EvenOdd 编码器设计

存储编码算法设计与实现

——Massive Storage 第一届大学生信息存储技术竞赛-挑战赛1

https://competition.huaweicloud.com/information/1000041751/circumstance

坎爷yyds队：
王永康
王新荣
王亚宁

1. 赛题分析-EvenOdd 编码

- EvenOdd 由一个质数p定义。它可以将一个规模为(p-1)*p的阵列经过编码成为一个规模为(p-1)*(p+2)的阵列。它可以容任意的两错，即**任意一列或者两列的数据丢失，都可以由剩余数据进行恢复**。我们称**前p列为“数据列” (data column)**，后两列为“**校验列” (parity column)**。其中，第0个校验列我们称为“**行校验列”(row parity)**，第1个校验列我们称为**“对角线校验列” (diagonal parity)**。
- 行检验列和对角线校验列生成的方式如下：
  - 例子中 p = 5, 记下面数据矩阵为 M, 第 5 列和第 6 列为校验列， 记 $\bigoplus_{i=i_0}^{n} a_i = a_{i_0} \bigoplus a_{i_0+1} \cdots \bigoplus a_{n-1} \bigoplus a_n$, $\bigoplus$ 为异或运算

	C0	C1	C2	C3	C4
R0	1	0	1	1	0
R1	0	1	1	0	0
R2	1	1	0	0	0
R3	0	1	0	1	1

每一行进行异或运算，结果写到 c5:

$$M[i][5] = \bigoplus_{j=0}^{4} M[i][j], 0 \le i \le 3$$

	C0	C1	C2	C3	C4	C5
R0	1	0	1	1	0	1
R1	0	1	1	0	0	0
R2	1	1	0	0	0	0
R3	0	1	0	1	1	1

	C0	C1	C2	C3	C4
R0	1	0	1	1	0
R1	0	1	1	0	0
R2	1	1	0	0	0
R3	0	1	0	1	1

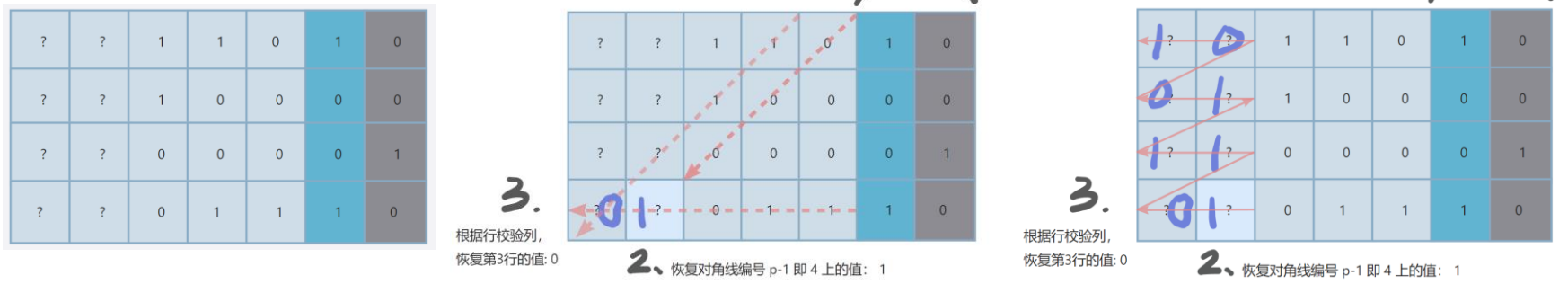
1. 先计算红色对角线 $s = \bigoplus_{(i+j)\%p==p-1} M[i][j]$
2. 然后计算对角线 0 ~ p-2 和 s 的异或结果

