许祥, 陈庆奎. 一种均衡的大规模数据流存储机制[J]. 智能计算机与应用,2025,15(12):151-156. DOI:10.20169/j.issn.2095-2163.251123

一种均衡的大规模数据流存储机制

许 祥, 陈庆奎
(上海理工大学 光电信息与计算机工程学院, 上海 200093)

摘 要: 为了避免造成数据库瞬时存储压力过大和系统存储负载不平衡等问题,针对海量并发数据流存储请求,本文提出一种面向并发数据流处理的存储机制。通过消息队列的异步传递缓解高并发数据阻塞问题,针对均衡存储并发数据流数据,将数据流进行数据预处理为任务单元,提出基于任务分配的一致性哈希算法实现任务单元的动态分配及处理节点间的负载均衡;同时使用 HBase 集群进行数据存储和持久化,为系统提供了可靠的存储基础。实验结果表明,该存储方法能够在持续的并发数据流下实现存储系统的负载均衡,保证了系统存储的稳定性,能够提高系统存储吞吐量。

关键词: 并发数据流; 负载均衡; 一致性哈希; 存储机制
中图分类号: TP391.9 **文献标志码**: A **文章编号**: 2095-2163(2025)12-0151-06

A balanced large-scale data streaming storage mechanism

XU Xiang, CHEN Qingkui
(School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China)

Abstract: Aiming at the massive concurrent data stream storage requests, in order to avoid the problems of excessive instantaneous storage pressure of the database and system storage load imbalance, this paper proposed a storage method for concurrent data stream processing. Through the asynchronous transmission of message queue, the problem of high concurrent data blocking is alleviated. In the system, the data preprocessing is taken as the task unit in the efficient processing of concurrent data stream data unit, and the consistent hash algorithm based on task allocation is proposed to realize the dynamic allocation of task units and realize the load balance between processing nodes. At the same time, the HBase cluster is used for data storage and persistence, which provides a reliable storage foundation for the system. Through experimental analysis, this storage method can realize the load balancing of the storage system under the continuous concurrent data flow, ensure the stability of the system storage, and improve the throughput of the system storage compared with the traditional method.

Key words: concurrent data streams; load balancing; consistent hash; storage mechanism

0 引 言

随着物联网和人工智能技术的高速发展,采集设备数量大幅增加,产生海量数据,需要实时计算并存储。一般来说,采集的数据通常都是时间序列数据即数据格式包含时间戳的数据,时间序列数据一般结构比较单一而数据量巨大[1]。每个终端设备产生的数据流包含唯一标识、时间戳、数据区和处理周期,当多个终端设备汇集就会形成并发数据流。并发数据流是指具有大量相同周期的数据流的集合[2]。传统存储方法是将并发数据流直接写入数据库,这样会给数据库造成巨大的存储压力,而面向数千个并发数据流的存储,如何使底层存储节点能够承受瞬时存储压力是当前面临的问题。

目前通用的关系型数据库越来越难以适应时间序列数据写入性能要求,主要体现在大量终端采集设备频繁采集数据写入关系型数据库导致存储效率低下。为了适应时序数据的存储,产生了时序数据库,比如 InfluxDB 等[3]。但这类数据库都局限于单机存储优化,而采用 InfluxDB 的原生集群方案较为单一,缺乏拓展能力。针对海量数据流,采用基于分布式数据库 HBase 存储是较为合适的方案,但对于

基金项目: 国家自然科学基金(61572325); 上海重点科技攻关项目(19DZ1208903)。
作者简介: 许 祥(1999—),男,硕士研究生,主要研究方向:数据流存储。
通信作者: 陈庆奎(1966—),男,博士,教授,博士生导师,主要研究方向:计算机集群,并行计算,人工智能等。Email:chenqingkui@usst.edu.cn。
收稿日期: 2024-02-18

不同业务场景的需求,建立存储机制是当前热门的研究方向[4-5]。文献[6]针对时序数据存储引发的负载倾斜问题,提出基于冷热数据分区以及访问行为的负载均衡优化策略,较好地降低了数据倾斜度;文献[7]设计分层次的分布式存储方案,提出基于 HBase 构建两层存储架构,应对海量数据存储问题。

