学号 2017202110038

密级_____

海量存储课程论文

院（系）名 称：计算机学院

专 业 名 称 ：计算机软件与理论

学 生 姓 名 ：黎 文 丹

二〇一七年十二月

摘　要

现有的分布式文件系统普遍使用单一的元数据管理器节点来管理多个数据存储节点的架构，这种系统结构限制的分布式文件系统元数据管理器的容错性和扩展性，论文“Scaling Distributed File Systems in Resource-Harvesting Datacenters”针对这一问题，提出了分布式文件系统的联合结构，这种联合结构实现了元数据管理器的水平扩展并且能够有效的实现子集群之间的访问负载均衡和空间使用均衡。除了扩展问题，在分布式环境下，文件的切分对分布式存储和分布式处理有很大的影响。现有的文件特别是数组结构数据切分主要依靠管理员的经验，这种依靠经验的切分方式严重影响的分布式系统效率的提升，论文“Spartan：A Distributed Array Framework with Smart Tiling”针对这一问题，提出了一种智能平铺策略，通过在前端分析用户使用数组类型数据的意图，进而确定一个最优的平铺策略，实验表明这种智能平铺生成的平铺策略能有效的优化网络访问延时。

关键词：分布式文件系统；平铺策略

目　录

第 1 章　绪论

1.1 论文概述

在大数据时代，很多应用领域如机器学习、科学计算等都需要进行大量的数据处理操作，此外数据的爆炸式增长要求我们利用更多的机器来存储和管理数据。为了存储海量数据，大量分布式文件系统应运而生，如 HDFS、GFS、AFS 等。这些现有的分布式文件系统基本解决了海量数据的存储问题。为了提高数据处理能力，结合分布式文件系统提出了同位置服务，即在存储数据的服务器里运行处理相应数据的批处理工作。同位置服务充分利用了数据本地化的优势并充分利用了服务器剩余的 CPU 资源，它减少了网络传输延迟对批处理任务性能提高影响，也使得服务器资源得到了充分的利用。

现阶段，分布式思想和分布式文件系统已经普遍应用于各个应用领域，并且基本满足了各个应用对数据存储、获取以及分析处理的需求。尽管如此，分布式环境中依然存在着大量的挑战。如： 如何合理的切分数据以便于更好的将数据存储的分布式文件系统中并加速数据分析处理的能力，如何在保证数据一致性和耐用性、减少数据移动的情况下扩展分布式文件系统。为了应对现有分布式环境中存在的诸多挑战，人们从多个方面作为切入点研究问题的解决策略。我从众多相关研究论文中选择了两篇论文进行了深入的学习，一篇是由来自纽约大学、北京大学和 IBM 华盛顿研究院的 Chien-Chin Huang 等人发表在 2015USENIX 年度技术会议上的“Spartan : A Distributed Array Framework with Smart Tiling”；另一篇是由杜克大学、微软研究院的 Pulkit A.Misra 等人发表在 2017USNIX 年度技术会议上的“Scaling Distributed File Systems in Resource-Harvesting Datacenters”。这两篇论文分别讨论了分布式智能数组切分策略和分布式文件系统的扩展策略。

1.2 本文内容组织结构

全文总共有七章，全文组织结构如下：

第一章主要对本次论文学习的方向——分布式文件系统的特点、挑战进行了

简单的介绍，列举了本次论文学习材料的作者、出处等相关信息，同时总结了本论文结构安排。

第二章简要介绍分布式文件系统扩展技术的研究现状，并对论文“Scaling Distributed File Systems in Resource-Harvesting Datacenters”中提出的扩展技术的动机、解决思路和实验评估进行了详细的介绍。

第三章简要介绍了分布式数组框架的研究现状，并对论文“Spartan ： A Distributed Array Framework with Smart Tiling”中提出的扩展技术的动机、解决思路和实验评估进行了详细的介绍。

