基于网络的分布式存储网络系统定序方法的研究与实现

1. 对指导教师下达的课题任务的学习与理解

1.1. 研究背景

分布式存储系统（Distributed Storage System）最早由谷歌提出，其目的是通过服务器来提供使用与大规模，高并发场景下的 Web 访问问题。它采用可扩展的系统结构，利用多台存储服务器分担存储负荷,利用位置服务器定位存储信息,它不但提高了系统的可靠性、可用性和存取效率，还易于扩展。在多对多数据交换模型中，服务器和存储设备通过交换机进行连接。交换机是数据中心架构中的关键组件，它负责将数据从一个设备传输到另一个设备。多对多数据交换模型的优点是可以实现高并发的数据传输和共享。多个服务器可以同时访问多个存储设备，提高了数据访问的效率和吞吐量。此外，这种模型还具有良好的可扩展性，可以根据需求增加服务器和存储设备的数量。

在传统的数据交换模型中，数据通常会被分割成多个数据包进行传输，每个数据包都会带有序列号来标识数据的顺序。然而，由于网络的不稳定性，有时候接收方可能会接收到序列号不连续的数据包，或者某些数据包丢失导致序列号出现缺失，这就产生了数据的空洞问题。当多个服务器同时尝试访问存储空间或者进行数据写入时，可能会发生数据竞争，导致一些服务器需要等待其他服务器完成操作才能进行自己的操作，从而增加数据等待时延。同时存储设备本身的性能也可能成为数据等待时延的原因。当多个服务器同时尝试访问存储设备进行数据写入时，设备的读写速度可能无法满足所有服务器的需求，从而导致数据等待时延增加。

1.2. 研究目的及意义

通过对分布式存储网络系统定序方法的优化，在服务器需要访问已存储的数据时，直接根据存储单元的信息进行数据访问，无需经过序号发生器的编号过程，可以提高 CPU 的 I/O 效率。当服务器需要访问数据中心时，它向数据中心发送请求。一旦服务器发起访问请求，数据中心接收到请求后不再依赖于序号发生器对服务器进行编号,而是立即将可用的存储空间分配给该服务器。同时避免了服务器访问数据中心之前控制信息的交互过程，这样可以确保各个服务器都能及时获取到所需的存储空间，减少排队等待时间。数据中心根据存储空间的可用性，选择合适的存储单元，并将存储单元的信息返回给服务器。服务器得到存储单元的信息后，直接将数据存储到相应的存储单元中。

这种方法的主要优势在于取消了序号发生器对服务器的编号,减少了服务器之间的排队时间和控制语句过程的交互。同时，采用先到先分配存储空间的方法，能够更加高效地利用存储资源，提高了 CPU 的 I/O 效率。

2. 阅读文献资料进行调研的综述

2.1. 国内研究现状

有关于数据定序问题方面的研究,在分布式的存储系统中常用的 WAL(Write Ahead Log，预写日志)机制可以保证数据操作的原子性和持久性，实现原理是数据操作不直接作用于存储系统，而是先写入预写日志，如果数据操作执行失败，那么预写日志中操作记录被忽略，数据操作被撤销；如果数据操作执行成功，记录在预写日志的该数据操作将在随后某个时间被写入存储系统，数据操作被提交。分布式的存储系统是采用将数据按多副本存放方式来保证数据可靠性，对于 WAL 机制不仅要保证数据操作的原子性和持久性，还要确保多副本的数据一致性，因此在 WAL 机制中需要对多个客户端的并发数据操作进行定序，然后将定序后的数据操作转至存储系统,使存储系统的多副本按定序顺序依次执行多个客户端的并发数据操作，保证多副本的数据一致性。基于此背景朱云峰等人(2023)公开了一种数据操作定序方法及系统[1]，方法包括：响应于客户端发起数据操作的广播请求，由各个日志服务器根据广播请求的类型确定逻辑时间戳，记录广播请求和数据操作的逻辑时间戳；基于各个日志服务

