汤浩, 吴若玲, 申莉, 等. 面向卫星遥测数据的高通量大数据存储系统设计与优化[J]. 智能计算机与应用,2026,16(7):100-106. DOI:10.20169/j.issn.2095-2163.24111803

面向卫星遥测数据的高通量大数据存储系统设计与优化

汤 浩[1,2], 吴若玲[3], 申 莉[1,2], 郭旦怀[3]

(1 北京轩宇智能科技有限公司, 北京 100019; 2 空天高性能处理器芯片北京市工程研究中心, 北京 100019;
3 北京化工大学 信息科学与技术学院, 北京 100029)

摘 要: 随着卫星通信数据量的爆发式增长,传统的数据存储技术越来越难以满足高通量卫星遥测数据存储需求。本文设计并实现了一种面向卫星遥测数据的高通量大数据存储系统(HTDSS)。该系统采用模块化架构,包括客户端组件、基于 Flink 的流处理程序、ClickHouse 分布式存储系统和可视化监控平台,支持集群横向扩展来提升吞吐量,能够支撑每秒百兆字节以上的数据存储需求,效率满足卫星应用需求,实现了高可用、可扩展和高吞吐等特性,高效支持大规模卫星数据的存储与处理。

关键词: 高通量; 分布式; 数据存储系统; 卫星数据

中图分类号: TP311 **文献标志码**: A **文章编号**: 2095-2163(2026)07-0100-07

Design and optimization of high throughput big data storage system for satellite telemetry data

TANG Hao[1,2], WU Ruoling[3], SHEN Li[1,2], GUO Danhuai[3]

(1 Beijing SunWise Intelligence Technology Co., Ltd., Beijing 100019, China;
2 Beijing Engineering Research Center of Aerospace High Performance Processor Chips, Beijing 100019, China;
3 College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China)

Abstract: With the exponential growth in satellite communication data, traditional data storage technologies are increasingly difficult to meet the requirements of high-throughput data. This paper designs and implements a High Throughput big Data Storage System (HTDSS) for satellite telemetry data. The system adopts a modular architecture comprising client components, a Flink-based stream processing program, a ClickHouse distributed storage system, and a visualization monitoring platform. It can support horizontal cluster expansion to improve throughput, achieving a storage capacity more than 100 megabytes per second. Its efficiency meets the needs of satellite applications and is significantly improved far higher than existing single machine performance, having achieved high availability, scalability and high throughput and other features, thereby efficiently supporting large-scale satellite data storage and processing.

Key words: high throughput; distributed system; data storage system; satellite data

0 引 言

随着新一轮信息化浪潮的到来和 5G 技术的发展与普及,全球的数字化信息呈现井喷式的增长,一个大规模生产、传播和应用数据的大数据时代已经开启[1-2]。在遥测卫星领域,采集手段的增加和每字节采集传输成本的下降使得卫星传感器元件数量与种类激增,数据传输量随之大幅上升,如气象卫星因气象状态的时空复杂性,其数据量更是呈几何式增长[3],远超传统存储和处理技术的承载范围。这些数据不仅包含多样化的非结构化信息,如遥测图像和气候数据等,而且对数据的实时性提出了更高的要求。此外,卫星大数据还展现出明显的流式传输特性,即从卫星发送、到地面接收、再到数据存储呈流式传输,这也对数据存储系统提出了新的要求。

过去几十年,基于安全考虑,卫星遥测数据存储主要采用单机实时存储与多机冷备份方式,受限于单机存储能力和吞吐量,大体量、多维度数据只能先

基金项目: 国家自然科学基金(42371476); 中央高校基本科研业务费资助项目(buctrc202132)。

作者简介: 汤 浩(1986—),男,高级工程师。主要研究方向:航天装备测控与健康管理研究与应用。

通信作者: 郭旦怀(1973—),男,博士,教授。主要研究方向:时空数据分析,大数据处理,数据智能。Email:gdh@buct.edu.cn。

收稿日期: 2024-11-18

存储后分析，导致时效滞后。而遥测数据分析，限于单机处理能力，只能采取基于阈值的方法。该方法高度依赖专家经验，存在较大的局限性。随着机器学习在 PHM 领域的广泛应用[4-7]，其对数据量的需求日益增大，单机存储显然难以满足这一需求。特别是对于高通量卫星数据，其数据包大小不等、信息量不均，每秒数据量巨大，单机存储不仅容易丢包，还无法实现实时解析。因此，研究如何利用分布式大数据技术来满足高通量卫星大数据存储需求无疑具有重要的现实意义。

