学号 2017202110020

**海量存储技术课程论文**

**浅析“基于历史的大规模数据中心的备用周期和存储空间获取”**

院（系）名 称：计算机学院

专 业 名 称 ：计算机体系结构

学 生 姓 名 ：李晓彤

指 导 教 师 ：何水兵 教授

摘 要

提高数据中心的利用率并降低成本的一个有效方法是将其延迟关键型服务和批量工作负载共同定位。在这篇文章里，作者为我们描述了一个为共置目的而搜集备用计算周期和存储空间的系统。主要的挑战是尽量减少对服务的性能影响，同时考虑其使用和管理模式。为了克服这个挑战，作者提出了使服务优先于资源并利用关于它们的历史信息的技术。基于这些信息，作者在服务器上安排相关的批处理任务，这些任务表现出相似的模式，并可能有足够的可用资源用于任务的持续时间，并将数据副本放置在展现出不同模式的服务器上。本文主要描述了服务如何在十个大型生产数据中心上被利用和管理。通过使用真实的实验和模拟验证了本文的技术消除了许多情况下的数据丢失和不可用性，同时保护了共存服务并提高了批处理作业的执行时间。

目 录

[1 绪论 4](#_Toc500688159)

[1.1 研究背景 4](#_Toc500688160)

[1.2 国内外相关工作 5](#_Toc500688161)

[1.3 文章主要工作 6](#_Toc500688162)

[2 表征行为模式 8](#_Toc500688163)

[2.1 数据来源和术语 8](#_Toc500688164)

[2.2 资源利用率 8](#_Toc500688165)

[2.3 磁盘重映像 9](#_Toc500688166)

[3 智能协同定位技术 12](#_Toc500688167)

[3.1 智能任务调度 12](#_Toc500688168)

[3.2 智能数据放置 13](#_Toc500688169)

[4 系统实现 16](#_Toc500688170)

[4.1 背景知识 16](#_Toc500688171)

[4.2 实现方针 17](#_Toc500688172)

[4.3 YARN-H 和 Tez-H 17](#_Toc500688173)

[4.4 HDFS-H 18](#_Toc500688174)

[5 仿真和实验结果分析 20](#_Toc500688175)

[5.1 实验结果分析 20](#_Toc500688176)

[5.2 仿真结果分析 21](#_Toc500688177)

[6 总结 25](#_Toc500688178)

[7 参考文献 26](#_Toc500688179)

# 绪论

## 研究背景

采购服务器在大规模数据中心的总体拥有成本（TCO）中占据主导地位，比如谷歌和微软运营的那些。但服务器的平均利用率通常很低，尤其是在面向用户的交互式服务的集群中。其原因包括：这些服务通常是延迟严重的（即，要求低尾响应时间），并且可能在用户负载方面表现出高峰值; 此外还必须保留容量来应对意外的负载峰值和故障。

从服务器中提取更多价值的一种有效方法是将有用的批量工作负载（例如数据分析，机器学习）和他们在相同的服务器上执行其他功能时所需的数据放在一起，包括那些运行延迟关键服务的任务。但是，为了使这些服务的协同定位成为可能，即使发生意外事件，我们也必须保护它们免受批处理工作负载或其存储访问所产生的非平凡的性能干扰。如果协同定位开始降低响应时间，则调度程序必须限制甚至杀死（并在别处重新启动）肇事者批处理工作负载。无论哪种情况，批处理工作负载的性能都会受到影响。尽管如此，协同定位最终降低了TCO，因为批量工作负载不是延迟关键型的，与服务共享相同的基础架构，而不需要自己的。

近年来的调度研究已经考虑了如何仔细地选择与每个服务共同定位哪个批量工作负载以最小化潜在的干扰（最常见的是最后一级高速缓存干扰）。但是，这些工作要么承担简单连续的批量应用，要么忽略实际服务的资源利用动态性。调度包含许多分布式任务（例如，数据分析作业）的大量的工作负载是具有挑战性的，因为调度决策必须将这些任务串联起来以实现最佳性能。资源利用动态使情况变得复杂。例如，长时间运行的工作负载可能会在共同定位的服务的负载增加时阻塞或终止其某些任务

此外，没有先前的研究详细探讨了服务与批量工作负载数据的共同定位。真正的服务通常会留下大量的闲置存储空间（和带宽），可用于存储批量工作负载所需的数据。但是，同位置存储会带来更多挑战，因为服​​务的管理和利用可能会影响数据的持久性和可用性。例如，服务工程师和管理系统本身可能重新映像（重新格式化）磁盘，删除所有的数据。重新映像通常来自持久状态管理，服务部署，健壮性测试或磁盘故障。同位置和重新映像可能会导致数据块的所有副本在重新生成之前被销毁。

## 国内外相关工作

数据中心表征：数据中心运营商之前的工作中已经研究了选定的生产集群，而不是整个数据中心。相反，本文将十个数据中心中的所有主要租户描述为服务开发和测试，以及等待使用或准备退役的数据中心，其中包括那些用于生产延迟重要和非关键服务的主要租户。

在地点不同的情况下收集资源：之前的工作已经提出在没有同时定位的延迟重要的服务的情况下为批量工作量收获资源。本文的工作主要集中在现代数据中心更具挑战性的共址场景。

延迟关键和批处理任务的共同定位：最近的研究针对协同定位的两个方面：

（1）性能隔离——确保在同一台服务器上，批处理任务在共同定位后不干扰服务;

（2）调度——该选择哪些任务与每个服务共同定位，以最小化干扰或提高打包质量。 Borg解决了Google数据中心的两个方面，使用基于Linux cgroup的容器，对延迟关键型任务的特殊处理以及从容器收集资源。

本文的工作与这些努力大不相同。由于隔离和干扰感知调度已经得到了充分的研究，因而将这些技术的实施留给未来的工作。相反的是保留不能提供给批处理任务的计算资源，尖峰主租户可以立即使用这个储备，直到我们的软件能够做出反应（最多几秒钟内）来补充储备。将本文的系统与更细粒度的隔离技术相结合将使储量更小。

而且，与谷歌的服务不同，文中的主租户“拥有”他们的服务器，而不是声明他们潜在的资源需求。这意味着我们必须小心收集资源，以防止干扰劳动力环境服务和降低批量工作绩效。因此，本文超越了以前的作品通过理解并且探索利用主要租户的资源使用动态来减少对批处理任务的需求。关于资源使用动态，一篇相关论文[5]，从历史利用率数据中获得资源可用性的服务水平目标（SLO）。作者利用类似的数据，但对于他们的论文没有涉及的动态任务调度。

同样重要的是，本文是第一个详细探讨从主租户收获的存储空间数据密集型批处理作业。这种情况涉及了解主要租户的管理方式以及资源使用情况。对于计算和存储收割，作者利用主要和次要租户的历史行为，这往往比用户注释或者估计更准确（例如[35]）。从延迟关键型工作负载中获取资源的任何系统都可以从中受益利用相同的行为。

