奥德赛：分布式数据序列相似性搜索领域的探索之旅

预印本

Manos Chatzakis洛桑联邦理工学院 emmanouil.chatzakis@epfl.ch *

Panagiota FatourouFORTH、ICS 和克里特大学，CSD faturu@ics.forth.gr †

Eleftherios KosmasFORTH、ICS、希腊地中海大学和克里特大学，CSD ekosmas@csd.uoc.gr

Themis Palpanas
Université Paris Cité & IU
themis@mi.parisdescartes.fr

Botao Peng中国科学院计算技术研究所
pengbotao@ict.ac.cn

arXiv:2301.11049v1 [cs.DC] 26 Jan 2023

摘要

本文介绍了 Odyssey，这是一种新颖的分布式数据序列处理框架。该框架充分利用由多核服务器组成的现代分布式系统的全部计算能力，高效应对数据序列处理中实现良好加速比并确保高可扩展性的关键挑战。Odyssey 解决了设计高效且高度可扩展的分布式数据序列索引时面临的诸多挑战，包括高效调度和负载均衡，同时避免付出搬移数据的高昂代价。它还支持灵活的部分复制方案，使 Odyssey 能够在数据可扩展性与查询应答的良好性能之间实现基本权衡。通过采用多种配置并使用多个真实和合成数据集进行实验分析，我们证明 Odyssey 实现了这些颇具挑战性的目标。

本文发表于 PVLDB 2023 年第 16 卷。

1 引言

研究动机。 如今，高效处理大规模现实世界数据序列集合，已成为金融、天体物理学、神经科学、工程学等众多不同应用领域中最具挑战性且至关重要的问题之一 Palpanas [2015], Zoumpatianos and Palpanas [2018], Palpanas and Beckmann [48(3]。这类应用会产生大量由数据点构成的有序序列，称为数据序列。生成数据序列集合后，需要对其进行分析，以提取有用知识 Kashino et al. [1999],Ye and Keogh [2009],Huijse et al. [2014], Raza et al. [2015], Boniol et al. [2020], Boniol and Palpanas [2020],Linardi et al. [2020], Wang et al. [2023a]。此类分析通常包括回答相似性搜索 查询 Palpanas [2015], Echihabi et al.[2018, 2019], , 而这些查询对各种下游分析任务都很有用 Echihabi et al. [2020, 2021a,b]。此外，不同领域的多种应用都对结果的准确性十分敏感 Palpanas and Beckmann [48(3], Bagnall et al. [9(7], , 因此要求实现精确查询回答 Echihabi et al. [2018], , 这正是本文关注的重点。

*与 ICS-FORTH 和 Université Paris Cité 合作开展的工作

†部分工作是在 P. Fatourou 于法国的 Université Paris Cité 担任 Marie Sklodowska-Curie 个人研究员期间完成的（PLATON 项目，GA 编号 101031688）。

随着数据序列集合规模不断扩大 Palpanas [2015, 2017], Palpanas and Beckmann [48(3], , 近期提出的最先进的数据序列索引通过使用多个线程以及利用现代硬件的单指令多数据能力来实现并行处理 Peng et al. [2018, 2020a], Echihabi et al. [2022]。然而，如今数据序列集合规模前所未有的增长，使得即使是最先进的并行数据序列索引也难以胜任 Palpanas [2017], Echihabi et al. [2018, 2019], Palpanas and Beckmann [48(3], Bagnall et al. [9(7], Gogolou et al. [2019], Echihabi et al. [2023],，这主要是因为精确查询回答需要大量随机磁盘页读取 Echihabi et al. [2018]。为解决这些问题，研究者提出了快速的内存解决方案 Peng et al. [2020b, 2021a,b]。然而，这些解决方案未能利用分布式系统，因此受限于单台机器的内存容量。这正是我们要解决的限制，从而使上述最先进的解决方案能够处理远超任何单个节点主存容量的数据集。

挑战。 在数据序列相似性搜索中，即使使用数据序列索引，精确查询回答也非常消耗资源。我们必须对索引进行剪枝，或者访问其中的每个叶节点。然而，Echihabi 等人 [2018], Gogolou 等人 [2019]以往的研究表明，剪枝的效果并不理想，尤其是在某些困难数据集上。

我们需要实现的主要目标（自然）是可扩展性。也就是说，增加可用的硬件资源（例如节点数）应能降低时间成本，理想情况下，降幅与资源增幅相当；或者能以大致相同的时间成本处理相应增加的数据量。为实现这一目标，我们需要确保在整个执行过程中，分布式系统的所有节点都能为完成工作做出同等贡献。为此，我们需要针对以下两个问题提出有效的解决方案：(i) 查询调度：给定查询工作负载，决定将哪些查询分配给各个系统节点；以及 (ii) 负载均衡：设计相应机制，使已完成工作的系统节点能够帮助其他系统节点完成工作。

这一背景下的挑战如下。首先，要实现有效的查询调度，我们需要设计能够估算数据序列相似性搜索查询执行成本的机制，而目前尚无此类机制。其次，这一观察表明负载均衡方案不可或缺；但这也意味着，为了使此类机制切实可行，我们需要复制数据，因为大规模移动数据序列的成本高得令人难以承受。数据复制不利于数据可扩展性，并且无论从哪方面看，都会增加索引创建时间的成本，但它能缩短查询应答时间，因此会带来有趣的权衡，而有效的解决方案应妥善应对这些权衡。第三，除了上述所有考量之外，我们还需要确保解决方案在每个系统节点内部的多核 CPU 上高效执行时，仍能保持良好的并行化特性，并在查询应答过程中实现较强的剪枝能力。

我们的方法。我们提出了一种名为*Odyssey*的新型分布式数据序列（DS）索引与处理框架。该框架充分利用计算平台的全部计算能力，从而高效实现高度可扩展这一目标。

为 Odyssey 制定合适的调度方案时，我们进行了查询分析，发现总执行时间与所考虑的单节点数据序列索引类别的某个参数之间存在相关性。这项分析推动了高效调度方案的设计：为输入查询批次中的每个查询生成执行时间预测。

即使在预测可能不准确的情况下，Odyssey 也能实现负载均衡（LB）。它提供了一种负载均衡机制，确保空闲节点可以从仍有工作要处理的其他节点那里接手（或窃取）任务（前提是这些节点存储着相似的数据）。将 Odyssey 调度器与这种负载均衡技术相结合，可使我们测试的所有查询批次都获得出色的性能和很高的可扩展性。

确保数据可扩展性，同时保证查询应答的良好性能，是相互矛盾的目标。不进行数据复制的方案可将空间开销降至最低，但实验表明，这种方法无法确保查询应答期间的最佳性能，因为不进行数据复制意味着无法使用 Odyssey 的负载均衡机制。

Odyssey 通过支持灵活的 部分复制方案，成功有效地统一了这两个相互矛盾的目标。这样，它便能在数据可扩展性与查询应答期间的良好性能之间的根本性权衡中游刃有余。复制度是 Odyssey 的一个参数。通过适当指定该参数，用户可以选择最适合其应用和设置的时间-空间权衡。实验表明，即使复制度较低，Odyssey 也能实现良好的性能。

在设计 Odyssey 时，我们还承担了一项更具挑战性的任务：在充分利用单个节点计算能力的索引之上，构建 Odyssey 所提供的高效分布式计算组件，使其性能不逊于最先进的并行索引 Peng 等人 [2020b, 2021a年,b], 。在每个节点上使用一个最先进的 MESSI 索引实例 Peng 等人 [2021a] 这一简单方法，未能取得良好的性能，主要有两个原因。首先，

不同的数据序列查询可能表现出不同程度的局部性（只有在运行时才能发现），导致某些节点的剪枝效果较差，进而造成严重的负载均衡问题和性能下降。其次，在这种简单方法之上实现负载均衡，需要移动数据，而这往往代价高昂，难以承受。Odyssey 的单节点索引方案借鉴了最先进索引 Peng 等人 [2018,2020a年,b, 2021a年,b], 中的一些技术，并将这些技术与新的组件和机制结合起来，从而实现负载均衡，并提出一种方案，使过载节点的工作能够转交给空闲节点，而无需承担移动任何数据的高昂代价。

Odyssey 在多个方面具有创新性。首先，在遍历索引树以生成无法剪枝的数据序列集合时，它采用了与所有现有方法不同的并行模式。其次，它提出了新的实现方法，用于填充和处理高效查询应答所需的数据结构。为实现线程间的负载均衡，选择这些数据结构规模的合适阈值至关重要，而 Odyssey 提出了一种有效机制来预测合适的阈值。此外，Odyssey 提供了高效的通信和记账机制，使节点之间能够快速交换信息，从而确保所有节点都能达到良好的剪枝效果。

Odyssey 的速度最高可达竞争方法的 6。6 倍，性能比其最佳竞争方法高出 3。5 倍以上。此外，Odyssey 的索引创建能够随着数据集大小和节点数量的增加而实现完美扩展。而且，Odyssey 性能最佳的调度策略比其初始调度策略快 2。5 倍以上。

贡献。 本文的主要贡献如下：

• 我们介绍了 Odyssey，这是一个可扩展的框架，支持在配备多核服务器的集群中进行分布式数据序列相似性搜索。这使我们的方法成为首个能够同时利用系统节点内部和节点之间并行化的定制化数据序列解决方案。

• 我们设计了一种调度算法，用于将查询分配给集群中的节点。该算法通过计算每个查询执行时间的（足够准确的）估计值，尽可能均衡各节点的工作负载。

• 我们提出了一种新颖的精确搜索算法，支持共享同一索引（完全复制）的节点之间进行工作窃取。因此，即使工作未能（或无法）在集群各节点之间均匀分配，我们的方法仍能实现高性能。我们还进一步扩展了该方案，使其在节点之间仅共享部分索引（部分复制）时也能运行。

• 我们的方法支持节点之间采用不同的复制度，使用户能够探索整个解决方案范围，在空间（复制度）与速度（查询应答时间）之间进行权衡。

• 我们还提出了一种密度感知型数据分区方法，能够高效地对数据进行分区，从而提升工作负载均衡能力。

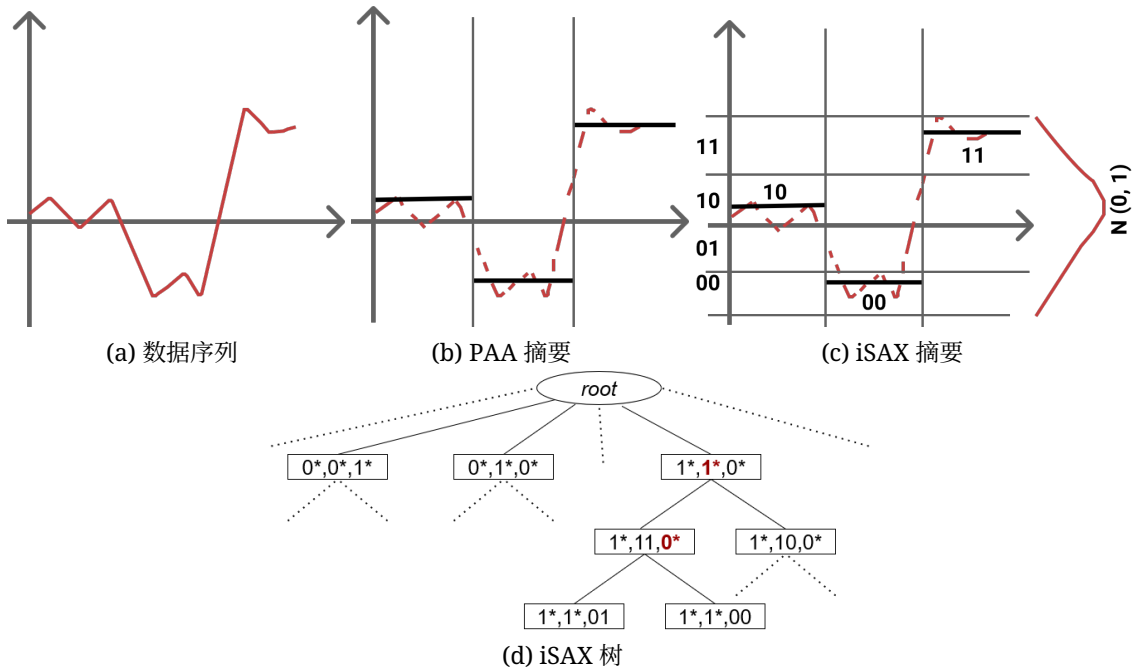
• 最后，我们使用大量配置和真实及合成数据集开展了实验评估（代码和数据可在线获取 cod [2022]）。评估结果表明，Odyssey 效率出色，几乎可实现线性扩展，精确查询应答时间比竞争方法最多快 6.6 倍。

