江波龙 - 广东工业大学
2025 技术联合开发项目介绍

2025/1/16

深圳市江波龙电子股份有限公司
Shenzhen Longsys Electronics Co., Ltd.

深圳市江波龙电子股份有限公司
Shenzhen Longsys Electronics Co., Ltd.

FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Sold State Drives

FLIN

SSD内部的不公平性

- 背景
  - **现代SSD：**固态硬盘（SSD）因其高吞吐量、低响应时间和低功耗被广泛应用
  - **性能瓶颈：**传统主机接口协议（如SATA）成为性能瓶颈，随着NAND闪存技术的发展，这些协议无法充分利用SSD性能。
  - **新协议：**使用新的高性能协议（如NVMe），消除了操作系统软件栈的需求，提高了性能，但也带来了新的问题
- 问题
  - **公平性控制缺失**：

    移除操作系统软件栈后，缺乏公平性控制机制，导致多个应用程序同时访问共享SSD时的性能不公平。

    a) 当两个应用程序共享同一SSD时，一个应用程序的执行时间显著增加。

    b) 向MQ-SSD发出大量I/O请求的应用程序可能会饿死其他应用程序的请求，影响整体系统性能。

研究问题：NVMe SSD 多租户的技术手段

干扰类型

1. **流强度**：流强度较高的I/O流（即该流生成事务的速率）会使流强度较低的I/O流变慢，在实验中，慢了最多49倍。
2. **访问模式**：当高并行性I/O流与低并行性I/O流同时运行时，低并行性I/O流会阻止高并行性I/O流的事务在多个芯片上同时完成，导致性能下降（最多2.4倍）
3. **读写比例**：因为SSD写操作通常比读操作慢10到40倍，一个具有较大写请求比例的流会不公平地减慢同时运行的、具有较小写请求比例的流。
4. **垃圾回收**：SSD执行GC，当一个I/O流需要频繁的GC操作时，这些操作会阻止另一个很少进行GC操作的流的I/O请求，在实验中，最多减慢3.5倍.

FLIN

Flash-Level INterference-aware scheduler (FLIN)

FLIN（Flash-Level INterference-aware scheduler）是一个新的MQ-SSD事务调度单元，旨在通过调度和重新排序事务，显著减少I/O流之间的干扰，从而提供公平性。
FLIN通过三个主要阶段来实现这一目标：

① 公平感知队列插入（Fairness-Aware Queue Insertion）

FLIN将每个I/O请求的新事务插入到相应的读/写队列中。插入的位置取决于I/O流的当前强度

② 优先级感知队列仲裁（Priority-Aware Queue Arbitration）

FLIN使用主机为每个I/O流分配的优先级来确定每个后端内存芯片的下一个读请求和下一个写请求

③ 等待时间平衡事务选择（Wait-Balancing Transaction Selection）

FLIN选择要派发到FCC的事务，减轻不同I/O流之间由于读/写比率差异和GC引入的干扰

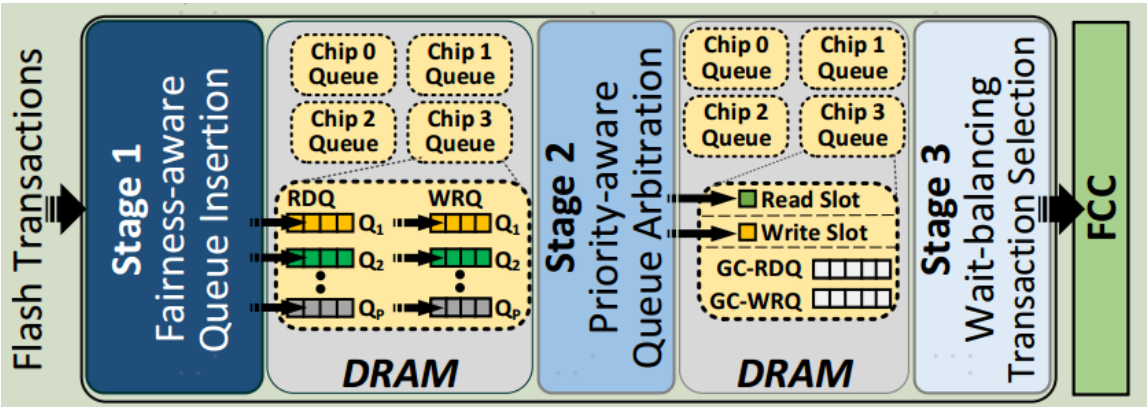


Figure 8: High-level overview of FLIN.