数据处理框架和协同定位：研究人员已经提出了对Hadoop栈的改进，但都是在没有共同定位的情况下的，例如文献[3，8，13，14，15，18，39]。其他人在使用虚拟机的托管方案中考虑了Hadoop（版本1），但在专用服务器上运行了HDFS [7,30,41]。 Lin等人[22]将数据存储在专用和自愿的计算机（空闲台式机）上，但没有主租户。我们没有意识到Mesos [16]在共址场景中的研究。Bistro [12]依靠为服务提供静态资源预定，并在剩余资源上安排批量作业。与这些工作相比，本文提出了Hadoop栈的动态调度和数据放置技术，并探讨了位于同一地点的主租户和二级租户的性能，数据可用性和数据持久性。

## 文章主要工作

在本文中，作者提出了收集分布式批处理工作负载中数据中心备用计算周期和存储空间的技术。本文将每个服务器的原始工作负载称为“主要租户”，并将服务器上的任何资源收集工作负载（即批量计算任务或其存储访问）称为“辅助租户”。我们优先考虑每个服务器的资源给主租户;当主租户需要资源时，二级租户可能会被取消（在任务的情况下）或被拒绝（在存储访问的情况下）。为了减少任务处理的次数并提高数据的可用性和持久性，作者提出了依靠历史资源利用和磁盘重映像模式的任务调度和数据放置技术。我们在逻辑上将在这些维度中表现出类似模式的主租户分组。使用这些利用组，我们的调度技术在服务器上调度相关的批处理任务，这些任务具有相似的模式和足够的资源用于任务的预期持续时间，从而避免了由于缺乏资源而造成的失败。使用利用和重映射组，我们的数据放置技术将数据副本放置在具有不同模式的服务器中，从而提高了存储资源的收获性质的持久性和可用性。为了创建团队，我们在十个生产数据中心中对主要租户的使用和重新映像模式进行了描述，其中包括一个流行的搜索引擎及其支持服务。每个数据中心最多可容纳数以万计的服务器。我们的表征显示，数据中心工作负载是周期性的常识是不准确的，因为大多数服务器通常不执行交互式服务。本文的目标是在所有的服务器上进行收集。

作者将技术实现到Apache Hadoop堆栈的YARN调度器，Tez作业管理器和HDFS文件系统中[11,29,36]。（主租户使用他们自己的调度和文件系统。）股票YARN和HDFS假设没有外部工作负载，所以我们也让这些系统知道主租户和他们的资源使用情况。本文使用生产数据中心中的102台服务器来评估我们的系统，并利用和缩放从中缩小的行为。作者也使用模拟来研究本文的系统的更长时间和更大的群集。

结果表明，本文提出的系统具有以下几个特点：

（1）可以将平均批处理作业执行时间提高达90％;

（2）在三次重复数据块的情况下，可以将数据丢失减少两个数量级以上，消除四路复制下的数据丢失，并且消除大多数利用率级别的数据不可用性。

（3）部署了本文的大规模生产文件系统

本文的主要贡献是：

•描述了十个生产数据中心如何使用和管理服务器的动态。

•基于主租户的历史行为以及如何管理，提出了改善任务调度和数据布局的技术。

•使用本文的技术，扩展了Hadoop堆栈以收集数据中心的备用周期和存储空间。

•使用真实的实验和模拟评估我们的系统，并显示批量作业性能，数据持久性和数据可用性方面的大幅改进。

•讨论大规模生产部署的经验。

从中得出的结论是，资源获取主要从工作负载的资源使用和管理模式的详细计算中获益。这种计算能够提高利用率并降低TCO。

# 表征行为模式

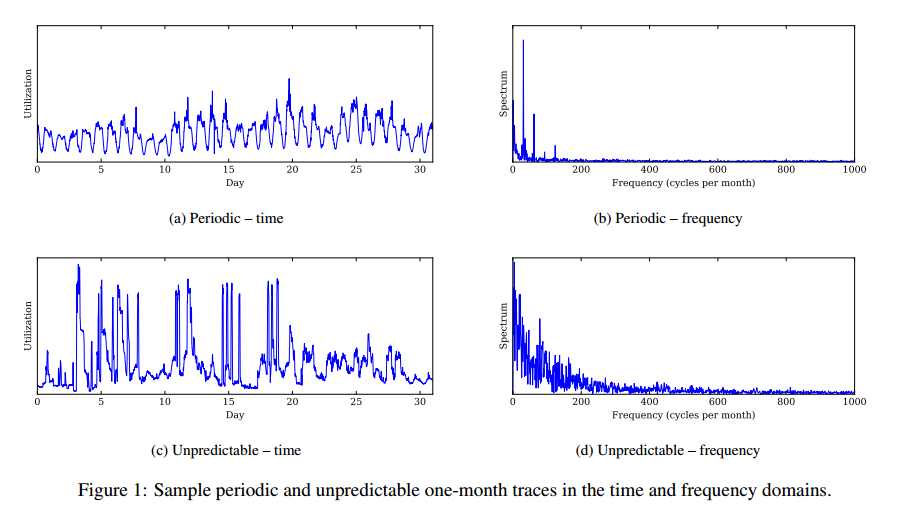
## 数据来源和术语

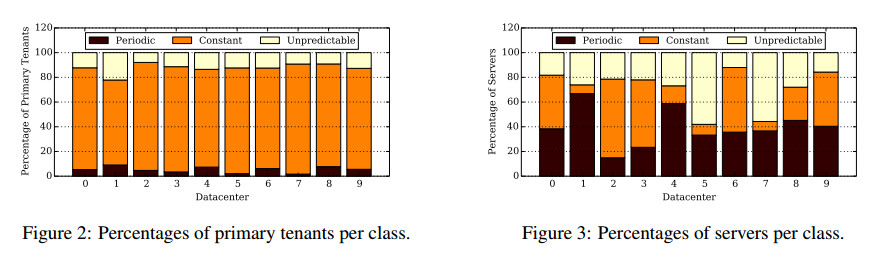
现在在十个生产数据中心中对主要租户进行了描述，并对协同定位技术和结果进行描述。作者利用AutoPilot [17]（数据中心中使用的主要租户管理和部署系统）收集数据，在AutoPilot下，每个服务器都是环境（逻辑上相关的服务器集合，例如搜索引擎的索引服务器）的一部分，并执行机器功能（特定功能，例如结果排名）。环境可以用于生产、开发或测试。在本文的术语中，每个主租户相当于一个<环境，机器功能>对。主租户在物理硬件上运行，没有虚拟化，每个数据中心有几百到几千个主租户。虽然本文的研究集中在AutoPilot管理的数据中心上，但本文的表征和技术应该也很容易适用于其他管理系统。事实上，类似的遥测技术通常被收集在其他生产数据中心，例如，谷歌的GWP [28]和Facebook的Scuba [2]。

