需求

1. SSD作为Cache容量有限、且目前所有写入系统的数据都需要经过Cache层，需要尽可能快速地将Cache中数据向下层迁移，避免Cache写满无可用空间，阻塞应用的写入
2. SSDCache数据向下层迁移的操作要尽可能不影响前端的业务IO

方案

本文为基本草案，实现之前需要更多细节的详细设计

触发机制

1. Timer：控制模块周期性(10s)地生成Redo请求，分发给本节点的所有RedoWorker，触发RedoWorker进行执行机制判断。
2. Usage：Cache使用模块(WriteWorker)占用Cache的Unit达到一定个数(64 or other)之后，生成Redo请求(携带SpeedUp标志)，分发给同一WorkerPair的RedoWorker，触发RedoWorker进行执行机制判断。
3. Feedback：Cache之下的层次使用AIO模块时生效。下层使用AIO时，可以批量的分发Redo内容到下层，直到下层反馈队列满。当下层Redo内容完成执行Callback时，可以生成新的Redo请求，加入RedoWorker的请求队列。

执行机制

由触发机制产生的Redo请求最终的执行调度有如下的情况：

1. 请求不被调度执行。
2. 请求调度执行的粒度根据当前的工作状态计算得出。
   请求是否被调度执行主要依赖下述条件的判断：
3. 当前是否有待Redo的日志数据？
4. 已分发的Redo内容请求是否已经达到下层的上限？
5. 已分发的Redo内容请求是否已达到预设的粒度？
6. 当前的Unit是否已分发完请求，等待回应(与实现相关，跨Unit的调度可能出现数据不一致)
   请求的粒度主要依赖下述条件的判断：
7. Redo模式
   - Normal Mode：正常模式，执行粒度稍小，如每次只分发16个请求到下层
   - SpeedUp Mode：加速模式，执行粒度增大，如每次分发64个请求到下层
   - ？是否需要更快的模式，什么时间触发(后文提出了Dead Mode)
8. 下层的处理能力，目前的架构中，Cache层之下的容量层通常使用HDD作为存储介

质，Redo下发的请求更偏向于HDD上的随机写，所以请求的处理能力是一定的。下层的IO处理能力主要使用在如下的逻辑中：
    - Redo请求的写IO
    - WriteCache的4K对齐的放大写的读IO
    - CacheMiss的读IO
      综上，为了避免影响正常业务的IO，下层存储介质的IO能力不能以最高负载的状态对Redo提供服务，目前给出的建议值是IO能力的一半的软限制，在SpeedUp模式下可以上浮到70%。
9. 下层的Feedback：下层根据自己的队列深度或者自己的服务能力算法计算的服务能力可能会拒绝请求的分发。不过当前的AIO的队列深度应该是远小于Cache层的粒度的。

粒度计算

分发的粒度影响了Cache层数据下沉的速度，由于Cache层与容量层的存储空间的巨大差异，系统当然是希望尽可能快的完成数据下沉，以释放更多的Cache空间，以提供高性能的前端业务的写IO。

但是从之前的描述中我们可以看到，数据越快的下沉，读IO可能会需要容量层提供服务的可能性就越大，此时在容量层的读写IO会加剧对有些的IO能力的竞争，所以为了使系统能够达到均衡、匀速的性能输出，并不是一味地增加Redo的速率就是最优方案。

基于10000rpm的HDD的随机IOPS在100+，目前预设了如下的几个值：

1. 16个日志请求
2. 64个日志请求
3. 是否还需新增一个请求个数，64 + 16？
4. 1个日志请求？当完成批量向下层分发数据之后，下层的Callback可能是以1个日志请求为基本单位，上层在每个Callback中都分发一个新的日志请求到下层？还是每4/8？个Callback分发相应的请求到下层。(实现相关：从目前的架构看，因为Unit的索引都在1次加锁之后Load完成，所以不管是1个还是多个应该不会对业务IO有太大的影响，当然如果跨Unit时需要满足前面的执行机制)

问题

极致写场景

前文说到了是否还需要更快的模式，比如性能测试或者特定应用场景中，用户会有持续海量的写入操作，此时前文提到的2级模式的数据下沉速度是否能满足类似场景，此时可以考虑新增第3种模式 Dead Mode，分发Redo粒度更大，接近硬盘IO能力甚至超过10% ~ 20%的请求加入队列。

此时应增加下层IO的统计信息，可以基于过去的周期的下层读IO的统计信息决定是否让系统

进入Dead Mode。

这种通过过去的统计来预测未来的IO行为有一定的滞后性，所以将周期分为小周期和大周期两个不同阶段，即需要结合小周期和大周期内的IO情况来得出最终的结论，基本流程如下：
1.每隔10s统计这10s的读IO数
2.累计统计10min中的读IO数
3.进行判断时，需要同时满足上一个10s的IO在一定范围且10min的IO在一定范围，才允许进入Dead Mode

之所以要统计大小2个周期，是避免IO感知太灵敏，每隔10s就可能进行一次mode切换