2 预备知识与相关工作

数据序列。一个数据序列记作 $S = \{p_1, ..., p_n\}$，是由点构成的序列，其中每个点 p_i 都是一个由实数值 u_i 和 t_i 在序列中的位置组成的二元组（u_i，t_i） $1 \leq i \leq n$； n 是该序列的大小（或维度）。当 t_i 表示时间时，我们称之为时间序列。在若干情况下，我们会省略 t_i，例如当这些点等间距分布，或仅作为数值 u_i 的索引时。Echihabi 等人 [2018]；为简便起见，我们也会省略它们。

iSAX 摘要。 数据序列的*iSAX* 摘要（Shieh 和 Keogh [2008] ）将 x 轴划分为等长区段，并用数据序列中各区段所包含数据点的均值表示每个区段（见图 1）。随后，它按照正态分布确定的区域大小将 y 轴划分为若干区域，并用一定数量的比特表示每个区域（基数）。各区域所用的比特数可以不同，由此便能构建分层索引树（基于 *iSAX* 的索引树，Palpanas [2020]；见图 1）。

相似性搜索。 给定一个数据序列集合 $\mathcal{C}$ 和一个输入数据序列 S（称为查询），相似性搜索的任务是找出 $\mathcal{C}$ 中与 S 最相似的数据序列。我们重点关注寻找单个最佳答案，即 *1-NN* 问题。我们也聚焦于欧几里得距离（ ED ）。两个时间序列 $T = \{t_1, ..., t_n\}$ 和 $S = \{s_1, ..., s_n\}$ 之间的欧几里得距离（或实际距离）定义为 $ED(T, S) = \sqrt{\sum_{i=1}^{n}(t_i - s_i)^2}$。我们将 T 和 S的*iSAX* 摘要之间的距离称为下界距离。任意两个数据序列之间的下界距离总是小于或等于它们之间的实际距离。



(a) 数据序列 (b) PAA 摘要 (c) iSAX 摘要

(d) iSAX 树

图 1：从数据序列到 iSAX 索引

单节点并行摘要式数据序列索引。 此类索引 Peng 等人 [2018, 2020a,b, 2021a,b], Echihabi 等人 [2022], Azizi 等人 [2023] 利用多个线程（以及单指令多数据）来构建索引树，并基于该树应答查询。它们通常由两个主要阶段组成：索引树构建 阶段和查询应答阶段。在索引树构建阶段，它们首先并行计算集合中所有数据序列的摘要。如果这些摘要是 iSAX 摘要，我们就称之为基于 *iSAX* 的数据序列索引。为了实现良好的局部性并降低同步开销，它们会将这些摘要存入一组摘要缓冲区中。摘要相似的数据序列会被放入同一个缓冲区。随后，每个缓冲区中的数据序列会被存入它们所构建的索引树的相应子树中。这些设计决策使它们能够以近乎完全并行的方式构建索引树（因此不会产生同步开销），并在树构建期间实现数据访问的局部性。因此，它们遵循了实现良好性能的关键原则；设计并行索引时也应遵循这些原则。

为回答查询，这些索引首先计算查询的摘要。随后，它们遍历索引树，根据 iSAX 摘要下界距离查找最合适的数据序列。这些数据序列与查询序列之间的距离存储在一个名为当前最优值（BSF）的变量中，作为当前查询的初始近似答案。然后，使用 BSF 对初始集合中的数据序列进行剪枝。当数据序列 S 与查询 S 之间的下界距离高于 BSF 的当前值时，就会将该数据序列剪枝。此过程会从初始数据序列集合中筛选出一个尽可能小的子集，其中包含需要进一步检查的序列。这些序列通常存储在一个或多个优先队列中（Peng 等人 [2020b，2021a年，b]）。多个线程并发处理优先队列中的元素，计算实际距离（如有需要），并在每次发现新的最小值时更新 BSF（见图2）。此过程完成后，答案的距离便保存在 BSF 中。

多节点系统与查询处理。 系统由多个异步节点组成，它们通过交换消息进行通信。每个节点都是一台多核机器，能够支持多个线程（也可能支持单指令多数据计算）。线程通过访问共享变量进行通信。共享变量可以被原子地读取和写入。系统还可能提供更强的原语，例如取值并递增。取值并递增(V， val)会将值 val 原子地加到变量 V 的当前值上，并返回此次更新前 V 的值。

系统接收一个规模任意大的查询批次作为输入。目标是利用系统的计算能力执行这些查询，并使总完成时间最小化；也就是说，从任一节点开始处理该批次中的某个查询，到所有节点完成计算的最早时刻之间所经过的时间。我们的方法很容易调整，也适用于动态到达系统的查询。

初始集合中的数据序列可以存储在所有节点上（完全复制），也可以分散到不同节点，使各节点存储数据的不相交子集（无复制）。也可以采用部分复制 方案，

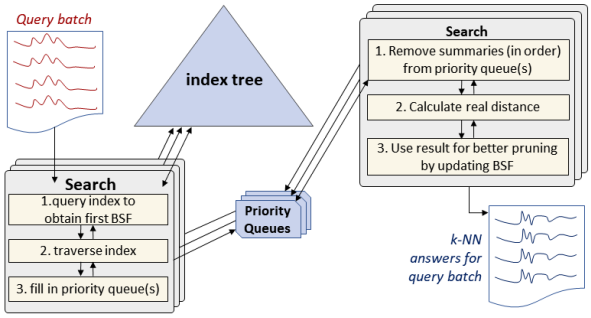


图2：并行数据序列索引的算法概述。

在这种方案中，各节点存储的数据子集不一定两两不相交（例如，多个节点可能存储相同的数据序列子集）。数据分区 机制决定如何拆分初始数据序列集合中的数据并将其分发到各节点。

查询调度算法旨在将输入查询分配给各个节点，使每个节点承担的工作量大致相同。对于完全复制，静态查询调度器（*SQS*）会将查询序列划分为 N 个子序列，每个节点获取其中一个子序列并负责回答其中的查询。动态查询调度器（*DQS*）使用一个协调器节点，其他节点则向协调器请求要执行的查询。协调器收到请求后，可以通过将下一个尚未处理的查询分配给工作节点来响应；也可以先对查询序列进行预处理（例如，根据某种属性重新排列查询），再开始向各节点分配查询。为避免浪费计算能力，协调器可以在响应其他节点的请求期间自行回答查询。

2.1 相关工作

要在规模极大的数据序列集合上快速执行数据序列相似性搜索查询，必须使用专门的索引结构。通常，数据序列索引会根据序列的摘要及相应的下界剪枝搜索空间，并且仅在过滤假阳性时使用序列的原始数据。

数据序列索引。 Agrawal 等人 Agrawal 等人 [1993] 首次提出使用空间索引结构为数据序列建立索引的方案，该方案基于 R树 Sellis 等人 [1987], ，之后由 Rafiei 和 Mendelzon [1998]进行了优化。文献中还提出了各种专用于数据序列的索引 Echihabi 等人[2021a]。DSTree Wang 等人 [2013] 是一种基于自适应分段常数近似（APCA）摘要的索引 Keogh 等人 [2001]。DSTree 可以根据需要增加 APCA 的细节，自适应地执行分裂操作。iSAX 索引基于符号聚合近似（SAX）摘要及其扩展 iSAX Shieh 和 Keogh [2008]。在这种情况下，数据序列摘要采用按位表示，从而得到简洁的表示和整体索引。文献中还提出了其他几种基于 iSAX 的索引 Camerra 等人 [2014], Zoumpatianos 等人 [2014, 2016], Linardi 和 Palpanas [2019, 2020], Palpanas [2020],Wang 和 Palpanas [2021], Wang 等人 [2023b]。这些索引是该领域最先进的解决方案之一 Echihabi 等人[2018], ，其中包括 MESSI Peng 等人 [2020b、2021a年], ，这是一种内存中、多核且支持 SIMD 的 iSAX 索引版本。

数据序列管理系统。 近年来，已经开发出多种数据序列管理系统 Jensen 等人 [2017], Echihabi 等人 [2021a].Beringei Pelkonen 等人 [2015] 采用定制的内存存储引擎。它按照每个序列单独处理的方案压缩和组织数据。CrateDB Crate [2018] 将数据划分为数据块，存储在分布式文件系统中，并使用 Apache Lucene 为其建立索引。InfluxDB InfluxDB [2018]使用时间结构合并树（LSM 树变体）。Prometheus Prometheus [2018] 基于 Beringei 的理念。QuasarDB QuasarDB [2018] 使用 RocksDB 或 Hellium Hellium [2018]。Riak TS RiakTS [2018]同时支持 LevelDB 和 Bitcask，后者是一种定制的日志结构哈希表。Timescale Timescale [2018] 是一个 Postgres 扩展。IoTDB Wang 等人 [2020] 专为流式数据序列设计。最后，OpenTSDB OpenTSDB [2015],、Timely Timely [2018] （专注于安全性）和 Warp10 Warp10 [2018] 等多种系统都是基于 HBase 开发的。上述所有系统都支持对指定位置进行范围扫描、聚合函数

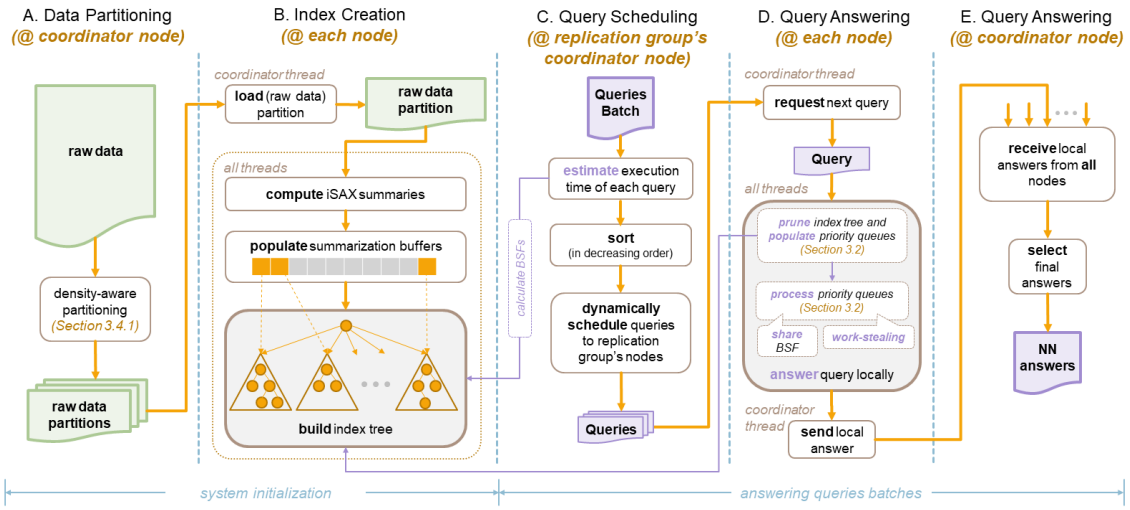


图3：Odyssey 流程图。

和筛选。InfluxDB 支持移动平均值、预测、转换等查询，Timescale 则支持间隙填充。尽管如此，上述系统都不支持精确的整体匹配相似性搜索查询。

分布式数据序列索引。 KV-Match Wu等人 [2019] 及其改进版 L-Match Feng等人 [2020], 是能够支持相似性搜索的索引结构。这些索引可以构建在 Apache HBase 之上，并在 Apache Spark 中以分布式方式运行。需要指出的是，这些解决方案仅支持子序列相似性搜索，不支持整序列匹配，而整序列匹配正是本文的研究重点（Echihabi 等人 [2018], ）。TARDIS Zhang等人 [2019]是一个用于相似性搜索的 Apache Spark 系统。它支持近似查询，也支持精确匹配查询，即判断查询内容是否与数据集中的内容完全相同。这种查询比我们研究的精确查询容易得多，而且无法高效地转换为精确查询。最后，DPiSAX Yagoubi 等人 [2017, 2020] 是一个使用 Scala 开发、基于 Apache Spark 的分布式数据序列相似性搜索解决方案。它旨在应答批量近似搜索查询，同时也支持精确搜索。DPiSAX 利用数据集一个小型样本的 iSAX 摘要，将数据平均分发到各节点。随后，在每个节点上针对本地数据构建 iSAX 索引，并用其进行查询应答。为生成精确搜索结果，所有节点都需要将各自的部分结果发送给协调器，由协调器合并这些结果并生成最终的精确答案。需要注意的是，DPiSAX 并非专为节点内并行化而设计，但它是文献中唯一支持精确搜索的分布式数据序列索引。

Cilk框架中采用了工作窃取方法 Cil [1996]。Blumofe 和 Leiserson [1999], Fatourou和Spirakis [1999, 2000]对工作窃取方法进行了形式化研究与分析。关于这一主题已经开展了大量工作（例如，Blelloch等人 [1999, 1997],Fatourou [2001]）。

3 Odyssey框架

我们首先介绍 Odyssey流程图的高层概览，该流程图由五个阶段组成（见图3） .