写到 c6: $M[k][6] = s \bigoplus_{(i+j)\%p==k} M[i][j],\ 0 \le k \le 3$

	C0	C1	C2	C3	C4	C5	C6
R0	1	0	1	1	0	1	0
R1	0	1	1	0	0	0	0
R2	1	1	0	0	0	0	1
R3	0	1	0	1	1	1	0

1. 赛题分析-EvenOdd 解码

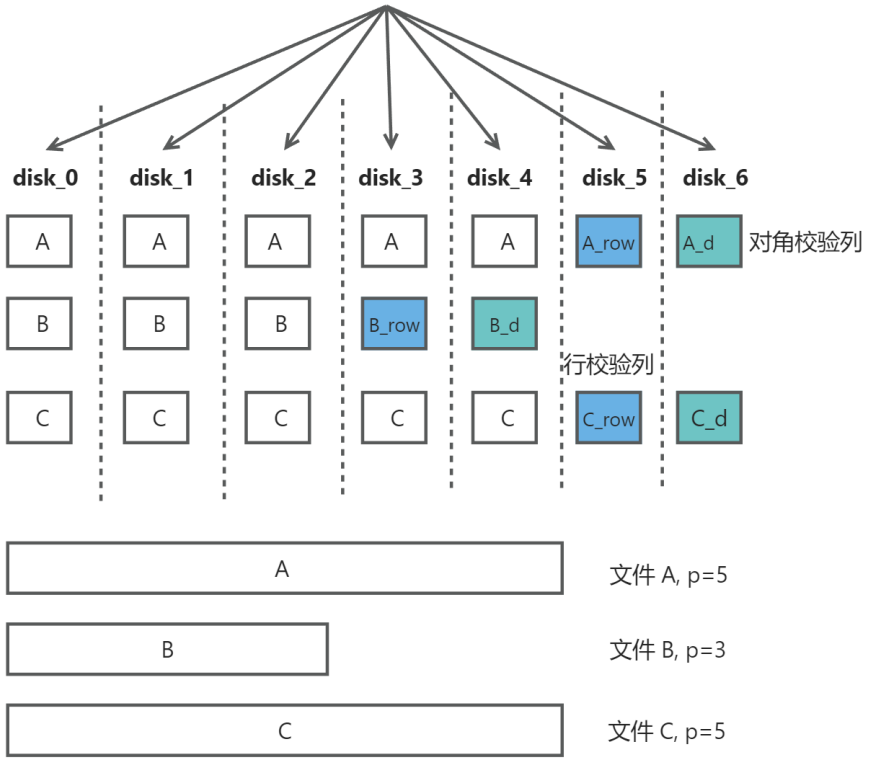
- 1 列数据丢失的恢复
  - 可以通过 row parity 或 diagonal parity 恢复，过程和生成 row parity 或 diagonal parity 一样
- 2 列数据丢失的恢复
  - 考虑极端情况：两列数据列丢失的情形
    - p = 5, 第 0、1列数据丢失



1. 赛题分析-write 操作

- **$ ./evenodd write filename p**
- **evenodd** 是要求生成的可执行文件，后面接操作参数
- 把 filename 文件，写到 *./disk_0/, ./disk_1/, ..., ./disk_{p-1}/, ./disk_{p}/* 目录下，保证任何不超过 2 个 disk 目录丢失情况下，能够恢复出原文件。（模拟磁盘损坏能恢复）