面向大规模 AI 数据流处理的集群系统,本文主要解决上游的高速并发数据流和下游低速写入 HBase 存储节点之间的速度不匹配问题,基于一致性哈希算法的任务分配算法,将数据流封装成的任务单元均衡传递给处理节点,有效维持了系统的负载均衡和稳定性,也提升了 HBase 存储集群的存储性能。

1 相关工作

流数据处理存储方面,文献[8]提出了一种基于 HBase 的交通流数据实时存储系统,在一定程度上解决了海量数据的实时存储,但可能存在处理大规模数据流的性能瓶颈;文献[9]提出利用缓存窗口阈值的面向流数据实时存储方法,能够满足数据低中速率的实时存储,对于高速数据流的处理有一定的局限性;文献[10]提出基于轮转数据桶模型,利用多缓存存储数据流延迟写入,缓解存储节点的写入压力,提供了不同的处理思路。基于上述的研究工作,针对大规模并发 AI 数据流的场景,通过设计相对应的存储机制能够有效缓解 AI 数据流的并发存储压力,提高数据流的实时存储性能。

在分布式系统中,任务分配涉及到如何有效地将任务分配给每个处理节点。经典的任务分配算法包括随机分配算法、轮询算法等。轮询任务分配算法以轮询的方式,将任务分配到不同的处理节点,实现较为简单,容易导致处理节点间负载不均衡现象[11]。另一种任务分配算法:一致性哈希算法,哈希范围是值在 $0\sim2^{32}$ 之间,形成环状结构,先采用哈希函数计算处理节点的哈希值,再根据每个数据关键字通过哈希函数将该数据关键字映射为某个值,如果该值没有映射到处理节点上,那么顺时针查找到第一个处理节点,即为目标处理节点。文献[12]在服务器集群系统中应用一致性哈希算法动态分配策略,实现了系统良好的负载均衡性;文献[13]基于一致性哈希算法在电力企业分布式数据存储中有效降低了存储数据延迟,保证了数据完整性。根据上述实践,将一致性哈希算法引入数据流任务单元分配处理节点的场景,通过该算法使处理节点能稳定高效处理任务单元。

本文采用一致性哈希算法分配任务单元到处理节点上,可能存在处理节点在哈希环上分布不均匀,带来数据倾斜问题,因此引入了虚拟节点的概念。虚拟节点允许一个物理节点对应多个虚拟节点,从而在环上占据多个位置,可以确保数据在环上更均匀分布,减少节点之间的数据不均衡。同时当一个物理节点故障时,也要从环中删除所有虚拟节点。

2 模型设计

面向公交视频识别及智能分析,通过实时并行化处理海量公交车图片,计算分析公交车厢内的实时拥挤度。AI 复合处理模块由多个 AI 复合处理单元组成,每个 AI 复合处理单元包括数据抓取、AI 模型计算以及状态监控各部分组成。AI 复合处理模块是数据流的来源,会根据数据抓取的特点产生多条相应的周期性数据流。

2.1 模型架构

根据上述系统的 AI 复合处理模块产生的数据流设计如图 1 所示。存储模型架构主要包括以下部门:

(1)数据源层

AI 复合处理单元包括图片抓取、AI 模型计算以及状态监控模块,形成一个综合性的处理单元。系统运行时,AI 复合处理单元生成 3 种数据流:图片流、结果流和监控流。图片流主要是图片抓取,产生的图片描述信息;结果流是图片经过 AI 模型计算输出的结果,包括各种指标值和计算结果;监控流包含对模型计算节点状态的实时监控数据。每个 AI 复合处理单元都会在不同的周期内生成这 3 种不同的数据流。

(2)消息队列

消息队列是用于处理异步消息传递的通信机制,不同类型的数据可以以不同的消息格式发送到队列中,而消费者则根据需要选择性订阅并处理消息。当多源数据流进入消息队列,可以异步发送到数据预处理层,常见的消息队列有 Redis、MabbitMQ 等。

(3)数据预处理层

从消息队列中获取数据流的数据单元,根据数据的结构信息进行分类,将到达的数据单元组成对应的任务单元,形成多个任务单元后进行后续任务分配,交给处理节点,提高存储效率。

