**磁电盘IO最优调度**

|  |  |
| --- | --- |
| **日期** | **更新内容** |
| 2024年8月9日 | BeltWearTimes接口函数  TotalAccessTime接口函数  TotalTapeBeltWearTimes接口函数 |

# 背景

磁带介质具备较高的容量/价格比，广泛用于归档和数据保护场景，主要包括磁头、带体和卷带电机。磁带组成结构如图1，读写数据一般覆盖1个Wrap，采用蛇形记录技术从磁带的起始端(BOT)到磁带的结束端(EOT)写完磁道后，再回过来写直到起始端，所以磁带是天然的顺序访问亲和介质。

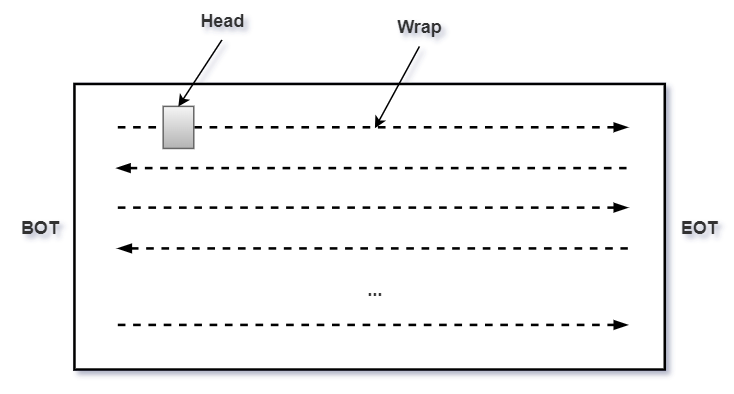
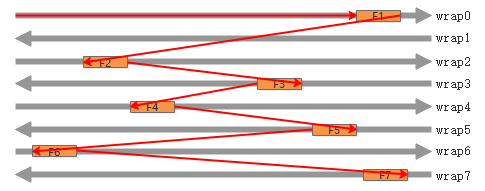
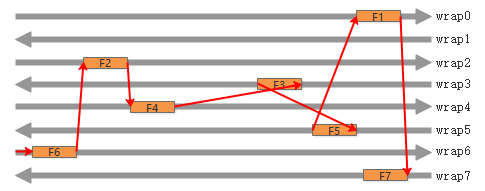


图1 磁带组成结构

磁带只能单并发读写操作、带体读写次数仅40K、电机启停次数仅500K，任何掉头、随机寻址或不合理的磁带访问顺序都会损耗硬件寿命，造成用户几十秒级/分钟级时延，从而影响磁带性能与寿命。在数据热点、读写关联场景下，磁带可利用SSD缓存、数据排布等系统性技术优化寿命与性能，但在随机读场景，关键技术是利用IO调度对一批数据访问序列进行最优排序，以大幅降低寻道时延与无效带体访问磨损，如图2对比线形访问调度和IO近邻调度如何在最短时间内完成所有读任务，线形访问调度明显比IO近邻调度访问带体长度与寻道时延更大。



线形访问调度



IO近邻调度

图2 磁带IO调度算法

磁电盘IO调度是典型的OTSP（Open Loop Travelling Salesman Problem）最优化调度问题，以最小化磁带寻道时延和磨损为优化目标，在IO请求数、IO请求时间、算法复杂度等多种变量约束下进行组合优化模型构建和求解IO最优推荐序列，逼近全局最优解。IO最优调度在基于磁电盘的归档存储系统中，使得随机寻道时延和读磨损降低50%以上，技术价值重要性进一步凸显。

# 基本概念

本题目模拟对随机访问的磁带数据进行IO调度，为了能更好地理解题目，我们需要对一些基本概念进行澄清，IO请求特征包括IO输入参数和磁带物理位置，用于构建磁带寻道模型；请求访问时间和磁带磨损用于评价IO调度指标。