## 资源利用率

AutoPilot为每个服务器记录所有硬件资源的主租户利用率，但为了简单起见，将重点放在本文中的CPU上，记录每两分钟的CPU利用率。由于主租户的所有服务器的负载并不总是平衡的，因此本文计算每个时间段的平均利用率，并使用此“平均”服务器的利用率来代表主租户。然后，使用信号处理来识别租户利用的趋势。具体而言，本文对每个主租户的数据分别使用快速傅立叶变换（FFT）。 FFT将利用率时间序列转换到频域，使得识别序列中的任何周期性（及其强度）变得容易。本文确定了三类主租户：定期的，不可预测的和（大致）不变的。图1显示了时间和频率域中周期性和不可预测的主租户的CPU利用率趋势。图1b显示频率为31的强信号，因为那个月有31天（负载峰值和谷值）。相比之下，图1d显示随着频率增加，信号强度呈下降趋势，因为大部分信号源自很少发生的事件（即，呈现较低的频率）。正如人们所期望的那样，面向用户的主要租户经常表现出定期的利用率（例如，白天高和夜晚低），而非面向用户（例如，网络爬行，批量数据分析）或非生产（例如，开发，测试）主租户往往没有。例如，Web爬行或数据清理租户可能会显示（大致）不变的使用率，而测试租户通常会显示不可预知的使用行为。更有意思的是，图2显示，面向用户（定期）的主租户实际上是一小部分。绝大多数主要租户表现出大致恒定的CPU利用率。尽管如此，图3显示了每个数据中心的服务器周期性主租户的比例很高（平均40％）。

不过，非定期主租户占租户和服务器的一半以上。最重要的是，绝大多数服务器（75％）运行主要租户（周期性和常量），历史利用率数据是未来行为的良好预测指标（利用率定期或一直重复）。 因此，利用这些数据应该可以提高我们的任务调度和数据放置的质量。





## 磁盘重映像

对于一些主租户而言，磁盘映像相对来说比较频繁，它本身可能威胁共同定位下的数据的持久性。更糟的是，磁盘重映像通常是相关的，即许多服务器可能同时被重新映像（例如，当服务器从一个主租户转到另一个主租户时）。因此，数据持久性对于重映像和相关性至关重要。 AutoPilot收集每个服务器的磁盘重新映像（重新格式化）数据。这些数据包括多种类型的重映像：

（1）由开发人员或者服务运营商，手动启动的打算重新部署其环境（主租户），或重新启动;

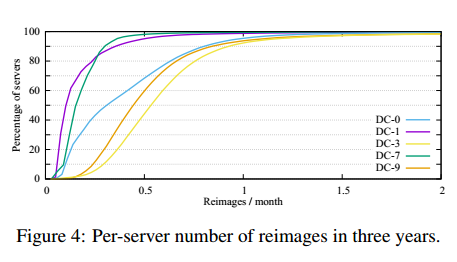
（2）由AutoPilot发起的测试生产服务弹性的测试;

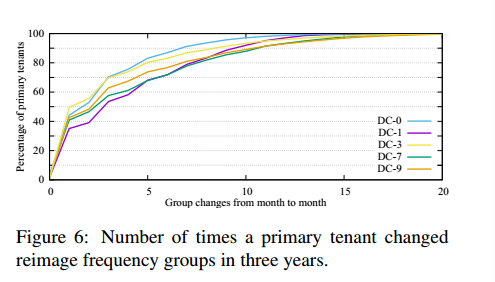
（3）在磁盘进行维护（例如，测试失败）时由AutoPilot发起的那些操作。

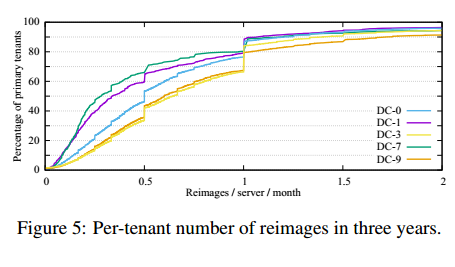
我们现在使用来自AutoPilot的三年数据研究重新成像模式。以我们观察到的重新映像频率为例，图4显示了我们样本中五个代表性数据中心在三年内每台服务器每月平均重映像数的累积分布函数（CDF）。图5显示了同一年份和数据中心每个主租户平均每台服务器的reimages数量的CDF。这个数字的不连续性是由于短租主租户。我们从这些数字做出三个观察。首先，也是最重要的一点，每个数据中心的主租户平均重映率频率差异很大（图5中几乎没有显示垂直线）。其次，所有数据中心的每月重映像频率都相当低。例如，至少90％的服务器平均每月重新映像一次或更少次，而至少80％的主要租户平均每台服务器每月重新映像一次或更少次。这表明，主要租户工程师和AutoPilot的重新造价平均而言并不是过于激进，但是服务器（10％）和主要租户（20％）的显着尾部会相对频繁地重新映像。第三，主数据中心的主租户重映像行为是相当一致的，尽管三个数据中心的重映像速率明显低于每台服务器（我们在图4中显示了其中两个数据中心）。

剩下的问题是，每个主要租户是否每个月都有大致相同的频率。在这方面，我们发现有很大的变化，频率有时会有很大的变化。

然而，相比之下，主要租户往往在频谱的同一部份。换句话说，在一个月内经历相对较少（大）数量的重映像的主租户往往在下个月经历相对较少（大）的重映像数量。为了验证这一趋势，作者将数据中心的主要租户分为三个频率组，每个频率组有相同的租户数量：不频繁，中间频繁和频繁。然后，我们追踪这些群体中主要租户的流动情况。图6绘制了主租户从一个月到下一个月改变组次数的CDF。至少80％的主要租户在三年内可能的35次变更中只有8次或更少的次数。这种行为在数据中心之间也是一致的。再次，这些数字表明，历史成像数据应该提供关于未来的有意义的信息。使用这些数据应该改善数据的位置。



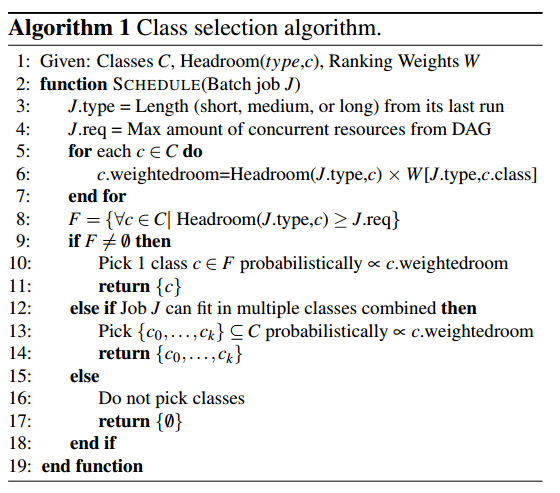




# 智能协同定位技术

## 智能任务调度

我们试图安排批量任务（二级租户）从本地运行交互式服务及其支持工作负载（主要租户）的服务器上获取备用周期。现代集群调度程序实现了高度的工作绩效和/或公平性，因此它们是这种用途的理想选择。但是，他们的设计通常假定专用服务器，即没有在同一台服务器上运行的主租户。因此，我们必须（1）修改它们，以了解主租户和主租户对服务器资源的优先级; （2）为他们提供调度算法，减少共同主租户对资源需求造成的任务处理数量。第一个要求相当容易完成，所以我们在第五部分描述我们的实现。在这里，我们关注第二个需求，即智能任务调度，并使用历史主租户利用率数据来选择最有可能具有所需资源的服务器可用于整个任务的整个执行。由于主租户数量庞大，在任务调度期间独立对待他们是不切实际的。因此，我们的调度技术首先将具有相似利用模式的主要租户聚集到相同的利用类中，然后为任务的任务选择一个类。接下来，我们依次讨论我们的聚类和类选择算法。



