

**课程论文**

**PipeDream: Generalized Pipeline Parallelism for DNN Training**

|  |  |
| --- | --- |
| **学院** | 计算机科学与技术 |
| **班级** | 计科本硕博2001班 |
| **老师** | 胡燏翀 |
| **姓名** | 柳子淇 |
| **学号** | U202015628 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **2024** | **年** | **1** | **月** | **4** | **日** |

目录

[论文概述 3](#_Toc155293019)

[1.1论文摘要 3](#_Toc155293020)

[1.2研究背景 3](#_Toc155293021)

[1.2.1 CXL技术带来的机遇 4](#_Toc155293022)

[1.2.2用闪存介质作为内存的挑战 4](#_Toc155293023)

[1.3问题描述与方法介绍 5](#_Toc155293024)

[1.3.1 内存跟踪工具 5](#_Toc155293025)

[1.3.2 CXL-flash设计 6](#_Toc155293026)

[1.4实验结果 9](#_Toc155293027)

[1.4.1缓存替换策略 9](#_Toc155293028)

[1.4.2预取策略 11](#_Toc155293029)

[论文理解 13](#_Toc155293030)

[1.1优点分析 13](#_Toc155293031)

[1.2缺点分析 13](#_Toc155293032)

[课程感想 14](#_Toc155293033)

[同学评议 15](#_Toc155293034)

[报告一：Amazon DynamoDB 15](#_Toc155293035)

[报告二：GL-Cache 15](#_Toc155293036)

[报告三：CXL分布式共享内存的部分故障弹性内存管理系统 16](#_Toc155293037)

[报告四：FUSEE 16](#_Toc155293038)

[报告五：使用Trio架构实现高性能和安全的用户空间NVM文件系统 17](#_Toc155293039)

[报告六：分布式训练中的检查点恢复 17](#_Toc155293040)

[报告七：RubbleDB 18](#_Toc155293041)

[报告八：eZNS: 新型存储设备的弹性接口 19](#_Toc155293042)

[参考文献 21](#_Toc155293043)

# 论文概述

本文探讨了利用成本较低的闪存内存（flash memory），结合最新的互连技术（如CXL），来应对“内存墙”问题。研究背景聚焦于现有内存系统的局限性，尤其是在处理大数据和AI应用时的性能瓶颈。通过引入CXL技术，来使得SSD实现更高的内存带宽和容量，同时降低延迟，以达到接近DRAM的性能。

## 1.1论文摘要

摘要部分提到了这篇论文调查了使用CXL（Compute Express Link）技术和廉价闪存内存来克服内存墙的可行性。重点在于探索CXL-Flash设备的设计空间，并展示了缓存和预取等技术如何帮助缓解闪存内存的性能和寿命问题。使用真实应用的追踪数据，论文展示了这些技术使得CXL设备的预计寿命至少为3.1年，能够在一微秒以下处理68-91%的内存请求。此外，还分析了现有技术的局限性，并提出了系统级的改变，以实现使用闪存达到DRAM级别的性能。

## 1.2研究背景

计算系统中计算能力与内存需求之间不平衡的增长，形成了所谓的“内存墙”挑战。例如，自然语言处理（NLP）模型的参数数量每年增长14.1倍，而内存容量增长却远远落后，仅为每年1.3倍。这迫使数据密集型应用程序采用更高效的内存使用策略或用户级内存管理以避免昂贵的页面交换。"内存墙"问题是指计算系统中内存带宽和容量的增长速度无法跟上处理器性能的提升速度。这导致了处理器在等待数据时的效率降低，从而限制了整个系统的性能。内存墙问题在处理大数据和高性能计算应用时尤为显著，因为这些应用需要大量、快速的数据处理和内存访问。论文强调了克服内存墙问题对提升数据中心和高性能计算系统性能的重要性。

### 1.2.1 CXL技术带来的机遇

探讨了CXL技术如何提供一种新的解决方案来克服内存墙问题。CXL技术能够整合CPU、加速器和内存设备，使得这些设备间的内存访问更为高效。文中还强调了CXL技术在降低延迟和提高带宽方面的潜力。CXL允许更紧密的集成和更高效的内存一致性协议，这对于提升数据中心和高性能计算系统的性能至关重要。CXL技术通过整合CPU、加速器和内存设备，能够显著提高内存带宽和降低延迟，从而解决传统内存系统中的瓶颈问题。

### 1.2.2用闪存介质作为内存的挑战

论文在这一部分中详细阐述了将闪存作为主内存使用时面临的挑战。这些挑战包括存储粒度不匹配、相对于DRAM的高延迟和有限的寿命。

**粒度不匹配**。闪存不是随机访问的：其数据以页粒度写入和读取，每个页的大小约为几千字节，导致大量的流量放大。此外，页不能被直接覆写，相反，必须首先擦除一个包含数百个页的块，然后才能写入数据到已擦除的页。这种接口限制导致任何64B缓存行刷新操作都会在读取-修改-写入操作流程中产生大量的写放大。作为一个块设备，访问粒度要大得多（4KiB）使得SSD有更多的开销。

**微秒级延迟。**闪存的速度比DRAM慢几个数量级，其读取速度仍在几十微秒范围内，而较慢的编程和擦除操作则在几百微秒到几千微秒之间。此外，闪存的延迟还取决于其单元技术。作为一个块设备，相较于软件实现的存储栈中存在的开销，微秒级的延迟是可以容忍的。然而，对于直接使用load/store指令访问的内存设备来说，延迟过高是一个严重的问题。

**有限的寿命。**写入和擦除操作期间施加在闪存上的高电压会使单元磨损甚至失效。由于应用程序级和内核级的Cache的存在，SSD的块接口的写入量并不大，因此当前的耐久性限制在作为SSD的寿命内使用通常是足够的。然而，作为内存设备，频繁的内存写入会使闪存介质迅速磨损。

文章指出，尽管在传统存储系统中已有解决方案，但在CXL-flash环境中，这些挑战需要通过更精细和高效的硬件设计来应对。这对于CXL-Flash设备的性能和寿命具有重大影响。

## 1.3问题描述与方法介绍

### 1.3.1 内存跟踪工具

传统跟踪工具的局限性：主存和CXL-flash通过物理内存地址进行访问。遗憾的是，目前我们所知，没有公开可用的工具可以在不进行硬件修改的情况下跟踪最后一级缓存（LLC）和内存控制器之间的物理内存事务。仅仅跟踪CPU中的load/store指令是不够的，有两点原因：（1）它只收集虚拟地址访问，（2）对CXL-flash的访问会被缓存层次结构过滤掉。

设计了新的内存跟踪工具，以追踪真实直达物理设备的访问。通过结合来自Valgrind的内存跟踪和页面错误事件信息来跟踪物理内存访问。如图1所示，使用Valgrind对应用程序的load/store指令进行工具化，并使用其缓存模拟器(Cachegrind)来过滤对内存的访问。然而，Cachegrind产生的这些内存访问仍然是虚拟地址，因此需要虚拟到物理（V2P）映射信息来生成物理内存跟踪。因此需要收集应用程序运行时由页面错误引起的页表更新。我们修改了安装页面表项的内核函数（do\_anonymous\_page()和do\_set\_pte()），并将目标应用程序的PID的V2P转换存储在/proc文件系统中。这样做可以捕获应用程序执行过程中页面表更新的动态特性，并且开销很小。最终，我们将来自Valgrind的虚拟访问和页面表更新结合起来生成物理内存跟踪。