1.3 本章小结

本章主要就分布式文件系统和分布式环境的现状和出现的问题做了一个概要性的介绍。此外还概括了本文的主要学习的两篇论文，并对本文的行文和组织结构进行了介绍。

第 2 章　分布式文件系统扩展技术

2.1 分布式文件系统结构的局限性

大多数现有分布式文件系统通常采用主从系统结构，例如 Hadoop 分布式文件系统（HDFS），一个提供元数据服务的 Namenode 节点（还有一个备用副本）和若干个提供数据存储的 Datanode 节点构成一个 HDFS 集群。这种系统结构存在三个重要的缺点：（1）扩展性差：HDFS 依赖于集中式元数据管理器的架构特点，使得 HDFS 的底层存储（即 Datanode 节点）可以通过简单的添加机器进行水平扩展，而元数据管理器（即 Namenode 节点）不可以。（2）性能：文件操作的性能制约与单个 Namenode 的吞吐量。（3）隔离性：单个 Namenode 难以提供隔离性。

2.2 分布式文件系统扩展技术研究现状

由于现有的分布式文件系统没有被设计成可以透明地或根本地扩展的模式，为了增加集群数量现有的解决方案有两种，一种扩展方法是依赖于管理人员人工的将数据切分为不相关的几个部分并为每个互相独立的数据块分别创建子集群，每个子集群都运行一个独立的管理器用于名称空间的管理；另一种扩展方法是实现多个一致性强的元数据管理器，例如 Google Colossus（GFS 的后续版本）。

然而这两种方案都分别存在着不同的缺陷，第一种方案存在四个主要的缺点：（1）不能向用户呈现完整的命名空间，只能呈现不同的分区视图，而且用户经常对纺织其文件夹或文件的子集群有完全的控制。（2）用户可能无意中填满子集群或频繁访问某一个子集群使得该集群访问需求超载。（3）为了缓解这些情况，管理员必须手动管理文件夹/文件放置（通过文件夹/文件移植和/或迫使用户使用更轻度使用的子群集）。（4）大多数情况下，特别是当服务复杂而多时，管理员（以及用户不可能）很难理解每个子集群中位于同一地点的服务的特征，以便做出适当的文件夹/文件放置决定。第二种方案也有两个主要的缺点：（1）系统变得更加复杂，而且这种复杂性只对于大型设备是必需的（更简单的系统对于更受欢迎的小型系统也是可行的）。（2）任何软件错误，故障或操作失误都会在没有子集群提供隔离的情况下产生更大的影响。

2.3 新的分布式文件系统扩展方法

鉴于这两种方法的缺点，很显然，分布式文件系统可伸缩性需要一个清晰的系统分层，自动化和可管理的方法。

为了实现避免同位置服务的干扰和促进子集群的行为多样性、良好的性能和良好的空间利用率的目标，Pulkit A.Misra 等人重点关注如何透明有效地“联合”子集群，提出了联合体系结构，通过在理解联合名称空间的客户端和元数据管理器之间插入一层软件，并将请求路由到相应的子集群来实现多个集群的扩展。

为了简化这个问题，他们将系统扩展问题分成两部分。一部分是如何将服务器分配到每个子集群，另一部分是如何将文件夹或文件分配到每个子集群。

首先选择要分配给每个子集群的服务器，以最大限度地提高数据可用性和持久性。具体而言，我们使用一致性哈希来实现这一新的目的，在调整子集群的大小时限制数据移动。 其次，当子集群开始用尽空间，或者子集群的元数据管理器开始接收过多的访问负载时，我们将文件夹/文件分配给子集群并进行高效迁移。 我们将这种再平衡建模为混合整数线性规划（MILP）问题，该问题足够简单，可以有效解决。迁移发生在后台，对用户透明。

2．3．1 联合结构

他们提出的架构假定为一个结构与 HDFS、Cosmos Store 和 GFS 类似的未经修改的底层分布式文件系统。它联合了多个分布式文件系统的子集群，每个子集群都有独立的元数据管理器、底层数据存储节点和客户端库。联合中每个子集群独立运行，不知道其他子集群的存在。

与其他分布式文件系统不同的地方在于该联合结构在客户端和服务器端之间有一层由状态存储器、路由器和再平衡器构成的中间层。该中间层主要有以下几点重要意义：一，路由器分发来自客户端的请求而且把这些请求信息记录下来以便于生成集群访问负载的流量报告，此外它还通过询问子集群的元数据管理器收集底层数据存储节点的使用情况；二，状态存储器存储了有路由器收集到的各种信息、全局装载表、路由器状态、再平衡器状态；三，再平衡器依据再平衡策略跨子集群迁移文件夹/文件，然后更新状态表。