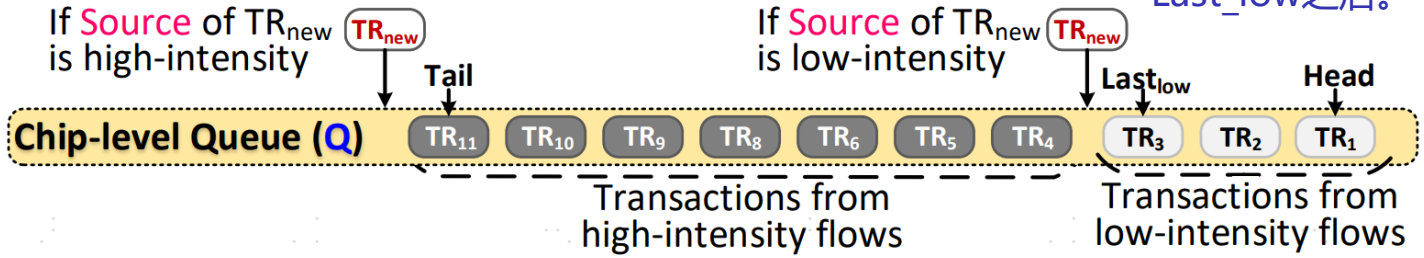
FLIN

Step1: Fairness-Aware Queue Insertion

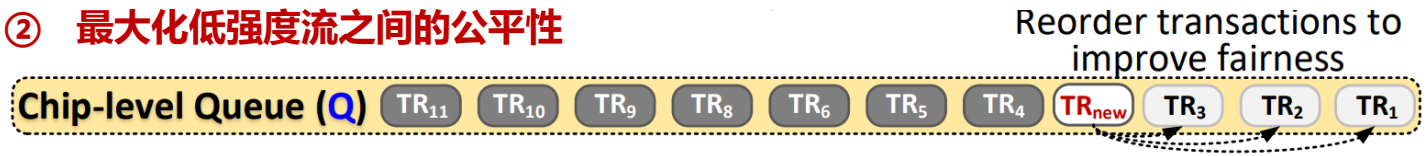
① **优先低强度流**

如果新事务来自高强度流，则将其插入到队列尾部。

① 如果**新事务（TR_new）**来自低强度流，则将其插入到Last_low之后。

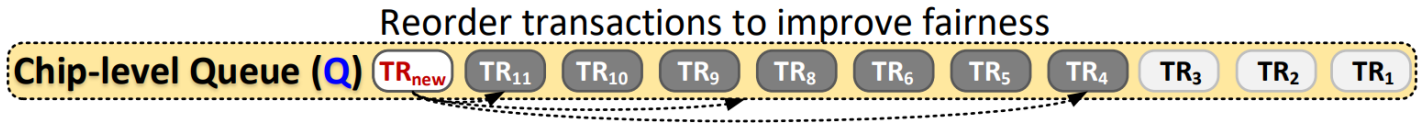


② **最大化低强度流之间的公平性**



② 根据估计的**等待时间**和**减速情况**，将低强度流的事务重新排序，以最大化所有低优先级流中待处理事务的最小减速与最大减速的比率。

③ **最大化高强度流之间的公平性**



③ 估计TR_new的等待时间，然后检查当前的公平性。如果公平性低于预定阈值（F_thr），且新事务属于减速最大的流，则将事务移到所有其他高强度流事务之前。如果公平性高于阈值，则只会重新排序高强度流的事务以实现公平性。

FLIN

Step1: Fairness-Aware Queue Insertion

- **动态分类I/O流的强度**

  在固定长度的时间间隔（例如 10 ms）内进行分类，并记录每个I/O流的读写事务数量。在时间间隔结束时，FLIN使用记录的事务计数计算每个I/O流的读写到达率。如果一个I/O流的 Read 到达率低于预定的阈值 α_{read}，则将其分类为低强度读流；否则，分类为高强度读流。Write 同理。