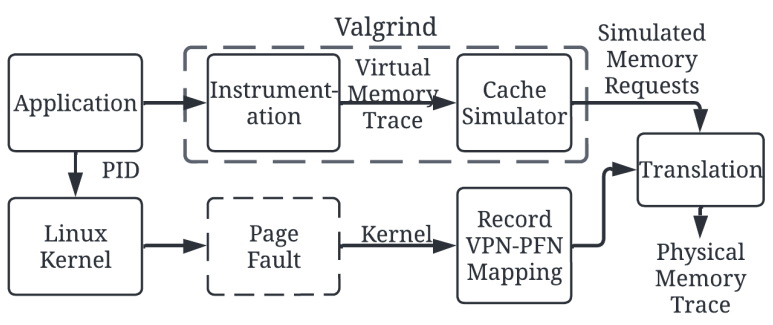


图 1 收集物理内存跟踪的工作流程

### 1.3.2 CXL-flash设计

探索CXL-flash的设计空间，特别关注其中的硬件模块。为了模拟硬件，基于MQSim 及其扩展MQSim-E 构建了一个CXL-flash模拟器，并使用五个合成应用程序的物理内存跟踪来评估设计带来的影响。CXL-flash的整体架构如图2所示，并在图3中给出了参数配置。

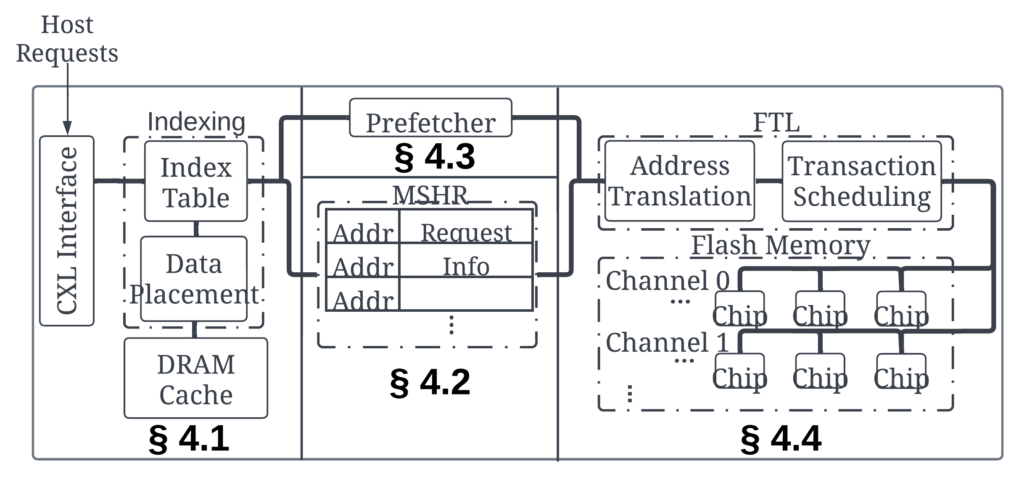


图 2 CXL-flash的架构图

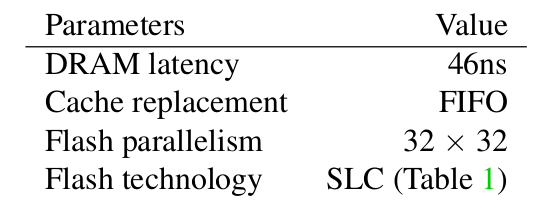


图 3 CXL-flash的配置参数

**缓存对性能的影响**

缓存在提高性能方面非常有效。通过在CXL-flash中引入缓存，可以减少对慢速闪存的访问次数，从而大大降低访问延迟。缓存能够存储最近使用的数据，以供快速访问，从而加快了读取和写入操作的速度。

**减少闪存访问量**

为了解决访问粒度不匹配的问题并有效地减少闪存访问量。为了解决这个问题，我们在CXL-flash中引入了一组MSHR（miss status holding registers），受到CPU缓存的启发。MSHR用于跟踪当前未完成的闪存内存请求，并将单个闪存内存读取服务多个64B内存访问。通过使用MSHR，可以显著降低长尾延迟，特别是对于具有大量重复读取的工作负载。图4显示了使用（实线）和不使用（虚线）MSHR时的延迟分布。通过添加MSHR，可以减少闪存内存流量，但无法通过主动提高缓存命中率来改善性能。

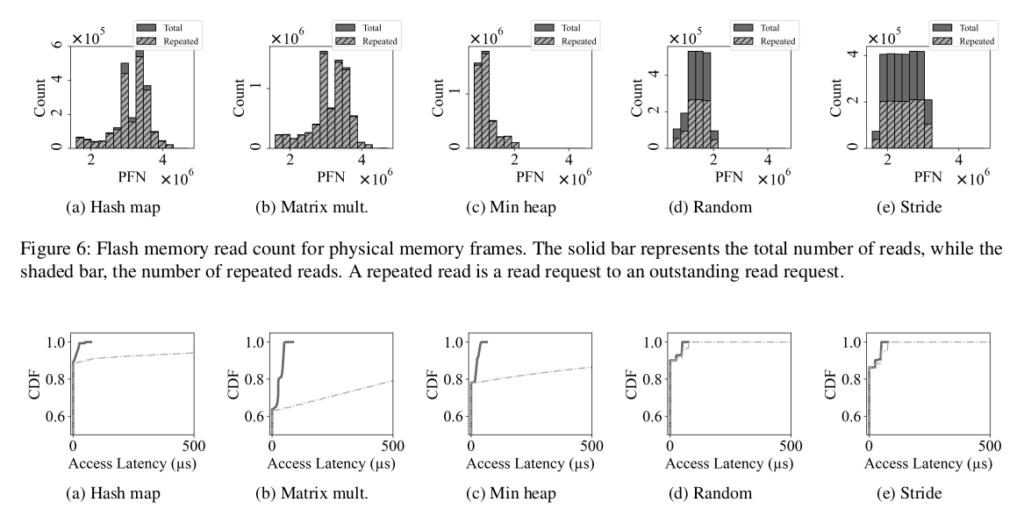


图 4 使用（实线）和不使用（虚线）MSHR的延迟分布。

**预取技术**

预取技术在隐藏长延迟的闪存访问方面非常有效。通过预先从闪存中提取数据并将其存储在高速缓存中，可以避免后续访问时的延迟。预取技术可以利用局部性原理，提前将可能需要的数据加载到缓存中，以减少对闪存的实际访问次数，并提高整体性能。效果如图5所示。

