一种高效的序列数据同步优化算法及应用

霍淑华[1] 田 丰[1] 李 颖[1]
HUO Shuhua TIAN Feng LI Ying

摘 要 在信息系统中保持客户端与服务端序列数据的有序性是一个基础而关键的需求，传统同步方法在处理长序列时存在显著的性能瓶颈。基于此，文章提出了一种针对序列数据同步场景的通用优化算法，并设计了一种基于局部匹配和双向扩展的启发式算法，通过识别最大公共子序列区域（SUPER 区间）最小化非必要的数据更新。算法通用性设计使其可广泛应用于电子商务、社交网络等多个领域的序列同步需求。实验结果表明，该算法在一定约束内保持数据一致性的同时，显著降低了计算复杂度。

关键词 数据排序；序列对齐；数据库优化；启发式算法

doi：10.3969/j.issn.1672-9528.2025.10.042

0 引言

在现代信息系统架构中，多端数据同步是一个基础但关键的技术挑战。随着移动互联网的普及，用户期望在各种设备上获得一致的数据体验，这要求系统能够高效处理客户端与服务端之间的数据同步问题。特别是在需要维护用户自定义排序的场景下，例如音乐播放列表、电子书阅读进度、项目管理看板等。传统的全量同步方法会导致系统承受巨大的性能压力。

数据需要在多个客户端间同步时，现有的数据同步解决方案主要分为两类，全量同步和增量同步，但算法均存在局限性。在保证合并顺序合理的前提下，简单的全量更新策略会带来三个主要问题：

（1）过度占用服务内存；

（2）过度的数据库写操作；

（3）客户端响应延迟。

增量同步有基于时间戳的增量同步和基于版本号的差异检测。然而，这些方法在维护有序列表时效果有限，因为它们无法有效识别列表中的相对位置变化。

本文提出的算法通过创新性地结合序列比对技术和启发式区间检测，实现了在 $O(n)$ 的时间复杂度和最小为 $O(n)$ 的空间复杂度内完成高效同步，这一突破性进展显著提升了序列数据同步的性能表现。相较于现有解决方案，本算法的主要创新点包括：

（1）高效启发式策略：采用“关键点定位 + 双向扩展”的启发式方法，通过精心设计的锚点选择机制和双向遍历策略在保证一致的前提下，从传统 LCS 算法的 $O(n^2)$ 平均时间复杂度降至接近线性水平。

（2）资源优化：通过创新的 SUPER 区间识别机制，显著降低内存消耗和数据库访问压力，使其适合在资源受限环境中部署。

（3）通用性：突破特定领域限制，算法适用于各类序列数据同步。通过抽象关键操作（如比较函数、定位策略等），算法可灵活适配不同应用场景。

1 相关工作

列表同步和排序维护问题在计算机科学多个领域都有深入研究。本文系统性地梳理了相关研究工作，并将本算法置于更广阔的学术背景中进行比较分析。

传统的序列同步算法主要基于最长公共子序列（LCS）方法 [1]。虽然 LCS 能够准确识别两个序列之间的最大匹配部分，但其 $O(mn)$ 的时间复杂度和 $O(mn)$ 的空间复杂度在处理大规模数据时变得不可行。特别是当 m 与 n 的数量达到万级别时，传统的 LCS 实现需要百兆以上的内存空间，这对现代 Web 应用来说是不可接受的。滚动数组技术 [2] 可以降低空间复杂度至 $O(\min(m, n))$，但实现复杂且难以维护。此外，完全匹配算法在处理部分修改的场景时效率不高。

分布式方向有研究在改进 LCS 的海量数据的差异检测 [3] 和一致性算法评估 [4]。这些研究都着眼于强一致性和差异检测方向，并不能降低算法的复杂度。

在数据库领域，维护有序列表是一个经典问题。B 树 [5] 及其变种为大型有序数据集提供了高效的索引结构。最近，Delta-CRDT[6] 等基于 CRDT（conflict-free replicated data types）的方法为分布式环境下的列表维护提供了新思路。但这些方法主要关注底层存储结构，而非优化同步过程。本文

1. 中国民航信息网络股份有限公司 北京 101318

从应用层出发，为上层应用提供了更轻量级的解决方案。表1总结了相关算法的主要特征对比。

表1 相关算法研究对比

算法类别	时间复杂度	空间复杂度	适用场景
Delta-CRDT	$O(n)$	$O(n)$	分布式系统
LCS	$O(mn)$	$O(mn)$	最长公共子序列
LCS- 滚动列表	$O(mn)$	$O(\min(m, n))$	最长公共子序列
本算法	$O(n)$	$O(n)$	高相似有序列表合并