(4)均衡分配

对于预处理层已缓存的任务进行任务分发,设计实现了一种基于一致性哈希任务分配算法,构建

任务单元映射处理节点表，实现任务单元与处理节点之间的匹配映射，同时保证处理节点间的负载均衡，提高系统稳定性。

(5)数据管理层

数据管理层由多个处理节点构成，其核心任务是根据上游任务分配算法接收任务单元，通过高效的缓存机制，对任务单元进行暂时存储和管理，再利用多线程并发的方式将数据写入存储层，能够更加高效的应对大规模数据流和任务处理需求。

(6)数据存储层

HBase 是一个基于 Hadoop 的分布式数据库，其架构允许数据水平切分存储，底层存储节点通过 HBase 搭建集群，将数据均匀分布在多个节点上，能够实现存储负载均衡和高效访问需求[14-15]。

(7)数据查询模块

根据用户查询，优先考虑在任务单元映射的处理节点缓冲区内进行检索，如果未能在该缓冲区找到相应数据，才会转向 HBase 集群进行进一步的查询。

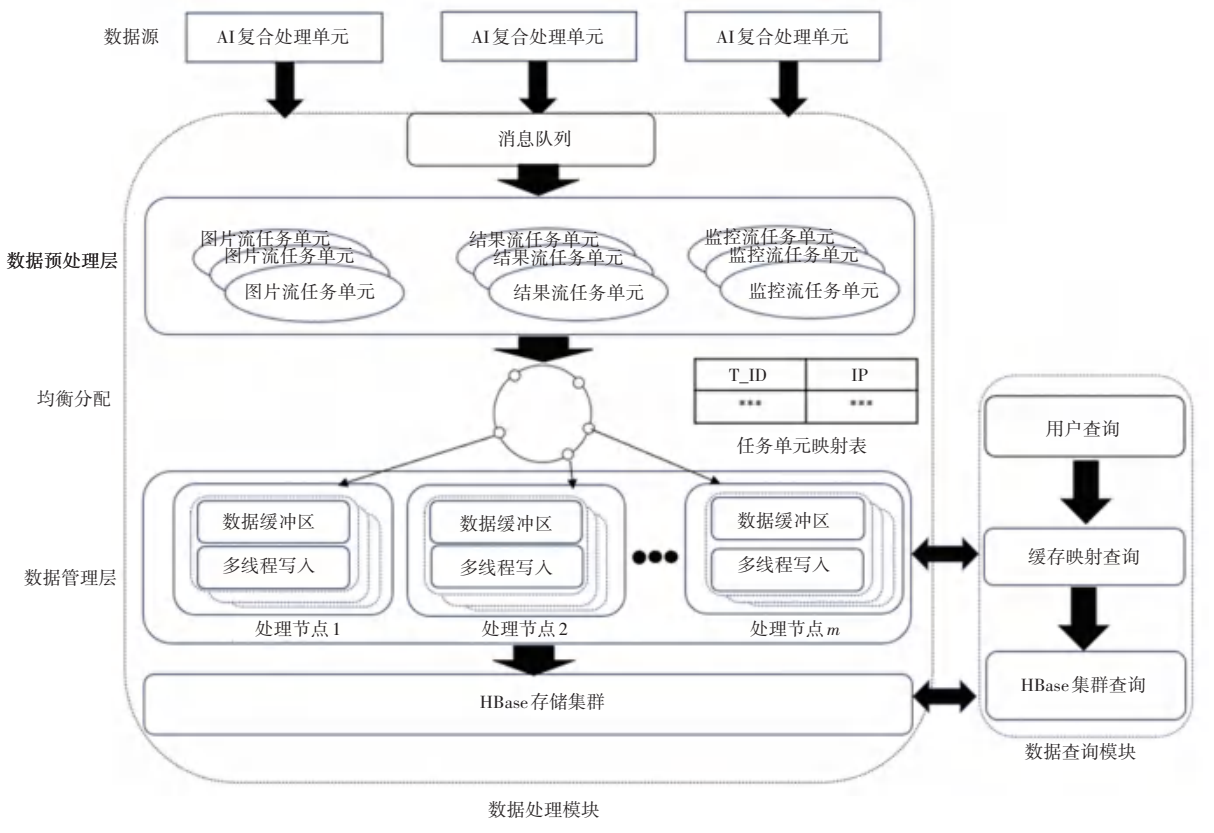

图 1　存储模型架构

Fig. 1　Storage model architecture

2.2　数据预处理层

每个 AI 复合处理单元会产生不同周期、不同流元素的数据流进入到消息队列中，将不同数据流的流元素进行分类打包，能够提高数据处理的效率。

(1)AI 数据单元(数据流编号，数据类型，周期时间，生成时间，数据)