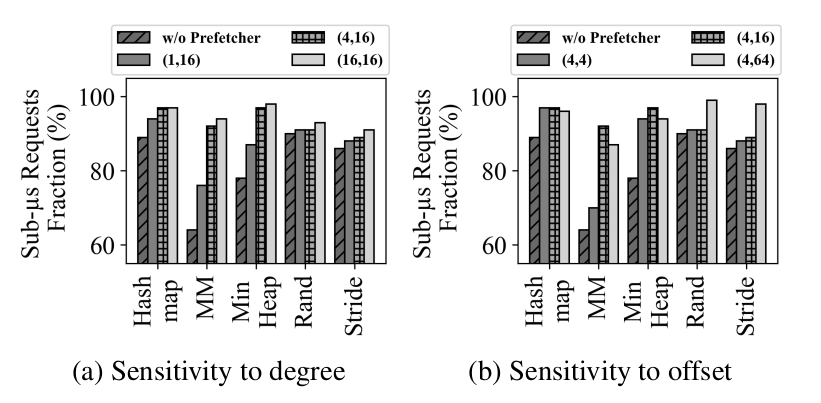


图 5  具有不同预取器配置的CXL闪存的性能。(X, Y)表示Next-N-line预取器的度和偏移。

**CXL-flash适合何种介质及其并行性**

在随机工作负载中，较小的缓存会导致更多的缓存未命中，从而增加了对闪存的访问次数，而较大的缓存可以更好地隐藏访问延迟。对于步幅工作负载，预取器可以有效地隐藏访问延迟，因此缓存大小的影响较小，但更高的闪存并行性可以提供更多的并行性，从而提高性能。

如图6，7所示，实验结果显示，对于CXL-flash，选择适当的闪存技术和并行性对性能和寿命都非常重要。在存在缓存的情况下，ULL和SLC技术之间的性能差异可以忽略，而MLC和TLC技术的性能较差。增加缓存大小可以改善性能和寿命，特别是对于MLC和TLC技术来说。闪存并行性对于较小的缓存大小特别重要，而对于较大的缓存和步幅工作负载来说，影响较小。因此，在设计CXL-flash系统时，需要综合考虑闪存技术、并行性和缓存大小，以获得最佳性能和寿命。

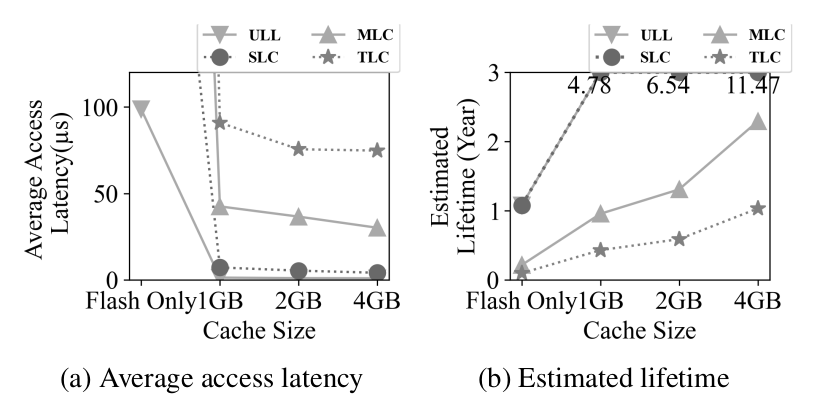


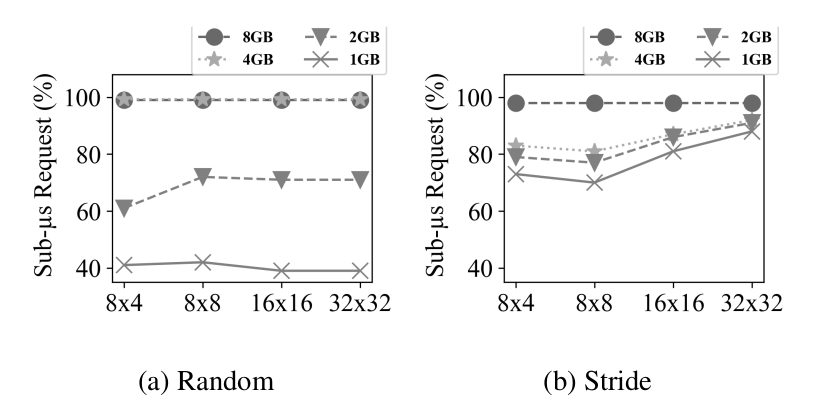
图 6 对CXL闪存性能和寿命的闪存技术和缓存大小敏感性测试。

图 7 不同闪存并行性和缓存大小的亚微秒请求百分比。x轴表示闪存内存并行性（通道数×路数）。线表示不同缓存大小的值。

## 1.4实验结果

实验评估部分使用了五个不同的真实应用程序进行评估，这些应用程序来自各个领域，并且均为内存密集型的。评估的重点是高级缓存和预取策略的效果和性能。

### 1.4.1缓存替换策略

与前文对缓存大小的考察不同，在这里我们固定缓存大小，并评估不同缓存替换策略在不同关联性下的影响。具体而言，我们实现了以下四种策略：

FIFO（先进先出）：淘汰最旧的数据。

Random（随机）：随机选择数据进行淘汰。

LRU（最近最少使用）：淘汰最近未被使用的数据。

CFLRU（Clean First LRU）：优先淘汰干净的缓存行，而不是被修改过的缓存行。

我们选择Random作为基准策略，而FIFO和LRU是两种可在硬件中实现的标准CPU缓存策略。为了进一步减少流量并延长设备的使用寿命，我们实现了CFLRU，以探索优先淘汰干净缓存行以减少闪存写入活动的好处。