- **估算独立等待时间和减速**

  ① 计算一个事务从进入队列到完成所有操作的总时间，包括执行、传输和等待时间

  $$T^{TR} = T^{TR}_{memory} + T^{TR}_{transfer} + T^{TR}_{wait}$$

  T^{TR}_{memory} 和 $T^{TR}_{transfer}$ 是在内存芯片中执行命令和数据传输所需的时间。
  T^{TR}_{wait} 是事务在读或写队列中等待的时间。

  ② 估算一个事务在没有其他流干扰的情况下的独立等待时间

  $$T^{TR}_{wait\_alone} = T^{last}_{enqueue} + T^{last}_{alone} - T_{now}$$

  $T^{last}_{enqueue}$：最后一个请求入队的时间戳。

  T^{last}_{alone}：最后一个请求的估算完成时间。

  T_{now}：当前时间戳

  减速计算公式： $S^{TR} = T^{TR}_{shared}/T^{TR}_{wait}$

  ③ 估算共享等待时间，一个事务在存在其他流干扰的情况下，总的等待时间

  $$T^{TR}_{wait\_shared} = T_{now} - T_{enqueued} + T_{chip\_busy} + \sum_{i=1}^{p-1}\left(T^{i}_{memory} + T^{i}_{transfer}\right)$$

  队列中所有未执行事务的执行和传输时间

  $T_{now} - T_{enqueued}$ 是事务入队以来的等待时间。

  T_{chip_busy} 是当前执行的事务剩余的服务时间。

FLIN

Step2: Priority-Aware Queue Arbitration

- **架构**
  1. **维护队列：**

     FLIN在SSD的DRAM中为每个优先级水平维护一个 Read 队列和一个 Write 队列。

     (如果协议定义了 p 个优先级水平，FLIN为每个闪存芯片包含 p 个 Read 队列和 p 个 Write 队列)
  2. **事务选择：**

     从 p 个 Read 队列的队列头部选择一个就绪的读事务，从 p 个 Write 队列的队列头部选择一个就绪的写事务。这两个选定的事务随后进入调度器的最后阶段进行处理
- **加权轮询策略**
  1. **策略**

     当多个队列在队列头部有就绪执行的事务时，FLIN使用**加权轮询策略**选择将要进入下一个调度阶段的读和写事务。
  2. **权重分配**

     若协议定义了p个优先级水平，0最低，p-1最高. FLIN为优先级水平 **i** 分配权重 2^i。在**加权轮询策略**下，第二阶段的每 2^i 次调度决策中，优先级水平 **i** 队列中的事务将获得 2^i 个调度时隙

FLIN

Step2: Priority-Aware Queue Arbitration

- **FLIN的优先级感知队列仲裁**

假设有三个I/O流：A, B和C, 优先级分别是0, 1和2.

权重分配：

- $W_A = 2^0 = 1$, 同时意味 A 获得 1 次调度机会
- $W_B = 2^1 = 2$, 同时意味 B 获得 2 次调度机会
- $W_C = 2^2 = 4$, 同时意味 C 获得 4 次调度机会

调度决策：

① 调度一个 C 的 **Read** 事务
② 调度一个 C 的 **Write** 事务
③ 调度一个 C 的 **Read** 事务
④ 调度一个 C 的 **Write** 事务
⑤ 调度一个 B 的 **Read** 事务
⑥ 调度一个 B 的 **Write** 事务
⑦ 调度一个 A 的 **Read** 事务

FLIN

Step3: Wait-Balancing Transaction Selection

在将读写事务的等待时间均衡分配，以最小化由于读/写比例和垃圾回收需求引起的干扰

1. 估计比例等待时间(Proportional Wait Time)

$$PW = \frac{T_{wait}}{T_{memory} + T_{transfer}}$$

T_{memory} 是事务在内存芯片中执行所需的时间。
$T_{transfer}$ 是数据传输所需的时间。
T_{wait} 是从主机发起I/O请求到事务完成的总等待时间。

如果PW_{read}大于PW_{write}，则优先处理 Read 事务；否则，优先处理 Write 事务

2. 派发事务(Dispatch Transactions)

- Read 事务优先，派发 Read slot 里的事务到FCC
- Write 事务优先，调度器考虑是否同时派发 GC操作

  如果内存中的可用Page低于预定阈值（GC_FLIN），且垃圾回收队列中有等待的事务：
  ① 确定要执行的垃圾回收事务数量（GCMigrationCount）。
  ② 执行GCM个垃圾回收事务（FLIN始终执行一对读写事务）。
  ③ 完成垃圾回收事务后，调度器派发写事务。