通过比较可以看出，本算法在保持线性时间复杂度的同时，兼具低空间开销和特定场景优化的特点。

2 问题描述

在现代 Web 系统应用中，维护客户端与服务端之间的有序数据列表同步是一个普遍存在且极具挑战性的问题。该问题的核心在于如何在资源受限的条件下，高效地合并两个可能存在差异的有序列表，同时保证数据一致性和系统性能。

2.1 问题定义

设用户在某客户端（如移动 App）上维护一个有序数据项列表 Params=[$p_1, p_2, \ldots, p_n$]，同时在服务端数据库存储的有序列表为 DB=[$d_1, d_2, \ldots, d_m$]，其中：

（1）每个数据项包含唯一标识符（如 itemID）和排序标识符（如 orderID）。

（2）客户端和服务端的列表可能存在部分差异（某些项仅存在于某一端）。

（3）用户可能对列表进行顺序调整（如将末尾项移至首部），但是其最频繁的操作在于置顶、置低等首尾数据的顺序调整。

目标是在保持 Param 列表顺序的情况下合并 DB 列表，并最大条件降低 DB 列表的 orderID 字段调整，同时保证 DB 列表的顺序性。

2.2 关键约束

（1）顺序敏感性，数据项的相对顺序至关重要，例如：在音乐播放列表中，歌曲顺序决定播放顺序；在任务管理看板中，卡片顺序影响工作流程。传统哈希比对方法失效，即使数据项相同，顺序变化仍需正确同步。

（2）部分数据差异，客户端和服务端的列表可能不完全相同，用户在客户端顶部新增了数据项（Params 中存在，但是 DB 中不存在）；服务端被其他操作增加了数据项（DB 中存在，但 Params 中不存在）。算法需同时处理数据增加和顺序调整，不能仅依赖简单的增量同步策略。

（3）高相似性假设，在大多数实际场景中，两个列表的差异较小（80% 以上的数据项相同）。差异通常集中在列表的头部或尾部。若相似度过低（如 <50%），算法需退化为全量覆盖策略。

（4）资源限制，算法时间复杂度和空间复杂度需控制在 $O(n)$ 甚至更低；低数据库写入压力，目标是将操作减少 85% 以上。

（5）顺序字段非连续性假设。即假设数据的 orderId 字段可以为非连续的，可以为负数，其值仅仅用于指示顺序，无业务含义。

客户端与服务端需求方法通常采用以下两种策略：

（1）全量覆盖策略：直接将整个 Params 列表覆盖 DB 列表，这导致 $O(n)$ 的数据库写操作，当 n 较大时（典型场景 n>1 000）会产生严重的性能瓶颈。

（2）增量更新策略：通过比较两个列表的差异，仅更新发生变化的部分。然而，这种方法在处理顺序调整时效率低下，特别是当用户对列表进行了大规模重排序时（如将末尾项移至首部），实际上需要更新几乎所有项的排序标识。

现有解决方案难以同时满足约束条件，特别是在处理大规模列表时。例如，经典的序列比对算法虽然可以准确识别差异，但其 $O(n^2)$ 的空间复杂度使其无法应用于实践；而基于哈希的快速比对方法又无法正确处理顺序变化。这些需求使得设计一个高效的同步算法变得更具挑战性。

3 算法设计

本文算法核心思想是通过识别 Params 和 DB 中的最长连续相同子序列（称为 SUPER 区间），仅对区间外的元素进行必要的 orderID 调整，从而减少数据库写操作和运行内存。详细步骤如下：

（1）相似性检测

①首先计算客户端（Params）和服务端（DB）列表的相似度。

②若相同项占比 >80%，进入优化路径；否则退化为简单合并。

③简单合并以 Params 顺序为准，DB 列表多余数据置于末尾。

此算法会导致 DB 中一些关键的排序靠前的数据丢失其正确的顺序。

（2）锚点选择

①在 Params 中 40% 位置 P 选择锚点项 A，记为 R。

②检查 R 是否存在于 DB，如果不存在，进一步在 40%

+5% 的位置选择 A'，记为 R。

③重复②的操作，直到检测 R 在 DB 中存在或者当前位置 >60%（最多选择 5 次锚点）。

④若 $R \in$DB，记录位置 DB 项 R 在的位置 Q，以及 R 的位置 P。

⑤否则，退化为简单合并算法（以 Params 顺序为标准，DB 多余的数据放在序列最末端）。

（3）SUPER 区间识别

①以锚点 P（Params 中的位置）和 Q（DB 中的位置）为中心，分别在列表中双向扩展（向前、向后遍历），直到发现不匹配的项，最终得到 SUPER 区间：$[P_{start}, P_{end}]\times[Q_{start}, Q_{end}]$。SUPER 区间的含义就是一个基于锚点选择的公共子序列（不一定为最长，但是在假设约束下有较高概率为最长）。