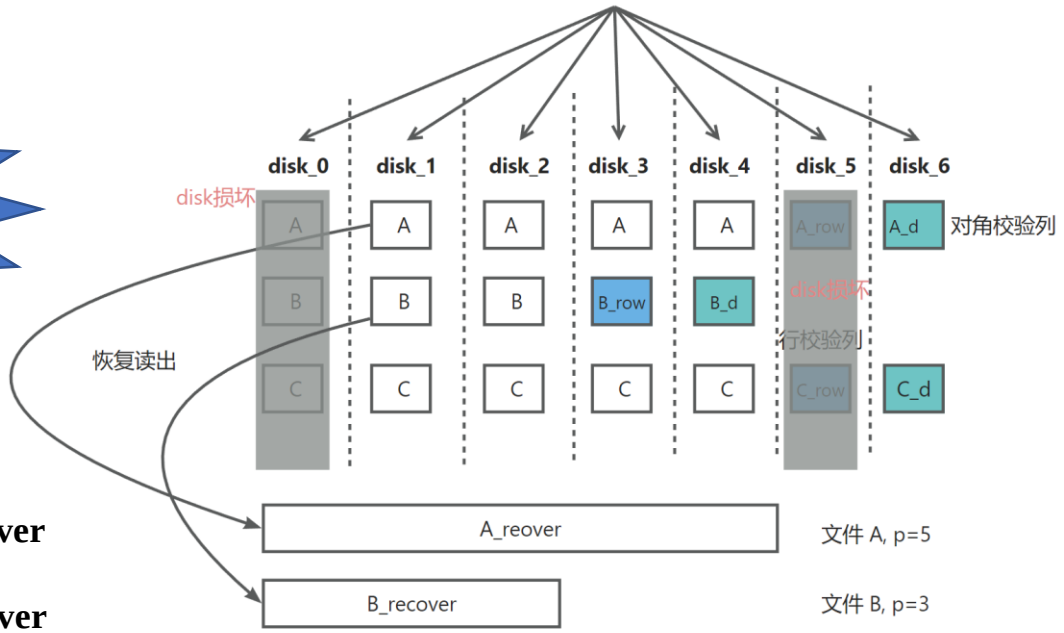
不必落盘



1. 赛题分析-read 操作

- **$ ./evenodd read filename save_name**
- 从 disk_* 中读出 filename 并保存为 save_name

需要在 write 阶段保留必要的 filename, p 等信息,

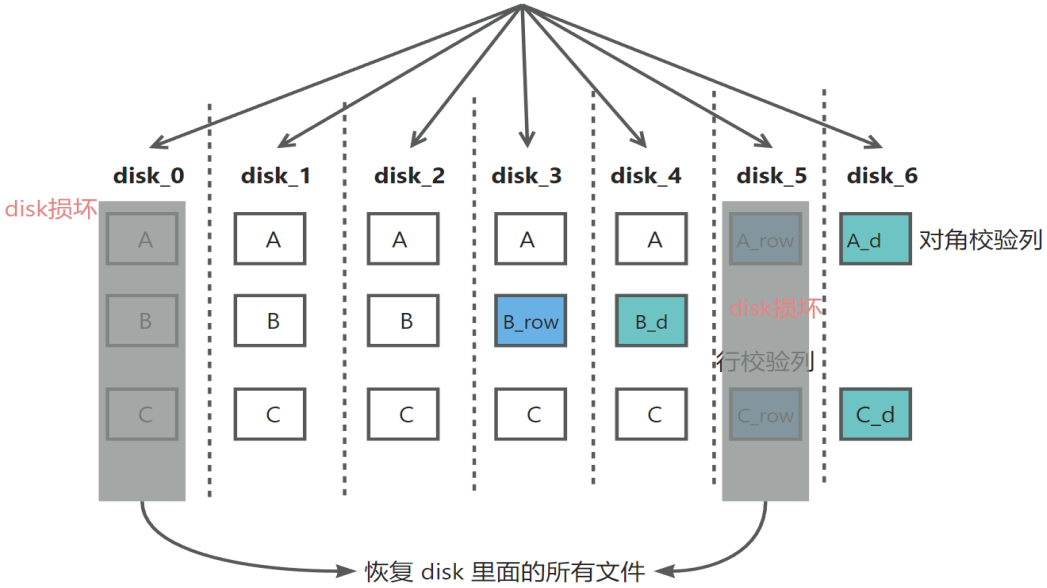


$ ./evenodd read A A_reover

$ ./evenodd read B B_reover

1. 赛题分析-repair 操作

- **$ ./evenodd repair i1 i2**
- 恢复 disk_i1 和 disk_i2 里面所有的文件



1. 赛题分析-测试平台

- 2核 CPU, 4G 内存
- 100G的SSD：
  - 吞吐量 R: 245MiB/s, W: 151MiB/s
- 语言：C/C++
- 默认环境为干净的 ubuntu 18.04 + build essentials

1. 赛题分析-测试数据

- 测试时 p 的范围 3-100 的质数
- 文件大小，最大可达 100 GiB
- 猜测：执行若干组 write, read, repair 测试，计算总时间。
  - 正确性 50% + 性能 50%

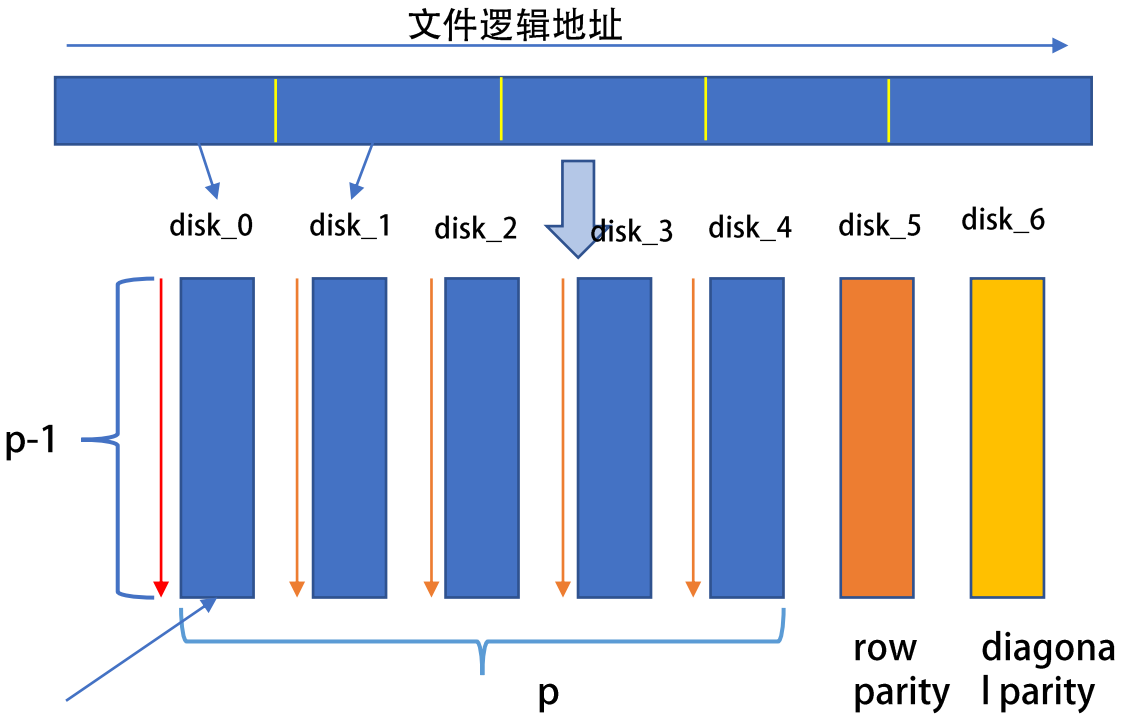
2. 瓶颈分析

- 瓶颈 1：异或运算（CPU密集）
  - 计算行校验列和对角校验列需要大量异或运算
- 瓶颈2：IO 瓶颈
  - 文件太大时，超出内存（4G)，不能一次性读入内存，可能会重复读盘
  - 写入数据和读出数据竞争 SSD 带宽?