## 2.1 IO请求特征

IO输入参数包括IO请求数、IO基础信息，磁带物理位置描述磁头位置与状态，两者主要包括wrap、Lpos、startlpos、endlpos：

* Wrap是带体横向存放数据的的物理对象，数据布局有两个方向（BOT往EOT，EOT往BOT），最大规格为280。
* Lpos是指相对BOT的具体带体位置，BOT lpos为0。
* Startlpos是指数据起始的位置，endlpos是指数据结束的位置，Startlpos与endlpos之间的长度为数据段大小（即IO大小），如wrap方向是BOT往EOT，则数据的endlpos > startlpos，如wrap方向是EOT往BOT，则数据的startlpos > endlpos。赛题中不考虑数据跨2个或2个以上wrap。

## 2.2 IO请求访问时间

IO请求访问时间 = IO排序时间 + 寻址时间 + 数据读取时间：

* IO排序时间指一批IO请求执行调度算法得到排序后IO序列所消耗的时间；
* 寻址时间指从磁头当前位置移动到数据起始位置的时间，该段时间只进行寻址，不进行读写，以磁头起始位置及状态和目标的位置及状态可确定寻址时间；
* 数据读取时间指从数据起始位置到数据结束位置的读写时间，通过数据在磁带上所占据的磁带长度可确定访问时间。

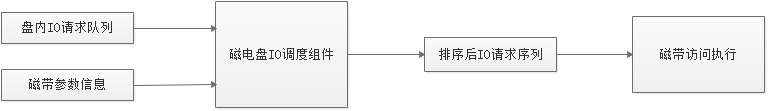
## 2.3 磁带磨损

磁带磨损由带体磨损和电机磨损两部分组成：

* 带体磨损指磁头通过寻址或读写划过当前位置带体，则当前位置的带体磨损消耗一次，统计粒度为lpos的pass次数；如图2中F6数据段到F2数据段，F6数据段的结束lpos为100，F2的起始lpos为110，从F6移动到F2经过10个lpos，该过程的磁带磨损为10。
* 电机磨损主要与电机启停次数和运行时间相关，刹车、启动加速均需消耗1次电机寿命，如磁带掉头，需要先刹车，再启动加速，故掉头需消耗2次电机寿命，统计粒度为一批IO的电机启停次数。如图2中F4数据段到F3数据段，F4数据段的方向和F3数据段的方向相反，因此从F4到F3要进行一次掉头操作，经过减速到0再加速的过程，电机磨损为2。

# 任务描述

参赛选手将会负责设计与编写归档存储系统中的磁电盘IO调度算法组件。**该算法组件需要决定当前磁带请求需要处理的先后顺序，输入一批盘内IO请求，要求在规定的时间内对该批请求进行排序，输出排序后的IO请求序列。**一旦请求序列输出后，便无法再对队列中的请求进行调整。磁带按照排序后的IO请求按序访问磁带上的数据，直到所有IO请求处理完毕**。**



* 盘内IO请求队列：磁带接收到需要访问执行的读IO请求，请求示例如下所示：

IOUint {

id: 4871,

wrap: 12,

startLpos: 1024,

endLpos: 2048,

}

该消息表示，请求ID为4871，Wrap序号为12，该IO请求数据起始位置为1024，该IO请求数据结束位置为2048。

* 磁带参数信息：磁头当前状态信息及其他磁带相关的参数，示例如下；

HeadInfo {

wrap: 4,

lpos: 950,

status: HEAD\_STATIC

}

该信息表示，当前磁头处于4号wrap上，位置为lpos 950，状态为静止（HEAD\_STATIC表示静止，HEAD\_SEEK表示寻址中，HEAD\_RW表示读写中）；