(2)任务单元(编号，数据单元个数，长度，类型，周期，生成时间，数据单元内容)

设计数据单元分类打包为任务单元算法：

算法 1　任务单元算法

1) 当数据单元进入消息队列中，根据数据单元生成时间的顺序队列进行数据分类，相同的数据单元可以打包为同一任务单元，可以预定义每个任务单元容纳数据单元个数和大小限制；

2) 根据当前任务单元剩余容纳个数，若有剩余则添加，否则分配其他同类型任务单元，同时设置当前任务单元周期为最小数据单元周期并记录当前任务单元大小；

3) 当前任务单元中数据单元个数达到阈值，则交给后续任务分配处理；

4) 返回步骤 1 继续执行。

2.3　均衡分配层

建立任务单元与处理节点的分配关系。在分配时应考虑处理节点间的负载均衡，本文提出了一致性哈希算法的任务分配算法。

设处理节点的缓冲区大小为 M，SIZE 表示任务

单元的大小,统计每个处理节点缓冲区任务单元的总个数 L, 则缓冲区负载率为:

$$D = \frac{L \times \text{SIZE}}{M} \times 100\% \quad (1)$$

综合考虑 CPU 利用率、内存利用率和缓冲区负载率, 对于每个处理节点 i 的负载评价指标, 公式如下:

$$P_i = (a \times C_i + b \times M_i + c \times D_i) \times 100\% \quad (2)$$

其中, C_i 为第 i 个处理节点的 CPU 利用率; M_i 为第 i 个处理节点的内存利用率;D_i 为第 i 个处理节点的缓冲区负载率;a、b 和 c 分别表示 CPU、内存和缓冲区负载率在整体的权重。

每个处理节点的初始节点个数 N_i 之比:

$$N_1 : N_2 : \cdots : N_i = \frac{1}{P_1} : \frac{1}{P_2} : \cdots : \frac{1}{P_i} \quad (3)$$

每个处理节点设置虚拟节点个数为 $N_i - 1$。相同物理节点的虚拟节点在同一组,根据 IP 地址划分物理节点。当在环中寻找到虚拟节点即可根据虚拟节点映射表查找到物理节点,虚拟节点映射表如图 2 所示。

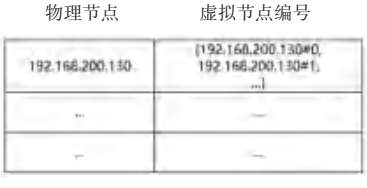

图 2 虚拟节点映射表

Fig. 2 Virtual node mapping table

算法 2 任务分配算法

1) 初始化节点列表:初始化处理节点个数并引入虚拟节点,确定每个处理节点的虚拟节点个数;

2) 处理节点映射:设计每个处理节点及虚拟节点组合字符串(IP 地址+编号)采用 FNV(Fowler-Noll-Vo)哈希算法计算映射到环上,并且构建虚拟节点映射表;

3) 任务单元处理:根据任务单元生成时间(TIME_STAMP)对任务单元进行排序,进入等待队列,若生成时间相同,则根据任务周期大小(T_Period)排序进入队列,构建任务单元映射到处理节点表,记录任务单元和处理节点的映射关系;

4) 任务单元映射:将任务单元根据任务单元编号(T_Id)使用相同的哈希算法映射,在哈希环中找到对应的处理节点分配任务,并实时添加到任务单元映射表,继续执行步骤 3。

标准差是一种衡量数据分散程度的统计指标,通过计算各节点上某个负载指标的标准差,可以评估节点之间的负载均衡情况,公式如下:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (P_i - \mu)^2}{n}} \quad (4)$$

其中, σ 为系统负载评价指标的标准差; n 为当前节点总个数; μ 为系统处理节点负载评价指标的平均值,公式如下:

$$\mu = \frac{\sum_{i=1}^{n} P_i}{n} \quad (5)$$

通过设置标准差的阈值进行动态负载均衡,当系统的负载状况指标超过阈值,此时系统根据所有处理节点的负载状况指标进行虚拟节点个数再分配,有效平衡处理节点的负载,维持系统的稳定性。