3. 我们的方案

- 按列划分文件

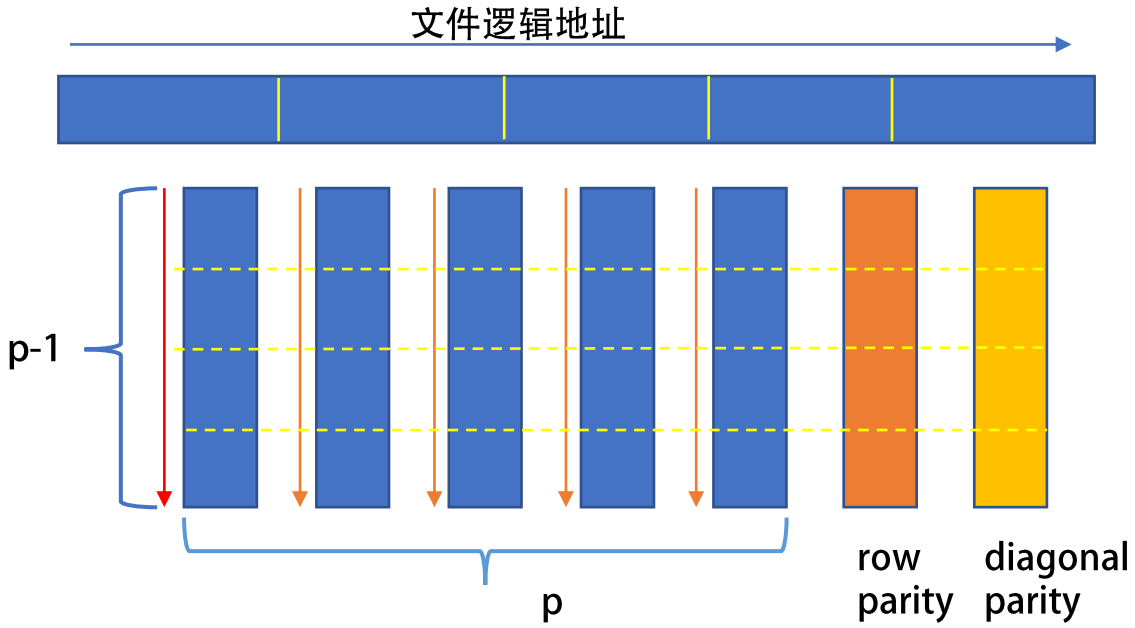
好处：
- 读连续
- 写连续



col file: 写入每个 disk 的文件

3. 我们的方案-write

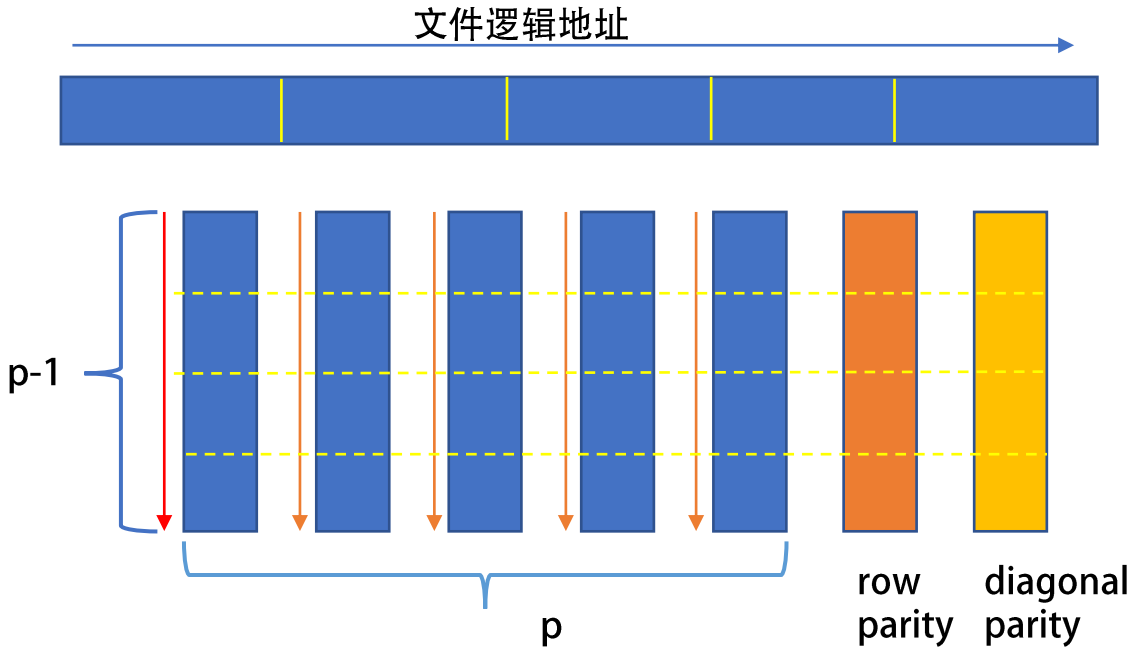
- 小文件 <= 3GiB: 可以全部读入内存，然后计算完 row parity 和 diagonal parity，再写入 SSD;
- 大文件 > 3GiB:
  - 按列读取，读完便写入一次；然后释放，重复使用该块内存空间;
- 超大文件:
  - 拆分成若干个大文件