* 磁电盘IO调度组件：算法组件需要在接收IO任务请求之后，基于IO数据布局进行访问最佳排序，输出排序后的IO执行序列。请求执行消息如下所示：

OutputParam {

len: 4,

sequence: [3, 1, 2, 0],

}

该输出序列表示，当前输出任务数量len为4个，sequence为IO任务请求id号，[3, 1, 2, 0]表示输出执行序列按3,1,2,0的请求序列执行。

**该算法组件主要包括磁带寻道模型、调度算法，寻道模型用于计算时延与磨损收益，为盘内IO排序提供基础特征；调度算法利用磁带寻道模型、IO请求特征和资源约束对一批IO进行排序。磁带寻道模型须以最小访问时延和最小磨损为目标函数**：

𝜔为不同代价的权重，详见第5章节评分标准；𝑐\_𝑠𝑒𝑒𝑘、 𝑐\_𝑎𝑏𝑟𝑎𝑠𝑖𝑜𝑛和𝑐\_𝑚𝑜𝑡𝑜𝑟分别表示磁带寻道时延代价、磁带带体磨损代价和磁带电机磨损代价。**寻道时延影响因子包括wrap改变个数、磁带运动方向、磁头掉头次数和磁带运动横向距离；带体磨损影响因子是lpos pass次数，电机磨损影响因子是电机启停次数**。

**调度算法须在低资源开销、数据离散分布下逼近IO调度全局最优解。参赛选手主要设计和实现调度算法，寻道模型由赛题方提供。**

* 磁带访问执行：磁带驱动器将会按输出序列依次执行IO请求任务，统计请求访问时间及磁带磨损情况。

# 接口样例

## 接口函数

1. **参数选手完成IOScheduleAlgorithm算法接口函数**

|  |
| --- |
| /\*\*   \* @brief  算法接口   \* @param  input            输入参数   \* @param  output           输出参数   \* @return int32\_t          返回成功或者失败，RETURN\_OK 或 RETURN\_ERROR   \*/  int32\_t **IOScheduleAlgorithm**(const InputParam \*input, OutputParam \*output) |

函数入参结构体InputParam

|  |
| --- |
| /\* 磁头状态 \*/  typedef enum {      HEAD\_STATIC = 0,  // 静止      HEAD\_RW,          // 读写      HEAD\_STATUS\_BUTT  } HEAD\_STATUS;  /\* 磁头位置与状态 \*/  typedef struct {      uint32\_t wrap;      uint32\_t lpos;      HEAD\_STATUS status;  } HeadInfo;  /\* IO 结构体 \*/  typedef struct {      uint32\_t id;            // IO序号      uint32\_t wrap;          // 起始wrap      uint32\_t startLpos;     // 起始lpos      uint32\_t endLpos;       // 结束lpos  } IOUint;  /\* IO 容器 \*/  typedef struct {      uint32\_t len;       // io数量      IOUint \*ioArray;    // io数组，访问方式ioArray[i]  }IOVector;  /\* 输入参数结构体 \*/  typedef struct {      HeadInfo headInfo;      IOVector ioVec;  }InputParam; |

函数出参结构体OutputParam，框架已经为变量output和sequence变量分配了空间，选手**不需要再malloc空间**。

|  |
| --- |
| /\* 输出数据结构体 \*/  typedef struct {      uint32\_t len;       // io数量      uint32\_t \*sequence; // io序号的排列，访问方式sequence[i]  }OutputParam; |

1. 寻址时间计算函数，用于计算磁头从一个位置移动到另一个位置的时间。

|  |
| --- |
| /\*\*   \* @brief 寻址时间计算   \* @param  start            磁头起始位置和状态   \* @param  target           目标位置和期望到达位置的状态   \* @return uint32\_t          返回时间，单位毫秒   \*/  uint32\_t **SeekTimeCalculate**(const HeadInfo \*start, const HeadInfo \*target); |