图8衡量了对CXL-flash的内存请求的微秒以下延迟的百分比，图9显示了闪存写入次数。

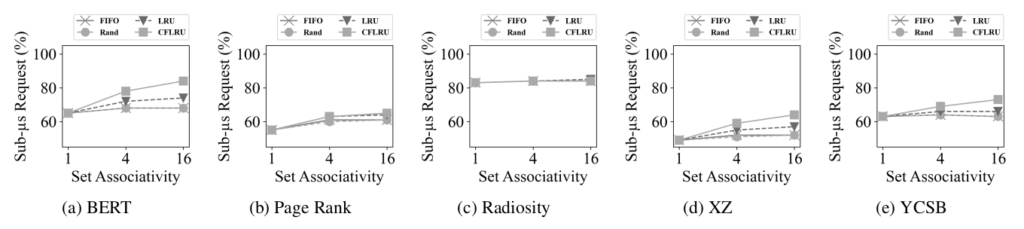


图 8  基于缓存替换策略和关联性的CXL闪存亚微秒延迟百分比

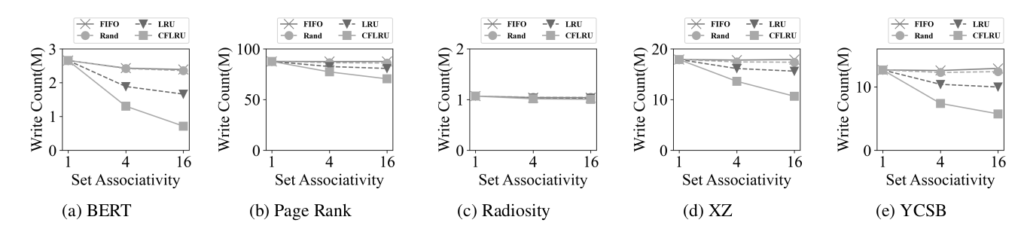


图 9 基于缓存替换策略和关联性的闪存写请求数量

可以从如上这些图表中得出五个观察结果：

* + - 1. 增加关联性可以提高性能，因为它增加了缓存命中率。对于缓存失效开销较高的系统来说，增加命中率比减少命中时间更具影响力。
      2. CFLRU在其他替换策略中表现最好，特别是在BERT、XZ和YCSB工作负载中。这得到了图12中写入流量显著减少的支持。
      3. 具有高局部性的工作负载（如Radiosity）对缓存替换策略不敏感：不论采用哪种策略，至少83%的请求具有亚微秒的延迟。
      4. 以读为主的工作负载通常比以写为主的工作负载表现更好，因为闪存程序延迟相对于读取来说更大。例如，BERT和Radiosity只产生了0.7M和1.0M的闪存写入，相应地，它们的亚微秒延迟部分分别高达84%和85%。
      5. 具有低局部性的工作负载不仅表现较差，而且对缓存策略不太敏感。特别是对于Page Rank工作负载，如图11b所示，最多只有65%的请求能够达到亚微秒的延迟，这是由于其低局部性和较大的占用空间。

### 1.4.2预取策略

在前文中使用一个大型的8GiB缓存来衡量简单的Next-N-line预取器的效果。在本节中将其缓存大小缩小到64MiB，将其关联性设置为16，并使用CFLRU算法进行管理，然后衡量以下五种预取器设置的性能：

1. NP（无预取）：不预取任何数据。
2. NL（Next-N-line）[41]：在发生需求缺失或预取命中时，预取接下来的N个数据。
3. FD（Feedback-directed）[65]：通过跟踪预取器的准确性、及时性和干扰程度，动态调整预取器的侵入性。
4. BO（Best-offset）[55]：通过跟踪最近请求的历史，学习连续访问之间的增量。它还有一个置信度的概念，可以禁用预取。
5. LP（Leap）[53]：采用基于多数投票的预取方法，并根据动态窗口大小调整适应性。它还根据预取器的准确性逐渐调整侵入性。

经过实验得出了7个观察结果：

**观察1：**预取器对不同工作负载的性能影响不同。最先进的预取器在某些工作负载上对性能产生了负面影响，而在其他工作负载上则有所帮助。特别是，在使用最佳偏移预取器时，Radiosity工作负载出现了延迟的增加。这表明预取器的性能取决于工作负载的特性，并且可能对应用产生不利影响。

**观察2：**CXL闪存的寿命在真实工作负载下至少为3.1年。根据写入闪存的数据量、耐用限制和容量估计，CXL闪存在最坏情况下可持续使用3.1年，在其他工作负载下可达到更长的寿命。因此，CXL闪存的耐久性可以满足内存请求的强度。

**观察3：**CXL闪存比纯DRAM设备具有更好的性能成本比。尽管CXL闪存在延迟方面稍逊于DRAM，但分析显示它在性能成本效益方面具有潜力。相比于最近的DRAM价格，CXL闪存具有更低的成本，并且可以为大部分内存请求提供低延迟的服务。这使得CXL闪存成为一种具有成本效益的内存扩展选项，尤其适用于内存密集型应用程序。

**观察4：**预取器的改进性能归功于高准确性。通过绘制评估的预取器的准确率、覆盖率、延迟和污染度指标，观察到预取器对于那些准确性高的工作负载（如Page Rank和Radiosity）表现得更好。例如，Leap（LP）预取器在Radiosity工作负载下达到85％的准确性，而在BERT下只有27％。这表明预取器的性能与其准确性密切相关，高准确性能够帮助提升性能。

**观察5：**低准确性时，缓存污染是性能下降的主要原因。对于准确性较低的工作负载（如BERT和YCSB），它们的污染率较高，这导致启用预取器时性能下降。例如，对于XZ工作负载，尽管最佳偏移（BO）预取器的准确性较低，但在几乎没有污染时，它与没有预取器时的性能相当。这表明缓存污染会降低CXL-flash的性能，因此预取器应当谨慎使用以避免对设备产生不利影响。

**观察6：**在虚拟内存地址转换到物理内存地址转换的过程中，CXL-flash的预取器的准确性和覆盖率都显著下降。这意味着预取器在物理内存访问下的性能较差，而在虚拟内存地址下的性能更好。例如，在Page Rank工作负载下，最佳偏移预取器在虚拟内存跟踪中的准确性为99％，但在物理跟踪中下降到42％。类似地，在BERT工作负载下，覆盖率从76％降至26％。

**观察7：**提出了一种假设，即如果内核能够为设备提供内存访问模式提示，那么CXL-flash的性能将得到改善。这里假设内核具有关于物理内存访问模式的信息，并且可以在实际访问之前将提示传递给设备。在BERT工作负载中，在前N％的密集访问地址的10％处生成提示时，性能的改善情况。随着对更多地址生成访问提示，性能得到了改善。

# 论文理解

本该论文探讨了使用CXL（Compute Express Link）技术的SSD（固态硬盘）来克服"内存墙"的挑战。"内存墙"是指由于内存速度与处理器速度之间的差异而导致的性能瓶颈。该论文的目标是通过使用CXL技术将闪存与内存连接起来，提供更高的性能和容量。

## 1.1优点分析

1、提高性能：使用CXL技术将闪存与内存连接起来，可以提供更高的性能，缓解"内存墙"问题。

2、扩展容量：通过将闪存作为扩展内存，可以增加系统的总容量，满足对大内存需求的应用程序。

3、成本效益：相比传统的DRAM内存，使用CXL-Enabled SSDs可以提供更高的容量，并且更具成本效益。

4、这篇论文有开源的代码，并且有极其丰富的实验数据，具有较强的说服力。

## 1.2缺点分析

不足之处主要集中在以下2点：

1. 延迟：尽管CXL-Enabled SSDs可以提供更高的容量和性能，但与传统的DRAM内存相比，它们可能具有更高的访问延迟[1]。
2. 兼容性：CXL技术是相对较新的技术，可能需要相应的硬件和软件支持，以确保与现有系统的兼容性[1]。

# 课程感想

通过选修操作系统设计课程，我深入学习了操作系统领域的前沿科研知识，极大地提升了我的专业实力。在课堂上，我积极参与同学和老师们对操作系统领域顶级会议论文的分享，这锻炼了我的独立思考能力和动手实践能力，并在许多其他方面也取得了显著进步。更重要的是，我学会了高效的学习方法，这对于未来的学习和成长至关重要，我从中获益匪浅。

在面对社会挑战的过程中，我深知不断学习和实践的重要性。无论遇到多么困难的问题，我都愿意积极迎接挑战，努力将困难转化为乐趣，寻找并探索其中的有趣和宝贵之处。

这门课程让我深刻认识到思路即出路的道理。在面对我不理解或不懂的问题时，我养成了及时请教他人或上网查询资料的习惯。通过认真钻研、动脑思考和实践操作，我相信没有什么知识是我无法掌握的，我从中获得了丰富的收获和成长。