聚类算法周期性地（例如，每天一次）从每个主租户的平均服务器获取CPU利用率的最近时间序列，在该系列上运行FFT算法，将租户分组成第三部分中描述的三种模式，恒定的，不可预知的），然后使用K-Means算法将每个模式中的轮廓聚类为类。聚类标记每个类的利用模式，平均利用率和高峰利用率。它还保持类与其主要租户之间的映射。正如我们在算法1中详细说明的，我们的类选择算法依赖于聚类算法定义的类。当我们需要为作业的任务分配资源时，算法根据预期的作业长度（第3行）和长度的预定分类来选择一个或多个类。我们使用权重代表所需的排名（第6行）;更高的分量意味着更高的排名长期工作，我们优先考虑不断的类，然后是阶段性类，最后是不可预测的类。在这种情况下，我们优先考虑常量类，因为具有足够可用资源的常量利用主租户不太可能在执行期间将资源从作业中分离出来。另一个极端的情况是，短期工作不需要长久的资源保证;知道当前的利用率就足够了。因此，对于一个短的工作，我们先排序不可预测的类，然后是周期性的，最后是不变的。对于一个中等工作来说，排名是先定期的，然后是不变的，最后是不可预测的。我们通过将上一次执行的时间与两个预定义的阈值（第3行）进行比较，将作业分为短、中或长。

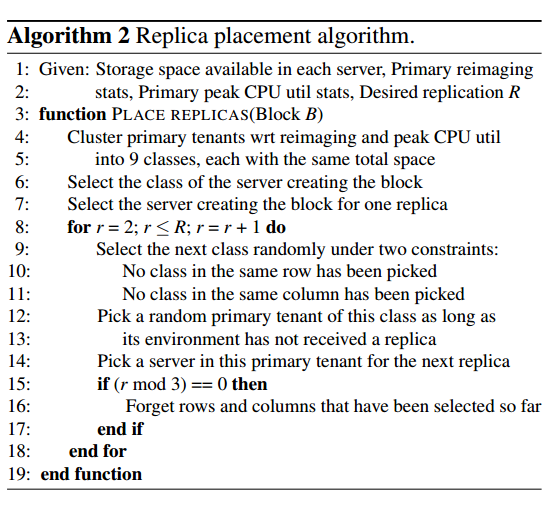
我们基于工作长度的历史分布和每个优选租户类别的当前计算能力来设置阈值（例如，长工作所需的总计算应该与不变的主租户的计算能力成比例）。重要的是，最后的持续时间不需要准确的执行时间估计。我们的目标要容易得多：将工作分为三大类。我们假设以前没有执行过的工作是中等工作。在第一次猜测出现可能的错误之后，我们发现作业始终属于同一类型。我们使用作业有向无环图（DAG）的广度遍历来估计作业将需要的最大并发资源量（第4行），DAG是许多框架中执行流的常见表示[1,29,40]。我们发现这个估计对于我们的工作量是准确的。图7显示了一个示例作业DAG（来自TPC-DS [34]的查询19），为此我们估计了最多469个并发容器。正如我们在下面所定义的，班级（第8行）中的某个作业是否“合适”取决于该班级中服务器当前展示的可用资源数量（或空间的大小）。当多个班级可以主持工作，算法选择一个与其权重的头部空间成正比（第9和10行）。如果需要多个类，则根据需要选择尽可能多的类，概率性地再次选择（第12行和第13行）。如果任何类的组合中没有足够的资源，则不会选择任何类（第16行）。空间取决于作业类型。对于短期工作，我们将其定义为1减去班级中服务器的当前平均CPU利用率。对于中等工作，我们使用1减去最大值（平均CPU利用率，当前CPU利用率）。对于长时间的工作，我们使用1减去最大值（峰值CPU利用率，当前CPU利用率）。

## 智能数据放置

现代分布式文件系统实现了高数据访问性能，可用性和持久性，因此在我们的采集场景中使用它们有强烈的动机。但是，像集群调度程序一样，他们假定没有主租户的专用服务器在同一台服务器上运行和存储数据，并且没有主租户所有者故意重新映像磁盘。因此，我们必须（1）对它们进行修改，使其成为共址感知; （2）为他们提供副本放置算法，以改善主要租户面临的数据可用性和持久性，以及如何管理这些算法。再一次，第一个要求相当容易完成，所以我们在第五部分讨论我们的实现。在这里，我们关注第二个要求，即智能副本放置。面临的挑战是主要租户和管理系统可能会损害任何区块的数据可用性和耐久性：

（1）如果块的副本存储在同时加载尖峰的主租户中，则该块可能变得不可用;

（2）如果开发人员或管理系统在短时间内重新映像包含块的所有副本的磁盘，块将丢失。然后必须考虑副本放置算法主要承租人和管理系统活动。直观的最佳优先方法是尝试寻找重新映像其磁盘的主租户，并从这些主租户中选择具有最低CPU利用率的主租户。但是，这种贪婪的方法有两个严重的缺陷。首先，它独立地处理耐久性和可用性，一个接一个地忽视它们之间的相互作用。其次，在所有“好”主租户的空间耗尽之后，必须在可能导致耐用性差，可用性差或两者兼而有之的地点创建新的复制品。我们更倾向于做出决定，同时提高耐用性和可用性，同时尽可能均匀地在所有类型的主租户中均匀分布副本。



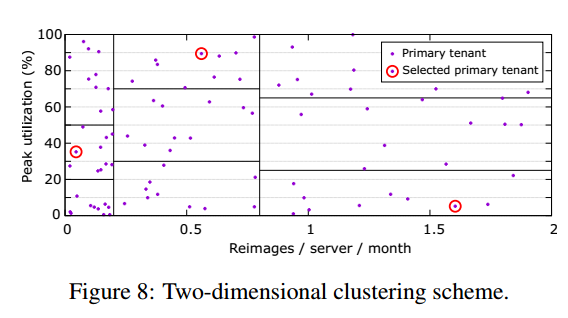
因此，我们的复制品放置算法（算法2）创建了一个二维聚类方案，其中一个维度对应于耐久性（磁盘再现），另一维度对应于可用性（峰值CPU利用率）。它将二维空间分为3类（不频繁的，中等的，频繁的图像与低，中，高峰利用率），每个类别有相同数量的可用存储量用于采集S / 9，其中S是总数当前可用的存储量（第4行和第5行）。只要它们提供足够的主要租户多样性，这个想法就可以被应用于3×3以外的分割。上述方法试图平衡不同班级的可用空间。但是，如果主租户的可用空间数量差异很大，文件系统开始变满，则可能无法实现完美的平衡。原因是完美的平衡空间可能需要把一个大的主租户分成两个或更多的班级。我们通过为每个租户选择一个班级来防止这种情况，以避免伤害安置多样性。副作用是小主租户得到更快填补，导致更大的主租户最终成为副本的唯一可能的目标。这种效果可以通过不填充文件系统来消除，即少于三个主租户作为副本可能的目标。