3. 我们的方案-read

分情况考虑

- 无数据列丢失：直接读取全部数据列
- 一列数据列丢失：用行校验列来修复
- 两列数据列丢失：用行校验列和对角校验列共同恢复



大多是顺序读与顺序写

按列读取，用于计算和入后即可释放空间，读取下一列时可复用该内存空间；只需维护部分额外数据

只解码出文件，不修复 disk 下缺失的文件

3. 我们的方案-repair

- 分情况修复：
  - 修复行校验列，或修复对角校验列：重新编码
  - 修复一列数据列：用行校验列或对角校验列来修复
    - 改进：混合行校验列和对角校验列来修复（详情见后面）
  - 修复两列数据列：用行校验列和对角校验列共同修复

4. 优化方案

优化考虑两方面：

- CPU 计算加速
  - 加速异或操作：利用 OpenMP 开双线程加速异或操作
  - 执行 64 位的异或操作

```
#pragma omp parallel for num_threads(2)
  for (off_t i = 0; i < symbol_size / 8; i++) {
    ((size_t *)(lhs))[i] = ((size_t *)(lhs))[i] ^ ((size_t *)(rhs))[i];
  }
```

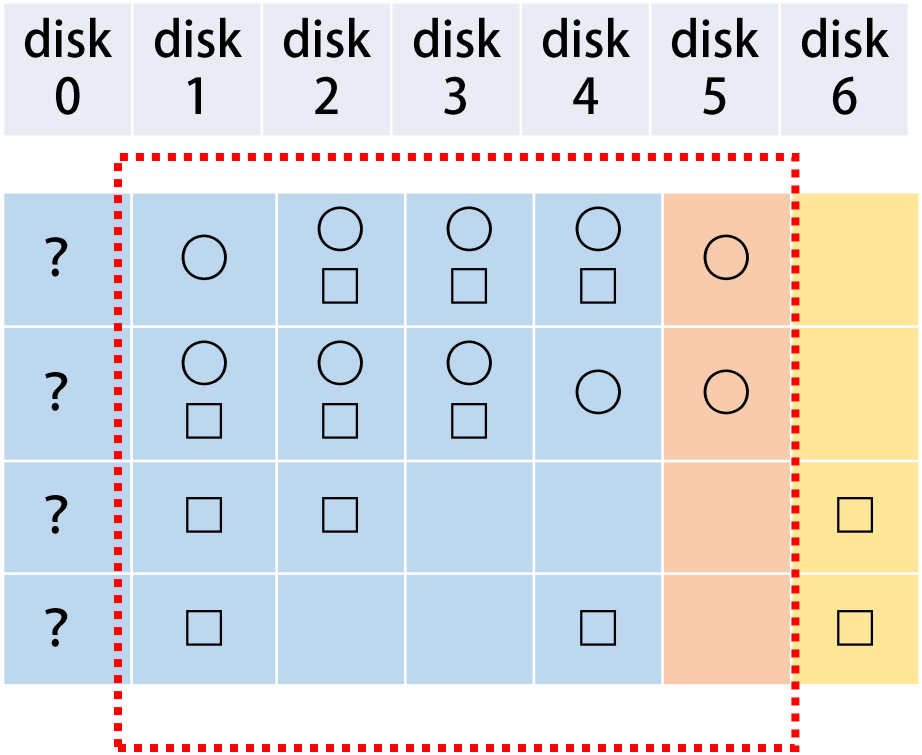
- IO 加速
  - 单盘损坏快速恢复: 尽可能减少读盘次数

4. 优化：单盘损坏混合恢复[1]

- 当数据列所在的一个盘(disk)发生故障时，可以让前 (p-1)/2 的数据块通过行校验列恢复；剩余 (p-1)/2 块通过对角校验列来恢复。
- 从而可以重复利用一些数据，减少从磁盘读取数据量。

$$\text{读取数据减少量} = \frac{1}{4}\left(1 - \frac{1}{p}\right)$$

- 右图为修复 disk_0 例子（p=5），其中 ○ □ 分别表示通过**行校验列、对角校验列**修复时需要读取的数据块；



- [1]常乾, 许胤龙, 项利萍,等. 基于EVENODD码的单盘故障快速恢复算法[J]

