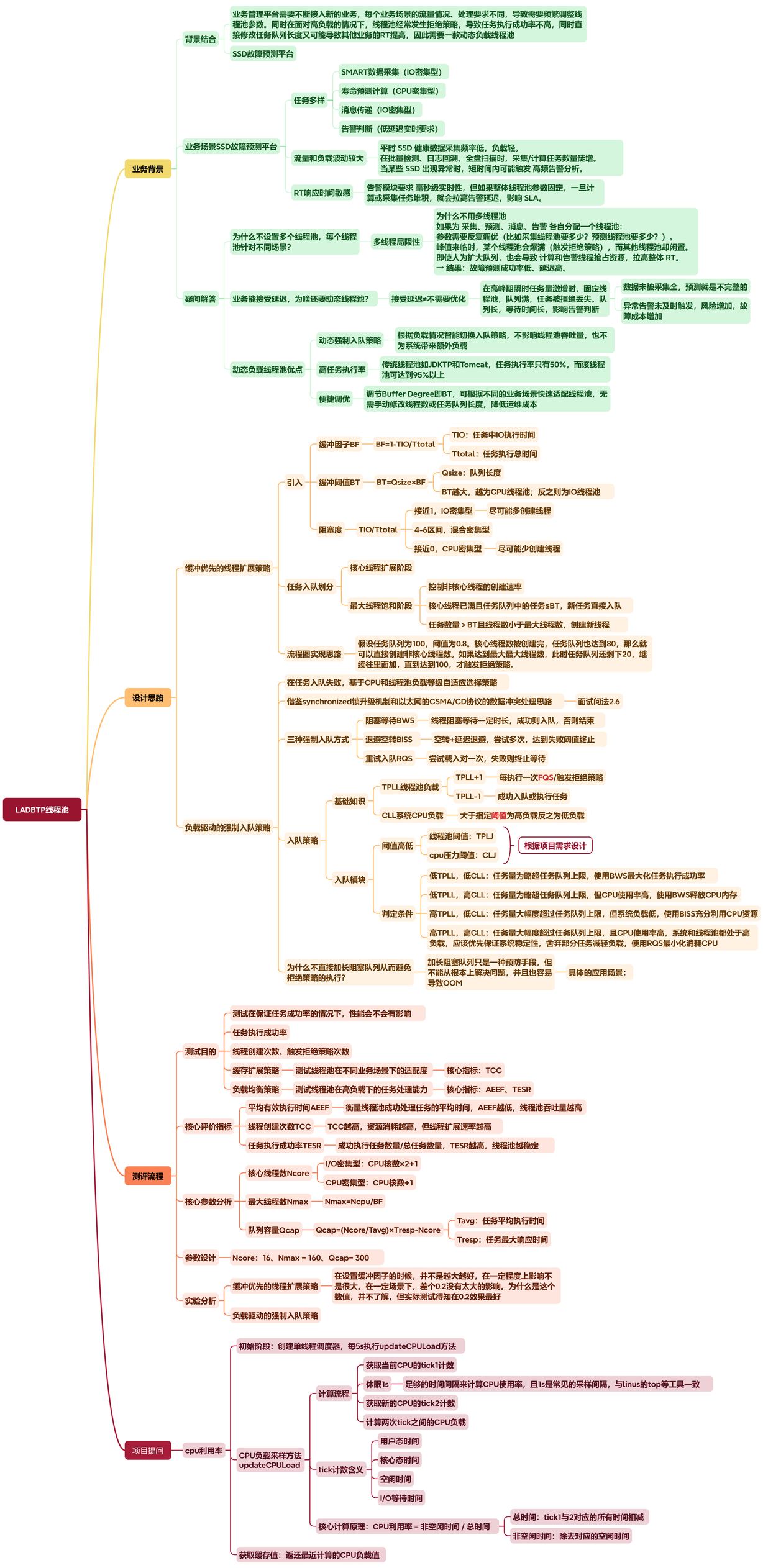
LADBTP线程池
业务背景
背景结合
业务管理平台需要不断接入新的业务，每个业务场景的流量情况、处理要求不同，导致需要频繁调整线程池参数。同时在面对高负载的情况下，线程池经常发生拒绝策略，导致任务执行成功率不高，同时直接修改任务队列长度又可能导致其他业务的RT提高，因此需要一款动态负载线程池
SSD故障预测平台
业务场景SSD故障预测平台
任务多样
SMART数据采集（IO密集型）
寿命预测计算（CPU密集型）
消息传递（IO密集型）
告警判断（低延迟实时要求）
流量和负载波动较大
平时 SSD 健康数据采集频率低，负载轻。
在批量检测、日志回溯、全盘扫描时，采集/计算任务数量陡增。
当某些 SSD 出现异常时，短时间内可能触发 高频告警分析。
RT响应时间敏感
告警模块要求 毫秒级实时性，但如果整体线程池参数固定，一旦计算或采集任务堆积，就会拉高告警延迟，影响 SLA。
疑问解答
为什么不设置多个线程池，每个线程池针对不同场景?
多线程局限性
为什么不用多线程池
如果为 采集、预测、消息、告警 各自分配一个线程池：
参数需要反复调优（比如采集线程池要多少？预测线程池要多少？）。
峰值来临时，某个线程池会爆满（触发拒绝策略），而其他线程池却闲置。
即使人为扩大队列，也会导致 计算和告警线程抢占资源，拉高整体 RT。
→ 结果：故障预测成功率低、延迟高。
业务能接受延迟，为啥还要动态线程池?