优化措施：

- Write事务挂起：当Read slot中的事务的*PW*大于当前执行的Write操作的*PW*时，FLIN支持Write事务挂起，以避免读请求被不必要地阻塞。
- GC事务的处理：当Read slot为空时，FLIN会派发一对GC读/写事务。如果在执行GC事务时有新事务到达读槽位，FLIN会中断GC写操作并执行Read slot中的事务，以避免新到的Read请求被不必要地阻塞。

FLIN

Step3: Wait-Balancing Transaction Selection

$$PW = \frac{T_{wait}}{T_{memory} + T_{transfer}}$$

$$T_{wait} = T_{now} - T_{enqueued} + T_{other\_slot} + T_{GC}$$

$T_{now} - T_{enqueued}$ 表示事务已经等待的时间。
T_{other_slot} 是首先执行另一个槽位中操作所需的时间。
T_{GC} 是执行待定垃圾回收事务所需的时间。

如果选择Read slot中的事务，不执行GC事务，因此 T_{GC}=0。
如果选择Write slot中的事务， T_{GC} 可能为非零。

$$T_{other\_slot} = T_{memory} + T_{transfer} + T_{GC}$$

若正在考虑的事务是 Write 事务, 且有待执行的GC操作，T_{GC}需要估算

$$T_{GC} = \sum_{i=1}^{GCM} (2T_{transfer} + T_{memory}^{read} + T_{memory}^{write}) + T_{memory}^{erase}$$

$$GCM = \frac{NumWrites_f}{\sum_i NumWrites_i} \times \frac{Valid_f}{\sum_i Valid_i} \times length_{GC-RDQ}$$

- $NumWrites_f$ 是自上次垃圾回收操作以来该流执行的写操作数量。
- $Valid_f$ 是SSD中属于该流的有效页面数量（通过FTL页面管理机制维护的有效页面历史记录确定）。
- $length_{GC\text{-}RDQ}$ 是排队的垃圾回收读事务的数量（表示所有待执行垃圾回收迁移的总数）。

WA-OPShare: Workload-Adaptive Over-Provisioning Space Allocation for Multi-Tenant SSDs

WA-OPShare

多租户SSD

- 多租户SSD——竞争有限的存储资源
  - 固态盘虚拟化技术支持
    - PCIe SR-IOV
    - NVMe namespace/set
- 多租户SSD中的性能公平
  - IO调度
    - ✓ 基于预算的I/O处理
    - ✓ 基于公平队列的I/O调度
    - ✓ 启发式I/O调度
  - 空间资源分配
    - ✓ 动态调整缓存
    - ✓ **OPS** 等资源的分配

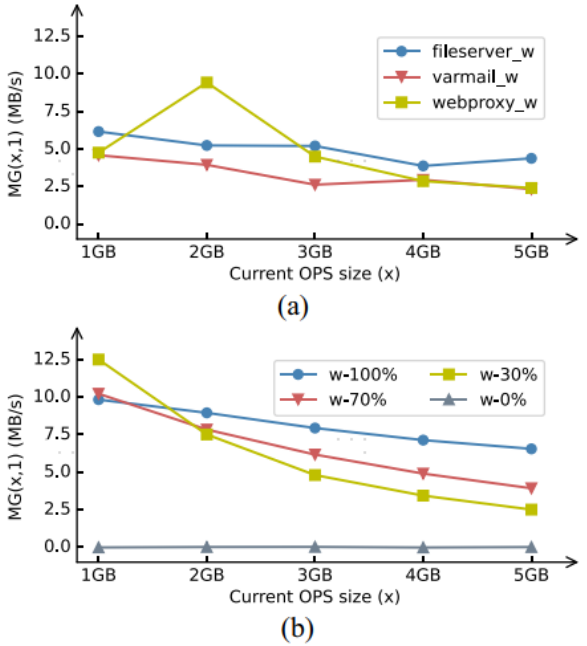
➢闪存具有 “先擦除后写入”特性，SSD 需要垃圾回收（GC）操作来回收无效页面
➢额外的用户不可见的闪存存储空间，预留空间 (Over-provising Space, OPS)，其大小会影响GC效率