器记录的不同数据操作的广播请求和逻辑时间戳对不同数据操作进行定序。他们将数据操作的复制逻辑与定序逻辑解耦，客户端发起数据操作的广播请求直接到达预写日志系统的各个节点实现了复制逻辑，预写日志系统中任一日志服务器通过异步收集各个日志服务器记录的广播请求和逻辑时间戳实现不同数据操作的定序逻辑，因此预写日志系统无需选举主节点作单点角色，可以提升客户端的访问稳定性，且还可避免主节点故障触发选举带来的服务不可用。

他们首先提出在现有技术中，采用的是基于有主共识的预写日志机制。参见图 1 所示，客户端将数据操作的请求提交给预写日志系统的主节点，主节点一方面将请求在其余节点之间做复制，另一方面根据接收请求的前后顺序对请求做定序，在完成复制和定序之后即可回复客户端，然后主节点将已经完成复制和定序的请求再存储到分布式存储系统中。上述实现方案在实际应用中，一方面，主节点成为性能单点，如果主节点故障，就会触发重新选举主节点过程，在新的主节点选出之前整个系统不可服务，并且如果主节点因为网络、负载等原因出现服务性能抖动时，会直接影响客户端数据操作的延迟、吞吐等指标；另一方面，数据操作的复制过程和定序过程都处于客户端提交数据操作路径上，这样对于客户端来说，为数据操作的提交带来了至少两跳的时间延迟。为了解决上述技术问题，他们提出一种无主共识的预写日志机制，以将数据操 作的复制逻辑与定序逻辑解耦，参见图 2 所示，对于复制逻辑，由客户端这一侧通过向预写 日志系统广播数据操作的请求，确认预写日志系统做了相应的持久化操作后，即认为请求 提交成功；对于定序逻辑，由预写日志系统中的任一节点通过异步收集各个节点做持久化 操作时记录的持久化信息，并根据这些持久化信息对多个请求做定序操作，并将定序后的请求存储到分布式存储系统，由分布式存储系统按照定序顺序依次执行各个请求。基于上述描述的无主共识的预写日志机制，由于无需在预写日志系统选举主节点作单点角色，因此就不存在主节点性能单点问题，提升了客户端的访问稳定性，并且还可避免主节点故障触发选举过程所带来的服务不可用窗口。另外，客户端向预写日志系统广播数据操作的请求，在确认预写日志系统做了相应的持久化操作后，不需要等待完成请求定 序过程即认为请求提交成功，因此对于客户端来说，缩减了请求提交延迟。