此外，在阅读论文的过程中，我不仅扩充了专业知识，还积累了宝贵的经验和成熟度。我积极查阅大量相关资料，并向老师和同学请教，这为我在专业知识和实践能力方面的提升做出了巨大贡献。在此，我要衷心感谢给予我帮助的所有同学和指导老师！课程的成功离不开老师的耐心指导和同学们的热心帮助，我再次由衷地感谢他们！

# 同学评议

## 报告一：Amazon DynamoDB

Amazon DynamoDB: A Scalable, Predictably Performant, and Fully Managed NoSQL Database Service

Amazon DynamoDB是一种可扩展、可预测性能和完全托管的NoSQL数据库服务。它是由亚马逊提供的数据库即服务（DBaaS）。

DynamoDB是一个完全托管的云服务，用户可以直接创建表格并进行读写操作。它采用多租户架构，可以在同一物理机上存储来自不同客户的数据。DynamoDB对表格的规模没有预定义的限制，可以存储任意量级的数据。

它提供可预测的性能，对于1KB的数据项，AWS区域内的应用程序可以在低个位数毫秒的延迟范围内完成请求。DynamoDB具有高可用性，可以在多个数据中心（AZs）之间复制数据，并在选定的区域之间进行地理复制。DynamoDB支持灵活的用例，表格没有固定的模式。

## 报告二：GL-Cache

FAST'23 GL-Cache：Group-level learning for efficient and high-performance caching

这是一篇关于高效和高性能缓存的研究论文。该论文提出了一种名为GL-Cache的缓存系统，利用组级别学习来提高缓存的效率和性能。

GL-Cache的主要特点和优势包括：

组级别学习：GL-Cache利用组级别学习来优化缓存的决策和策略。通过对缓存中的数据进行分组，并针对每个组应用机器学习算法，GL-Cache能够更好地理解和预测数据的访问模式，从而提高缓存的命中率和性能。

高效的缓存替换策略：GL-Cache采用了一种基于组级别学习的缓存替换策略。通过学习每个组的访问模式和重要性，GL-Cache能够智能地选择哪些数据应该被保留在缓存中，以及哪些数据应该被替换出去。这种智能的替换策略可以显著提高缓存的效率和性能。

高性能的缓存管理：GL-Cache通过并行处理和优化的数据结构，实现了高性能的缓存管理。它能够有效地处理大规模的数据集和高并发的访问请求，同时保持低延迟和高吞吐量。

可扩展性和灵活性：GL-Cache具有良好的可扩展性和灵活性。它可以适应不同规模和类型的应用程序，并能够根据实际需求进行配置和调整。

总之，通过组级别学习和智能的缓存替换策略，GL-Cache能够提供高效和高性能的缓存服务。

## 报告三：CXL分布式共享内存的部分故障弹性内存管理系统

SOSP'23 Partial Failure Resilient Memory Management System for (CXL-based) Distributed Shared Memory

这篇文章的主要内容是关于一种针对分布式共享内存系统的部分故障容错内存管理系统。文章介绍了一种基于引用计数的自动分布式内存管理系统，名为CXL-SHM。CXL-SHM使用了一种特殊的基于时代的非阻塞算法来实现引用计数的维护。因此，即使一些参与的客户端意外失败并且没有释放其所拥有的内存引用，CXL-SHM也能够避免阻塞同步、内存泄漏、双重释放和野指针等问题。研究人员通过在真实的CXL硬件上进行微基准测试和端到端应用程序的评估，展示了CXL-SHM的效率以及使用CXL-SHM构建高效分布式应用程序的简单性和灵活性。

## 报告四：FUSEE

FUSEE: A Fully Memory-Disaggregated Key-Value Store

这是一篇关于完全内存分离的键值存储系统的文章。该系统旨在解决传统键值存储系统中内存容量限制的问题。文章介绍了FUSEE的设计和实现，以及其在性能和可扩展性方面的优势。

FUSEE系统架构：文章详细描述了FUSEE的系统架构，包括客户端、存储节点和内存节点之间的通信和协作方式。FUSEE使用了内存分离的设计，将存储和计算分开，从而允许存储节点和内存节点在不同的物理服务器上运行。

内存分离技术：文章介绍了FUSEE使用的内存分离技术，包括内存远程访问和数据迁移。内存远程访问允许存储节点通过网络直接访问内存节点上的数据，而数据迁移则可以在不同的内存节点之间动态地迁移数据，以实现负载均衡和容错性。

## 报告五：使用Trio架构实现高性能和安全的用户空间NVM文件系统

Enabling High-Performanceand Secure Userspace NVM File Systems with the Trio Architecture

这篇文章介绍了一种名为Trio架构的技术，它可以实现高性能和安全的用户空间非易失性内存（NVM）文件系统。该架构旨在解决传统文件系统在利用NVM的性能优势和保护数据安全方面的挑战。以下是该文章的主要内容：

Trio架构概述：文章首先介绍了Trio架构的基本原理和设计目标。Trio架构通过将文件系统的关键功能分为三个独立的组件来实现高性能和安全性。这三个组件分别是元数据服务（Metadata Service）、数据服务（Data Service）和安全服务（Security Service）。

元数据服务：元数据服务负责管理文件系统的元数据，包括文件和目录的结构、权限和属性等信息。Trio架构通过将元数据服务放在用户空间中，可以避免传统文件系统中内核态和用户态之间的频繁切换，从而提高性能。

数据服务：数据服务负责管理文件系统中的实际数据。Trio架构通过将数据服务与元数据服务分离，可以实现更好的并行处理和高性能。此外，数据服务还利用NVM的特性，通过直接访问NVM来提高数据访问速度。

安全服务：安全服务负责保护文件系统中的数据安全。Trio架构通过使用硬件加密和访问控制策略来保护数据的机密性和完整性。此外，安全服务还提供了对文件系统的审计和监控功能，以便及时发现和应对安全威胁。

## 报告六：分布式训练中的检查点恢复

Gemini: Fast Failure Recovery in Distributed Training with In-Memory Checkpoints

这是一篇关于分布式训练中快速故障恢复的论文。该论文提出了一种名Gemini的系统，它使用内存中的检查点来实现快速的故障恢复。

Gemini的主要目标是解决分布式训练中的故障恢复问题。在传统的分布式训练中，当一个节点发生故障时，整个训练过程都需要重新开始。这会导致时间和计算资源的浪费。而Gemini通过使用内存中的检查点，可以快速地从故障节点的上一个检查点恢复训练过程，而无需重新开始。

Gemini的关键思想是将训练过程中的状态保存在内存中的检查点中。当一个节点发生故障时，Gemini可以快速地将其他节点的状态恢复到故障节点上一个检查点的状态。这样，训练过程可以在故障节点的上一个检查点处继续进行，而无需重新开始。

Gemini还引入了一种称为"快速恢复"的机制，可以进一步加快故障恢复的速度。快速恢复利用了内存中的检查点，可以在故障节点上快速地恢复训练过程，而无需从磁盘中读取数据。

Gemini的实验结果表明，与传统的故障恢复方法相比，Gemini可以显著减少故障恢复的时间和计算资源的消耗。这使得Gemini成为分布式训练中快速故障恢复的有效解决方案。