WA-OPShare

多租户SSD预留空间资源

- 预留空间 (Over-provising Space, OPS)：租户不可使用的SSD物理空间
  - ➢ 固态盘垃圾回收需要迁移数据，OPS越多，闪存块内平均有效页数越少，GC开销越低
  - ✓ **提高GC效率**（迁移更少的页，完成更快
  - ✓ **降低写放大**（GC导致的数据迁移更少）
  - ✓ **提升固态盘性能及寿命**

- ➢ **边缘收益MG(x,1)**：额外使用1GB OPS资源带来的带宽增加
  - ✓ **写入模式**不同，边缘增益不同
  - ✓ OPS资源越多，增加OPS带来的提升越有限；随OPS资源的增加时，该限制逐渐减小
  - ✓ **写入占比**更高的工作负载具有更高增益



WA-OPShare

多租户SSD空间资源分配优化方法

- 传统空间分配方案难以最大化资源利用率，存在性能损失
  - ➢ 多租户共享SSD:
  - ✓ 固态盘时空资源：IO时间+OPS空间

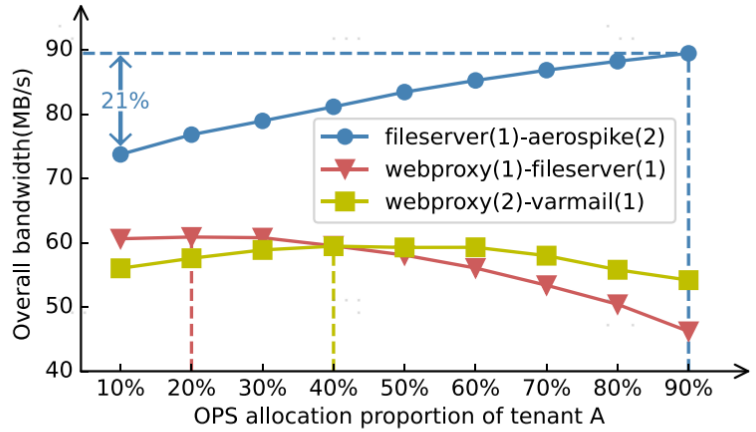


Fig. 1. Overall bandwidth of the Partition scheme under different OPS allocation configurations between two tenants working concurrently in the simulated SSD. "$X(a) - Y(b)$" means tenant A runs the workload X with weight a and tenant B runs the workload Y with weight b.

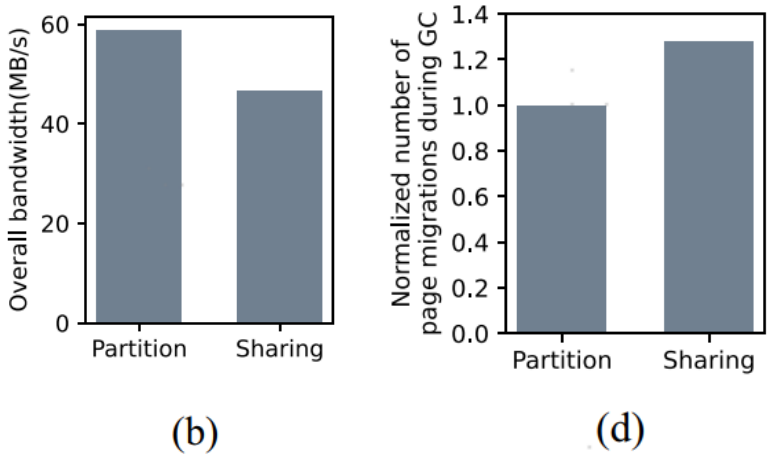
- ➢ **传统分区方案不能适应负载的动态变化**
- ➢ **最佳分区配置比不同**
- ✓ 分别为2:8、4:6和9:1
- ➢ **不恰当的分区配置会导致糟糕的性能**
- ✓ 蓝色线：最差性能与最佳性能相差21%

WA-OPShare

多租户SSD空间资源分配优化方法

- 在多租户间分享OPS空间资源:

Partition方案	静态分区 OPS资源，隔离性好 难以适应负载变化，难以预测最佳配置
Sharing方案	自由竞争 OPS 资源 无法利用负载特征最大化 OPS 资源利用率