在第一阶段，协调器节点 将原始数据序列集合划分为与系统节点数量相同的数据块，并为每个节点（包括其自身）分配一个数据块。（第 3.4 节详细介绍了 Odyssey 的分区方案。）在第二阶段，每个节点 (i) 将其数据块加载到内存中，(ii) 计算其 iSAX 摘要并将其存储到若干个摘要缓冲区中，以实现局部性，以及 (iii) 构建其索引树。为提升查询应答性能，Odyssey 采用数据复制机制。它将节点划分为若干组（复制组，详见第 3.3 节），每组中的所有节点都存储相同的数据块。每个复制组都有一个协调器节点，称为组协调器，负责向组内节点调度查询。待执行的查询批次（例如，源自*k-NN* 分类任务）会提交给所有组协调器（因为不同组存储着不同的数据块）。在第三阶段，组协调器首先估计每个查询的执行时间，然后按估计执行时间降序排列查询，并动态 地将它们调度到组内节点（第 3.1 节介绍了查询调度）。在第四阶段，每个节点处理分配给它的查询。它首先计算一个初始 BSF，然后使用该 BSF 剪枝索引树，并将无法剪枝的叶节点放入优先队列。最后，它处理优先队列中的元素，以找到最佳局部答案（对应于其数据

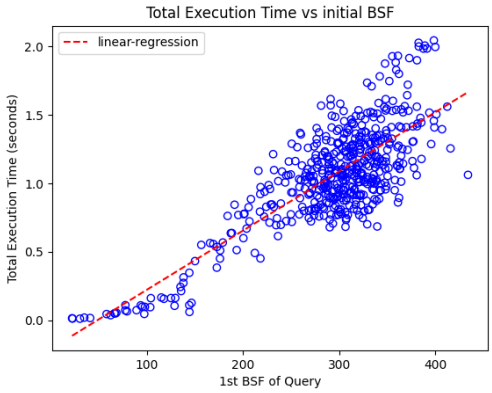


图4：Seismic 查询预测的线性回归。

块）。在此阶段，Odyssey 支持*BSF* 共享 和工作窃取 （详见第 3.2 节）。在最后阶段，协调器节点从各组协调器收集局部答案，并生成最终答案。

3.1 查询调度

为了正确回答一个查询，应将其转发至至少一个系统节点集合，这些节点共同存储全部数据。我们在第 3.3 节将这样的集合称为节点 集群。因此，在无复制的情况下，该集合包含所有系统节点，所以调度算法应将所有查询转发至所有节点。其他复制配置（尤其是 FULL 复制）则更有意义，因为它们支持采用不同的调度技术。

为了设计 Odyssey 调度器，我们尝试了一系列调度技术，包括第 2 节讨论的、适用于 FULL 复制配置的简单 STATIC 和动态方案（SQS 和 DQS）。遗憾的是，对于许多类别的查询批次，这些方案都会产生严重的负载不均衡问题。以静态情况为例，考虑一个由逐渐变得更加困难的查询组成的查询序列（即，每个查询的运行时间都比下一个查询短）。SQS 会将简单查询分配给前面的系统节点，而最后面的节点则需要处理更多工作。动态方法（DQS）同样可能导致负载不均衡：即使在一些简单情况下，例如查询批次的末尾包含一个困难查询，大多数节点也可能处于空闲状态，而单个节点正在运行该困难查询。这可能会显著降低性能。

如果我们知道每个查询的执行时间，就可以避免其中一些负载不均衡。Gogolou 等人 [2020], Echihabi 等人 [2023] 近期的研究表明，初始 BSF 与单节点索引树中访问的顶点数量之间存在相关性。我们进行了相应的查询分析，结果显示，初始 *BSF* 较高的相似性搜索查询，其执行时间往往也较长。在本研究中，我们使用线性回归模型（也可以采用其他预测方案）来估算每个查询的执行时间。图4展示了一个示例（针对 Seismic；我们对其他数据集也采用相同的流程）。

这些观察促使我们设计了两种调度算法。第一种是基于静态预测的调度，根据查询的估算执行时间，将查询静态分配给各节点。每个节点都维护一个负载 变量，用于存储分配给该节点的查询的执行时间估算值之和。该算法采用贪心方法分配查询，以实现负载均衡。该算法有两种变体：第一种（未排序）按照查询在序列中的顺序进行调度；第二种（已排序）则根据执行时间估算值按降序排列查询序列。。第二种调度算法称为动态预测调度，是 DQS 的增强版本：它根据估算值（按降序排列）对整个查询批次排序后，再将查询分配给各节点。

考虑一个由两个节点 sn_1 和 sn_2，组成的系统，并设 $\mathcal{Q} = \{q_1, q_2, q_3, q_4, q_5\}$ 为待执行的查询批次。假设 $\mathcal{ES} = \{100, 50, 200, 250, 80\}$ 是估计执行时间的集合，其中 $\mathcal{ES}$ 的第 i 个元素是 q_i, $1 \leq i \leq 5$ 的估计执行时间。使用负载变量 l_1 和 l_2 （分别对应 sn_1 和 sn_2，）的未排序静态预测调度过程如下：将 q_1 分配给 sn_1 （因此， $l_1 = 100$），并将 q_2 分配给 sn_2 （因此， $l_2 = 50$）。由于 $l_2 < l_1$， q_3 被分配给 sn_2 （因此， $l_2 = 250$）。采用类似策略，将 q_4 分配给 sn_1，，将 q_5 分配给 sn_2。因此， sn_1 接收 $\{q_1, q_4\}$，而 sn_2 接收 $\{q_2, q_3, q_5\}$。在排序静态预测调度中，首先按估计时间降序对 $\mathcal{Q}$ 中的查询进行排序，得到 $Q' = \{q_4, q_3, q_1, q_5, q_2\}$ （对应于 $\mathcal{ES}' = \{250, 200, 100, 80, 50\}$）。在这些集合上应用静态预测调度算法（如

上文所述）后，将 $\{q_4, q_5\}$ 分配给 sn_1，将 $\{q_3, q_1, q_2\}$ 分配给 sn_2。最后，动态预测调度也会对 Q中的查询进行排序。在这种情况下，将 q_4 分配给节点$_1$，将 q_3 分配给节点$_2$,，其余查询则在收到请求时按顺序动态分配给节点（因此依据实际执行时间进行分配）。
Odyssey 框架支持上述所有调度算法。事实证明，在大多数情况下，Odyssey 索引采用的动态预测调度是最佳方案。

3.2 负载均衡

Odyssey 提供了一种负载均衡（LB）机制，可应用于第 3.1 节所述的任何调度方案之上。具体而言，空闲节点可以窃取其他仍有待处理工作的节点上的工作（前提是它们存储了相似的数据）。

这是必要的，因为预测未必总是准确，或者查询批次可能在运行时动态生成，在这种情况下，无法对整个查询批次进行排序。实现高可扩展性也需要这一机制。随着所用节点数量增加，每个节点需要处理的批量查询数量会越来越少。因此，可能会出现第 3.1 节所述的问题场景：只有一个或少数几个节点处理困难的查询，而其他节点则处于空闲状态。
我们方法的概述。我们进行了一系列实验，以分析查询应答时间的构成。分析结果表明，查询应答的大部分时间都花在了优先队列的处理上。因此，我们着重设计一种方法，使节点能够在执行这一阶段时窃取工作。为简单起见，我们首先关注全复制情形，即每个节点都拥有初始数据集合；随后将在第 3.3 节讨论部分复制。

Blumofe 和 Leiserson [1999], Cil [1996] 提出的简单工作窃取方案并不适用，主要是因为在节点之间移动数据（存储在优先队列中）的代价很高，应当避免。因此，在我们的场景中，主要挑战在于如何将工作从一个节点转交给另一个节点，同时完全不移动任何数据。

Odyssey 的负载均衡机制如下所示。一个空闲的系统节点 sn 会随机选择另一个节点 sn'，并向其发送窃取请求。如果 sn' 仍有工作要做，它就会选择若干个优先队列交给 sn。为了避免承担传输数据的开销，Odyssey 采用了一种技术，告知 sn 如何根据自己的索引在本地构建要处理的优先队列。节点 sn 遍历索引树中指定的部分，并重新构建这些优先队列。由于创建优先队列所需的时间相较于处理它们的时间要短得多，这种方案效果相当不错。

请注意，现有最先进的索引所采用的创建和处理优先队列的方法（Peng 等人 [2020b，2021a年，b] ）过于简单，无法在不移动任何数据的情况下支持工作窃取。在 Odyssey 中，我们提出（见第 3.2.1 节）一种新的单节点多线程索引实现。该实现遵循第 2 节中为并行索引描述的良好设计原则，同时也解决了上述问题。

3.2.1 单节点查询应答

考虑任意一个系统节点 sn，并假设已创建一棵基于 iSAX 的索引树，且已在 sn 中计算出 BSF 的初始值。图 5 展示了 Odyssey 的单节点查询应答算法概要。伪代码见算法 1 和算法 2。

描述。 节点 sn 依次执行分配给它的查询批次中的每个查询（算法 1）。对于每个查询 Q，它都会创建若干搜索工作线程来执行该查询（第 8 行）。当 sn 的查询批次中的所有查询都处理完毕后， sn 会通知其他节点它已完成（第 12 行）。随后，它会尝试通过执行 PerformWorkStealing 来帮助其他活动节点（第 13 行）。每个节点都会分配一个线程，充当工作窃取管理器（第 6 行）。该线程只负责处理该节点收到的所有工作窃取请求（算法 3）。（工作窃取将在第 3.2.2 节中讨论。）
sn 中的查询回答算法会将树划分为根子树（*RS*）批次，即由连续的根子树组成的集合（见图 5），并分配若干线程处理这些批次。每个线程首先使用取值并递增（算法 2）获取一个要处理的 RS 批次。随后，该线程执行批次处理 例程，该例程递归遍历树，并将无法剪枝的叶节点插入属于该 RS 批次的一组优先队列之一。