从本质上讲，空间利用和多样性之间有一个折衷。我们在第7节中进一步讨论这个折衷。当客户创建一个新块时，我们的算法为每个副本选择一个类。第一类是创建块的服务器的类;该算法在该服务器放置一个副本来提升局部性（第6和7行）。如果所需的复制大于1，则它会随机地重复选择类，这样二维空间的行或列就不会有两个选择（第9,10和11行）。它在这个类中随机选择一个主租户（随机选择的服务器）中放置一个副本，同时确保同一环境中没有两个主租户接收副本（第12,13和14行）。最后，对于大于3的所需复制级别，它会进行额外的选择。在每一轮开始时，它忘记了前一轮（第15,16和17行）的行和列选择的历史。环境约束是我们的技术的唯一方面是AutoPilot特定的。然而，这些限制概括为任何管理系统：避免将多个复制品放置在引起资源使用，重新成像或失败的相关性的任何逻辑（例如，环境）或物理（例如机架）服务器分组中。图8显示了我们的集群方案和主要租户选择的例子，假设所有主要租户都有相同数量的可用存储空间。定义高峰利用率类的行不对齐，因为我们确保所有类中的可用存储空间都是相同的。

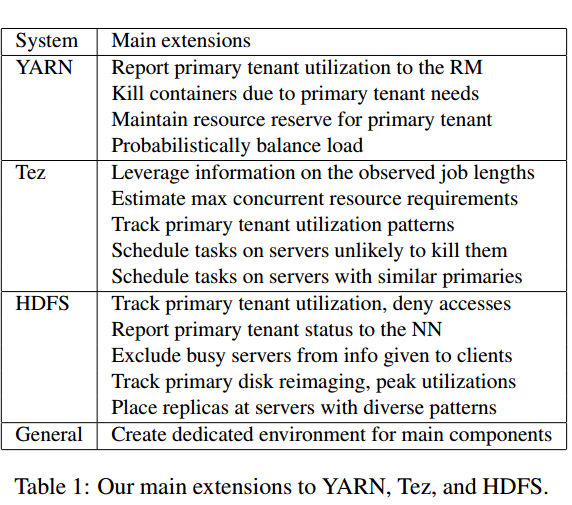
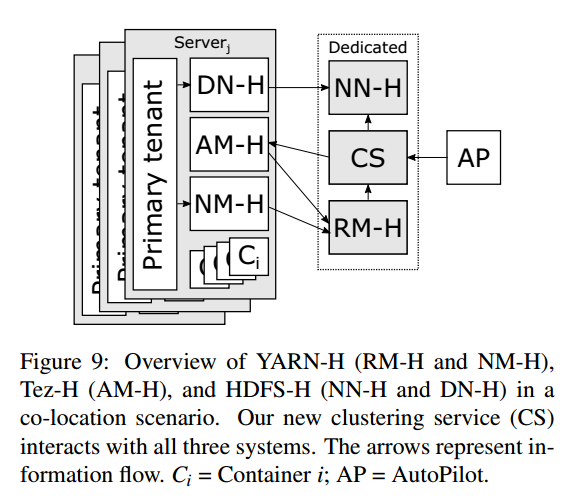
# 系统实现

我们将技术实施到YARN，Tez和HDFS。接下来，首先讲一下各个系统，然后介绍实施指南和系统，称为YARN-H，Tez-H和HDFS-H（“-H”是指历史）。

## 背景知识



YARN [36]包括在专用服务器上运行的全局资源管理器（RM），每个服务器上的节点管理器（NM）以及在其中一个服务器上运行的每个工作应用主机（AM）。 RM在整个集群中仲裁资源（当前，核心和内存）的使用。如果失败，（主）RM通常由辅助RM进行备份。每个AM都从RM请求容器来运行其作业的任务。每个容器请求为它指定所需的核心和内存分配，并且可选地指定“节点标签”。 RM为具有可用资源和相同标签的每个容器选择目标服务器。AM决定在每个容器中执行哪些任务。AM还跟踪任务的执行情况，适当排序它们，并重新启动任何被杀死的任务。每个NM都创建容器，并定期将“本地可用资源”数量报告给RM。 NM杀死任何试图利用更多内存而不是分配的容器。Tez [29]是可以构建MapReduce，Hive，Pig和其他应用程序的流行框架。Tez提供了一个以DAG执行复杂工作的AM。 HDFS [11]包括在专用服务器上运行的全局名称节点（NN）和每个服务器的数据节点（DN）。 NN管理命名空间和文件块到DN的映射。 （主）NN通常由辅助NN来备份。默认情况下，NN复制每个块（256 MBytes）三次：在创建块的服务器中的一个副本，在同一机架的另一个服务器中的一个副本，以及在远程机架中的一个副本。在块访问时，NN通知客户端关于存储块副本的服务器。客户端然后直接联系任何这些服务器上的DN来完成访问。。DN对NN的心跳;在一些DN丢失心跳之后，NN开始在其他服务器上重新创建对应的副本，而不会使网络过载（每​​小时30块）。



## 实现方针

我们首先必须修改系统，以了解主要租户及其对服务器资源的优先级。由于这个优先权，我们必须确保这些系统（RM和NN）的关键组件不与任何主租户共享他们的服务器。其次，我们希望将基于历史的任务调度和数据放置算法整合到这些系统中。图9概述了我们的系统。图中的箭头表示信息流。每个共享服务器都会收到我们系统的一个实例 其他工作量被视为主要租户。表1概述了我们对的主要的扩展。接下来的部分描述我们的系统。

## YARN-H 和 Tez-H

**设计目标**：（G1）确保主租户总是得到他们想要的核心和记忆;（G2）确保主租户永远存在资源的储备; （G3）在由于相应主租户的资源需求而不太可能被取消的服务器上安排任务。

**主要租户意识**：我们通过修改NM来实现YARN-H中的目标G1和G2（1）跟踪主租户的核心和内存利用情况; （2）将它们四舍五入到下一个整数的内核和下一个整数MB的内存; （3）将这些四舍五入值和次租户的核心和记忆分配的总和报告给RM-H。如果NM-H检测到没有足够的预留资源，则通过将足够的容器从最小到最大的容器进行杀死，从而将储备补充回预定的数量。

**智能任务调度**：我们通过执行一个执行我们的聚类算法的服务来实现目标G3，并将我们的类选择算法集成到Tez-H中。我们在4.1节描述了这两种算法。 Tez-H向RM-H要求估计最大并发容器数量。当Tez-H选择一个类时，请求将为该类指定节点标签。当Tez-H选择多个类时，它使用一个分隔表达式来命名标签。 RM-H将容器安排到正确类别的心跳服务器，其概率与服务器的可用资源成比例。如果Tez-H未命名标签，则RM-H将使用其默认策略选择目标服务器。

**开销：**我们的修改引入了微不足道的开销。对于主租户意识，我们添加一些系统调用的NM来获得资源利用率，执行一些算术运算，并搭载使用现有的心跳结果到RM-H。集群服务在工作执行的关键路径上工作，使用一些算术运算来计算头部空间，并且在RM-H上施加很小的负载。与RM-H每分钟查询一次相比，每3秒钟服务器就会心跳RM-H。Tez-H需要与每个作业的集群服务进行一次交互。