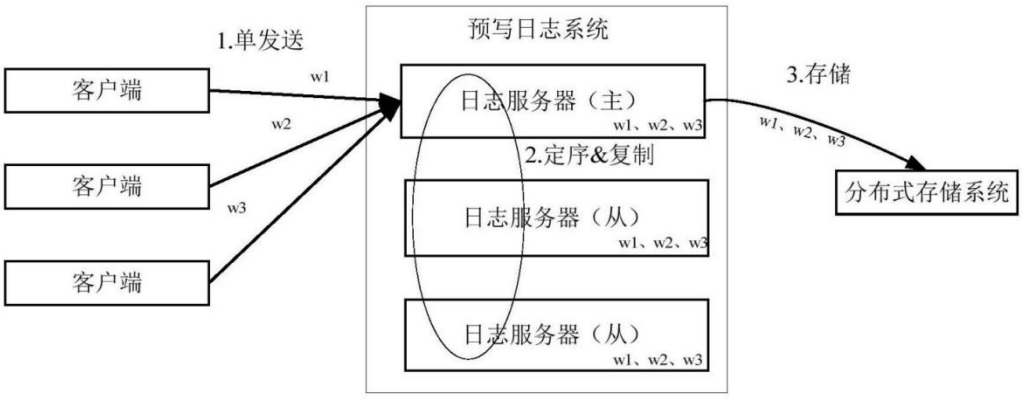

图 1 有主共识的预写日志机制实现架构

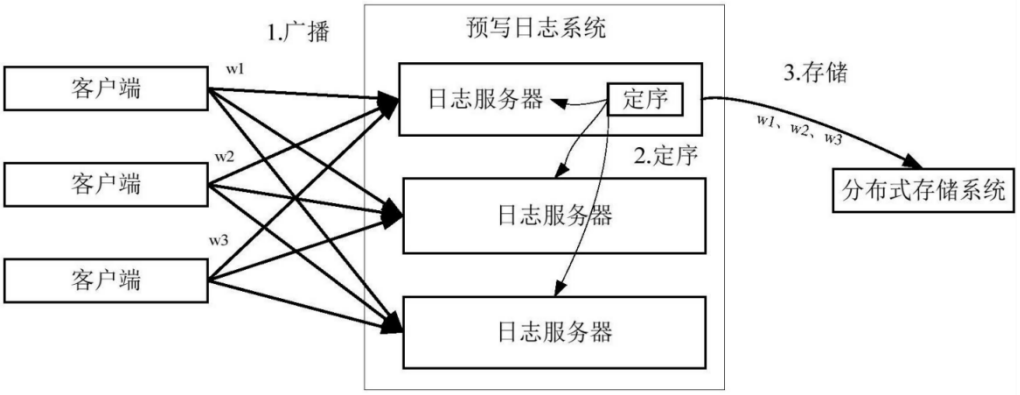

图 2 无主共识的预写日志机制实现架构

在关于提高分布式存储网络 CPU 性能以及 IO 吞吐率方面，国内也有很多学者提出了不同的方法。在大数据计算领域，数据规模大，数据维度高，数据种类多是其典型特征，内存计算技术逐渐开始发挥重要作用。因此，增加分布式存储网络系统的存取效率以及其吞吐量的提升受到了学术界的广泛关注和深入研究。加州大学伯克利分校开发的 Apache Spark[5]以及 SAP 公司在 2012 年推出的 HANA 内存计算[6]已经得到工业界的广泛关注。国内学者陈游旻等人（2019）提出基于 RDMA 的分布式存储系统[2]，远程内存直接访问(remote direct memory access, RDMA)作为一种新兴的跨网数据传输技术逐渐从高性能计算走进大数据领域，并开始受到广泛关注，RDMA 允许本地 CPU 绕过操作系统，直接读写远端节点内存，该过程无需远端 CPU 的参与。它支持在对方主机 CPU 不参与的情况下远程读写异地内存，并提供高带宽、高吞吐和低延迟的数据传输特性，从而大幅提升分布式存储系统的性能。同时本文指出基于 RDMA 设计高效的系统软件，在 CPU 调度层面需要考虑负载均衡的问题：服务端线程静态映射的方式能提升并行度，但每个客户端负载具有差异性，且相应的远程调用开销不尽相同，因此有可能导致个别 CPU 核会产生处理繁忙的问题，而其他 CPU 核空闲等待的现象。针对这个问题，陈游旻等人利用 Stuedi P[7]（2014）提出的一种“Work Stealing and Load Balancing” 的管理方法，在服务端引入监控器，用于实时统计各 CPU 核的工作负载状态，当某 CPU 核的工作负载超过某阈值，则将新来的部分请求放入到 1 个全局队列中，而其他 CPU 核在不繁忙时查看全局队列，并及时处理相应请求。这种方式有效解决了各 CPU 核负载不均衡的问题，但同时也引入了时序问题，导致早到达的 RPC 请求在晚到的请求之后被处理。这种乱序响应需要在客户端处理逻辑中被谨慎考虑。同时，RDMA 可以直接访问远端内存数据，这使得分布式键值存储系统中的数据索引模式发生改变。关于分布式键值存储系统的进一步优化，崔玉龙等人（2023）就键值存储提出了一种面向跨区域架构的无协调分布式键值存储系统 Elsa[4]。Elsa 在保证高性能和高可拓展性的基础上， 采用无冲突备份数据结构 (CRDT) 技术来无协调的保证副本间的强最终一致性， 降低了系统节点间的协调开销。采用了一种基于 CRDT 技术的无协调一致性模型而非读写锁、共识协议等机制来解决并发冲突和副本一致性问题。同时 Elsa 采用了 Actor 的编程模型，线程间无协调的处理用户请求，并保证副本间的强最终一致性，保证系统将多数时间花费在处理用户请求上，从而提升系统的吞吐量。