算法 3 动态负载均衡算法

1) 定时监测:定期检查系统负载均衡状况,当系统中某个节点故障或失效时,将故障节点以及虚拟节点从环上移除,删除虚拟节点映射表,如果发现负载评价指标超过预定义的阈值,执行步骤 2;

2) 负载排序:当超出系统预先设定的阈值,则将每个节点的负载指标进行排序,执行步骤 3;

3) 虚拟节点再分配:对于负载指标最高的节点,可以将某一个虚拟节点转移给下一顺序的节点,同时更新虚拟节点映射表。

2.4 数据管理层

在系统运行时,每个处理节点都需要处理来自上游的多个任务单元,这些任务单元通过哈希环上的虚拟节点映射到每个处理节点,因此每个节点都可能同时接收到多个任务单元。

对于每个处理节点,根据其性能和硬件配置,设置了预定义大小的缓存区,性能较高的节点可以配置更大的缓存区,以容纳更多的任务单元。每个缓存区都记录着接收到的任务单元的个数和大小,并实时计算缓存区的负载率,这样的设计有助于监控节点的负载情况 ,用于上游均衡分配任务单元进行调整,保持节点的负载均衡状况。

针对已经在缓存区中的任务单元,采用了多线程方式写入数据存储层。通过线程池技术,可以灵活管理和调度多个线程,以最大限度地利用处理节点的计算资源。每个线程独立处理一个任务单元,从中提取数据单元并进行持久化处理,这种并发处理方式提高了数据的处理效率,使得系统能够更快地响应和处理大规模的数据流。

根据系统负载和业务需求情况,可以动态增加处理节点,有效分担系统当前节点的负载压力,提高

系统的整体性能和吞吐量。通过将新节点动态映射到哈希环上，使上游数据有效地分配到新节点进行处理，实现系统的可扩展性。

2.5　数据查询模块

在本文的应用场景中，一般会在特定的时间窗口内处理数据。查询请求通常包含时间属性范围，例如，实时查询当前时刻的公交车的拥挤度情况。优化查询流程的策略：查询当前时刻的任务单元映射表中的处理节点缓冲区记录，如果在该缓冲区内找到所需数据，则查询成功；反之，则转向 HBase 集群进行查询。

在 HBase 数据库中，整张表包括了行键(RowKey)、列族、列以及时间戳等几个重要部分。对数据库进行访问最主要是通过行键进行访问，如果设置的行键无法满足检索条件，或用户使用的检索条件不匹配时则会进行全表扫描，查询效率大大降低。因此，针对数据单元特点设计合适的行键有利于提高查询效率。

设计 RowKey 需要遵循唯一性、散列性、简短性、可读性。本文选用 MD5(Message Digest 5)算法可满足散列性、唯一性。通过 MD5 算法可以将任意长度的数据加密成固定长度，并且加密结构不可逆，同时具有较高离散度，避免写入热点问题，公式如下：

$$\text{RowKey} = \text{MD5}(\text{S\_id})\%16 + \text{Long.MAX\_VALUE} - \text{TIMESTAMP} \tag{6}$$

其中，S_id 表示数据单元编号。

由于 RowKey 按照 ASCII 排序原则，Long.MAX_VALUE-TIMESTAMP 使用递减时间戳可以保证最后插入的数据可以先查询到，符合当前应用场景，加快了数据实时查询速度。

客户端查询 HBase，一般有 3 种方式：

(1)通过 GET 方式。HBase 系统提供了单条 GET 数据和批量 GET 数据，单条 GET 通常是通过请求表名+RowKey，批量 GET 通常是通过请求表名+RowKey 集合来实现；

(2)通过 SCAN 方式，设置 startRow 和 stopRow 参数进行范围匹配；

(3)全表扫描，即直接扫描整张表中所有行记录。

3　实验与分析

3.1　实验环境

本实验抓取某城市公交车的实时监控图片流，通过 AI 复合处理单元服务器产生周期性数据流。部署了 8 台服务器，操作系统为 CentOS 7.9.2009(Core)，CPU 型号为 AMD Opteron(tm) Processor 6320，8 核，16 G 内存，40 G 硬盘，其中处理节点个数为 3 个，HBase 集群环境使用 Hadoop 3.1.3，副本数为 3，Zookeeper 3.5.7，HBase 2.0.5。初始时，处理节点的负载状况评价指标影响因子 a、b、c 分别设置为{0.3,0.3,0.4}。