对于每个 RS 批次，在任意时刻都存在一个活动优先队列。当该优先队列的大小超过阈值时，就会弃用该队列，并为该 RS 批次初始化另一个队列。

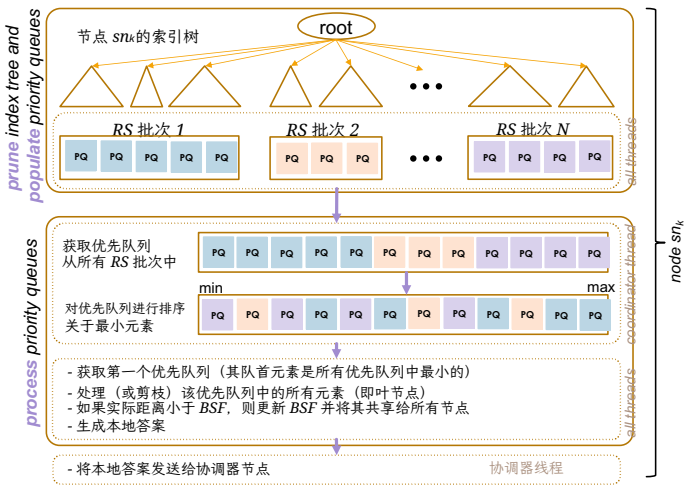


图 5：Odyssey 单节点查询应答流程概述。

算法 1： Odyssey 单节点查询应答——节点 sn的代码

1 **共享变量：共享指针数组** $PQueues = NULL$
输入：查询序列批次 $QBatch$，**索引** $Index$，**整数** $NThreads$
2 **个数组** $BSFArray[]$ ▷，大小为 $|QBatch|$
3 **对于**每个查询序列 *ID* Q，在 $QBatch$ **执行**
4 $iSAX_Q$ = 计算 Q的 iSAX 摘要
5 BSF =近似搜索($iSAX_Q$, $Index$)
6 创建一个线程来执行 WorkStealingManager(Q) 的实例
7 **对于** $i \leftarrow 0$ **到** $NThreads - 1$，**执行**
8 创建一个线程来执行 SearchWorker(Q, $Index$, N_{sb}, i, $PQueues$) 的实例
9 等待所有线程完成
10 $FinishFlag[Q]$:= 真
11 BSF 数组[Q] := BSF;
12 向所有节点发送(DONE, sn)
13 PerformWorkStealing()
14 **返回** ($BSFArray$)

空闲线程 th 一旦发现所有 RS 批次都已分配进行处理，就会尝试帮助其他仍处于活动状态的线程 th'完成其所分配 RS 批次的处理（算法 2，第 11–14 行）。为降低同步开销，每个 RS 批次可获得帮助的线程数设有一个阈值 帮助阈值（第 12 行）。当所有 RS 批次的子树都已遍历且所有优先队列都已填充后，此阶段结束。实验表明，当 RS 批次数量 N_{sb} 等于工作线程数时，性能最佳。

树遍历阶段结束后，我们会为每个 RS 批次获得一个优先队列，并将它们存储在一个数组中。出于性能考虑，该数组按照每个优先队列顶端元素的优先级排序。这就是优先队列预处理阶段（第 15-21 行）。这样，算法会优先处理查询下界距离最小的优先队列。这些队列中包含的数据序列与查询的实际距离更近的可能性更大，因此能够进一步进行剪枝。

接着，开始优先队列处理阶段（第 23-29 行）。每个线程都从 PQueues 数组中取出一个优先队列进行处理（使用取值并递增）。例程处理优先队列会处理存储在优先队列中且无法剪枝的数据序列。每当计算出这些序列中任意一个与查询序列之间更小的实际时间距离时，就会更新 BSF，使其包含该距离。这个改进后的 BSF 会提交给系统的所有节点。最后，所有答案都会传送到协调器节点，BSF 的全局最小值就是对查询的响应。

算法 2： SearchWorker - 线程 tid▷的代码

1 共享变量2 **共享整数** $BCnt = 0, PQCnt = 0, TotPQ = 0$;

输入：查询序列 Q，索引 $Index$，整数 N_{sb}，整数 tid，队列 $PQueues$[]3 整数 $bindex, pqindex$； 4 ▷树遍历阶段 5 while ($TRUE$) do6 $bindex \leftarrow$取值并递增($BCnt$,1); 7 if $bindex \geq N_{sb}$ then 8 break; 9 $ProcessRSBatch$ ($Q, bindex, Index.RSBatches$); 10 $Index.RSBatches[bindex].complete \leftarrow TRUE$; 11 for $bindex \leftarrow 0$ to N_{sb} do 12 if !$Index.RSBatches[bindex].complete$ AND 取值并递增($Index.RSBatches[bindex].helped$, 1) $< HelpTH$ then 13 $ProcessBatch(Q, bindex, Index.RSBatches)$;
14 $Index.RSBatches[bindex].complete \leftarrow TRUE$; 15 所有线程屏障； 16 ▷优先队列预处理阶段 17 if $tid == 0$ then 18 遍历所有 RS 批次，并将它们的优先队列放入 $PQueues$[];19 $SortByRootPriority(PQueues)$;
20 $TotPQ \leftarrow PQueues$的有效元素数量； 21 所有线程屏障； 22 ▷优先队列处理阶段 23 while ($TRUE$) do
24 $pqindex \leftarrow$取值并递增($PQCnt$, 1); 25 if $pqindex \geq TotPQ$ then 26 break; 27 if $PQueues[pqindex].stolen$ then 28 continue;29 $ProcessPriorityQueue(PQueues[pqindex])$;

算法 3： 工作窃取管理器——节点 sn的代码

输入：整数 N_B

1 **收到来自节点** sn'**的 StealingRequest 类型消息时：**
2 S：= 满足取走属性的 RS 批次 ID 集合，至多包含 N_{send} 个
3 将(S、 Q个 sn、 Q的当前 BSF)发送给 sn'
4 将 ID 在 S 中的 RS 批次的优先队列标记为已窃取

5 ▷始终启用的事件：重复执行
6 **未收到消息时：**
7 **如果** $FinishFlag[Q]$ 中设置了 sn **则**
8 终止

优先队列的大小。 每个优先队列的大小不得超过特定阈值 TH。如果向优先队列添加元素后，队列大小达到 TH，则线程会放弃该优先队列，并为 RS 批次新建一个队列。这样，每个优先队列只包含来自一个 RS 批次的叶节点，并且最多包含与该 RS 批次对应的树部分中的TH 个叶节点。

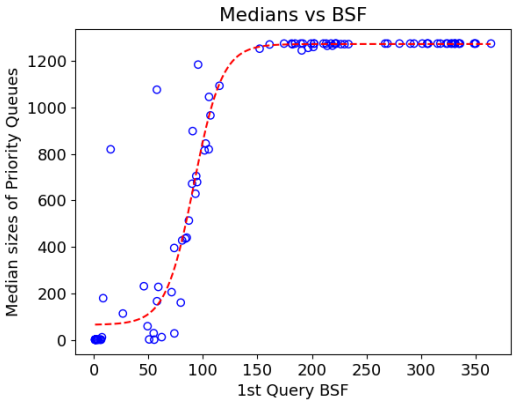
为不同线程实现负载均衡，选择合适的阈值 TH 至关重要。我们的目标是开发一种方法来确定阈值，使得生成的一组优先队列大小大致相同。对于我们使用的每个数据集，都会根据运行的查询来确定并配置阈值。我们以 Seismic 真实数据集 iri [2016],为例说明如何确定 TH，但我们在实验中使用的所有其他数据集（真实数据集或合成数据集）也采用类似的流程。在运行多个难度各异的查询后，我们发现，查询的初始 BSF 与用于回答该查询所生成的优先队列的中位数大小之间再次存在相关性。随后，我们使用

算法 4： PerformWorkStealing - 节点 sn的代码

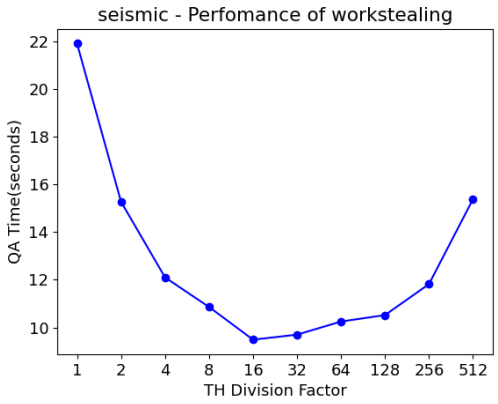
输入：索引 $index$，**函数** $exact_search_workstealing_func$，**查询序列** $queries[]$，**Integ er** $total_nodes_per_nodegroup$

收到来自节点 sn'**的 DONE 消息时：**
将 sn' 添加到集合 $DoneNds$ 中
如果 $DoneNds$ 包含系统的所有节点 **则**
    终止

收到来自节点 sn' **的** $msg = \langle S, Q_s, BSF_s \rangle$ **时：**
如果 $|S| > 0$ **则**
    创建线程，遍历 ID 位于 S中的 RS 批次 填充并处理相应的优先级队列 BSFArray$[Q_s]$:= BSF_s； ▷由上述线程计算 等待所有线程完成
    响应标志：= 0
▷始终启用的事件：重复执行**U**
收到时 ivi ng 无消息：
如果 !(响应标志)**那么**
    sn' := 随机选择一个不在 $DoneNds$中的节点
    将(StealingRequest, sn)发送至 sn'
    响应标志 := 1



(a) 用于确定 TH的 Sigmoid 函数拟合。



(b) 不同阈值划分因子下的性能。

图 6：Odyssey 单节点问答算法配置。

以下参数化公式：

$$f(Z) = m + (M - m)\frac{1}{1 + b \cdot exp(-c(Z - d))}$$

其中 $M \in [0, 1], m \leq M, b, c \in \mathbf{R}^*$, 以及 $d \in \mathbf{R}$ 是 Sigmoid 函数的参数（图 6a）。每个查询的最终阈值是 Sigmoid 函数得出的中位数估计值再除以一个因子（例如，根据图 6b 所示的图表，Seismic 的该因子必须为 16）。

实验表明，树遍历阶段完成后，我们会得到一组 RS 批次，其中包含多个优先队列，且大多数队列的大小相同。这使得线程在处理优先队列时能够实现负载均衡。

3.2.2 工作窃取算法

如果系统节点 sn b e 变为空闲，sn 会启动工作窃取协议（算法 4，第 15–17 行）。它会随机选择一个系统节 e sn' 从其知道仍处于活动状态的节点集合中 sn 选择一个节点，并向其发送窃取请求[3] .

[3]算法 3 和 4 的代码采用事件驱动风格 Attiya 和 Welch [2004], Lynch [1996]

每个节点中的一个线程充当工作窃取管理器（算法 3）。sn'的工作窃取管理器一收到请求，就会尝试将工作分配给sn（算法 3 的第 2–4 行）。
早期研究表明，在正确答案处理完毕后，查询应答的执行时间中仍有很大一部分用于验证是否不存在更优答案 Gogolou 等人 [2019, 2020], Echihabi 等人 [2023]。基于这些发现，Odyssey 的工作窃取机制选择让出一个满足取走属性的 RS 批次 B，即 B 尚未被窃取，且其第一个优先队列位于优先队列 数组中尽可能靠右的索引处。随后，该优先队列会被标记为已窃取。如果要让出多个批次，则重复此过程以选择其他 RS 批次。请记住，$PQueue$ 数组按各优先队列顶部元素的优先级排序。因此，通过这种方式让出批次，sn' 会将可能仍包含待处理工作的优先队列分配给辅助节点。此外，它让出的 RS 批次最有可能尚未处理。在整个过程中，每次当前 BSF 更新时，都会在各节点之间共享，因为辅助节点可能窃取到包含更优答案的优先队列（或者所有者之后可能计算出更优的 BSF）。
节点在窃取过程中让出的 RS 批次数量 N_{send} 会影响性能。从理论上讲，我们希望让出的 RS 批次数量一方面足以让窃取节点完成相当数量的工作，另一方面又不至于使让出的工作导致查询应答时间变长。实验表明，将 N_{send} 固定为 4 是最佳选择（因此，Odyssey 中的 $N_{send} = 4$ 为 4）。

3.3 数据复制

Odyssey 旨在确保数据可扩展性，同时保证良好的查询应答性能。要实现最佳的数据可扩展性，需要采用无复制方式，但实验表明，全复制设置下的查询应答性能最佳。Odyssey 通过提供灵活的部分复制方案，有效地权衡数据可扩展性与良好的查询应答性能。

其思路是将系统节点划分为集群，使每个集群共同存储整个数据集（见图 7）。每个集群节点存储（并为其建立索引）数据集的一个数据块。同一集群中各节点存储的数据块彼此不相交。复制组 是由一组节点构成的，其中每个节点存储的数据集都相同。（我们对大小相同的复制组进行了实验，但 Odyssey 也可以使用大小不同的复制组。）复制组中的节点基于相同的数据块构建各自的 iSAX 索引。因此，在每个复制组内，我们可以采用第 3.1 节和第 3.2 节中分别介绍的调度和负载均衡方案。我们将集群数量称为系统的复制度 。