张晓等人（2020）对 RDMA 的进一步应用提出了 DCache：一种基于 RDMA 的 HDFS 分布式缓存机制[8]，HDFS (Distributed File System)是 Hadoop 的重要组成部分，提供数据存储服务。IO 子系统的性能对数据处理效率有很大的影响。在 Hadoop 系统中，计算任务被调度

到数据所在的节点，以减少 IO 时间。作业调度和数据分布对数据处理的效率有很大的影响。HDFS 为用户提供了一种机制来指定要缓存的文件或目录。但是缓存的数据只有在相同数据节点上运行的作业才能访问。他们提出了一种支持 rdma 的分布式 HDFS 缓存机制。将缓存功能从原始数据节点中分离出来。设计并实现了一种新的缓存功能组件。集群中的任何节点都可以通过 RDMA (Remote Direct Memory Access)直接访问这些节点中的缓存数据。与现有的缓存机制相比，该方法提高了 IO 进程的性能和稳定性。并且使用缓存机制加速了数据写入过程。

从我国目前的发展来看，关于提高分布式网络系统的 I/O 率在应用方面的研究较为丰富，但是具体采用优化定序方法从而提高分布式网络定序方法的研究较少。近些年关于数据定序问题有更为详尽的研究，进一步的实验以及具体效果研究有着重要的意义。

2.2. 国外研究现状

国外目前关于提高 CPU 的 I/O 效率的研究重心在于通过预取和缓存技术的改进来提高读取操作效率从而提高 I/O 效率。提高分布式文件系统的读操作效率是一个具有挑战性和重要的研究问题。在分布式文件系统中，用于提高读操作性能的两个主要过程是预取和缓存。研究表明，预取技术可以解决大规模计算系统中的 I/O 瓶颈问题。预测性预取[11]和知情预取[12]这两种流行的方法都通过在数据访问之前将数据从磁盘预加载到主存来提高 I/O 性能。证据[12]表明，知情预取可以弥合处理器和磁盘 I/O 之间的性能差距。在 Patterson 等人的研究中，TIP 通知预取算法通过应用成本效益分析为预取和缓存分配缓冲区，提高了 I/O 密集型应用程序的性能[12]。Maen M. Al Assaf 于 2017 年提出了一种称为 IPODS 的知情预取技术[10]，他们研究的重点是知情预取，其中预取决策是基于应用程序提供的未来访问提示做出的。研究的目的是将流水线技术整合到 IPODS 中，为分布式多级存储系统提供预测性预取。它利用应用程序公开的访问模式来预取分布式多级存储系统中的暗示块。在 IPODS 中开发了一个预取管道，其中一个知情预取过程被划分为一组独立的预取步骤，并在分布式系统中分散在多个存储层中。在 IPODS 系统中，当数据块从硬盘预取到远程存储服务器的内存缓冲区时，服务器中缓冲的数据块通过网络预取到客户端的本地缓存中。他们展示了这两个预取步骤可以以流水线方式处理，以提高分布式存储系统的 I/O 性能。他们的 IPODS 技术与现有的预取方案在两个方面有所不同。首先它通过将暗示的数据保存在客户端的本地缓存和存储服务器的快速缓冲区(例如，固态磁盘)中来减少应用程序的 I/O 延迟。其次，在预取管道中，多个知情预取机制半独立地协调以获取块(1)从服务器中的低级(慢)存储设备到高级(快)存储设备，(2)从服务器中的高级设备到客户端的本地缓存。IPODS 中的预取管道明智地隐藏了分布式存储系统中的网络延迟，从而减少了分布式系统中的总体 I/O 访问时间。使用广泛的实际 I/O 跟踪，根据其实验表明 IPODS 可以显着提高分布式存储系统的 I/O 性能 6%。关于预测性预取，我国学者廖健伟（2017）提出了一种新的机制来对块访问时间序列进行建模[13]，以分析块访问模式，从而有意地指导块数据预取。该方案背后的基本思想是，某个偏移域内的块访问可能具有一些相关性，这些相关性可能有助于在文件系统级别上对访问模式进行分类。此外，采用邻接矩阵技术表示访问模式，加速模式匹配，最终实现 I/O 优化。通过对磁盘上多个真实块迹进行的一系列仿真实验，实验结果表明，所提出的预取机制优于其他比较机制。具体来说，与常用的顺序预取方案相比，它可以将平均 I/O 响应时间减少 14.6%-17.9%，与基于频繁序列挖掘的预取相比减少 4.1%-10.5%，但空间和时间开销更少。