## HDFS-H

**设计目标**：（G1）确保我们永远不会在主服务器上使用更多的空间， （G2）确保HDFS-H数据访问在需要服务器资源时不会干扰主租户;和（G3）考虑到主租户的资源使用情况以及他们如何管理，将每个区块的副本放在尽可能耐久和可用的位置。请注意，使用收获存储时无法保证完整的数据耐久性。例如，服务工程师或管理系统可能会重新映像

在同一时间的磁盘数量，销毁一个块的多个副本。显然，可以通过使用更多的副本来增加耐用性。我们在第六部分探讨这一点。

**主要租户意识：**对于目标G1，我们使用HDFS中的现有机制：主要租户声明HDFS-H可以在每个服务器中使用多少存储。实现目标G2更加困难。为了使我们的更改与客户端无缝对接，我们修改了DN，以便在其副本不可用时拒绝数据访问（即允许访问会消耗一些资源预留），导致客户端尝试其他副本。（如果所需块的所有副本都处于忙状态，则该块将变为不可用，并且Tez将无法执行相应的任务。）此外，DN-H报告“忙”，或者NNH在其心跳中可用。如果DN-H表示正忙，NNH将停止将其作为副本的潜在源（并停止将其用作新副本的目的地）。当CPU利用率低于预留阈值时，NN-H将再次列出服务器作为副本源（并将其用作新目标的目的地）。

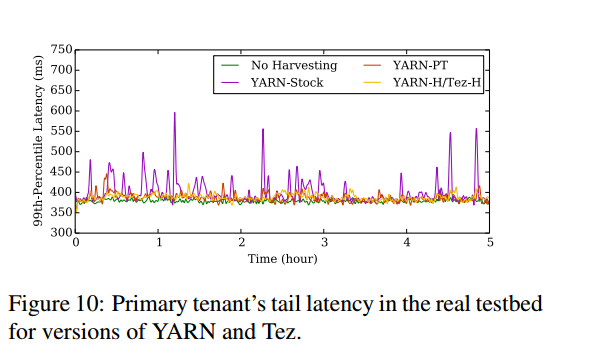
**智能副本放置：**对于目标G3，我们将我们的副本放置算法（4.2节）整合到NN-H中。

**开销：**我们对HDFS的扩展强加了可以忽略的开销。对于主租户感知，我们向DN添加一些系统调用以获取主租户CPU利用率，并在心跳中将结果搭载到NN-H。在重负载下拒绝请求增加了两个网络传输，但是与磁盘访问相比，这个开销是最小的。对于智能副本放置，我们的修改将聚类算法添加到NN，并且需要额外的通信来接收算法输入。集群和数据结构更新发生在后台，离开关键路径。

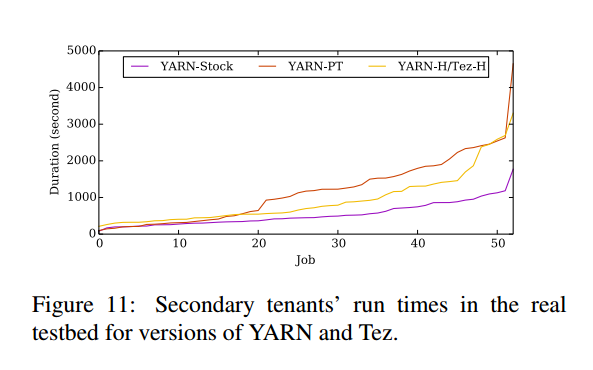
# 仿真和实验结果分析

## 实验结果分析

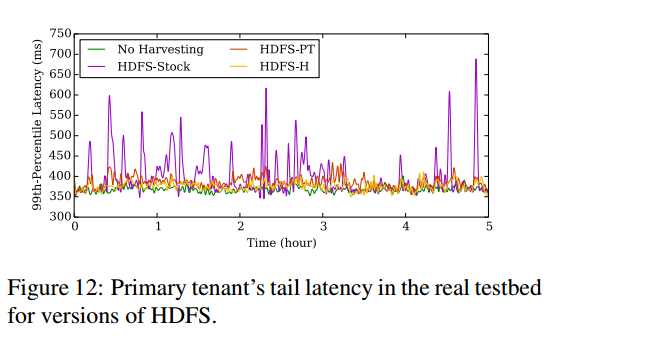
**任务调度比较**：我们从调查收集备用计算周期对主租户性能的影响开始。图10显示了在五个小时的实验中，服务器每分钟响应时间的平均值（百万分之九十九）。标有“No Harvesting”的曲线描述了我们单独运行Lucene时的尾部延迟。



其他曲线描绘Lucene尾部潜伏期在不同的系统下，当TPC-DS工作收获整个群集的备用周期。该图显示YARN-Stock显着影响尾部延迟，因为它忽略了主租户。相比之下，YARN-PT使尾部潜伏期显着降低并且更一致。主要原因是YARN-PT实际上杀死了任务，以确保主租户的负载可以爆发而没有延迟惩罚。最后，YARNH / Tez-H的尾部延迟几乎与“无收获”的尾部延迟一致。最大尾等待时间差异仅为44ms，这与无收获执行中的变化量相当（平均尾延迟范围从369ms到406ms）。 YARN-H集群容量利用率更高，尾部延时更好。 YARN-H / Tez-H的另一个关键特性是它可以很轻松地将任务安排到服务器上，而这些服务器不太可能被杀死。图11显示了针对YARN-Stock，YARN-PT和YARN-H / Tez-H的TPC-DS中所有作业的执行时间。正如人们所期望的那样，YARN股票的执行时间最短。不幸的是，这种表现的代价是破坏了主租户的承诺，这是不可接受的。由于YARN-PT必须在主租户的负载突发时终止（并重新运行）任务，因此执行时间要高得多，平均为1181秒。 YARN-H / Tez-H平均下降了938秒。在这些实验中，YARN-H / Tez-H将CPU的平均利用率从33％提高到了54％，这是一个显着的改善，因为我们为主要租户突发保留了33％的CPU。改善利用率取决于主要租户的利用情况（利用率越低，可以获得的资源越多），来自二级租户的资源需求（需求越高，可以安排的任务越多），资源储备（储备越小，我们可以收获的资源就越多）。总的来说，这些结果清楚地表明，YARN-H / TezH既能保护主租户的性能，又能提高批量工作的性能。

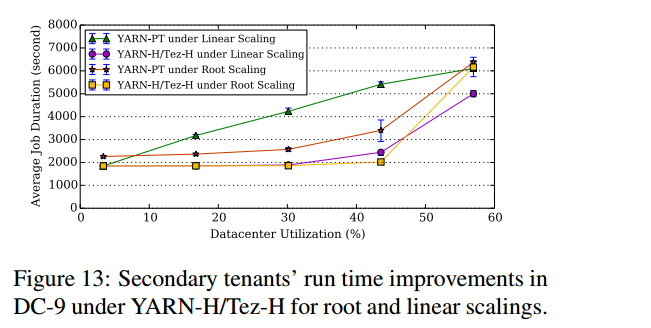


**数据放置和访问比较：** 我们现在调查HDFS-H是否能够保护主租户的性能，并提供比同行更高的数据可用性。 图12显示了在另一个5小时的实验中，服务器每分钟的第99百分位响应时间（以毫秒为单位）的平均值。 正如预期的那样，该数字显示HDFS-Stock显着降低了尾部延迟。 HDFS-PT和HDFSH降低到最多47毫秒。 原因是这些版本避免在繁忙的服务器上访问/创建数据。 但是，HDFS-PT实际上导致了47次访问失败，即这些块因为所有的副本都忙而无法访问。 通过使用我们的智能数据放置算法，HDFS-H消除了所有失败的访问。