分区方案与共享方案下的SSD**性能/GC开销**对比

- ✓ Sharing方案比Partition方案:
- ✓ 有20%的带宽损失
- ✓ GC 开销增加 28%

WA-OPShare

动态自适应OPS资源分配策略 WA-OPShare

- WA-OPShare: 负载自适应动态调整OPS资源分配方案

➢ **OPS回收策略：Lazy Superblock Compaction**

✓ 根据闪存页无效时间，建立闪存块空间利用率模型，定期擦除回收利用率最低的闪存块

➢ **OPS动态分配策略：OPS Allocator**

✓ 检测各租户对OPS资源的需求，计算OPS资源给各租户带来的性能收益

✓ 将回收得到的空闲闪存块分配给性能收益最大的租户，最大限度地利用资源

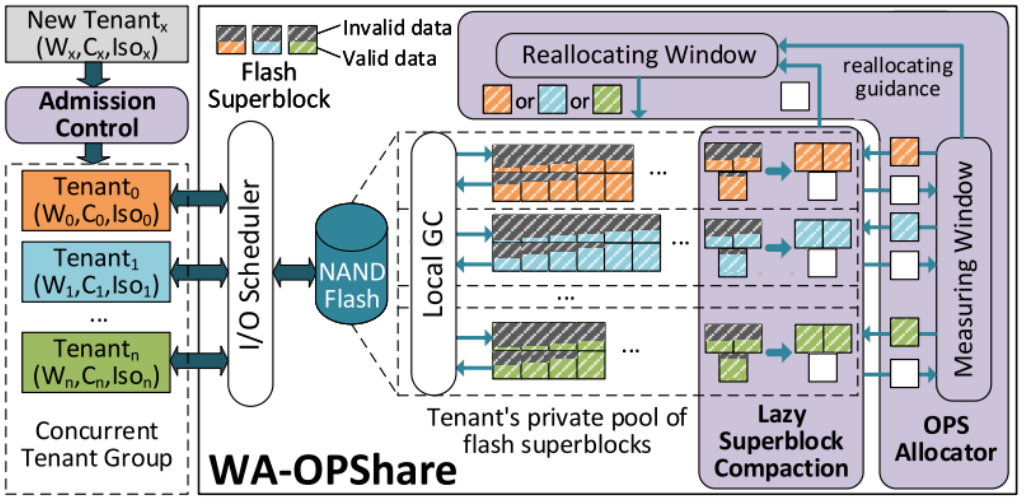


Fig. 4. Architecture of WA-OPShare.

WA-OPShare

动态自适应OPS资源分配策略 WA-OPShare

- **OPS回收策略：Lazy Superblock Compaction**
  - ✓ Lazy superblock：由于工作负载的访问局限性与可变性，可能存在长期未擦除的闪存超级块，并包含大量的无效闪存页，降低了SSD空间利用率
  - ✓ 周期性回收 efficiency 最低的block

$$\text{Efficiency}_i = \frac{\text{Ratio}_{\text{valid},i}}{\text{Laziness}_i} = \frac{N_{\text{valid},i}}{N \times \sum_{j=1}^{N} \text{ZombieTime}_{i,j}} \quad (1)$$

$$\sum_{j=1}^{N} \text{ZombieTime}_{i,j} = N \times \left(T_{cur,i} - T_{\text{written},i}\right) - \text{ALT}_i. \quad (2)$$

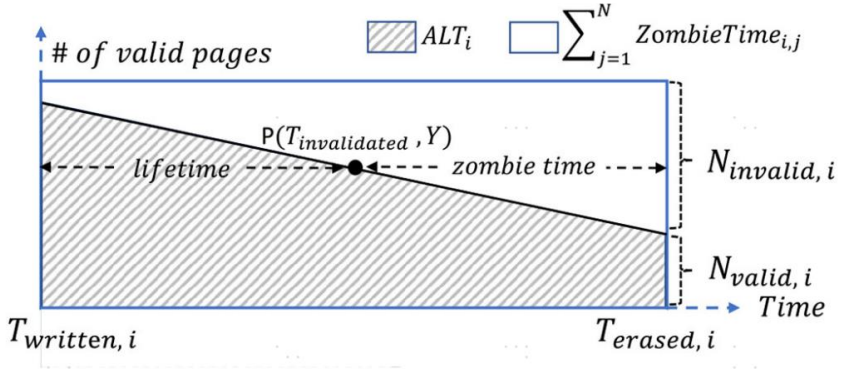


Fig. 5. Lifetime and zombie time of data pages in superblock i.