现代 web 应用程序部署在云计算系统中，因为它们支持无限的存储和计算能力。该云计算系统的主要后端存储组件之一是分布式文件系统，它允许存储和访问大量数据。在这些系统中部署的大多数 web 应用程序中，读取操作比写入操作执行得更频繁。A. Nalajala 等人（2022）关于缓存问题提出了一种基于客户端应用程序访问文件块的访问频率和访问近时性

排序的预取和多级缓存算法[9]。通过结合文件块的 Access-Frequency 和 Access Recency 排序，提出了新的增强排序算法用于预取文件块。使用基于排名的替换算法来替换缓存中的文件块。仿真结果表明，所提算法在分布式文件系统上的读操作性能提高了 29% ~ 77%。

存储在本地 ffb 列表和全局 ffb 列表中的每个 fid(bid)的 Access-Frequency rank(AF_rank)计算如式(1)所示:

$$AF\_rank(fid(bid)) = \frac{listsize - position(fid(bid))}{listsize} (1)$$

Access-Recency rank (AR_rank)使用式(2)计算:

$$AR\_rank(fid(Bid)) = \frac{logsize - position(fid(bid))}{logsize} (2)$$

算法 1 基于访问频率的排序过程

```
for each fid(bid) in DND_FFB_list do
  Assign a postion value from 0 to n-1 for all fid(bid)s in
  the list
end for
for each fid(bid) in DND_FFB_list do
  Calculate AF_rank(fid(bid)) //equation (1)
  Add fid(bid) to DND_FFB_Freq_ranklist
end for
for each fid(bid) in CCN_FFB_list do
  Assign a postion value from 0 to n-1 for all fid(bid)s in
  the list
end for
for each fid(bid) in CCN_FFB_list do
  calculate AF_rank(fid(bid)) //equation (1)
  Add fid(bid) to CCN_FFB_Freq_ranklist
end for
```

算法 2 基于访问频次的排序过程

```
for each fid(bid) in the DND_AR_list do
  Calculate AR_rank(fid(bid)) //equation (2)
  Add fid(bid) to DND_FFB_ARranklist
end for
for each fid(bid) in the CCN_AR_list do
  calculate AR_rank(fid(bid)) //equation (2)
  Add fid(bid) to CCN_FFB_ARranklist
end for
```

但是服务器上的运行时性能可变性一直是一个主要问题,它阻碍了现代分布式系统中可预测和可扩展的性能。在多个服务器上冗余地执行请求或作业已被证明可以有效地减轻可变性，无论是在理论上还是在实践中。采用冗余的系统引起了极大的关注，许多论文分析了在各种服务模型和运行时可变性假设下冗余的痛苦和收获。Mehmet Fatih Aktas（2019）介绍了通过使用复制或擦除编码冗余来执行许多任务的作业的成本(痛苦)与延迟(收益)分析[14]。服务时间变异性的尾重对冗余的痛苦和收获起决定性作用，通过推导成本和延迟的表达式来量化其影响。具体来说，他们的研究回答了四个问题:1)在成本与延迟权衡方面，复

制冗余和编码冗余是如何比较的? 2)我们能否等一段时间后引入冗余，并期望降低成本? 3)重新启动那些在一段时间后出现混乱的任务是否有助于降低成本和/或延迟? 4)裁员和重新启动一起使用是否有效？并通过使用从 Google 集群数据中提取的经验分布的模拟来验证每个问题的答案。