## 报告七：RubbleDB

RubbleDB: CPU-Efficient Replication with NVMe-oF

RubbleDB的主要目标是利用NVMe-oF技术为数据库提供高效的复制功能。NVMe-oF技术可以实现远程存储设备与CPU之间的高速通信，从而实现低延迟和高吞吐量的数据访问。然而，传统的复制技术可能会对CPU造成较大的负载，限制了NVMe-oF的性能优势。

RubbleDB通过引入一种CPU高效的复制机制来解决这个问题。它利用NVMe-oF的独特特性，将复制任务分担到NVMe-oF设备本身，从而减轻了CPU的负担。

论文详细介绍了RubbleDB的设计和实现，包括架构、数据结构和算法等。同时，论文还提供了实验结果，证明了RubbleDB在减少CPU负载的同时保持高复制性能的有效性。

总体而言，RubbleDB为利用NVMe-oF实现CPU高效复制提供了一种有前景的方法，可以显著提高分布式数据库的性能。

## 报告八：eZNS: 新型存储设备的弹性接口

eZNS: An Elastic Zoned Namespace for Commodity ZNS SSDs

这篇文章介绍了一种名为eZNS的新型弹性分区命名空间，用于通用ZNS（Zone Namespace）SSD。该命名空间的设计旨在提高SSD的性能和可扩展性

主要内容包括：

弹性分区命名空间的设计：eZNS通过将SSD分为多个逻辑分区，每个分区具有不同的性能和容量特性，从而实现了更高的灵活性和可扩展性。

分区映射和管理：文章介绍了eZNS如何将逻辑分区映射到物理区域，并管理分区之间的数据迁移和负载均衡。

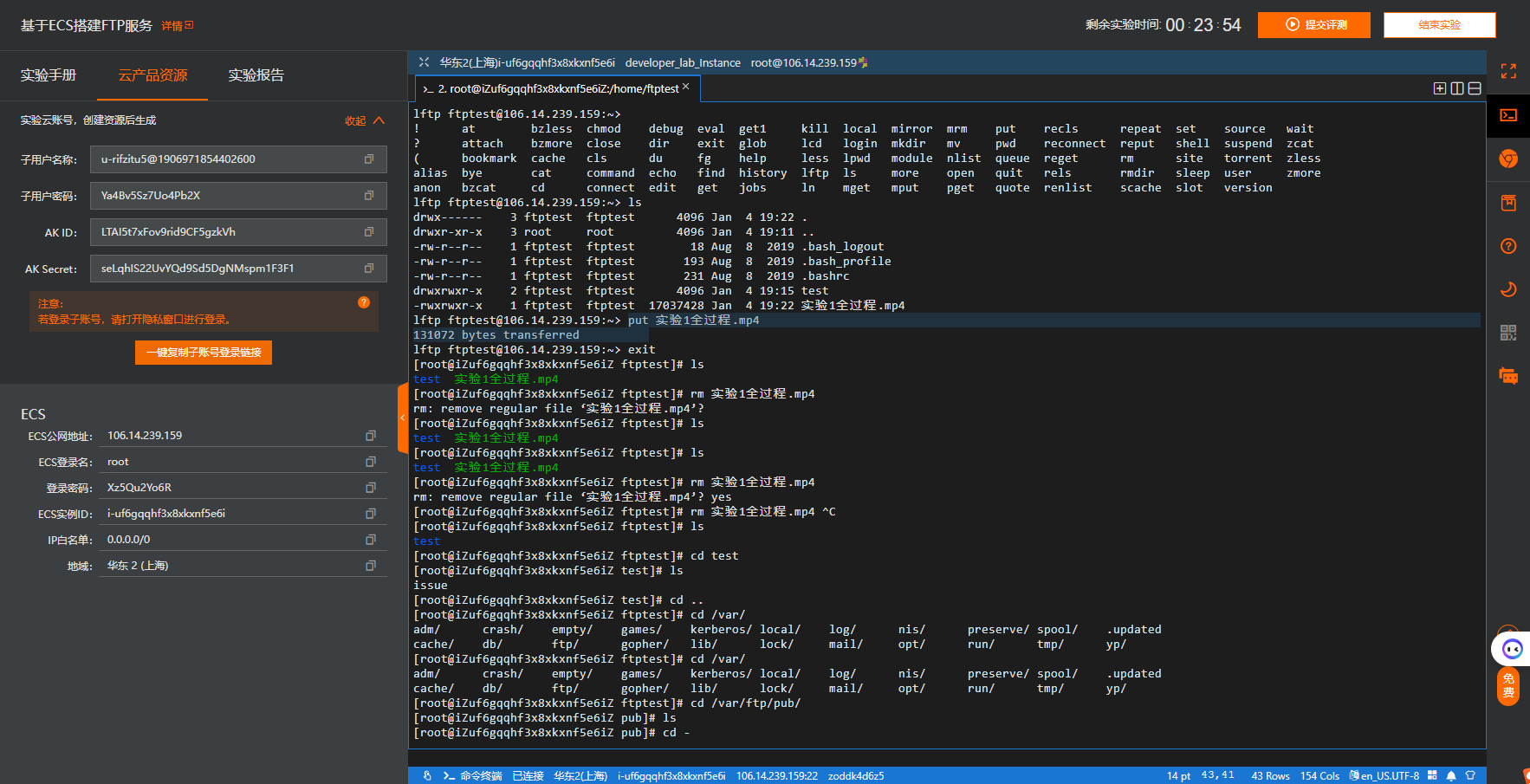
数据访问优化：eZNS通过优化数据访问模式和预取策略，提高了数据访问的效率和性能。