入参结构体HeadInfo：

|  |
| --- |
| /\* 磁头状态 \*/  typedef enum {      HEAD\_STATIC = 0,  // 静止      HEAD\_RW,          // 读写      HEAD\_STATUS\_BUTT  } HEAD\_STATUS;  /\* 磁头位置与状态 \*/  typedef struct {      uint32\_t wrap;      uint32\_t lpos;      HEAD\_STATUS status;  } HeadInfo; |

应用举例

|  |
| --- |
| /\* 磁头在位置start静止，移动到目标位置end，期望到达end时速度为读写速度 \*/  HeadInfo start = {10, 1000, HEAD\_STATIC};  HeadInfo end = {15, 3000, HEAD\_RW};  unt32\_t seekT = **SeekTimeCalculate**(&start, &end); |

1. 计算两点间带体磨损

|  |
| --- |
| /\*\*   \* @brief  带体磨损计算   \* @param  start            起始位置和状态   \* @param  target           目标位置和状态   \* @param  segWearInfo      带体的分段磨损信息，即记录磁头划过每个lpos的次数   \* @return uint32\_t         返回磁头所经过的LPOS数量，划过一个LPOS计数1次   \*/  uint32\_t BeltWearTimes(const HeadInfo \*start, const HeadInfo \*target, TapeBeltSegWearInfo \*segWearInfo); |

入参结构体TapeBeltSegWearInfo：

|  |
| --- |
| /\* 带体分段磨损 \*/  typedef struct{      uint16\_t segWear[MAX\_LPOS];       // 每个lpos的磨损，即每个lpos的划过次数  }TapeBeltSegWearInfo; |

1. 计算两点间电机磨损

|  |
| --- |
| /\*\*   \* @brief  电机磨损计算   \* @param  start            起始位置和状态   \* @param  target           目标位置和状态   \* @return uint32\_t         返回电机的启停次数   \*/  uint32\_t **MotorWearTimes**(const HeadInfo \*start, const HeadInfo \*target); |

1. 计算从磁带读取数据的时间

|  |
| --- |
| /\*\*   \* @brief 计算读数据的时间   \* @param  startLpos        lpos范围   \* @return uint32\_t          毫秒   \*/  uint32\_t **ReadTimeCalculate**(uint32\_t lposRange); |

应用举例

|  |
| --- |
| /\* 计算从fromLpos到toLpos之间的数据所需的时间 \*/  unt32\_t rwT = **ReadTimeCalculate**(abs(fromLpos - toLpos)); |

1. 计算总寻址时间和读IO时间，用于计算排序后的IO，实际的访问时间，包括寻址时间和读数据的时间。

|  |
| --- |
| /\*\*   \* @brief  计算批量IO的总访问时间，寻址时间+读IO数据时间   \* @param  input            磁头和批量的IO信息   \* @param  output           排序后的IO序列   \* @param  accessTime   输出参数，访问时间结构体   \*/  void TotalAccessTime(const InputParam \*input, const OutputParam \*output, AccessTime \*accessTime); |

入参结构体AccessTime：

|  |
| --- |
| /\* 访问时间 \*/  typedef struct{      uint32\_t addressDuration;           // 寻址时间      uint32\_t readDuration;              // 读数据时间  }AccessTime; |

1. 统计带体磨损函数，用于统计排序后的IO，对带体的磨损情况。（注：带体磨损指磁头划过的距离之和，距离单位是lpos，划过一个lpos计数一次，划过同一个lpos也计数一次）

|  |
| --- |
| /\*\*   \* @brief  统计磁带带体的磨损次数，即，划过一个LPOS计数1次   \* @param  input            磁头和批量的IO信息   \* @param  output           排序后的IO序列   \* @param  segWearInfo      带体的分段磨损信息，即记录磁头划过每个lpos的次数   \* @return uint32\_t         带体磨损次数   \*/  uint32\_t TotalTapeBeltWearTimes(const InputParam \*input, const OutputParam \*output, TapeBeltSegWearInfo \*segWearInfo); |