考虑一个包含 N_{sn} 个系统节点的系统。我们将 PARTIAL-k, $k \in \{1, 2, 4, \ldots, N_{sn}\}$ 称为一种包含 k 个复制组和 N_{sn}/k 个集群的数据复制配置。请注意，PARTIAL-N_{sn}（即 EQUALLY-SPLIT）对应无复制（每个节点存储数据集中的一个互不重叠的数据块），而 PARTIAL-1（即 FULL）对应完全复制（每个节点存储整个数据集）。请注意，Odyssey 的数据复制方案支持 $1 + \log N_{sn}$ 种不同的复制度。复制度越低，空间开销越小（因此数据可扩展性越好）。因此，Odyssey 的数据复制方案使我们能够解决内存限制问题。此外，复制组越多，索引创建的可扩展性越好。

一个包含 8 个节点的系统支持 $1+\log\ 8 = 4$ 种不同的复制度：FULL（PARTIAL-1）、PARTIAL-2、PARTIAL-4 和 EQUALLY-SPLIT（PARTIAL-8）。图 7 展示了 PARTIAL-4 的情况：共有 4 个复制组，分布在 2 个集群中；复制度为 2。

3.4 数据分区

Odyssey 框架支持多种分区方案。在 EQUALLY-SPLIT 方案下，每个系统节点都会分配到一个离散数据块，并建立相应的索引，因此每个节点都在其自身的数据部分上保有一个本地索引。查询会转发至所有节点。每个节点根据其本地索引和数据生成答案，其中的最小值即为最终答案。在分发数据之前，可以应用随机打乱（*RS*），以随机重排初始集合中的序列。
使用部分数据复制（或无复制）来回答查询批次时，每个查询都会发送到每个复制组。每个节点使用本地数据回答查询，最后汇总每个查询的部分答案，以找出最小答案。对于真实数据，查询的近似答案往往集中在数据集的一小部分中。拥有这些数据的组会得到较好的初始答案，因此能够剪枝更多，并极快地回答每个查询；而其他组则未必能算出较好的初始 BSF 值。因此，它们需要进行更多工作，导致负载不均衡。为此，我们为分布式索引添加了一种支持 BSF 共享的记账方法。当节点处理查询并找到更优的 BSF 值时，就会通过公共的 BSF 共享通道共享该值（如图 7 所示）。每个节点都会定期检查此通道，查看是否有某个查询的

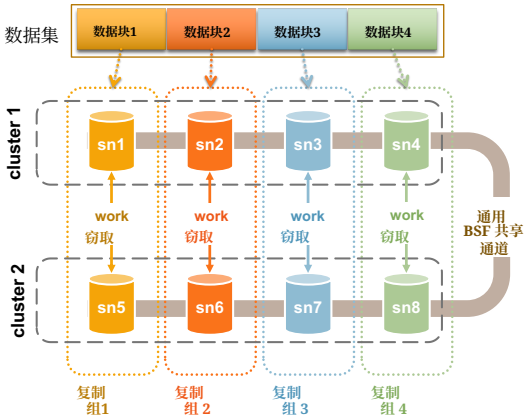


图 7：数据复制 PARTIAL-4（$N_{sn} = 8$，数据规模 80GB）。

答案到达。由于此过程并行运行，节点可能会收到一个更优的答案，而该查询要到之后才会处理。Odyssey 的记账方法解决了这类同步问题。每个节点都持有一个数组，用于存储从通道收到的、针对每个查询的 BSF 改进值；在回答查询之前，节点会检查该数组中的数据。因此，每个节点都能获取从所有节点收集到的最佳答案，我们的实验评估表明，使用此方法对性能至关重要。

除了这些简单的技术之外，Odyssey 还提供了一种复杂的数据分区方案。该方案基于对初始数据序列集合的预处理，在可用节点之间实现密度感知分布。所需的预处理会带来一定的时间开销。不过，这项处理只需执行一次，即可回答所需数量的查询。因此，随着待处理查询数量的增加，这项开销会逐渐摊销。我们将在第 3.4.1 节介绍该方案。

3.4.1 DENSITY-AWARE 数据分区

我们发现，良好的分区策略不应将所有相似序列都分配到同一个系统节点。否则，我们可能会造成工作负载不平衡，原因如下。假设我们需要应答一个相似性搜索查询，数据集中所有与查询相似的候选序列都存储在某个系统节点上，而其他节点存储的序列都与查询不相似。那么，在查询应答过程中，存有相似序列的节点需要进行大量计算（下界和实际距离），才能确定哪个候选序列是查询的最近邻，而几乎无法进行剪枝（如果能剪枝的话）。另一方面，存储不相似序列的其他节点则可以积极剪枝，因此能更快地完成各自的计算任务。

上述观察促使我们设计了 DENSITY-AWARE 分区策略，其目标是在不产生高计算开销的前提下，将相似序列分散到所有系统节点上。为此，我们利用格雷码 Gardner [1986] 排序来提升效果（因为它有助于拆分相似序列），并利用索引的摘要缓冲区来提升效率（因为我们必须在缓冲区层面而非单个序列层面进行操作）。

图8展示了一个示例，说明如何根据一种使用二进制码的简单策略，以及一种基于格雷码的策略，对摘要缓冲区中的数据序列进行分区。在前一种情况下，最终分配到同一节点的缓冲区中包含相似的序列：它们的 iSAX 表示（缓冲区的 iSAX 字）彼此非常接近。例如，节点 1 存储缓冲区“000”和“100”，因此其中序列的 iSAX 摘要仅相差一位。在后一种情况下，这个问题得到了解决。格雷码排序会将相似的缓冲区排列得彼此接近（根据定义，排序中相邻的两个缓冲区仅相差一位），因此可以轻松地以轮询方式将它们分配到不同的系统节点。

我们在图9中展示了 DENSITY-AWARE 分区策略的流程图。首先，我们计算数据序列集合的 iSAX 摘要，并将每个摘要分配到相应的汇总缓冲区。这些缓冲区按格雷码排序，然后开始实际的数据分区（采用轮询调度）。我们首先对 λ 最大缓冲区中的序列进行分区；这是必要的，因为少数缓冲区往往会包含异常大量的序列（我们不希望将它们全部分配给同一个系统节点）。然后，我们对剩余缓冲区进行分区，并检查分区是否均衡。如果不均衡，则选择最大节点中最大的

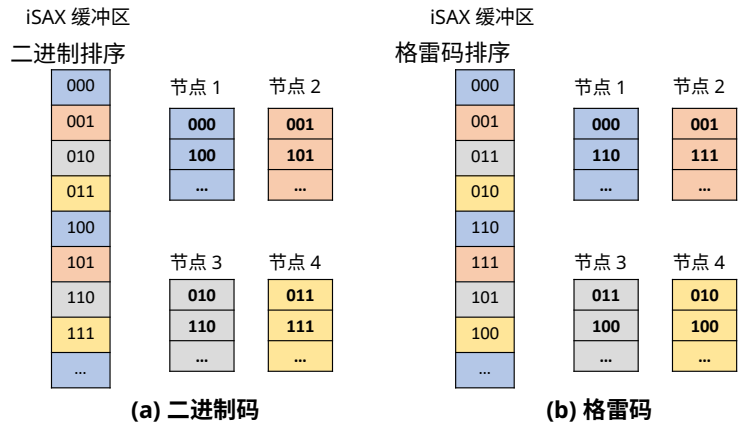


图8：示例 将 iSAX 缓冲区中的数据划分到 4 个系统节点，依据 (a) 简单 iSAX 和 (b) 格雷码。

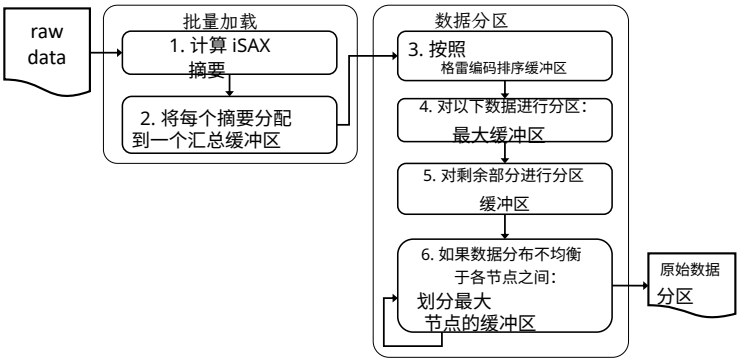


图9：DENSITY-AWARE 数据分区流程图。

缓冲区，并对该缓冲区中的序列进行分区。我们对多个真实数据集进行的实验（为简洁起见，此处省略）表明，随着 λ 从几百变化到几千，DENSITY-AWARE 始终表现得非常稳定。在本研究中，我们使用 $\lambda = 400$。

4 扩展

现在，我们将讨论 Odyssey 的两种扩展，以支持*k-NN* 搜索和动态时间规整（DTW）距离。

***k-NN*搜索**。扩展 Odyssey 以支持*k-NN* 相似性搜索的扩展非常简单。我们不再计算单个最佳至今值，而是计算ply 需要跟踪最小的 k 个 BSF 值。

DTW 距离。我们还扩展了 Odyssey，使其能够使用动态时间规整（DTW）进行相似性搜索。动态时间规整是一种弹性距离度量方法（Keogh和Ratanamahatana [2005]）。请注意，为此无需更改索引结构：我们构建的索引既能回答欧几里得相似性搜索查询，也能回答 DTW 相似性搜索查询。支持 DTW 查询只需修改查询回答算法，并使用 LB_Keogh（Keogh和Ratanamahatana [2005],），它是 DTW 距离的紧下界。我们注意到，可以通过计算候选序列上的对应点与查询的 LB_Keogh 包络上的点之间的距离，来求得查询与候选序列之间 DTW 距离的下界。

5 实验评估

环境配置。实验在由 16 个 SR645节点组成的集群上进行，这些节点通过 HDR 100Infiniband 网络连接。每个节点有 128 个核心（未启用超线程）和 200GB 随机存取存储器（在 256GB 物理内存中，可供用户使用的

表1：实验所用数据集的详细信息。

数据集	序列数量	长度（浮点数）	大小（GB）	描述
Seismic	100M	256	100	地震记录
Astro	270M	256	265	天文数据
Deep	1B	96	358	深度嵌入
Sift	1B	128	477	图像描述符
Yan-TtI	1B	200	800	图像和文本
Random	100M-1600M	256	100-1600	随机游走

内存），并运行 Re 红帽企业版 Linux r8.2。所有评估的算法均以 C 编写，并使用 MPICC 编译 ，
Intel(R) MPI 库 f or Linux 操作系统，v2021.2。

算法。我们的实验分析涵盖了 Odyssey 的全部数据分布策略，包括 k复制组、PARTIAL-k、 $k \in \{1, 2, 4, \ldots, N_{sn}\}$，以及密度感知数据分区算法（DENSITY-AWARE）。请回想一下，PARTIAL-N_{sn} 或 EQUALLY-SPLIT 对应无复制，而 PARTIAL-1 或 FULL 对应完全复制。此外，我们还评估了 Odyssey 的查询调度算法：(i) 静态调度，将大小相同的查询集合分配给各节点（STATIC）；(ii) 使用协调器的动态调度（DYNAMIC）；以及 (iii) 基于预测的调度，包括：无排序的静态调度（PREDICT-ST-UNSORTED）、有排序的静态调度（PREDICT-ST）和动态调度（PREDICT-DN）。此外，我们还评估了 Odyssey 在 DYNAMIC 和 PREDICT-DN 两种调度算法下的工作窃取机制，分别得到 WORK-STEAL 和 WORK-STEAL-PREDICT 算法。后者是我们效果最佳的调度算法（参见"查询调度"段落）。需要指出的是，Odyssey 的查询调度和工作窃取机制只能与提供某种复制能力的 FULL 或 PARTIAL 数据分布策略结合使用。