5. 结果

- Write: 110.311MB/s
- Read: 119.149MB/s
- 目前，在预赛上取得第一的成绩。

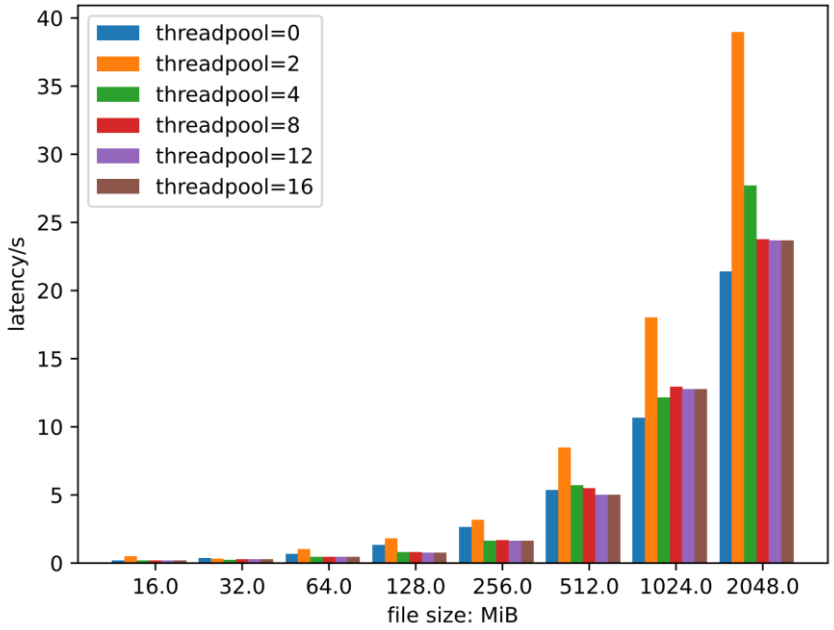
1	队伍名称	最新提交日期	正确性	性能	总分	性能测试时间
2	坎爷yyds	2022/11/28	50	48.44	98.44	383.142
3	嗯对对队	2022/12/5	50	46.88	96.88	387.576
4	HITszECOne	2022/12/5	50	45.31	95.31	393.184
5	GeekPie	2022/12/5	50	43.75	93.75	393.489
6	软泥帽与德比安	2022/12/5	50	42.19	92.19	395.484
7	WHUACM2022新生群-team026	2022/12/5	50	40.63	90.63	399.398
8	A-EO	2022/12/5	50	39.06	89.06	399.474
9	leedagee	2022/11/16	50	37.50	87.50	408.144

6. 一些问题

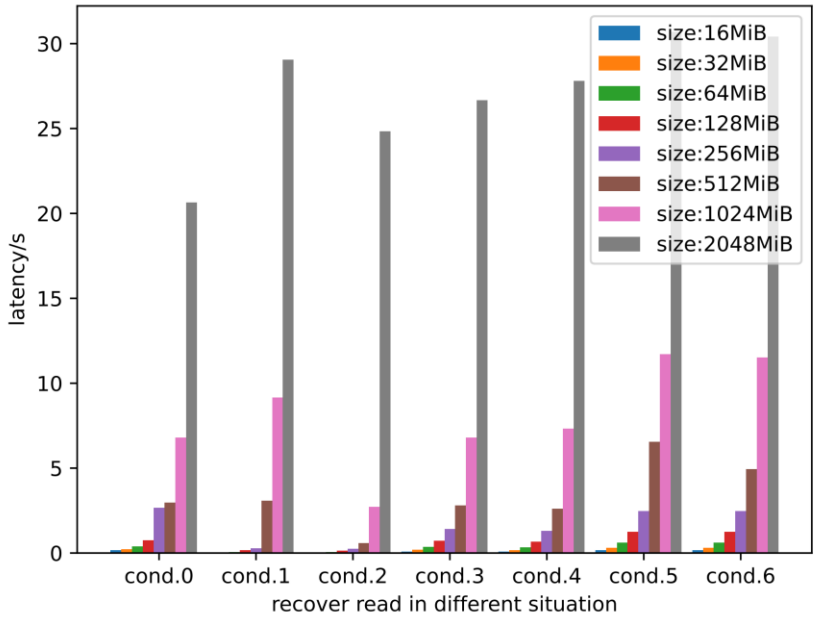
- 同步读、写会导致进程阻塞?
  - write 和 read 会导致整个进程阻塞?
- 异步读、写无法掩盖 cpu 上的异或运算
  - 尝试 IO 和 xor 计算并行:

- write 受 pagecache 影响，无法定量优化? 但是确实会和读竞争 SSD 带宽;
- mmap 会有性能提升吗?
  - 试过对大文件>=4GiB 的文件，没有明显效果。