接受延迟≠不需要优化
在高峰期瞬时任务量激增时，固定线程池，队列满，任务被拒绝丢失。队列长，等待时间长，影响告警判断
数据未被采集全，预测就是不完整的
异常告警未及时触发，风险增加，故障成本增加
动态负载线程池优点
动态强制入队策略
根据负载情况智能切换入队策略，不影响线程池吞吐量，也不为系统带来额外负载
高任务执行率
传统线程池如JDKTP和Tomcat，任务执行率只有50%，而该线程池可达到95%以上
便捷调优
调节Buffer Degree即BT，可根据不同的业务场景快速适配线程池，无需手动修改线程数或任务队列长度，降低运维成本
设计思路
缓冲优先的线程扩展策略
引入
缓冲因子BF
BF=1-TIO/Ttotal
TIO：任务中IO执行时间
Ttotal：任务执行总时间
缓冲阈值BT
BT=Qsize×BF
Qsize：队列长度
BT越大，越为CPU线程池；反之则为IO线程池
阻塞度
TIO/Ttotal
接近1，IO密集型
尽可能多创建线程
4-6区间，混合密集型
接近0，CPU密集型
尽可能少创建线程
任务入队划分
核心线程扩展阶段
最大线程饱和阶段
控制非核心线程的创建速率
核心线程已满且任务队列中的任务≤BT，新任务直接入队
任务数量 > BT且线程数小于最大线程数，创建新线程
流程图实现思路
假设任务队列为100，阈值为0.8，核心线程数被创建完，任务队列也达到80，那么就可以直接创建非核心线程数。如果达到最大最大线程数，此时任务队列还剩下20，继续往里面加，直到达到100，才触发拒绝策略。
负载驱动的强制入队策略
在任务入队失败，基于CPU和线程池负载等级自适应选择策略
借鉴synchronized锁升级机制和以太网的CSMA/CD协议的数据冲突处理思路
面试问法2.6
三种强制入队方式
阻塞等待BWS
线程阻塞等待一定时长，成功则入队，否则结束
退避空转BISS
空转+延迟退避，尝试多次，达到失败阈值终止
重试入队RQS
尝试载入对一次，失败则终止等待
入队策略
基础知识
TPLL线程池负载
TPLL+1
每执行一次FQS/触发拒绝策略
TPLL-1
成功入队或执行任务
CLL系统CPU负载
大于指定阈值为高负载反之为低负载
入队模块
阈值高低
线程池阈值：TPLJ
cpu压力阈值：CLJ
根据项目需求设计
判定条件
低TPLL，低CLL：任务量为略超任务队列上限，使用BWS最大化任务执行成功率
低TPLL，高CLL：任务量为略超任务队列上限，但CPU使用率高，使用BWS释放CPU内存
高TPLL，低CLL：任务量大幅度超过任务队列上限，但系统负载低，使用BISS充分利用CPU资源
高TPLL，高CLL：任务量大幅度超过任务队列上限，且CPU使用率高，系统和线程池都处于高负载，应该优先保证系统稳定性，舍弃部分任务减轻负载，使用RQS最小化消耗CPU
为什么不直接加长阻塞队列从而避免拒绝策略的执行?
加长阻塞队列只是一种预防手段，但不能从根本上解决问题，并且也容易导致OOM
具体的应用场景：
测评流程
测试目的
测试在保证任务成功率的情况下，性能会不会有影响
任务执行成功率
线程创建次数、触发拒绝策略次数
缓存扩展策略
测试线程池在不同业务场景下的适配度
核心指标：TCC
负载均衡策略
测试线程池在高负载下的任务处理能力
核心指标：AEEF、TESR
核心评价指标
平均有效执行时间AEEF
衡量线程池成功处理任务的平均时间，AEEF越低，线程池吞吐量越高
线程创建次数TCC
TCC越高，资源消耗越高，但线程扩展速率越高
任务执行成功率TESR
成功执行任务数量/总任务数量，TESR越高，线程池越稳定
核心参数分析
核心线程数Ncore
I/O密集型：CPU核数×2+1
CPU密集型：CPU核数+1
最大线程数Nmax
Nmax=Ncpu/BF
队列容量Qcap
Qcap=(Ncore/Tavg)×Tresp-Ncore
Tavg：任务平均执行时间
Tresp：任务最大响应时间
参数设计
Ncore：16、Nmax = 160、Qcap= 300
实验分析
缓冲优先的线程扩展策略
在设置缓冲因子的时候，并不是越大越好，在一定程度上影响不是很大。在一定场景下，差个0.2没有太大的影响。为什么是这个数值，并不了解，但实际测试得知在0.2效果最好
负载驱动的强制入队策略
项目提问
cpu利用率
CPU负载采样方法
updateCPULoad
初始阶段：创建单线程调度器，每5s执行updateCPULoad方法
计算流程
获取当前CPU的tick1计数
休眠1s
足够的时间间隔来计算CPU使用率，且1s是常见的采样间隔，与linus的top等工具一致
获取新的CPU的tick2计数
计算两次tick之间的CPU负载
tick计数含义
用户态时间
核心态时间
空闲时间
I/O等待时间
核心计算原理：CPU利用率 = 非空闲时间 / 总时间
总时间：tick1与2对应的所有时间相减
非空闲时间：除去对应的空闲时间
获取缓存值：返还最近计算的CPU负载值