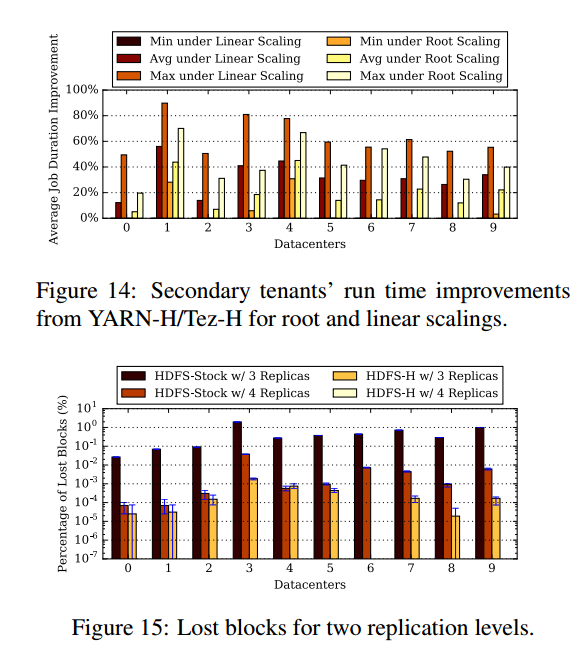


## 仿真结果分析

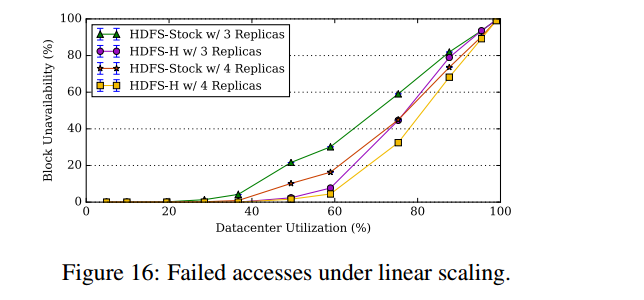
**任务调度比较：**假设我们的真实生产数据中心的规模和行为，我们开始我们的模拟研究，考虑CPU利用率的全部范围。回想一下，我们使用两种方法来从实际使用中调整利用率（向上和向下）：线性和根比例。为了隔离我们使用历史主租户利用的好处，我们比较YARN-H / Tez-H和YARN-PT。图13描述了在DC-9系统和缩放下的平均批处理作业执行时间与利用率的关系。曲线上的每个点都显示了五次运行的平均值，而间隔的范围则是运行中的最小平均值到最大平均值。正如人们所期望的那样，高利用率会导致更高的排队延迟和更长的执行时间。



（回想一下，我们为主要租户保留了33％的资源，所以当我们接近60％的利用率时，排队已经很长了）。然而，线性缩放下的YARN-PT表现不同，在较低的利用率下，平均执行时间开始显着增加。原因在于，线性缩放比每个主要租户的CPU利用率在根比例上产生更大的时间变化。更高的利用率变化意味着YARN-PT更可能不得不杀死任务，因为它不知道主要租户的历史使用模式。例如，在利用率为45％时，线性缩放下的YARN-PT比其他系统缩放组合多杀死4X个任务。因为YARN-H / Tez-H使用我们的聚类和智能任务调度，所以在大多数利用率范围内，它显着提高了工作绩效。在线性缩放下，平均执行时间缩短范围从0％到55％，而在根缩放下，范围在3％到41％之间。YARN-H / Tez-H在线性缩放下的优势更大，因为每个主租户的使用模式随着时间而变化更大。要查看具有不同于DC-9特性的主租户的影响，图14显示了YARN-H / Tez-H在每个数据中心的利用率范围内的最低，平均和最高作业执行时间的改进情况利用率）。在线性缩放下的平均改进范围从12％到56％，在根缩放下平均改善从5％到45％。DC-0和DC-2的平均改进程度最低，其中主要租户利用率随时间变化最小。在另一个极端，DC-1和DC-4的平均改善幅度最大，因为许多主要租户表现出显着的时间利用率变化。无论缩放类型如何，这两个数据中心最大的最大改进（分别是线性和根扩展下的〜90％和〜70％）也都来自这两个数据中心。



**数据放置和访问比较：**我们现在考虑HDFS-H中的数据持久性。 图15显示了在两个复制级别（每块三个和四个副本）下丢失块的百分比，因为我们模拟了一年的reimages和4M块。 每个条形图描绘了五次运行的平均值，并且这些模拟中的数据损失百分比范围从最小值到最大值。 丢失的条意味着在相应的五个模拟中没有数据丢失。单个丢失块表示丢失块的10 \* 5（<100X 1 / 4M）百分比，即6个耐用性9个。该图显示，与HDFS-Stock相比，HDFS-H在三路复制下减少了两个数量级的数据丢失。而且，对于其中一个数据中心，HDFS-H消除了三路复制下的所有数据丢失。任何数据中心的HDFS-H的最大丢失数量只有81块（DC-3）。



在四路复制的情况下，HDFS-H完全消除了所有数据中心的数据丢失，而HDFS-Stock仍然在整个板块都出现亏损。这些结果表明，尽管磁盘空间具有收获性质，而且许多主租户的重新映像率相对较高，但我们的数据放置算法在耐用性方面却有显着的提高。事实上，除了一个数据中心外，HDFSH和三向复制的损失要低于HDFS-Stock和四向复制的损失;即我们的算法几乎总是在比HDFS-Stock更低的空间上实现更高的耐用性。我们的数据可用性结果也是正面的。图16显示了在两个复制级别和线性缩放下失败访问的百分比，作为平均利用率的函数。这个数字包括五次范围内的范围，但是它们都很小，看不到。该图显示，HDFS-H在高达50％的利用率（X40％）下比HDFS-Stock没有数据不可用性，在两种复制水平下都没有更高的利用率（50％）。在50％的利用率下，HDFSStock在两种复制水平下都表现出相对较高的不可用性。大约66％的利用率，不可用性开始增加更快（如果CPU利用率高于66％，访问不能进行）。更有意思的是，我们在三路复制下的智能数据放置比四路复制下的HDFS-Stock在低于75％的利用率下实现了更低的不可用性。除了HDFSH在线性缩放比例较高（50％）的情况下，HDFSH不具有不可用性。无论扩展类型如何，HDFS-H都可以在比大多数应用的HDFS-Stock更低的空间开销下实现更高的可用性。

# 总结

在这篇文章中，作者首先描述了十个大型数据中心的所有服务器，然后引入了技术和系统，从批处理负载中有效地获得数据中心的备用计算周期和存储空间。 本文的系统体现了对现有主要工作负载的深入，并且合理德利用了其历史使用和管理信息。根据一个实验测试平台和十个数据中心的模拟结果显示，本文的系统可以在很多情况下消除数据丢失和数据不可用，同时保护主要工作的负载并显著提高批处理作业性能。基于这些结果，可以看出本文的系统，特别是任务调度和数据放置策略，使得数据中心运营商提高利用率并降低TCO。

# 参考文献

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: A System for Large-Scale Machine Learning. In Proceedings of the 12th USENIX Symposium on Operating System Design and Implementation, 2016.