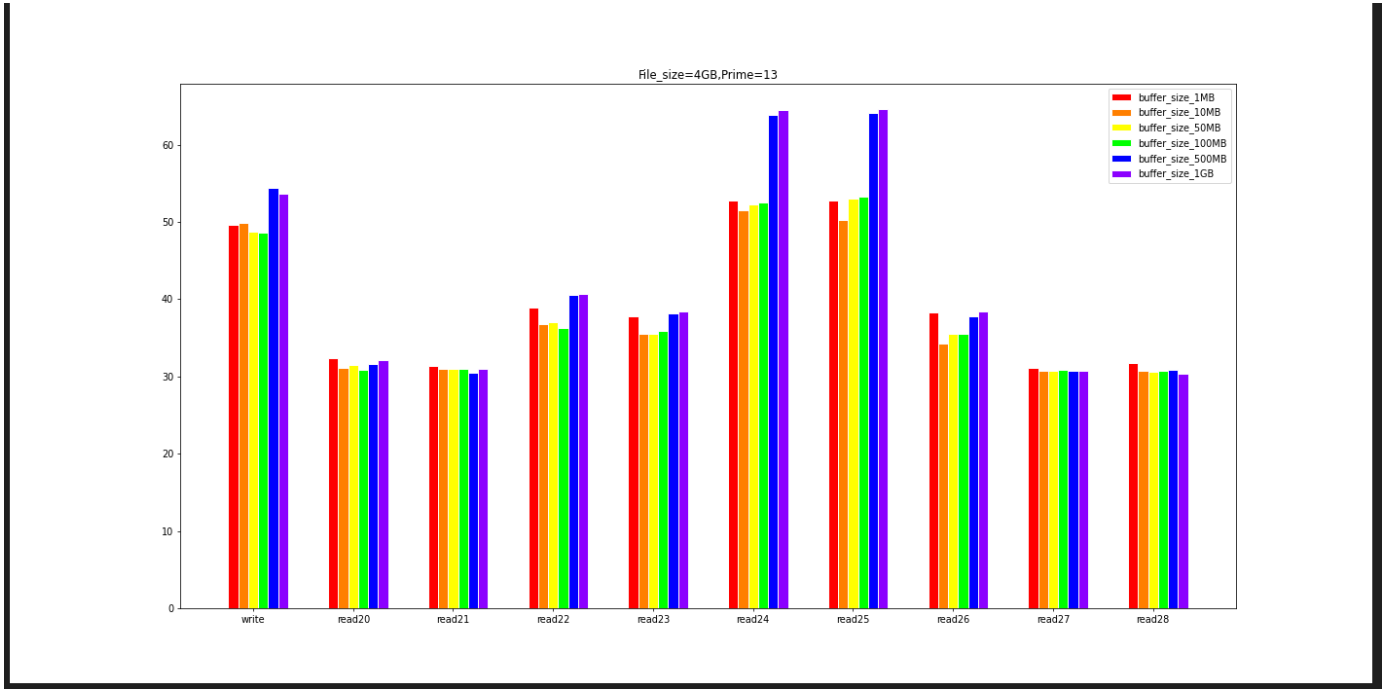
7. 存档-使用 threadpool 效果



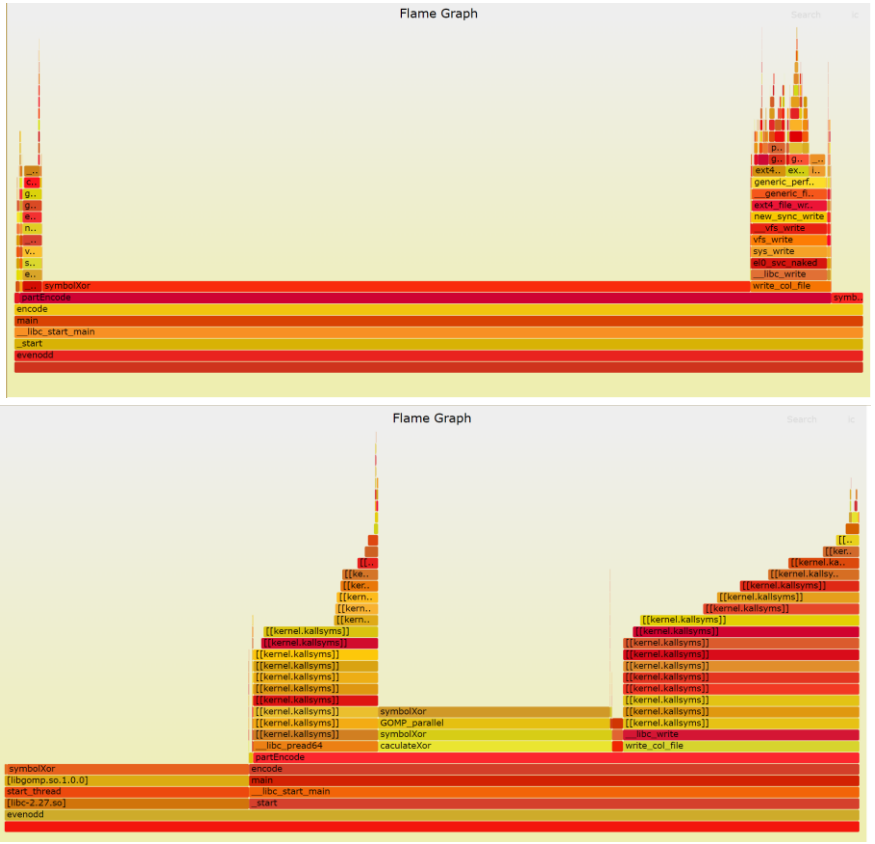
7. 存档-col file size 影响1



7. 存档-col file size 影响2



7. 存档-火焰图 xor 优化前后



7. 存档-IOzone

./iozone -a -n 512m -g 16g -i 0 -i 1 -i 2 -f /mnt/vdb/wyk_test/io_test/iozone -Rb /mnt/vdb/wyk_test/iozone.xls

The top row is records sizes, the left column is file sizes

Writer Report

	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
524288	0	0	0	0	1430116	1499040	1507030	1401839	1379698	1355499	1368041	1368440	1361971
1048576	0	0	0	0	553594	871663	609093	554524	527452	786096	555649	552582	798734
2097152	0	0	0	0	352212	367976	365343	366607	364121	364743	402137	364396	365288
4194304	0	0	0	0	300838	312602	311974	309421	291831	306205	309676	309599	291443
8388608	0	0	0	0	274756	277977	276379	275933	276048	276267	276507	275579	276186
16777216	0	0	0	0	268469	267795	267404	267107	266898	267179	267074	266964	266829

Re-writer Report

	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
524288	0	0	0	0	2942991	3131294	3144306	2769115	2680962	2582165	2585131	2563691	2556715
1048576	0	0	0	0	504851	700263	519050	560930	807777	649300	515733	498613	681457
2097152	0	0	0	0	333396	350544	348441	348448	348854	348701	349275	349746	347608
4194304	0	0	0	0	300973	292929	292463	309931	305336	300710	298470	297617	300943
8388608	0	0	0	0	278570	269857	271505	271539	270090	269793	271371	271528	277700
16777216	0	0	0	0	264590	265690	265647	265691	266313	265717	265729	266372	266354

Reader Report

	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
524288	0	0	0	0	5566163	5667676	5655810	5522536	5421069	4874739	4373991	4270140	3962633
1048576	0	0	0	0	5859891	5951358	5963414	5757134	5571069	5115079	4571846	4389347	4102024
2097152	0	0	0	0	255013	6119911	6091397	5895627	5768078	5228565	4695575	4511381	4190188
4194304	0	0	0	0	244509	255307	256475	257695	257792	259893	259765	259485	258573