2．3．2 再平衡器

再平衡器跨子集群迁移文件夹/文件，然后更新装载表。因为常规的客户端流量可能被定向到正在迁移的文件并由此导致多种失败类型，再平衡器应该确保联合文件系统的一致性。再平衡器使用了预写日志（记录平衡操作以及防止系统无意中运行多个并发再平衡器实例）、写入租约（防止管理员或再平衡器的多个实例对装载表项的更改）来确保平衡过程中的一致性。具体思路如下：

（1）每个再平衡器实例在执行迁移操作前检查日志，如果发现另一个实例正在主动更改命名空间的相同部分，则会中止迁移操作；否则它在状态存储器中记录它即将开始的操作的预写日志。每个操作完成后，它会更新日志以反映完成情况。失败的重新平衡可以使用日志完成或回滚。

（2）在相应的装载表条目上写入租约（在长期迁移期间可能需要续租），并记录要迁移的整个子树的状态（如最后修改时间）。

（3）再平衡器将数据复制到目标子集群。在复制结束时，它会检查复制期间源数据是否被修改。在比较子树的元数据之前，通过状态存储，再平衡器指示路由器防止写入源和目标子树。如果源数据不变，则再平衡器更新装载表，等待所有路由器确认更改，停止阻止写入源和目标子树，删除来自数据源子集群的数据。如果源数据在复制过程中被修改，则再平衡器将重新复制已更改的文件。再平衡器尝试完成三次重新复制，如果这些重新尝试均不足以完成复制，则重新平衡器将回滚，并且在复制开始之前阻止写入数据。

（4）迁移成功后，再平衡器放弃对装载项的限制。

2．3．3 服务器到子集群的映射

首先，需要确定整个联合体系结构中子集群个数，目前子集群的数量定义为服务器总数除以元数据管理器可以有效容纳的服务器数量（默认情况下每个子集群 4000 个服务器）。接着，利用机架名称的一致性哈希值将服务器机架分配到子集群。

选择利用整个机架名称的一致性哈希值的主要原因有四个：一是为了保证添加或删除子集群时减少需要移动的数据量；二是为了保留机架内网络位置的本地化优势（减轻网络传输的压力）；三是为了提高容错性以确保数据的耐用性，由于所有数据的不同备份数据块随机分布在不同机架的服务器上，某个机架的损坏或移除不会使得存储于该机架的数据块丢失；四是为了实现主租户（以数据存取为主要目的的负载）和二级租户（以批处理操作如数据分析为主要目的的负载）

在访问负载和空间消耗的平衡。不同机架使用多个不同的交换机进行信息传递，交换机的数据传输速度往往成为限制访问负载的瓶颈，每个主要租户分布在不同机架上使得每个机架对应的交换机的访问负载得到缓解。

2．3．4 文件夹/文件到子集群的映射

为了在某个子集群访问负载过大或存储空间使用率或高时实现负载均衡，他们提出了两个平衡策略：重新平衡策略和再平衡优化策略。

重新平衡策略是指周期性的重启再平衡器，并将每个子集群近期的元数据访问负载和空闲空间与预定义的阈值进行比较，找出需要进行平衡调整的子集群。那么每个子集群的信息是如何收集的呢？由于路由器拦截所有对元数据管理器的访问，他们可以很容易地积累这些信息并将其存储在状态存储器中。此外路由器还会定期询问每个子集群的元数据管理器，以了解每个子集群中的可用空间量，并在心跳期间将这些信息存储在状态存储区中。因此，重新平衡策略中需要子集群的信息可以由重新平衡器在状态存储中查找获得。