3.2　实验过程和分析

3.2.1　缓存区分析

连续比较某 10 个周期内，系统每个处理节点的缓存区任务单元数量及缓存区负载率变化。

不同节点缓存区分析如图 3 所示，每个周期到达的数据流任务中，通过基于一致性哈希算法的任务分配，3 个处理节点的任务分配数量呈现整体均匀分布的特征。一致性哈希算法通过将任务单元映射到哈希环上的节点，保证了节点的均匀分布，同时各个缓存区负载率一直在 50%~70%范围内，对于动态环境中的任务分配表现出良好的适应性，当有新的任务到达或处理节点发生变化时，系统能够动态调整任务的分配，确保各个节点的负载相对平衡。

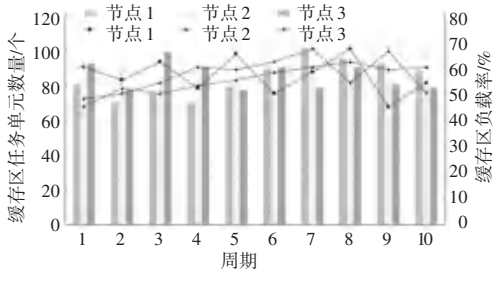

图 3　不同节点缓存区分析

Fig. 3　Different node cache analysis

3.2.2　系统负载评价指标

采用了轮询法、随机分配法以及本文提出的一致性哈希算法方案进行对比分析。连续比较了某 10 个周期内，不同任务单元分配方案对系统负载评价指标的影响，如图 4 所示。可见在不同的任务单元分配方案下，系统的负载均衡指标会受到不同影响，本文提出的任务单元分配方案相较于传统的轮询法和随机分配法，表现出更为明显的优势。随机分配法采用随机的方式将任务单元分配给处理节点，可能导致节点之间负载不均衡，难以保证任务的合理分配，在高并发环境下，随机分配法容易出现部分节点负载过重，系统整体性能下降的情况；轮询法作为一种基本的任务分配方式，按照固定的顺序将任务单元分配给不同的处理节点，无法适应动态变化的负载状况，也会影响系统整体的性能；本文提出的一致性哈希算法通过在哈希环上均匀分布节点，实现了对任务单元的高效分配，同时也可以根据系统负载状况进行动态调整，能够实现节点间负载均衡，提高了系统的整体性能。

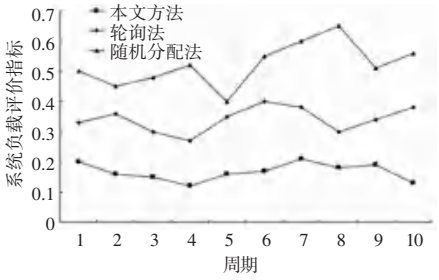

图 4 不同算法下系统负载评价指标对比

Fig. 4 Comparison of system load evaluation indexes under different algorithms

3.2.3 系统吞吐量

使用本文所提方案和传统存储方案在 8 种不同数据流任务速率下存储吞吐量对比实验,存储吞吐量的变化情况如图 5 所示。可见低速数据流任务下,本文所提出的存储方案与传统存储方案均能够满足实时存储需求。然而,随着数据任务速率的提高,本文所提存储方案在高速数据流环境中表现出明显的吞吐量优势,相对于传统存储方案更为高效可靠。本文的存储方案基于一致性哈希算法任务分配,能够有效地将任务分配到每个处理节点,充分发挥集群的并行处理能力,而传统存储方案在面对高速数据任务时可能遇到性能瓶颈。