8388608	0	0	0	0	251693	253097	253276	253967	253240	254048	253887	254199	260179
16777216	0	0	0	0	255923	256490	256785	257127	257354	257283	257341	257411	257408

Re-reader Report

	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
524288	0	0	0	0	4864021	4950828	4929320	4820825	4716850	4319001	3974285	3822455	3621175
1048576	0	0	0	0	5021244	5115251	5129343	4982753	4905917	4481995	4072805	3962947	3714613
2097152	0	0	0	0	5757642	5205852	5206228	5047976	4987957	4573873	4162420	4030132	3779048
4194304	0	0	0	0	258511	252953	251940	250391	250012	260090	259767	260684	250352
8388608	0	0	0	0	255516	255368	255287	259260	254753	259604	254958	259909	254913
16777216	0	0	0	0	255507	255409	255193	254944	254793	254899	254855	254830	254983

Random Read Report

	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
524288	0	0	0	0	5389871	5529062	5636890	5482292	5396180	4909434	4371878	4228575	3969199
1048576	0	0	0	0	5525310	5813762	5879546	5721945	5634477	5098612	4581972	4416490	4123755
2097152	0	0	0	0	5609954	5905360	6010202	5823561	5732397	5218781	4688856	4502530	4239361
4194304	0	0	0	0	159274	243556	371583	452685	491844	407780	459432	473735	505895
8388608	0	0	0	0	91962	139085	191611	246237	280123	209281	246258	262761	270763
16777216	0	0	0	0	84344	129329	176893	231119	261558	176655	213887	238580	249734

Random Write Report

	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
524288	0	0	0	0	2928540	3098023	3162515	2799414	2675637	2582750	2563391	2583855	2573152
1048576	0	0	0	0	1521714	948685	714375	706072	1308143	785825	815735	1758010	838372
2097152	0	0	0	0	393821	355980	356657	358924	358552	418945	431641	431382	441437
4194304	0	0	0	0	307518	310857	305489	304708	305482	301934	298836	300646	298352
8388608	0	0	0	0	282775	281801	277094	280927	270907	273803	279079	272035	271711
16777216	0	0	0	0	267868	267003	267483	268127	266726	267201	266541	266212	266539