再平衡优化策略是指确定哪些文件夹/文件要迁移到哪个子集群。他们将这个问题建模为一个混合整数线性规划问题（MILP），构造联合命名空间（树形结构）作为数据结构，通过对命名空间树的剪枝操作限制 MILP 问题的大小，提高计算效率。具体过程描述如下：首先，创建一个联合命名空间的树形结构，树中的每个节点包含以下三个信息：（1）该节点自上次重新平衡以来的短时间间隔（例如，5 分钟）收到的峰值负载量，（2）该节点子树的大小，（3）该节点所在子集群 ID。接着，剪除与管理员定义的低端阈值相比表现出更低的负载和更小的空间利用率的节点，这些节点不需要进行平衡处理。最后，将剪枝树作为 MILP 问题的输入。MILP 的目标函数为使得包含多个加权因子的效用函数最小化，使用的加权因子分别为每个子集群的访问负载、每个子集群的已用存储容量、重新平衡中要移动的数据量以及重新平衡后的装载表中的条目数。这些加权因子都是依靠管理人员配置的。

2.4 实验评估

他们在 HDFS 上实现了联合结构和联合技术，并将新的分布式文件系统命名为 DH-HDFS。为了展示主租户，他们使用了 10 个真实的大规模的数据中心的 CPU 利用率和磁盘重映像的统计信息。为了模拟二级租户，他们使用一个来自 Yahoo！

的真实的 HDFS 迹。分别对集群内副本放置策略、服务器分配策略、文件夹/文件分配策略与现有策略进行了对比，结果表明他们提出的各种策略均提高了数据的可用性和耐用性。接着，他们又针对再平衡器对参数（负载阈值、空间使用率阈值以及重启时间间隔）的敏感程度进行了实验。实验结果表明，改变参数阈值会使得负载峰值或空间使用不均衡，但是两者均在阈值之下，因此再平衡器能够有效的平衡集群。最后，他么分别对路由器的性能、再平衡器的性能和文件系统性能进行了实验，结果表明：

（1）路由器性能良好，对元数据操作增加了相对较低的延迟，并且可以忽略不计的延迟来阻止访问。

（2）重新平衡本身是有效的，但完成移植的总体时间可能差异很大，主要是由于主要租户流量。 不过，请记住，再平衡是在后台进行的，对用户来说是透明的，所以迁移时间的可变性不太可能成为问题。

（3）较低优先级（二级）工作量收集剩余资源，使得它们的性能受到主要租户资源需求的影响。大多数辅助工作负载的性能要求比较宽松，所以变化只在极端时才成为问题。然而，如果数据中心操作员希望其一些辅助工作负载具有更高的性能可预测性，那么他们必须（1）在其资源供应中考虑这些工作负载，例如，网络带宽;或(2)确保这些工作负载得到比尽力而为的服务质量更好的工作。

2.5 本章小结

本章主要对论文“Scaling Distributed File Systems in Resource-Harvesting Datacenters”中提出的扩展技术的动机、解决思路和实验评估进行了详细的介绍。首先介绍了现有分布式环境的局限性，接着介绍了论文中提出的分布式文件系统扩展技术，并着重介绍了分布式文件系统扩展技术中提出的联合结构、再平衡器、服务器到子集群的映射策略和文件夹/文件到子集群的映射策略，最后简要介绍了论文的实验部分。论文充分融合了多种现有分布式文件系统扩展技术的优势，提出并实现了一个有效的分布式文件系统扩展技术。

第 3 章　智能平铺的分布式数组框架

3.1 研究现状

为了帮助数组程序跨机器扩展，HPC 和系统社区提出了许多建议来开发分布式数组框架。然而，尽管做了这些努力，一个易于使用，高性能的分布式阵列框架仍然是难以捉摸的。 在分配阵列程序时，开放的挑战是如何最大限度地扩展存储在许多机器内存中的阵列数据的访问位置。 为了改善局部性，需要巧妙地将阵列分区，并将计算与数据共同定位。 我们称之为“平铺”问题。 平铺对于性能至关重要; 为地方而优化的程序可能比那些没有的程序要快一个数量级。

现有的分布式数组框架没有充分解决平铺问题。 大多数系统依靠用户手动指定数组分区; 例子包括 Pydron，Presto，MadLINQ，Global Arrays 和 Petsc。虽然 SciDB 可以自动选择一个好的块大小来优化从磁盘加载数组，但它仍然依赖于用户定义的平铺策略。 手工拼贴可以实现良好的局部性，但是使得所得到的系统比单个机器的使用更加繁琐和复杂。 理想情况下，分布式数组框架应该支持以最少的用户输入进行自动平铺，以实现易用性和高性能。