[2] L. Abraham, J. Allen, O. Barykin, V. Borkar, B. Chopra, C. Gerea, D. Merl, J. Metzler, D. Reiss, S. Subramanian, J. L. Wiener, and O. Zed. Scuba: Diving into Data at Facebook. Proceedings of the VLDB Endowment, 2013.

[3] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. N. Vijaykumar. Tarazu: Optimizing MapReduce on Heterogeneous Clusters. In Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems, 2012.

[4] L. A. Barroso, J. Clidaras, and U. Holzle. The Dat- ¨ acenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. Synthesis Lectures on Computer Architecture, 2013.

[5] M. Carvalho, W. Cirne, F. Brasileiro, and J. Wilkes. Long-term SLOs for Reclaimed Cloud Computing Resources. In Proceedings of the ACM Symposium on Cloud Computing, 2014.

[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A Distributed Storage System for Structured Data. ACM Transactions on Computer Systems, 2008.

[7] R. B. Clay, Z. Shen, and X. Ma. Accelerating Batch Analytics With Residual Resources From Interactive Clouds. In Proceedings of the 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, 2013.

[8] C. Curino, D. E. Difallah, C. Douglas, S. Krishnan, R. Ramakrishnan, and S. Rao. Reservationbased Scheduling: If You’Re Late Don’T Blame Us! In Proceedings of the ACM Symposium on Cloud Computing, 2014.

[9]C. Delimitrou and C. Kozyrakis. Paragon: QoSAware Scheduling for Heterogeneous Datacenters. In Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems, 2013.

[10] C. Delimitrou and C. Kozyrakis. Quasar: Resource-Efficient and QoS-Aware Cluster Management. In Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, 2014.

[11] A. Foundation. HDFS Architecture Guide, 2008.

[12] A. Goder, A. Spiridonov, and Y. Wang. Bistro: Scheduling Data-Parallel Jobs Against Live Production Systems. In Proceedings of the USENIX Annual Technical Conference, 2015.

[13] I. Goiri, R. Bianchini, S. Nagarakatte, and T. D. Nguyen. ApproxHadoop: Bringing Approximations to MapReduce Frameworks. In Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems, 2015.

[14] I. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. GreenHadoop: Leveraging Green Energy in Data-processing Frameworks. In Proceedings of the 7th ACM European Conference on Computer Systems, 2012.

[15] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-Resource Packing for Cluster Schedulers. In Proceedings of the 2014 ACM SIGCOMM Conference, 2014.

[16] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In Proceedings of the 8th USENIX Conference on Networked Systems

Design and Implementation, 2011.

[17] M. Isard. Autopilot: Automatic Data Center Management. SIGOPS Operating Systems Review, 2007.

[18] K. Karanasos, S. Rao, C. Curino, C. Douglas, K. Chaliparambil, G. M. Fumarola, S. Heddaya, R. Ramakrishnan, and S. Sakalanaga. Mercury: Hybrid Centralized and Distributed Scheduling in Large Shared Clusters. In Proceedings of the USENIX Annual Technical Conference, 2015.

[19] H. Kasture and D. Sanchez. Ubik: Efficient Cache Sharing with Strict Qos for Latency-Critical Workloads. In Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, 2014.

[20] M. A. Laurenzano, Y. Zhang, L. Tang, and J. Mars. Protean Code: Achieving Near-Free Online Code Transformations for Warehouse Scale Computers. In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, 2014.

[21] J. Leverich and C. Kozyrakis. Reconciling High Server Utilization and Sub-Millisecond Quality-ofService. In Proceedings of the 9th European Conference on Computer Systems, 2014.

[22] H. Lin, X. Ma, J. Archuleta, W.-C. Feng, M. Gardner, and Z. Zhang. MOON: MapReduce On Opportunistic eNvironments. In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, 2010.

[23] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor-A Hunter of Idle Workstations. In Proceedings of the 8th International Conference on Distributed Computing Systems, 1988.

[24] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis. Heracles: Improving Resource Efficiency at Scale. In Proceedings of the 42nd Annual International Symposium on Computer Architecture, 2015.

[25] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations. In Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, 2011.

[26] M. McCandless, E. Hatcher, and O. Gospodnetic. Lucene in Action: Covers Apache Lucene 3.0. Manning Publications Co., 2010.

[27] D. Novakovic, N. Vasic, S. Novakovic, D. Kostic, and R. Bianchini. DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments. In Proceedings of the USENIX Annual Technical Conference, 2013.

[28] G. Ren, E. Tune, T. Moseley, Y. Shi, S. Rus, and R. Hundt. Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers. IEEE Micro, 2010.

[29] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, and C. Curino. Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2015.

[30] B. Sharma, T. Wood, and C. R. Das. HybridMR: A Hierarchical MapReduce Scheduler for Hybrid Data Centers. In Proceedings of the 33rd International Conference on Distributed Computing Systems, 2013.

[31] L. Tang, J. Mars, and M. L. Soffa. Compiling for Niceness: Mitigating Contention for QoS in Warehouse Scale Computers. In Proceedings of the 10th International Symposium on Code Generation andOptimization, 2012.

[32] L. Tang, J. Mars, W. Wang, T. Dey, and M. L. Soffa. ReQoS: Reactive Static/Dynamic Compilation for QoS in Warehouse Scale Computers. In Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems, 2013.

[33] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: A Warehousing Solution Over a Map-Reduce Framework. Proceedings of the VLDB Endowment, 2009.

[34] Transaction Processing Performance Council. TPC Benchmarks.

[35] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Backfilling Using System-Generated Predictions Rather Than User Runtime Estimates. IEEE Transactions on Parallel and Distributed Systems, 2007.

[36] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O’Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache Hadoop YARN: Yet Another Resource Negotiator. In Proceedings of the 4th Annual Symposium on Cloud Computing, 2013.

[37] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale Cluster Management at Google with Borg. In Proceedingsof the 10th European Conference on Computer Systems, 2015.  
[38] H. Yang, A. Breslow, J. Mars, and L. Tang. Bubbleflux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers. In Proceedings of the 40th Annual InternationalSymposium on Computer Architecture, 2013.  
[39] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In Proceedings of the5th European Conference on Computer Systems, 2010.  
[40] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. In Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing, 2010.  
[41] W. Zhang, S. Rajasekaran, S. Duan, T. Wood, and M. Zhuy. Minimizing Interference and Maximizing Progress for Hadoop Virtual Machines. SIGMETRICS Performance Evaluation Review, 2015.  
[42] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes. CPI2: CPU Performance Isolation for Shared Compute Clusters. In Proceedings of the 8th ACM European Conferenceon Computer Systems, 2013.  
[43] Y. Zhang, M. A. Laurenzano, J. Mars, and L. Tang. SMiTe: Precise QoS Prediction on Real-System SMT Processors to Improve Utilization in Warehouse Scale Computers. In Proceedings of the47th Annual IEEE/ACM International Symposiumon Microarchitecture, 2014.