这项研究对于提高通用ZNS SSD的性能和可扩展性具有重要意义，为未来的存储系统设计提供了有价值的参考。

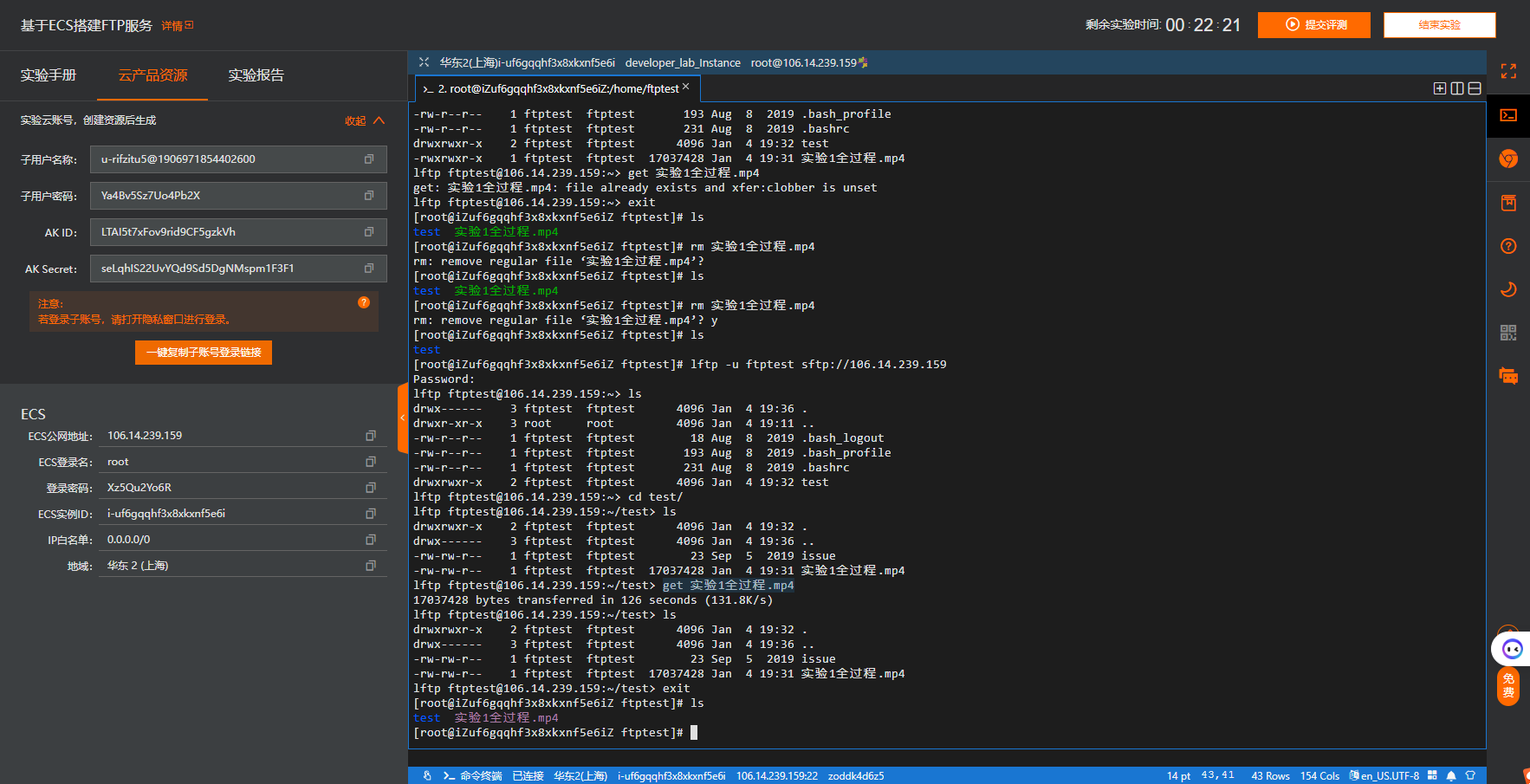
实践报告（一页）：

实验一

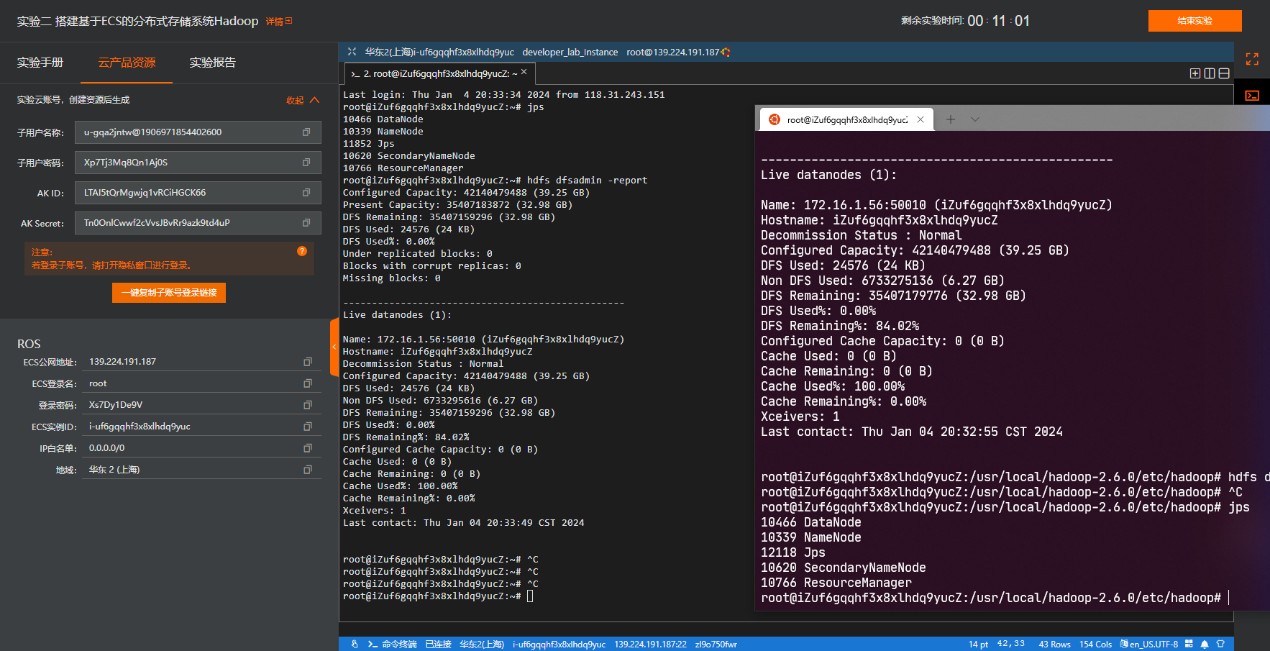
Put 视频 “实验1全过程.mp4”



Get视频



Jps + Hadoop 集群状态



# 参考文献

[1] 2019. Gloo. https://github.com/facebookincubator/gloo.

[2] 2019. MLPerf. https://www.mlperf.org/.

[3] 2019. NCCL. https://developer.nvidia.com/nccl.

[4] 2019. NVLink. https://www.nvidia.com/en-us/data-center/nvlink/.

[5] 2019. PyTorch. https://github.com/pytorch/pytorch.

[6] 2019. PyTorch DDP. https://pytorch.org/docs/stable/\_modules/torch/ nn/parallel/distributed.html.

[7] 2019. VGG-16 target accuracy using Caffe model. https://gist.github. com/ksimonyan/211839e770f7b538e2d8#gistcomment- 1403727.

[8] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. Ten- sorFlow: A System for Large-Scale Machine Learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16).GA,265–283. https://www.tensorflow.org/

[9] Baidu Inc. 2017. Bringing HPC Techniques to Deep Learning. http: //research.baidu.com/bringing- hpc- techniques- deep- learning/

[10] Léon Bottou and Olivier Bousquet. 2008. The Tradeoffs of Large Scale Learning. In Advances in Neural Information Processing Systems. 161– 168.

[11] David L Chen and William B Dolan. 2011. Collecting Highly Parallel Data for Paraphrase Evaluation. In Proceedings of the 49th Annual Meet- ing of the Association for Computational Linguistics: Human Language Technologies-Volume 1. Association for Computational Linguistics, 190– 200.

[12] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. 2016. Revisiting Distributed Synchronous SGD. arXiv preprint arXiv:1604.00981 (2016).

[13] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. MXNet: A Flexible and Efficient Machine Learning Library for Het- erogeneous Distributed Systems. CoRR abs/1512.01274 (2015). http: //arxiv.org/abs/1512.01274

[14] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training Deep Nets with Sublinear Memory Cost. arXiv preprint arXiv:1604.06174 (2016).