②如果 $P_{end} - P_{start}/\text{size(Params)}>30\%$，退化为简单合并算法，否则进入分区处理。

（4）分区处理

① SUPER 区间内保持 DB 数据的 orderID 不变。

② SUPER 区间上方：

a. DB 列表中对应位置 $<Q_{start}$ 的项，检测是否在 Params 中存在，如果不存在，则紧贴 SUPER 区间上界调整其 orderID，最终到 DB 中的第一项，其 orderID 记为 upper$Q$。

b. Params 列表中对应位置 $<P_{start}$ 的项，在 upper$Q$ 基础上递减 orderID。

③ SUPER 区间下方

a. DB 列表中对应位置大于 Q_{start} 的项，检测是否在 Params 中存在，如果不存在，则紧贴 SUPER 区间下界调整其 orderID，记为 lower$Q$。

b. Params 列表中对应位置大于 P_{end} 的项：在 lower$Q$ 的基础上递增 orderID。

（5）生成最终列表

非公共区间外的数据即为差异数据，即需要保存的数据。

4 实验评估

在实际生产环境抽取出 10 000 个用户的 APP 操作参数和对应数据库数据过敏后对该算法进行了全面测试，从多个维度与传统方法进行对比分析。测试环境配置：应用服务器使用 Intel Xeon Gold 6248R CPU @ 3.00 GHz，16 GB 内存，CentOS 7.9 操作系统；数据库使用 Oracle 11g 版本，SSD 硬盘。

4.1 数据集说明

本文构建了 3 个不同规模的数据集进行测试，小规模数据集的 Params 列表 20~100 个项，模拟普通用户场景；中规模数据集的 Params 列表 300~1 000 个项，模拟活跃用户场景；大规模数据集的 Params 列表 2 000~10 000 个项，模拟重度专业用户场景。其中每个数据集包含三种修改模式：轻度修改为数据差异在 0.1%~1% 的变动，表示日常调整；中度修改为数据差异在 1%~3% 的变动表示列表部分重构；重度修改为数据差异在 5%+ 变动，表示用户列表数据整体重构。这种多层次的测试设计能够全面评估算法性能，确保测试结果具有实际参考价值。

4.2 多维度对比指标

本文从性能效率（执行时间）、数据库写操作对比（实际执行的数据库操作数量）、并发性能对比（使用 JMeter 进行压力测试）3 个关键维度进行系统化评估，实验结果如表 2~4 所示。

从表 2 可以看出，3 种算法的时间复杂度随数据规模增长呈现显著差异。简单合并算法呈现近似线性增长 $O(n)$，但其基础耗时较高。表现出典型的 $O(n^2)$ 特征，在千级数据时耗时最高已达 1 000 ms 以上。优化算法保持稳定的亚线性增长，千级数据下仅 50 ms 左右，验证了其设计的线性时间复杂度优势。

表 2 执行时间对比（均值 ± 标准差）

单位：ms

数据规模	修改程度	执行时间		
		简单合并	LCS- 滚动列表	优化算法
小规模	轻度	2.2±2.5	2±0.8	0.9±0.3
小规模	中度	2.8±0.6	3.5±0.4	2.9±0.7
中规模	轻度	9.5±5.1	22.1±3.1	2.0±0.7
中规模	中度	9.5±5.9	22.8±3.5	7.8±1.6
大规模	轻度	78.8±25.9	807±312	3.5±0.8
大规模	中度	78.9±26.8	806±305	49.4±22

通过表 3 的数据库写操作对比显示，在中规模数据集（300~1 000 项）的中度修改场景下，传统全量更新需 682 次写操作，本算法只需要修改 46 次，平均降低 93%。

表 3 中度修改条件下数据库写操作统计对比

数据规模	方法	平均写操作次数
中规模	简单合并	682
	LCS- 滚动列表	43
	优化算法	46
大规模	传统方法	3 572
	LCS- 滚动列表	176
	优化算法（轻度）	182