ALT_i Super block i 的累积生命周期

$\sum_{j=1}^{N} \text{ZombieTime}_{i,j}$ Super block I zombie时间累和

- **白色部分越大，块内无效空间和未被利用的时间累和越长，与efficiency成反比**

WA-OPShare

动态自适应OPS资源分配策略 WA-OPShare

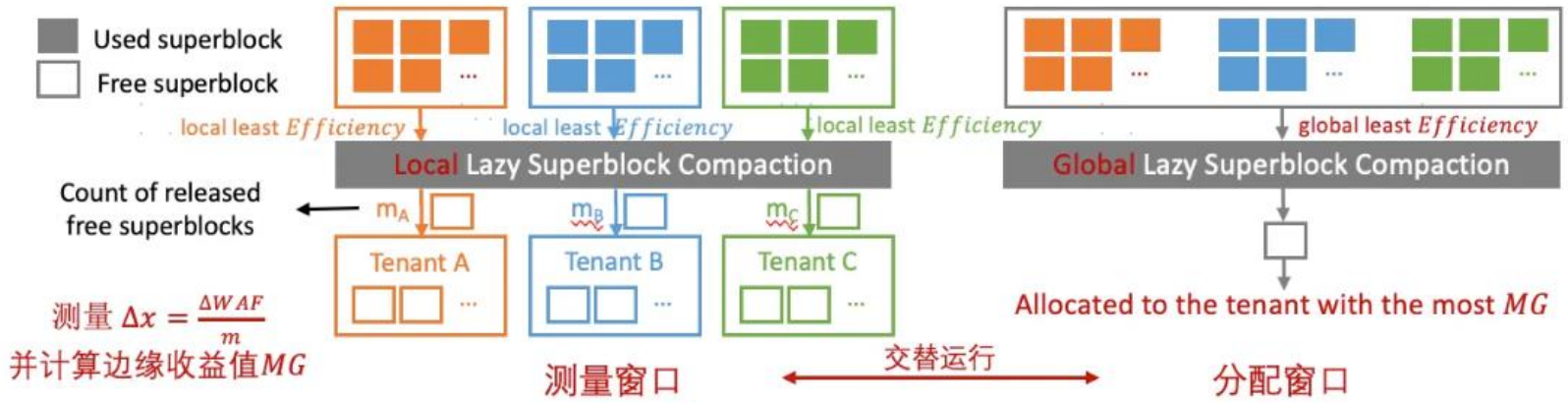
- **OPS动态分配策略 OPS Allocator：分配给OPS资源收益最大的租户**
  - ✓ 边缘收益（Marginal gain，MG）：增加OPS资源，可以获得的带宽收益

$$\text{Bandwidth} = N_{\text{host}} \times \text{PageSize} \propto \frac{\text{weight}}{\alpha_w \times \left[\text{WAF} \times \left(t_r + t_w + \frac{t_e}{P}\right) - 2t_r\right] + t_r}$$

$$\text{MG} = \text{Bandwidth}(\text{WAF} - \Delta x) - \text{Bandwidth}(\text{WAF})$$

tw,tr,te: page写入、读取、擦除的时间
P: block中page的个数
α_w：host write的占比

Δx：增加OPS资源带来的WAF降低



在测量窗口和再分配窗口间交替运行，测量窗口用于测量每个租户的边缘收益值，再分配窗口期间将回收的空间资源重新分配给边缘收益值最大的租户，最终实现负载自适应的空间资源动态调整，提升系统性能。

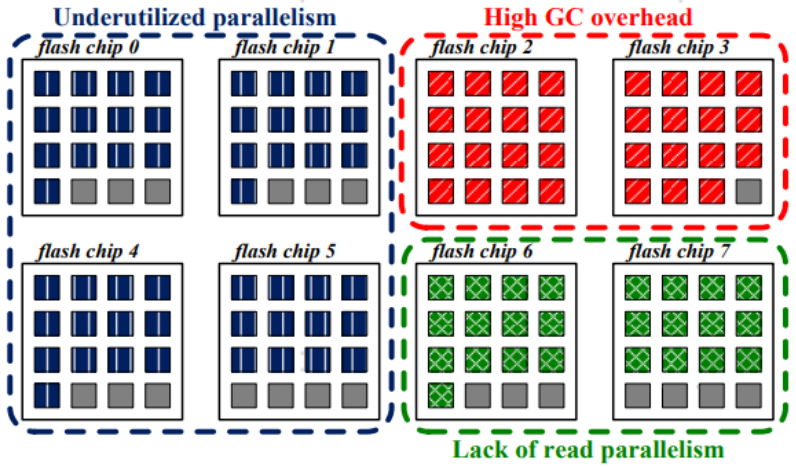
Utilitarian Performance Isolation in Shared SSDs