[15] Xie Chen, Adam Eversole, Gang Li, Dong Yu, and Frank Seide. 2012. Pipelined Back-Propagation for Context-dependent Deep Neural Net- works. In Interspeech.

[16] Trishul M Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project Adam: Building an Efficient and Scal- able Deep Learning Training System.. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’14), Vol. 14. 571–582.

[17] Cody Coleman, Daniel Kang, Deepak Narayanan, Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. 2019. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. ACM SIGOPS Operating Systems Review 53, 1 (2019), 14–25.

[18] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. 2017. DAWNBench: An End-to-End Deep Learning Bench- mark and Competition. NIPS ML Systems Workshop (2017).

[19] Henggang Cui, James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Abhimanu Kumar, Jinliang Wei, Wei Dai, Gregory R Ganger, Phillip B Gibbons, et al. 2014. Exploiting Bounded Staleness to Speed Up Big Data Analytics. In USENIX Annual Technical Conference. 37–48.

[20] Henggang Cui, Hao Zhang, Gregory R Ganger, Phillip B Gibbons, and Eric P Xing. 2016. GeePS: Scalable Deep Learning on Distributed GPUs with a GPU-Specialized Parameter Server. In Proceedings of the Eleventh European Conference on Computer Systems. ACM, 4.

[21] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. 2012. Large Scale Distributed Deep Networks. In Advances in Neural Information Processing Systems. 1223–1231.

[22] Michael Denkowski and Alon Lavie. 2014. Meteor Universal: Lan- guage Specific Translation Evaluation for Any Target Language. In Proceedings of the Ninth Workshop on Statistical Machine Translation. 376–380.

[23] DGX-1 [n. d.]. NVIDIA DGX-1. data- center/dgx- 1/.

https://www.nvidia.com/en-us/

[24] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. arXiv preprint arXiv:1706.02677 (2017).

[25] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. 2018. PipeDream: Fast and Efficient Pipeline Parallel DNN Training. arXiv preprint arXiv:1806.03377 (2018).

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. CoRR abs/1512.03385 (2015). http://arxiv.org/abs/1512.03385

[27] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. 2013. More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server. In Advances in Neural Information Processing Systems. 1223–1231.

[28] Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Ji- quan Ngiam, Quoc V Le, and Zhifeng Chen. 2018. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. arXiv preprint arXiv:1811.06965 (2018).

[29] Zhouyuan Huo, Bin Gu, Qian Yang, and Heng Huang. 2018. Decoupled Parallel Backpropagation with Convergence Guarantee. ICML-18, arXiv preprint arXiv:1804.10574 (2018).

[30] Animesh Jain, Amar Phanishayee, Jason Mars, Lingjia Tang, and Gen- nady Pekhimenko. 2018. Gist: Efficient Data Encoding for Deep Neural Network Training. In ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA ’18).

[31] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. 2018. Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes. arXiv preprint arXiv:1807.11205 (2018).

[32] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. arXiv preprint arXiv:1408.5093 (2014).

[33] Zhihao Jia, Sina Lin, Charles R Qi, and Alex Aiken. 2018. Exploring Hidden Dimensions in Parallelizing Convolutional Neural Networks. In Proceedings of the 28th International Conference on Machine Learning (ICML ’18).

[34] Zhihao Jia, Matei Zaharia, and Alex Aiken. 2019. Beyond Data and Model Parallelism for Deep Neural Networks. In Proceedings of the 2nd SysML Conference, SysML ’19. Palo Alto, CA, USA.

[35] Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. arXiv preprint arXiv:1412.6980 (2014).

[36] Alex Krizhevsky. 2014. One Weird Trick for Parallelizing Convolutional Neural Networks. arXiv preprint arXiv:1404.5997 (2014).

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Ima- geNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems. 1097–1105.

[38] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’14), Vol. 1. 3.

[39] Dominic Masters and Carlo Luschi. 2018. Revisiting Small Batch Training for Deep Neural Networks. arXiv preprint arXiv:1804.07612 (2018).

[40] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Reg- ularizing and Optimizing LSTM Language Models. arXiv preprint arXiv:1708.02182 (2017).

[41] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocky`, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. In Eleventh Annual Conference of the International Speech Com- munication Association.

[42] Azalia Mirhoseini, Hieu Pham, Quoc Le, Mohammad Norouzi, Samy Bengio, Benoit Steiner, Yuefeng Zhou, Naveen Kumar, Rasmus Larsen, and Jeff Dean. 2017. Device Placement Optimization with Reinforce- ment Learning. https://arxiv.org/abs/1706.04972

[43] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gra- dient Descent. In Advances in Neural Information Processing Systems. 693–701.

[44] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. ImageNet Large Scale Visual Recogni- tion Challenge. International Journal of Computer Vision 115, 3 (2015), 211–252.

[45] Frank Seide and Amit Agarwal. 2016. CNTK: Microsoft’s Open-Source Deep-Learning Toolkit. In Proceedings of the 22Nd ACM SIGKDD Inter- national Conference on Knowledge Discovery and Data Mining (KDD ’16). NewYork,NY,USA,2135–2135. https://github.com/Microsoft/CNTK

[46] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1- Bit Stochastic Gradient Descent and its Application to Data-Parallel Distributed Training of Speech DNNs. In Fifteenth Annual Conference of the International Speech Communication Association.

[47] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. On Parallelizability of Stochastic Gradient Descent for Speech DNNs. In International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE SPS.

[48] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolu- tional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556 (2014).

[49] Evan R Sparks, Ameet Talwalkar, Daniel Haas, Michael J Franklin, Michael I Jordan, and Tim Kraska. 2015. Automating Model Search for Large Scale Machine Learning. In Proceedings of the Sixth ACM Symposium on Cloud Computing. ACM, 368–380.

[50] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Opti- mization of Collective Communication Operations in MPICH. The International Journal of High Performance Computing Applications 19, 1 (2005), 49–66.

[51] Uber Technologies Inc. 2017. Meet Horovod: Uber’s Open Source Distributed Deep Learning Framework for TensorFlow. https://eng. uber.com/horovod/

[52] Leslie G. Valiant. 1990. A Bridging Model for Parallel Computation. Commun. ACM 33, 8 (Aug. 1990).

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. At- tention is All You Need. In Advances in Neural Information Processing Systems. 5998–6008.

[54] Subhashini Venugopalan, Marcus Rohrbach, Jeffrey Donahue, Ray- mond Mooney, Trevor Darrell, and Kate Saenko. 2015. Sequence to sequence-video to text. In Proceedings of the IEEE International Confer- ence on Computer Vision. 4534–4542.

[55] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arXiv preprint arXiv:1609.08144 (2016).

[56] YangYou,IgorGitman,andBorisGinsburg.2017.LargeBatchTraining of Convolutional Networks. arXiv preprint arXiv:1708.03888 (2017).

[57] Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P. Xing. 2017. Poseidon: An Efficient Communication Architecture for Distributed Deep Learning on GPU Clusters. In 2017 USENIX Annual Technical Conference (USENIX ATC 17). USENIX Association, Santa Clara, CA, 181–193.