3.2 智能平铺分布式数组框架

Spartan 使用分层的方法将执行分为前端和后端步骤。运行在客户端机器上的前端捕获用户代码，并将其转换为表达式图，其节点对应于高级操作符。 接下来，前端运行平铺优化器，以确定表达式图形中每个节点的良好平铺策略。 最后，前端将平铺的表达式图发送到后端。 后端提供高性能的高级操作符分布式实现。对于每个操作符，它都会调度在许多计算机上运行的任务的集合。任务根据优化器确定的平铺提示创建，提取和更新分布式内存数组。

3．2．1 高级操作符

Spartan 的高级操作符和分层设计有助于收集自动平铺所需的信息。高级操作符存在的必要性有以下三点：第一，高级操作符是对很多不同类型的计算表达式的抽象。它将各式各样的计算表达式分为 5 个基本操作（map、filter、fold、scan、join_update）和 3 个原始创建数操作（newarray、slice、swapaxis），所有其他的表达式都是这 8 种高级操作符的组合。与分析纷繁复杂、形式多变的

计算表达式相比，分析每种高级操作符在不同平铺方式下的网络传输代价是很简单的。通过叠加高级操作符的网络传输代价，使得确定一个计算表达式的网络传输代价变得可行，因此可以利用高级操作符蕴含的信息明确的分析表达式中每个数组的数据访问模式。第二，前端在讲用户代码转换为用高级操作符表示的过程中，可以获取关于数组形状和大小的运行时信息，这些信息能辅助选择平铺策略。第三，表达式图表示一个很大的执行上下文，从而允许前端理解一个数组是如何被多个表达式使用的。 这对于良好的平铺至关重要。

此外，Spartan 的高级操作符能接受用户自定义的函数作为参数，这使得 Spartan 能更好的兼容和扩展。

3．2．2 表达式图

在用户程序执行过程中，Spartan 的前端通过扫描用户代码捕获数组表达式，并将它们转换为一系列表达式图。在表达式图中，每个节点对应一个高级操作符，从一个节点到另一个节点的边表示它们之间的数据依赖关系。因为 Spartan 的高级操作符创建不可变数组，表达式图是非循环的。前端在捕获数据表达式的过程中使用了延迟评估策略，即只有在遇到控制流程的变量、用于程序输出的变量和用户明确的调用请求评估方法时前端停止增长表达式图形，否则表达式图将持续增长直到预先配置的限制。

3．2．3 基于图的平铺优化器

得到了高级运算符的表达式图，平铺优化器的目标是为每个运算符节点选择一个平铺策略，以最小化总体成本。由于表达式图可能非常大，这个优化问题是 NP 完全问题，通过穷举搜索方法寻找最好的平铺策略是不实际的。因此，他们提出了一个基于图的近似算法来快速识别一个好的平铺策略。

该算法分两个阶段工作。首先，基于表达图和每个高级操作符的成本概况构建一个平铺图。 接下来，它使用贪婪策略来搜索低成本的排列组合。

（1）构建平铺图：构建平铺图的目标是进一步分析表达式图中的平铺选择和成本。对于表达式图中的每个运算符，优化器依据可能的平铺策略将其转换成节点组，即多个平铺节点的集群，每个平铺节点表示一个对高级操作符输出或中间步骤的具体平铺策略。连接两个平铺节点的每条边的权重表示潜在的代价。

（2）搜索好的平铺策略：确定高级操作符的平铺策略相当于在平铺图中的对应的节点组中挑选一个节点，并且平铺节点的不同组合使得执行相应高级操作

有不同的网络传输成本。因此，平铺优化器的下一步就是分析平铺图并找到能最大限度降低总体成本的平铺策略组合。

平铺优化器采用贪婪搜索算法。该启发式算法首先确定最大连接的节点组对应平铺策略。这里，节点组的连通性是指其相邻节点组的数量。在决定节点组 X 的平铺时，算法选择使得 X 有最小网络传输代价的平铺节点。为什么要先优化这些节点呢？一个高级操作符平铺的代价取决于相邻高级操作符的平铺策略选择。因此，相邻高级操作符较多的高级操作符对整体成本影响较大。因此，该算法首先使连通性更高的节点组的成本最小化。该贪婪搜索算法由两个子算法构成，一个是计算每个平铺节点成本的 FindCost 算法，另一个是决定整个平铺图平铺策略组合的 FindTiling 算法。