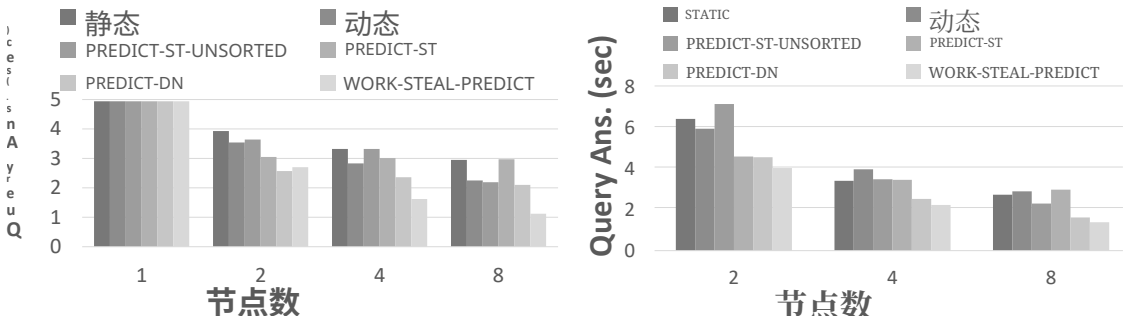
我们将 Odyssey 与以下方法进行比较：(i) MESSI Peng 等人 [2021a], ，其中我们在每个系统节点上独立运行 MESSI 索引；(ii) MESSI SW BSF，在该方法中，我们通过启用系统范围内共享 BSF 值来扩展前一种方案；以及 (iii) DPiSAX Yagoubi等人 [2020], ，我们实现（使用 C 语言）DPiSAX 数据分区策略，并且（为公平起见）使用 MESSI 在每个节点上实现查询应答。

数据集。我们使用不同规模的真实数据集和合成数据集评估了 Odyssey 的策略和算法（参见表1）。名为*Random*的合成数据序列由遵循高斯分布（0,1）的随机游走（即累积和）步骤生成。这类数据过去已被广泛使用 Faloutsos 等人 [1994],Camerra 等人 [2014], Zoumpatianos 等人 [2015, 2018], Echihabi 等人 [2018, 2019], ，并用于模拟股票市场价格的分布 Faloutsos 等人 [1994]。我们的五个真实数据集分别来自地震学（*Seismic*）、天文学（*Astro*）、深度学习（*Deep*）、图像处理（*Sift*）和信息检索（*Yan-TtI*）领域。*Seismic* 包含地震仪器记录，由 1 亿条大小为 256 的数据序列组成，数据来自 Seismology with Artificial Intelligence[2018]。*Astro* 表示天体，由 1 亿条大小为 256 的数据序列组成 Soldi 等人 [2014]。*Deep* Vision[2018] 包含从卷积神经网络的最后几层提取的 10 亿个大小为 96 的 Deep 向量。*Sift* Jégou 等人 [2011] 由图像描述符构成，而 Yandex Text-to-Image（*Yan-TtI*）Simhadri 等人 [2022] 包含 10 亿个向量，其中图像嵌入和文本嵌入位于同一空间；它代表了典型的跨模态信息检索任务。

评估指标。 在每次实验期间， E，对于每个节点sn，我们测量：(i) 计算 iSAX 摘要并填充接收缓冲区所需的缓冲区时间； (ii) 将接收缓冲区中的项目插入索引树所需的树时间；以及 (iii) 应答分配给sn的查询所需的查询应答时间 。这些时间的总和构成sn 在 E期间的总时间； 此外，缓冲区时间和树时间构成创建索引所需的时间，称为索引时间。为了计算 E期间的上述所有时间，我们取参与 E的每个节点对应时间的最大值。我们报告 10 次实验的平均时间。

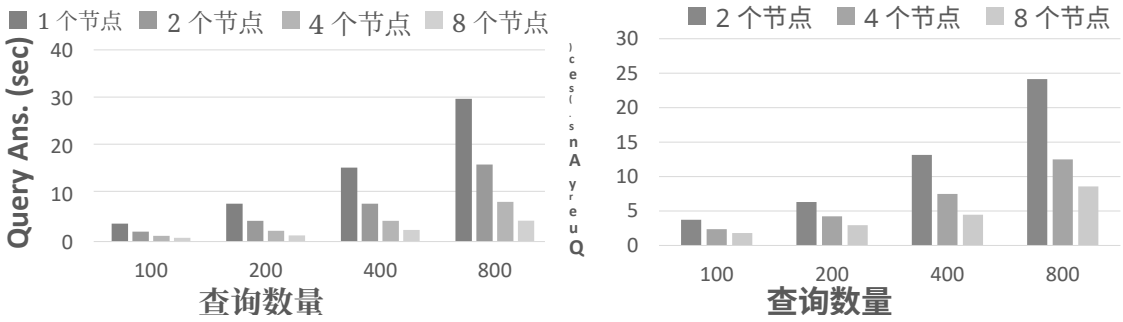
查询调度。 为了比较 Odyssey 的查询调度算法，我们选择 FULL 复制策略，以避免测量部分复制策略带来的任何开销。请注意，调度算法不能与无复制策略配合使用。我们使用 Random（合成数据集）和 Seismic（真实数据集）进行了实验，与 STATIC 相比，我们的所有算法都提升了性能。此外，对于合成数据集，我们发现各调度算法之间没有明显差异，因为生成数据集的数据序列和查询集时的随机性，使得查询应答所需的工作量几乎相同。我们展示 Seismic 的结果，该数据集中查询应答的工作量有所不同。具体而言，图 10 显示，随着节点数增加，PREDICT-DN 在所有情况下都是最佳调度策略，其表现最高可比 STATIC 好 150%。

工作窃取。图 10a 显示，在使用 FULL 复制时，对于节点数较多的情况，WORK-STEAL-PREDICT 的性能远优于 PREDICT-DN（最高接近 2 倍），也就是说，我们的工作窃取技术能够有效提升性能



(a) FULL 复制 (b) 部分复制-2

图 10：Odyssey 的调度算法（Seismic）。



(a) FULL 复制 (b) 部分复制-2

图 11：随着查询数量增加，查询回答的可扩展性（Random）。

在这些情况下也是如此。对于部分复制-2，情况相同，但效果较弱。回想一下（第 3.2.2 节），这是因为所有未使用工作窃取技术的算法都会受到负载不均衡问题的影响。具体而言，当查询集合中只有少数查询（数量远少于节点数）需要耗费显著更多的工作量才能得到应答（相比大多数查询）时，随着节点数增加，在相应的查询回答阶段结束时，会有更多节点处于空闲状态，因为没有分配给它们这类难以应答的查询。

查询可扩展性。为了评估 Odyssey 的算法在查询数量不断增加时的可扩展性，我们使用合成数据集和真实数据集，对 WORK-STEAL 进行了实验。在图 11a 中，我们展示了 Random 数据集在 FULL 复制下、总查询数量分别为 100、200、400 和 800 时的结果（其他受测数据集的结果相似）。可以看到，随着查询数量增加，WORK-STEAL 几乎实现了完美扩展，因为在 1 个节点上执行 100 个查询所需的时间，与在 j 个节点上执行 $j * 100$ 个查询所需的时间相同，$j \in \{2, 4, 8\}$。对于部分调度算法，我们也观察到了相同的趋势（图 11b）。请注意，部分复制只能应用于两个或更多节点。此外，我们还在图 12 中展示了通过增大数据集大小进行的可扩展性实验，涵盖 Random（100–1600GB）和 Yan-TtI（100–800GB）。我们测量了使用 8 个节点时回答 100 个查询的总时间。请注意，由于节点的内存容量有限，我们无法针对所有数据集大小执行所有复制策略。结果表明，随着数据集大小增加，查询应答时间平稳扩展；而提高复制度则能带来更好的性能。此外，我们观察到，随着节点数增加，Odyssey 的查询回答算法仍具有良好的可扩展性。图 13 更直观地展示了这一点，其中给出了 WORK-STEAL 在 Random 数据集上的吞吐量。

复制。我们现在使用 Seismic 数据集和 WORK-STEAL-PREDICT（这是我们效果最佳的调度算法）来研究 Odyssey 的不同复制策略，以避免节点之间负载不均衡带来的开销。具体而言，我们针对不同的查询数量测试 EQUALLY-SPLIT、PARTIAL-4、部分-2 和 FULL。图 15a–15b 展示了查询应答时间[4]，，可以看到，数据集的复制程度越高，查询应答所需时间越短，而且这一趋势适用于所有查询数量。因此，FULL 复制策略的查询应答时间最短。另一方面，图 15c–15d 展示了总执行时间，其中

[4]我们仅报告小型工作负载在 16 个节点上的结果，因为我们的集群调度器不允许在超过 8 个节点上运行长时间运行的作业。

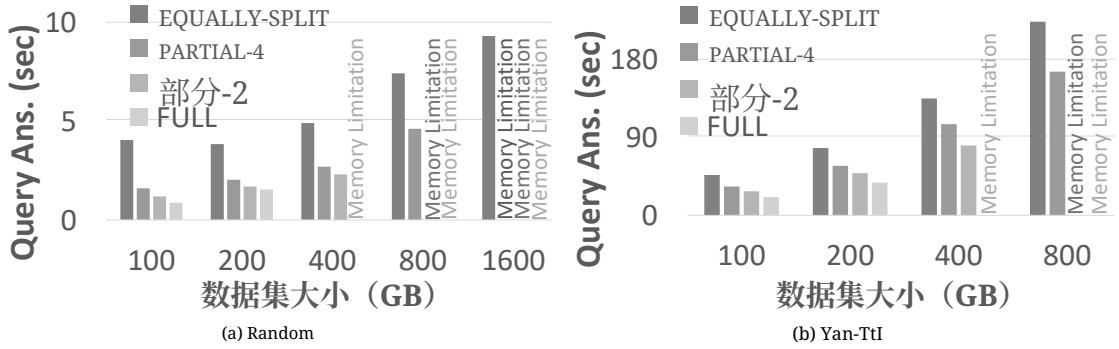


图 12：8 个节点上，100个查询的查询时间与数据规模的关系。

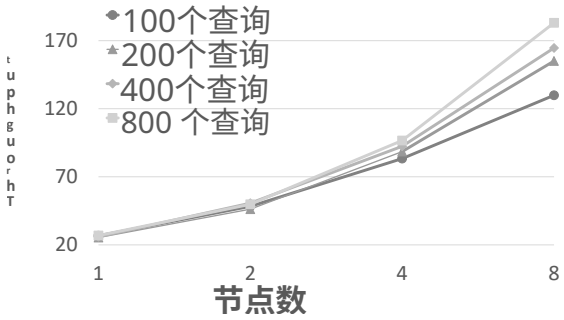


图 13：查询吞吐量（Random，FULL 副本）。

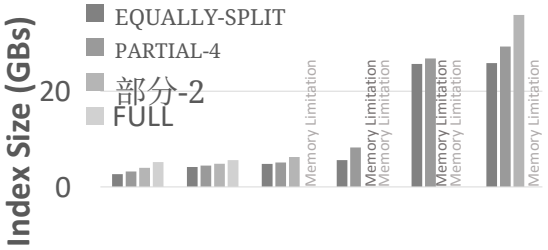
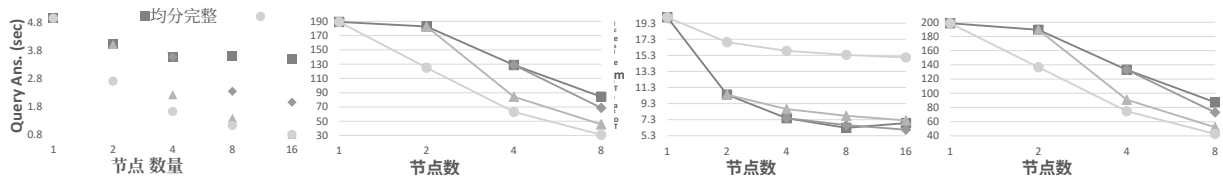