Utilitarian Performance Isolation in Shared SSDs

多租户环境的静态隔离

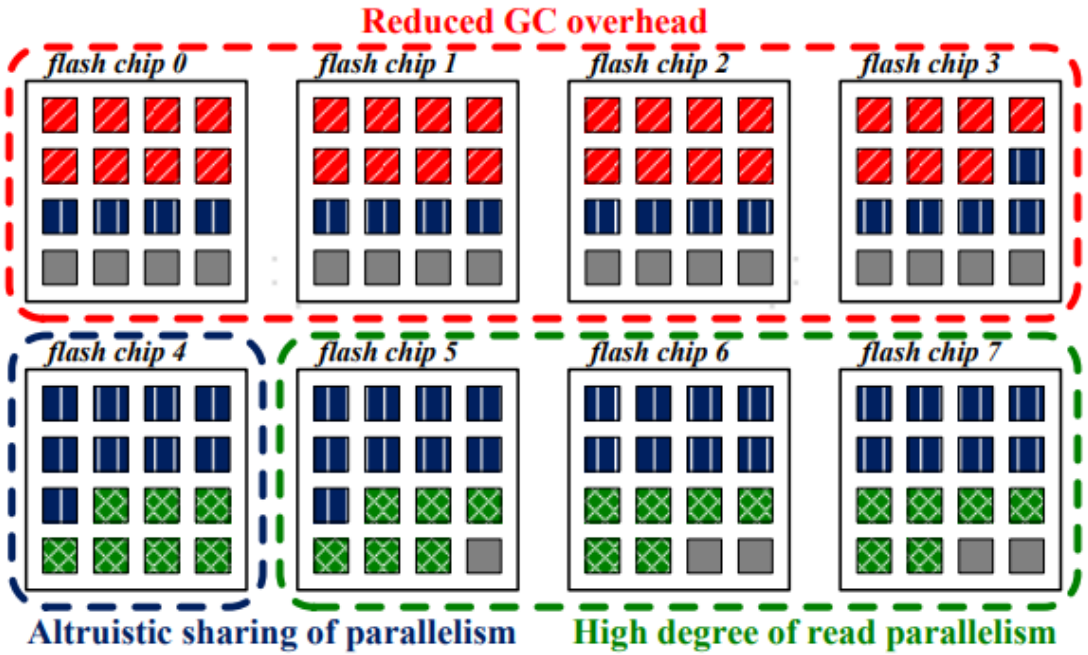
- 多租户存储需求
  - **性能隔离**：确保一个租户的操作不会影响其他租户的存储性能。。
- 传统解决方法
  - **静态资源分配方法**（预先分配固定数量的资源给每个租户）

    往往会导致资源利用率低下，尤其是在资源需求波动的情况下。



Utilitarian Performance Isolation in Shared SSDs

研究问题： 功利性的性能隔离



(b) Dynamic allocation in UPI.

初步分布写数据的时候，不把某个chip具体地分配给某个租户，相反允许各个租户的数据写到每一个nand flash block。

Utilitarian Performance Isolation in Shared SSDs

研究问题： 功利性的性能隔离

$$Util(t,S) = \frac{\sum_{c}^{chips} N_r(t,c)}{\sum_{c}^{chips} \left(\frac{N_r(t,c)}{1 - Traffic(c,S)}\right)} \qquad (1)$$

$$Traffic(c,S) = \frac{\sum_{t}^{tenants} \left(N_r(t,c)\cdot\tau_r + N_p(t,c,S)\cdot\tau_p\right)}{Time_{window}}$$

Nr(t, c):从租户t和芯片c的期望的读
Tra ffic(c,S):给定S芯片c的繁忙程度

利用效率函数，找到占用率最大的租户，移动一个nand flash芯片到另外占用率最低的租户。通过重移动数据，得到大致相同的占用率，提高多租户间的性能公平和性能隔离

Everything for Memory