1）FindCost 算法：通过计算每个相邻节点组与 T（当前结点）之间的最小边权重的总和来获得平铺节点的成本。如果相邻节点组是 swapaxis 这样的视图操作符，则它的平铺节点将由 T 决定。为了得到受 T 影响的精确的代价，FindCost 递归地查找视图操作符相邻节点组的成本。结果对应于平铺节点 T 的最佳可能成本。

2）FindTiling 算法：给定平铺图 G，该算法按照边连通性的顺序处理节点组。对于每个节点组，用 FindCost 计算每个平铺节点的成本，并以最小的代价选择平铺节点。在决定节点组 x 的平铺策略之后，该算法去除连接到所有其他平铺节点的所有边缘。这意味着算法不能自由地为 x 的相邻节点组选择平铺策略——它必须考虑 x 的选择平铺。

平铺算法的复杂度为 O（E×N），其中 E 是平铺图中边的数量，N 是节点组的数量。不能保证找到最佳的平铺。但是，我们发现贪婪策略在实践中运作良好。

3.3 实验评估

由于 NumPy 在机器学习和科学计算中广受欢迎，他们的实现目标是尽可能复制 NumPy 的“感觉”。 他们的原型目前支持 50 多个最常用的 Numpy 内置方法。

Spartan 的实现分为前端和后端两个部分，Spartan 前端用 Python 编写，用于捕获表达式图并执行平铺优化；Spartan 后端由一个指定的主服务器和一组机器上的许多工作进程组成。后端提供所有高级操作符的高效实现。给定一个表达式图形，主服务器一次负责协调一个节点（一个高级操作符）的执行。

他们对智能平铺算法的性能和可伸缩性进行了测量和评估。为了评估智能平铺策略的性能，他们比较了智能平铺算法产生的平铺策略和最佳平铺策略对应用程序的运行时间的影响，结果表明，应用智能平铺算法产生的平铺策略和最佳平铺策略的应用程序运行时间是不一样的，有时候 Spartan 的智能平铺性能优于最佳平铺策略。主要原因是 Spartan 的优化器为所有应用程序提供了最好的平铺选择。

尽管智能平铺为应用程序提供了最好的平铺策略，但不能保证智能平铺对各种应用程序都能很好地运行。因此，他们研究了随机生成程序的智能平铺的性能。结果表明，Spartan 的智能平铺可以为大多数程序提供最好的平铺策略。

为了评估智能平铺策略的可伸缩性，他们以两种方式评估所有应用程序的可伸缩性。 首先，这些应用程序使用固定大小的输入并跨越不同数量的 worker。其次，应用程序使用大小与 worker 数量成线性比例的输入。结果表明，许多应用程序的运行时间实现了完美的收缩。

3.4 本章小结

本章主要对论文“Spartan : A Distributed Array Framework with Smart Tiling” 中提出的扩展技术的动机、解决思路和实验评估进行了详细的介绍。首先介绍了现有分布式数组框架的局限性，接着介绍了论文中提出的分布式数组框架智能平铺技术，并着重介绍了智能平铺技术中提出的高级操作符、表达式图和基于图的平铺优化器，最后简要介绍了论文的实验部分。

参考文献

[1]. Misra, Pulkit A., et al. "Scaling Distributed File Systems in Resource-Harvesting Datacenters." 2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17). USENIX} Association}, 2017.

[2]. Huang, Chien-Chin, et al. "Spartan: A Distributed Array Framework with Smart Tiling." USENIX Annual Technical Conference. 2015.

致谢

时间转瞬即逝，不知不觉中半个学期的学习生活已经过去。感谢老师的悉心指导，尽管我们的研究方向各有不同，我依然学习到了很多研究方法和海量存储相关知识。