需要注意，LCS 算法由于每次都是准确寻找最长连续公共序列，本算法是启发式寻找，不能保证每次寻找的都是最长连续的，所以在数据库次数方面略低于 LCS 算法。

表 4 的压力测试结果显示，算法具有显著的并发优势，在 500 并发场景下仍可以保持 818 的 TPS，显著优于其他两种算法。

表 4 大规模轻度修改条件下每秒事务处理量（TPS）对比

并发用户数	传统方法 TPS	LCS- 滚动队列 TPS	优化算法 TPS
50	212	15	1 028
100	209	14	1 026
200	153	12	988
500	91	8	818

最终通过实验的对比观察下，本算法具有 3 种显著特性：

（1）数据规模适应性：算法性能与数据规模呈近似线性关系，在 100 000 只规模的测试中仍保持稳定。

（2）修改模式适应性：对头尾修改（如拖动分组）效率最高，对随机分散修改仍有较良好表现。

（3）硬件资源利用率：基本不限制硬件资源，在较低硬件配置下仍可以在合理时间返回。

5 结论与展望

本文提出了一种高效的动态列表排序同步优化算法，通过创新性地结合启发式锚点检测和 SUPER 区间识别技术，成功解决了有序列表同步的性能瓶颈问题。与传统方法相比，本算法在性能和资源方面具有显著优势。在保持 $O(n)$ 时间复杂度的同时，减少了 90% 以上的数据库写操作，显著降低了系统负载，提升了系统吞吐率。同时，算法具有领域通用性。虽然起源于民航领域的航班自选排序问题，但算法设计具有普适性，可以扩展到金融领域自选股管理、娱乐领域播放列表管理、电商领域收藏夹等多个场景使用。

基于当前研究成果，未来有几个方向值得进一步探索：

（1）机器学习增强的锚点选择 [7]。当前算法采用固定比例（40%~60% 之间每 5% 取一次锚点）的启发式锚点选择策略。未来可以引入轻量级机器学习模型，通过学习用户的历史编辑模式，动态优化锚点选择位置。例如，对于习惯从列表中部开始编辑的用户，可以自动调整锚点检测区域。

（2）多设备协同同步。扩展到更复杂的多设备同步场景，研究如何在本算法基础上实现跨多个终端的实时协同排序。可以结合 CRDT 等分布式数据类型理论，解决并发修改带来的冲突问题。

基于研究成果，本文提出以下实践建议：对于弱一致性的列表同步操作，建议优先考虑本算法而非传统同步方案。在实现时，可以设置相似度阈值（如 80%）为可配置参数，方便针对不同场景调优，对于关键业务系统，可以采用算法组合策略。

参考文献：

[1]HIRSCHBERG D S. Algorithms for the longest common subsequence problem[J]. Journal of the ACM, 1977, 24(4): 664-675.

[2] 刘凤鸣 . 动态规划的应用 [J]. 科技视界 ,2013(7):40-41.

[3] 王树刚 , 郭涛 , 朱超 , 等 . 基于 FastText 模型和 LCS 匹配算法的虚端子自动关联方法 [J]. 电力信息与通信技术 , 2025, 23(4):70-76.

[4] 韦涛 , 潘阳 , 张小辉 , 等 . 基于窄带弱连接条件下的数据同步方法 [J]. 信息化研究 ,2024,50(6):40-45.

[5]BAYER R, MCCREIGHT E. Organization and maintenance of large ordered indices[C]//SIGFIDET '70: Proceedings of the 1970 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control. NewYork:ACM,1970:107-141.

[6]ALMEIDA P S, SHOKERET A, BAQUERO C. Delta state replicated data types[J].Journal of parallel and distributed computing, 2018,111:162-173.

[7] 吴沁停 , 封与哲 , 潘金艳 , 等 . 基于锚点加速机制的聚类算法综述 [J/OL]. 上海交通大学学报 ,1-37[2025-05-15]. https://doi.org/10.16183/j.cnki.jsjtu.2024.425.

【作者简介】

霍淑华（1990—），男，山东滨州人，硕士，研究方向：算法设计与数据库优化，email:huoshuhua@travelsky.com.cn。

田丰（1980—），男，北京人，硕士，副高，研究方向：云计算、大模型应用，email: tianfeng@travelsky.com.cn。

李颖（1977—），女，北京人，本科，副高，研究方向：民航运价管理，email: liy@travelsky.com.cn。

（收稿日期：2025-05-15 修回日期：2025-09-29）