1. 统计电机磨损函数，用于统计排序后的IO，对电机的磨损情况，函数参数结构体与算法接口函数一致。（注：电机磨损指电机的启停次数，启、停各计数1次，掉头计数2次）

|  |
| --- |
| /\*\*   \* @brief  统计电机磨损次数，即，启、停各计数1次，掉头计数2次，   \* @param  input            磁头和批量的IO信息   \* @param  output           排序后的IO序列   \* @return uint32\_t         电机磨损次数   \*/  uint32\_t **TotalMotorWearTimes**(const InputParam \*input, const OutputParam \*output); |

## 代码编译

在代码根目录下依次执行以下命令，如果代码无误，可以生成可执行文件project\_hw，如下图：



## 运行测试

输入命令格式：“./project\_hw -f 测试数据文件地址”，例如：



# 评分标准

1. 测试标准：
   * **测试环境为华为云ECS鲲鹏通用计算增强型kc2，920 2vCPU、4GB内存。**
   * **测试工具采用IO调度算法仿真平台，比赛共提供初赛和终赛两组数据。**
   * **编程语言为C。**

2、测试用例：所有用例均基于不同的实际场景进行设计，以接近真实业务，用例中包含不同大小的请求数量（10 <= IO请求数 <=10000）以及不同分布特征的请求，覆盖多个场景，考察算法泛化性与鲁棒性：

* + **初赛用例只考虑IO请求访问时间指标。**
  + **决赛用例综合考虑IO请求访问时间和磁带磨损指标，由测试用例给出不同指标的权重，以充分验证不同算法优势。**

3、 时限、空间设计

a) 时间约束：要求单个用例IO排序时间 <= 20s。

b) 空间约束：要求单个用例算法运行空间 <= 10M。

4、 得分设计

**优先考虑规定时间内对比基线算法（SCAN、SORT）的求解效果（读时延、磁带磨损），在二者接近时通过算法执行时间确保最终排名的可区分性**。

5、评分程序会记录磁带执行情况，并对算法合规性进行校验，积分规则如下：

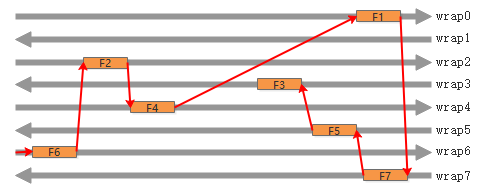
* + 初赛调度算法加分 =（基线读时延 - 排序后读时延）\* 10。
  + 决赛调度算法加分 =（基线读时延 - 排序后读时延）\* 10 +（基线读磨损 - 排序后读磨损）\* 10。
  + 调度用时加分 =（20s – IO排序时间）\* 10。
  + 调度超时罚分 =（执行时间 - 规定时间）\* ⌈请求大小 / 50⌉。
  + 空间超限罚分 =（实际空间 - 规定空间）\* ⌈请求大小 / 100⌉。
  + 调度错误罚分 = 10 \* 错误IO请求数。
  + 最终总分 = 调度算法加分 + 调度用时加分 – 调度超时罚分 - 空间超限罚分 - 排序错误罚分。

# 算法示例

常见磁带IO调度算法有先来先服务（FCFS）、电梯调度（SCAN）、最近邻调度、RAO（Recommended Access Ordering）和TAOS（Time-Based Access Order System），下面以电梯调度和最近调度举例。

## 6.1 电梯调度（SCAN）算法

按照一个方向(比如从BOT向EOT扫描)，扫描的过程中依次访问该方向的IO请求。当扫描到最远的一个IO请求后反向扫描，直至所有请求访问完成。



如上图，F6、F2、F4、F1为一个wrap方向，扫描到最远F1后，开始反向扫描F7、F5、F3，直至结束。

## 6.2 最近邻调度算法

1．获取或者估算IO请求物理地址：

2. 构建模型建立两个IO请求物理位置（）之间的代价函数

3. 基于贪心算法，搜索距离当前位置寻址时间最短的IO请求。