国外研究学者在针对分布式存储网络提高性能方面的研究在理论和实践上都取得了较为成熟的研究成果。提供了相应算法和实验结果证明，为本课题后续的研究提供很好的参照以及较为完备的理论准备。

2.3. 总结

国内外在关于分布式存储网络提高其性能的问题上取得了显著的成功，主要集中在通过预取和缓存技术的改进来提高读取操作效率从而提高 I/O 效率。然而关于选题领域的研究相对较少，深入研究选题领域，有望实现 CPU 存储 I/O 效率的进一步提升。具有重要的理论和应用价值，能够推动相关领域的进步和发展。

3. 主要参考文献和资料

[1] 阿里巴巴（中国）有限公司. 数据操作定序方法及系统、数据操作提交方法[P].中国,CN116737680A，20230912, 1.

[2] 陈游旻,陆游游,罗圣美,等.基于 RDMA 的分布式存储系统研究综述[J].计算机研究与发展, 2019, 56(2):13.DOI:10.7544/issn1000-1239.2019.20170849

[3] 何沅泽,王晓京,张景中.应用于分布式存储系统的网络编码方法研究[J].计算机应用,2013,33(A01):15-19

[4] 崔玉龙,付国,张岩峰等.Elsa:一种面向跨区域架构的无协调分布式键值存储系统[J].软件学报,2023,34(05):2427-2445.DOI:10.13328/j.cnki.jos.006437.

[5] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: cluster computing with working sets. In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing (HotCloud'10). USENIX Association, USA, 10.

[6] Franz Färber, Sang Kyun Cha, Jürgen Primsch, Christof Bornhövd, Stefan Sigg, and Wolfgang Lehner. 2012. SAP HANA database: data management for modern business applications. SIGMOD Rec. 40, 4 (December 2011), 45–51. https://doi.org/10.1145/2094114.2094126

[7] Stuedi P, Trivedi A, Metzler B, Pfefferle J. DaRPC: Data center RPC. In Proceedings of the 5th ACM Symposium on Cloud Computing, SOCC 2014. Association for Computing Machinery, Inc. 2014. (Proceedings of the 5th ACM Symposium on Cloud Computing, SOCC 2014). doi: 10.1145/2670979.2670994

[8] X. Zhang, B. Liu, Z. Gou, J. Shi and X. Zhao, "DCache: A Distributed Cache Mechanism for HDFS based on RDMA," 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Yanuca Island, Cuvu, Fiji, 2020, pp. 283-291, doi: 10.1109/HPCC-SmartCity-DSS50907.2020.00035.

[9] A.Nalajala, T. Ragunathan and R. Naha, "Efficient Prefetching and Client-Side Caching Algorithms for Improving the Performance of Read Operations in Distributed File Systems," in IEEE Access, vol. 10, pp. 126232-126252, 2022, doi: 10.1109/ACCESS.2022.3221117.

[10]Assaf, Maen M. Al, Xunfei Jiang, Xiao Qin, Mohamed Riduan Abid, Meikang Qiu

and Jifu Zhang. “Informed Prefetching for Distributed Multi-Level Storage Systems.” Journal of Signal Processing Systems 90 (2018): 619-640.

[11]Griffioen, J., & Appleton, R. (1994). Reducing file system latency using a predictive approach. In Proceedings of the 1994 USENIX annual technical conference (pp. 197–207). Berkeley, CA, USA.

[12]Patterson, R.H., Gibson, G., Stodolsky, D., & Zelenka, J. (1995). Informed prefetching and caching. In Proceedings of the 15th ACM symposium on operating system principles (pp. 79–95). CO, USA.

[13]J. Liao and S. Chen, "Optimization of Reading Data via Classified Block Access Patterns in File Systems," in IEEE Access, vol. 4, pp. 9421-9427, 2016, doi: 10.1109/ACCESS.2016.2642918.

[14]M. F. Aktaş and E. Soljanin, "Straggler Mitigation at Scale," in IEEE/ACM Transactions on Networking, vol. 27, no. 6, pp. 2266-2279, Dec. 2019, doi: 10.1109/TNET.2019.2946464.