图 14：索引大小。

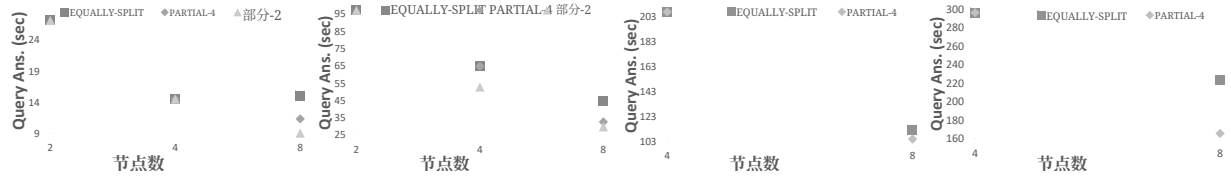
还包括索引树构建时间。有趣的是，对于较少的查询数量（100），我们观察到的趋势恰好相反：数据复制量越大，总时间越长，此时 FULL 的索引树构建时间最长。这是因为增加的索引树构建时间在总时间中占据主导地位。然而，随着查询数量增加，各算法总执行时间之间的差异会逐渐缩小。值得注意的是，当查询数量足够大时（例如 800），增加的索引树构建开销会被较短的查询应答时间抵消，因此 FULL 复制策略的表现优于 EQUALLY-SPLIT。这项分析揭示了查询应答成本与索引树构建成本之间一个有趣的权衡（涉及复制程度）；后者可以通过足够大的查询集来摊销。图 16 展示了其余真实数据集上进行 100 个查询的查询应答实验结果。我们观察到的趋势与 Seismic（图 15a）类似。总体而言，当需要优化查询应答时，我们建议在给定数据集大小和计算集群特性的条件下，尽可能提高 Odyssey 的复制度。

索引可扩展性。 图 14 展示了在使用 8 个节点时，针对我们使用的所有真实数据集以及 Random 100GB（Ran.100），采用每种复制策略时的索引总大小（GB）。在所有情况下，与数据集大小相比，索引大小都非常小。图 17a 和图 17b 分别展示了 Odyssey 对 1B 序列 Deep 数据集采用 EQUALLY-SPLIT 时，随着数据集大小增加（在一个由 16 个节点组成的系统上）以及随着节点数增加（使用完整大小的数据集）的索引创建时间。两种情况下，索引创建都实现了理想的加速比。此外，图 17c 展示了 Odyssey 在 Random 数据集上的可扩展性，其中数据集



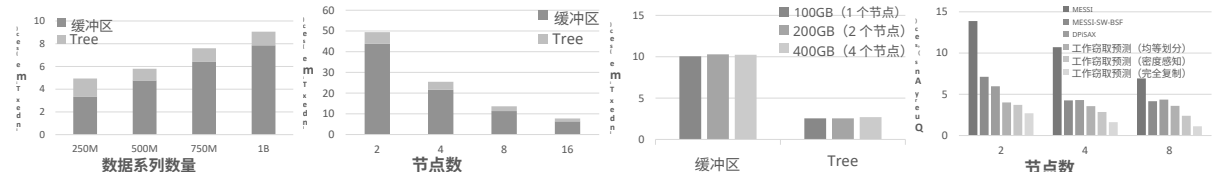
（a）100 个查询　（b）800 个查询　（c）100 个查询　(d) 800 个查询

图 15　：使用 WORK-STEAL-PREDICT 与 Seismic 比较 Odyssey 的复制策略　.



(a) Astro (b) Deep (c) Sift (d) Yan-TtI

图 16：使用真实数据集和 100个查询，比较 Odyssey 的复制策略（采用 WORK-STEAL-PREDICT）。



(a) (b) (c) (d)

图 17：(a) 在 16 个节点上，使用 EQUALLY-SPLIT 时 Deep 的索引可扩展性随数据集大小增加的变化。(b) 使用 EQUALLY-SPLIT 时，Deep 的索引可扩展性随节点数增加的变化。(c) 使用 EQUALLY-SPLIT 时，Random 数据集的大小和节点数均线性增加时的索引可扩展性。(d) 使用 Seismic，对比 WORK-STEAL-PREDICT 与 Odyssey 的不同数据分区方案，以及其他实现。

大小和节点数均线性增加，同样采用 EQUALLY-SPLIT。如图所示，Odyssey 实现了完美的可扩展性，因为相应的缓冲时间和索引时间几乎保持不变。

数据分区及与竞争方法的比较。图 17d 展示了：(i) 将 WORK-STEAL-PREDICT（Odyssey 性能最佳的算法）与 DMESSI、DMESSI-SW-BSF 和 DPISAX 进行比较；(ii) 使用 Seismic 时，Odyssey 的不同数据分区方案（即 EQUALLY-SPLIT 和 DENSITY-AWARE）以及 FULL 复制策略的性能。有趣的是，DMESSI 的性能显著低于所有其他实现，这表明，在多节点系统上仅运行多个最先进的单节点算法（如 MESSI）实例，以扩大其对更大数据集的适用性，在真实数据集上的表现并不好；因此，需要更复杂的方法。另一方面，采用 FULL 复制策略的 Odyssey WORK-STEAL-PREDICT 显著优于所有竞争方法。具体而言，其速度分别比 DMESSI、DMESSI-SW-BSF 和 DPISAX 快至 6.6 倍、3.7 倍和 3.8 倍。此外，关于 Odyssey 的数据分区技术，图 17d 显示，采用 DENSITY-AWARE 分区的 WORK-STEAL-PREDICT 优于 EQUALLY-SPLIT。

对*k-NN* 和 DTW 的扩展。最后，我们展示了使用 *k-NN* 查询和 DTW 距离的实验，其中我们在采用不同复制策略时，随着节点数增加，测量 100个查询的查询应答时间。对于*k-NN*，我们在 k 介于 1 到 20 之间变化时评估了所有复制策略；对于 DTW，则在变形窗口大小介于序列长度的 1%-15% 之间变化时进行评估。图 18 展示了 $k = 10$ 的 *k-NN* 结果，图 19 展示了变形窗口大小为 5% 时的 DTW 结果（其余参数取值下的结果相似）。正如预期的那样，两种情况下的查询应答时间都比之前更长；与此同时，增加节点数和提高复制度能够以我们在先前实验中观察到的相同方式提升性能。Seismic 的结果呈现出类似趋势，因篇幅所限，此处省略。

6 结论

在这项工作中，我们介绍了 Odyssey，这是一种新颖的分布式数据序列处理框架，能够充分利用由多核服务器组成的现代集群的全部计算能力。Odyssey 解决了设计高效且高度可扩展的分布式数据序列索引时面临的诸多挑战，包括高效调度、负载均衡和灵活的部分复制，并成功应对了数据可扩展性与良好查询应答性能之间的根本权衡。未来，我们计划扩展 Odyssey，使其支持 Linardi 和 Palpanas [2020],提出的子序列相似性搜索以及近似相似性搜索。

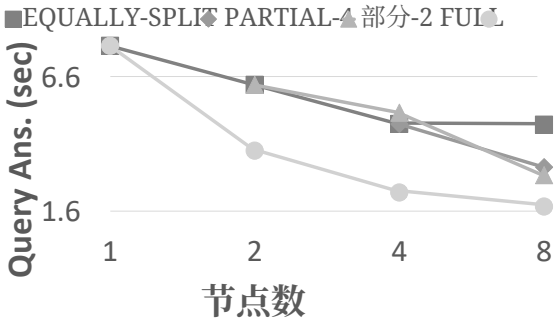


图 18：*10-NN* 查询应答（随机 100GB）

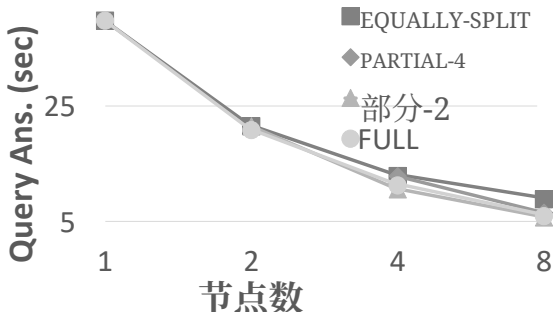


图 19：随机 100GB 上的动态时间规整（5% 扭曲）.

致谢

本研究得到国家自然科学基金委员会项目（编号：62202450）、欧盟地平线2020玛丽·斯克沃多夫斯卡-居里项目（编号：101031688）以及希腊研究与创新基金会（HFRI）“支持教职人员和研究人员的 HFRI 研究项目第二轮征集”项目（编号：3684）的资助。数值计算在法国巴黎地球物理研究所（IPGP）的 S-CAPAD/DANTE平台上完成。

参考文献

Themis Palpanas. 数据序列管理：迈向大规模序列分析之路。*SIGMOD Record*，2015。Kostas Zoumpatianos 和 Themis Palpanas。数据序列管理：满足大规模序列分析的需求。发表于*ICDE*，2018。Themis Palpanas 和 Volker Beckmann。第一届和第二届跨学科时间序列分析研讨会（ITISA）报告。*SIGREC*，48(3)，2019。Kunio Kashino、Gavin Smith 和 Hiroshi Murase。用于快速检索音频和视频的时间序列主动搜索。发表于*ICASSP*，1999。Lexiang Ye 和 Eamonn Keogh。时间序列形状片段：一种用于数据挖掘的新型基本单元。发表于*SIGKDD*。ACM，2009。Pablo Huijse、Pablo A Estevez、Pavlos Protopapas、Jose C Principe 和 Pablo Zegers。大规模天文时间序列数据库中的计算智能挑战与应用。*CIM*，2014。Usman Raza、Alessandro Camerra、Amy L Murphy、Themis Palpanas 和 Gian Pietro Picco。面向真实世界无线传感器网络的实用数据预测。*TKDE*，2015。Paul Boniol、Michele Linardi、Federico Roncallo 和 Themis Palpanas。大型序列中的自动异常检测。发表于*ICDE*，2020。Paul Boniol 和 Themis Palpanas。Series2Graph：基于图的时间序列子序列异常检测。*PVLDB*，2020。Michele Linardi、Yan Zhu、Themis Palpanas 和 Eamonn J. Keogh。矩阵轮廓走向 MAD：数据序列中的可变长度模式与异常片段发现。发表于*DAMI*，2020。Qitong Wang、Stephen Whitmarsh、Vincent Navarro 和 Themis Palpanas。iEDeaL：用于检测高度不平衡发作间期癫痫样放电的深度学习框架。*PVLDB*，16(2)，2023a。Karima Echihabi、Kostas Zoumpatianos、Themis Palpanas 和 Houda Benbrahim。数据序列相似性搜索中的勒尔那九头蛇：对最先进技术的实验评估。*PVLDB*，2018。Karima Echihabi、Kostas Zoumpatianos、Themis Palpanas 和 Houda Benbrahim。勒尔那九头蛇归来：数据序列近似相似性搜索的实验评估。*PVLDB*，2019。Karima Echihabi、Kostas Zoumpatianos 和 Themis Palpanas。高维向量上的可扩展机器学习：从数据序列到深度网络嵌入。发表于 *WIMS*：第十届国际网络智能、挖掘与语义会议，第 1–6 页。ACM，2020。Karima Echihabi、Kostas Zoumpatianos 和 Themis Palpanas。大规模序列管理：横向与纵向扩展。发表于 第*24*届扩展数据库技术国际会议论文集，*EDBT*，第 714–717 页。OpenProceedings.org，2021a。