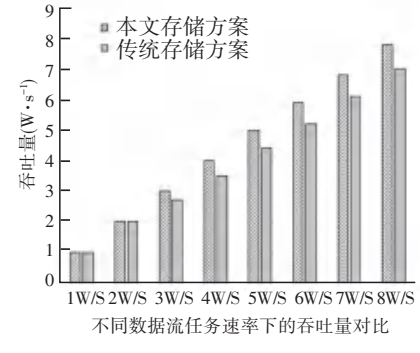

图 5 对比传统存储吞吐量结果图

Fig. 5 Comparison of traditional storage throughput results

4 结束语

针对处理海量并发数据流实时存储面临的问题,本文提出了一种均衡的存储机制,包括数据预处理模块、均衡分配模块,数据管理模块以及数据查询模块,旨在改善传统存储方案的性能,并使系统能够更好地应对庞大的数据流。首先,通过消息队列将数据源的数据流单元传递给数据预处理模块;其次,预处理模块对来自多源的数据流进行接收与解析,并根据其类型信息进行分类,构建相应的任务单元;均衡分配模块基于一致性哈希算法,实现了对任务单元的均衡分配,确保每个处理节点负载均衡,提高了系统整体的处理效率;数据管理模块通过多线程并发处理,使得系统能够以高吞吐量实时存储大规模数据流,满足实时性和稳定性的要求;最后,通过数据查询模块能够进行数据实时查询,满足当前项目的应用场景。

本文存储模型能够应对海量并发数据流的问题,还能够提高系统吞吐量,降低系统的写入压力,为提升数据流处理能力和数据存储系统性能提供了有益的方法和思路。

参考文献

[1] WANG C, HUANG X, QIAO J, et al. Apache IoTDB: Time-series database for internet of things[C]// Proceedings of the Very Large Data Bases Endowment. 2020: 2901-2904.

[2] 涂聪, 陈庆奎. 面向 AI 数据流处理的边缘 GPU 集群通信系统[J]. 小型微型计算机系统, 2022, 43(6): 1147-1153.

[3] RHEA S, WANG E, WONG E, et al. Littletable: A time-series database and its uses [C]//Proceedings of the 2017 ACM International Conference on Management of Data. New York: ACM, 2017: 125-138.

[4] WANG K, LIU G, ZHAI M, et al. Building an efficient storage model of spatial - temporal information based on HBase [J]. Journal of Spatial Science, 2019, 64(2): 301-317.

[5] AZQUETA-ALZÚAZ A, PATIÑO-MARTINEZ M, BRONDINO I, et al. Massive data load on distributed database systems over HBase [C]// Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). Piscataway, NJ: IEEE, 2017: 776-779.

[6] 杨力, 陈建廷, 向阳. 基于 HBase 的工业时序大数据分布式存储性能优化策略[J]. 计算机应用, 2023, 43(3): 759-766.

[7] 陈庆奎, 周利珍. 基于 HBase 的大规模无线传感网络数据存储系统[J]. 计算机应用, 2012, 32(7): 1920-1923.

[8] COLUZZI M, BROCCO A, CONTU P, et al. A survey and comparison of consistent hashing algorithms[C]// Proceedings of 2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). Piscataway, NJ: IEEE, 2023: 346-348.

[9] SUN Y, FANG J, HAN Y. A distributed real - time storage method for stream data [C]// Proceedings of 2013 10th Web Information System and Application Conference. Piscataway, NJ: IEEE, 2013: 314-317.

[10] 陆铭琛, 吕晏齐, 刘睿诚, 等. 基于水车模型的时序大数据快速存储[J]. 计算机科学, 2023, 50(1): 25-33.

[11] 韩朋花, 叶青, 姜晓明, 等. 改进加权轮询负载均衡算法研究[J]. 长春理工大学学报(自然科学版), 2018, 41(3): 131-134.

[12] 张开琦, 刘晓燕, 王信, 等. 基于动态权重的一致性哈希微服务负载均衡优化[J]. 计算机工程与科学, 2020, 42(8): 1339-1344.

[13] 曹熙. 基于一致性哈希算法的电力企业分布式数据存储研究[J]. 长江信息通信, 2022, 35(6): 147-149.

[14] 彭成辉. 基于 HBase 分布式数据库集群系统构建方法[J]. 信息技术与信息化, 2022(7): 95-98.

[15] ZHANG H, RONG-LI G A I. Distributed HBase cluster storage engine and database performance optimization[C]// Proceedings of 2021 IEEE 23rd International Conference on High Performance Computing & Communications. Piscataway, NJ: IEEE, 2021: 2274-2277.