Karima Echihabi、Themis Palpanas 和 Kostas Zoumpatianos。高维向量相似性搜索的新趋势：人工智能驱动、渐进式与分布式。*Proceedings of the VLDB Endowment*，14(12):3198–3201，2021b。Anthony J. Bagnall、Richard L. Cole、Themis Palpanas 和 Konstantinos Zoumpatianos。数据序列管理。*DagstuhlReports*，9(7)，2019。Themis Palpanas。数据序列挖掘的并行与分布式未来。发表于*HPCS*，2017。Botao Peng、Themis Palpanas 和 Panagiota Fatourou。Paris：序列索引与查询应答的新目的地。*IEEEBigData*，2018。Botao Peng、Themis Palpanas 和 Panagiota Fatourou。Paris+：多核架构上的数据序列索引。*TKDE*，2020a。Karima Echihabi、Panagiota Fatourou、Kostas Zoumpatianos、Themis Palpanas 和 Houda Benbrahim。Hercules 对决数据序列相似性搜索。*PVLDB*，2022。Anna Gogolou、Theophanis Tsandilas、Themis Palpanas 和 Anastasia Bezerianos。时间序列数据上的渐进式相似性搜索。发表于*EDBT*，2019。Karima Echihabi、Theophanis Tsandilas、Anna Gogolou、Anastasia Bezerianos 和 Themis Palpanas。ProS：具有概率质量保证的数据序列渐进式 k近邻相似性搜索与分类。*VLDBJ*，2023。Botao Peng、Themis Palpanas 和 Panagiota Fatourou。Messi：内存数据序列索引。发表于*ICDE*，2020b。Botao Peng、Panagiota Fatourou 和 Themis Palpanas。面向内存中数据的快速数据序列索引。*VLDB J.*，30(6)：1041–1067，2021a。Botao Peng、Panagiota Fatourou 和 Themis Palpanas。SING：使用GPU进行序列索引。发表于 数据工程国际会议论文集（*ICDE*），2021b。Odyssey 代码和数据集。https://helios2.mi.parisdescartes.fr/~themisp/odyssey/，2022。Jin Shieh 和 Eamonn Keogh。iSAX：TB 级时间序列的索引与挖掘。2008。doi:10.1145/1401890.1401966。URL https://doi.org/10.1145/1401890.1401966。Themis Palpanas。数据序列索引的演进——iSAX数据序列索引系列。发表于*Communications inComputer and Information Science(CCIS)*，第 1197 卷，2020。Ilias Azizi、Karima Echihabi 和 Themis Palpana。Elpis：用于可扩展数据科学的基于图的相似性搜索。*PVLDB*，2023。Rakesh Agrawal、Christos Faloutsos 和 Arun N. Swami。序列数据库中的高效相似性搜索。发表于 David B. Lomet 编，*FODO*，1993。Timos K. Sellis、Nick Roussopoulos 和 Christos Faloutsos。r+-树：一种用于多维对象的动态索引。发表于*VLDB*，1987。Davood Rafiei 和 Alberto O. Mendelzon。使用离散傅里叶变换高效检索相似时间序列。发表于 Katsumi Tanaka 和 Shahram Ghandeharizadeh 编，*FODO*，1998。Yang Wang、Peng Wang、Jian Pei、Wei Wang 和 Sheng Huang。一种用于时间序列整体匹配的数据自适应动态分段索引。*PVLDB*，6(10)，2013。Eamonn J. Keogh、Kaushik Chakrabarti、Sharad Mehrotra 和 Michael J. Pazzani。用于大型时间序列数据库索引的局部自适应降维。发表于 Sharad Mehrotra 和 Timos K. Sellis 编，*SIGMOD*，2001。Alessandro Camerra、Jin Shieh、Themis Palpanas、Thanawin Rakthanmanon 和 Eamonn J. Keogh。超越十亿条时间序列：使用 iSAX 对超大型时间序列集合进行索引与挖掘。*Knowl. Inf. Syst.*，39(1):123–151，2014。URL http://dblp.uni-trier.de/db/journals/kais/kais39.html#CamerraSPRK14。Kostas Zoumpatianos、Stratos Idreos 和 Themis Palpanas。用于交互式探索大数据序列的索引。发表于*SIGMOD*，2014。Kostas Zoumpatianos、Stratos Idreos 和 Themis Palpanas。ADS：自适应数据序列索引。*VLDB J.*，2016。Michele Linardi 和 Themis Palpanas。数据序列中的可扩展可变长度相似性搜索：ULISSE 方法。*PVLDB*，2019。Michele Linardi 和 Themis Palpanas。使用 ULISSE 进行可扩展数据序列子序列匹配。*VLDBJ*，2020。Qitong Wang 和 Themis Palpanas。用于数据序列相似性搜索的深度学习嵌入。发表于 *KDD '21*：第*27*届 *ACM SIGKDD*知识发现与数据挖掘会议，线上举行，新加坡，*2021*年*8*月*14*日至*18*日，第 1708–1716 页。ACM，2021。

Zeyu Wang、Qitong Wang、Peng Wang、Themis Palpanas 和 Wei Wang。Dumpy：面向大型数据序列集合的紧凑自适应索引。发表于*SIGMOD*，2023b年。Søren Kejser Jensen、Torben Bach Pedersen 和 Christian Thomsen。时间序列管理系统：综述。*IEEE Transactions on Knowledge and Data Engineering*，29(11):2581–2600，2017年。Tuomas Pelkonen、Scott Franklin、Paul Cavallaro、Qi Huang、Justin Meza、Justin Teller 和 Kaushik Veeraraghavan。Gorilla：快速、可扩展的内存中时间序列数据库。*VLDB*，2015年。Crate。CrateDB：面向机器数据与物联网的实时 SQL 数据库，2018年。网址：http://crate.io/。InfluxDB。InfluxDB——开源时间序列、指标和分析数据库（http://influxdb.com/），2018年。网址：http://influxdb.com/。Prometheus。Prometheus——监控系统与时间序列数据库，2018年。网址：http://prometheus.io/。QuasarDB。QuasarDB：高性能分布式时间序列数据库，2018年。网址：https://www.quasardb. net/。Hellium。Hellium：超高性能键值存储，2018年。网址：https://www.levyx.com/helium。RiakTS。Riak TS——Basho Technologies，2018年。网址：http://basho.com/products/riak-ts/。Timescale。Timescale——开源时间序列管理系统，2018年。网址：http://timescale.com/。Chen Wang、Xiangdong Huang、Jialin Qiao、Tian Jiang、Lei Rui、Jinrui Zhang、Rong Kang、Julian Feinauer、Kevin Mcgrail、Peng Wang、Diaohan Luo、Jun Yuan、Jianmin Wang 和 Jiaguang Sun。Apache IoTDB：物联网时间序列数据库。*Proceedings of the VLDB Endowment*，13(12):2901–2904，2020年。OpenTSDB。OpenTSDB——分布式、可扩展的监控系统（http://opentsdb.net/），2015年。网址：http: //opentsdb.net/。Timely。Timely——基于 Accumulo 和 Grafana 的安全时间序列数据库，2018年。网址：https://code.nsa. gov/timely/。Warp10。Warp 10——最先进的时间序列平台，2018年。网址：https://www.warp10.io/。Jiaye Wu、Peng Wang、Ningting Pan、Chen Wang、Wei Wang 和 Jianmin Wang。KV-Match：一种支持归一化和时间扭曲的子序列匹配方法。发表于第*35*届*IEEE*国际数据工程会议（*ICDE 2019*），中国澳门，*2019*年*4*月*8*日至*11*日，第866–877页。IEEE，2019年。Kefeng Feng、Peng Wang、Jiaye Wu 和 Wei Wang。L-Match：一种轻量且有效的子序列匹配方法。*IEEEAccess*，8:71572–71583，2020年。Liang Zhang、Noura Alghamdi、Mohamed Y. Eltabakh 和 Elke A. Rundensteiner。TARDIS：大型时间序列数据的分布式索引框架。发表于*ICDE*，2019年。Djamel Edine Yagoubi、Reza Akbarinia、Florent Masseglia 和 Themis Palpanas。DPiSAX：大规模分布式分区 iSAX。第 1135–1140 页，2017 年。doi:10.1109/ICDM.2017.151。Djamel-Edine Yagoubi、Reza Akbarinia、Florent Masseglia 和 Themis Palpanas。大规模分布式时间序列索引与查询。*IEEETransactions on Knowledge and Data Engineering*，32(1):108–120，2020年。doi:10.1109/TKDE.2018.2880215。Cilk：一种高效的多线程运行时系统。*Journal of Parallel and Distributed Computing*，37(1):55–69，1996年。Robert D. Blumofe 和 Charles E. Leiserson。通过工作窃取调度多线程计算。*Journal of the ACM*，46(5):720–748，1999年9月。ISSN 0004-5411。doi:10.1145/324133.324234。网址：https://doi.org/10.1145/324133.324234。Panagiota Fatourou 和 Paul Spirakis。一种用于一般严格多线程计算的新调度算法。发表于 Prasad Jayanti 编，分布式计算，第297–311页，柏林、海德堡，1999年。Springer Berlin Heidelberg。ISBN 978-3-540-48169-0。Panagiota Fatourou 和 Paul Spirakis。严格多线程计算的高效调度。*Theory of ComputingSystems*，33:173–232，2000年。Guy E. Blelloch、Phillip B. Gibbons 和 Yossi Matias。为具有细粒度并行性的语言提供可证明高效的调度。*Journal of the ACM*，46(2):281–321，1999年3月。ISSN 0004-5411。doi:10.1145/301970.301974。网址：https://doi.org/10.1145/301970.301974。Guy E. Blelloch、Phillip B. Gibbons、Yossi Matias 和 Girija J. Narlikar。使用同步变量进行并行计算的节省空间调度。发表于第九届*ACM*并行算法与体系结构年会论文集，SPAA '97。计算机协会，1997年。ISBN 0897918908。

Panagiota Fatourou。使用一次写入同步变量的并行计算低争用深度优先调度。发表于第十三届*ACM*并行算法与体系结构年会论文集，SPAA '01，美国纽约州纽约市，2001年。计算机协会。ISBN 1581134096。doi:10.1145/378580.378639。网址：https://doi.org/10.1145/378580.378639。Anna Gogolou、Theophanis Tsandilas、Karima Echihabi、Anastasia Bezerianos 和 Themis Palpanas。具有概率质量保证的数据序列渐进相似性搜索。发表于*2020*年*ACM SIGMOD*国际数据管理大会论文集，2020年。地震学研究机构联合会——地震数据访问。http://ds.iris.edu/data/access/，2016年。Hagit Attiya 和 Jennifer Welch。分布式计算：基础、模拟与高级主题。John Wiley and Sons, Inc.，美国新泽西州霍博肯，2004年。ISBN 0471453242。Nancy A. Lynch。分布式算法。Morgan Kaufmann Publishers Inc.，美国加利福尼亚州旧金山，1996年。ISBN 9780080504704。Martin Gardner。《打结的甜甜圈及其他数学趣闻》。W. H. Freeman，1986年。Eamonn Keogh 和 Chotirat Ann Ratanamahatana。动态时间规整的精确索引。*KIS*，2005年。Christos Faloutsos、M. Ranganathan 和 Yannis Manolopoulos。时间序列数据库中的快速子序列匹配。发表于*SIGMOD*，第419–429页，美国纽约州纽约市，1994年。ACM。doi:10.1145/191839.191925。网址：http: //doi.acm.org/10.1145/191839.191925。Kostas Zoumpatianos、Yin Lou、Themis Palpanas 和 Johannes Gehrke。数据序列索引的查询工作负载。发表于第*21*届*ACM SIGKDD*国际知识发现与数据挖掘大会论文集，澳大利亚新南威尔士州悉尼，*2015*年*8*月*10*日至*13*日，第1603–1612页，2015年。doi:10.1145/2783258.2783382。网址：http://doi.acm.org/10.1145/2783258.2783382。Kostas Zoumpatianos、Yin Lou、Ioana Ileana、Themis Palpanas 和 Johannes Gehrke。生成数据序列查询工作负载。*VLDBJ.*，2018年。结合人工智能的地震学研究机构联合会。地震数据访问。http://ds. iris.edu/data/access/，2018年。S Soldi、Volker Beckmann、WH Baumgartner、Gabriele Ponti、Chris R Shrader、P Lubi´nski、HA Krimm、F Mattana 和 Jack Tueller。活动星系核硬X射线的长期变化。《天文学与天体物理学》，563:A57，2014年。斯科尔科沃科技学院计算机视觉。深度十亿级索引。http://sites.skoltech.ru/compvision/noimi，2018年。Hervé Jégou、Romain Tavenard、Matthijs Douze 和 Laurent Amsaleg。搜索十亿个向量：使用源编码重新排序。发表于*IEEE*国际声学、语音与信号处理会议（*ICASSP 2011*），*2011*年*5*月*22*日至*27*日，捷克共和国布拉格布拉格会议中心，第861–864页。IEEE，2011年。doi:10.1109/ICASSP.2011.5946540。网址：https://doi.org/10.1109/ICASSP.2011.5946540。Harsha Vardhan Simhadri、George Williams、Martin Aumüller、Matthijs Douze、Artem Babenko、Dmitry Baranchuk、Qi Chen、Lucas Hosseini、Ravishankar Krishnaswamny、Gopal Srinivasa、Suhas Jayaram Subramanya 和 Jingdong Wang。NeurIPS?21十亿级近似最近邻搜索挑战赛结果。发表于*NeurIPS 2021*竞赛与演示专题论文集，第177–189页，2022年。