本文从航天领域的卫星遥测数据存储出发，研究基于大数据架构设计高通量数据存储系统的可行性，并提出高效的设计方案以满足日益增长的卫星数据存储和查询分析需求，为今后的航天领域大数据技术发展提供支撑。

1　研究现状

在航天领域，随着在轨航天器增多，需要实时检测的遥测数据呈爆发式增长，单纯依靠人力以及专家经验对遥测数据进行实时监测已经不能满足现实的技术发展需求[8]。近年来，许多学者将目光转向了数据驱动下的 PHM 和数据处理[9]。并且已有学者采用 Hadoop、HBase 等技术实现对卫星数据的分布式数据挖掘和流挖掘[10]。针对面向多星多任务场景，国内航天学者设计了大数据处理系统，实现了秒级数据响应[11]。但目前实时数仓技术在航天领域应用较少，并且已有的流数据处理能力也无法达到毫秒级响应。面对航天大数据，NASA 建立了开源云服务平台 Nebula[12]，并且可以通过 Hadoop MapReduce 来处理月表图片[13]，但未见针对高通量遥测数据的数据存储系统。对于高通量数据的存储与处理，近些年也有众多学者展开研究[14-20]，主要针对高效、高吞吐等特性进行优化，提升系统性能。

2　系统设计

针对上述需求，本文设计了面向卫星遥测数据的高通量大数据存储系统，以下简称系统或 HTDSS（High Throughput Data Storage System）。本系统的针对场景是卫星突发性或持续性的高通量通信数据存储。在卫星通信方式上，数据压缩包首先会通过高频电磁波被地面接收站接收，之后经过软硬件方式会对数据进行解码解压。最后，解压后的数据通过 HTDSS 进行存储，HTDSS 需要提供一个存储接口或者包含数据存储 API 的客户端存储软件包。解压数据在 HTDSS 中还需要进行解析，并封装为基础数据类型在底层数据库上持久化存储。

2.1　系统设计目标

针对目前高通量卫星数据存储的需求，分布式高通量数据存储系统在设计指标上，需要能够接受 40 000～50 000 帧/s 数据包，每个数据包大小在 1.5～5.0 KB 之间，数据包内包含 90～200 个传感器和遥感参数以及参数的附加信息。系统能够以 100 MB/s 吞吐速率接收数据包，并对数据包进行解析，抽离出遥感参数信息，并将参数转换为可存储格式后存储在分布式 OLAP 数据库中。同时，数据表应该建立时间索引以加速后续的查询需求，能够提供方便的基于时间的查询方式。此外，通过资源管理框架、负载均衡和组件规划部署等手段，充分利用集群的硬件性能，最大化利用物理节点的固态硬盘（SSD）、机械硬盘（HHD）、CPU、内存及网络带宽，同时兼容廉价的低性能节点。

2.2　系统总体架构

HTDSS 的总体架构如图 1 所示，主要包含了四大服务层和 6 种开源软件和框架，分别是基于 Spring Boot 的客户端层、基于 Kafka 消息通信中间件的数据接入层、基于 Flink 流处理引擎的数据处理层和基于 ClickHouse 和 HBase 的数据存储层。在系统中 Zookeeper 起到分布式管理的角色，同时也是系统组件关键功能的依赖，如管理和感知 Kafka 成员、管理 ClickHouse 分布式表、选举 HBase 中的 HMaster 节点。

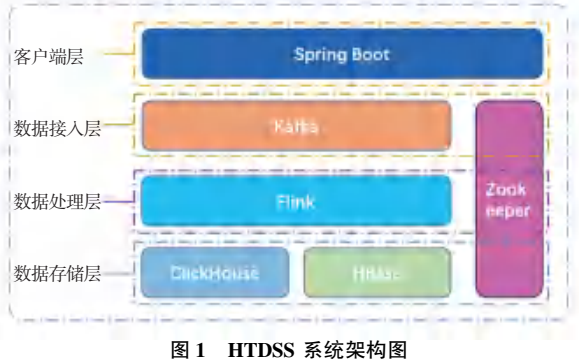


图 1　HTDSS 系统架构图

Fig. 1　Architecture diagram of HTDSS system

HTDSS 的整体架构由实时数据仓库架构改进和精简而来，相对于传统基于 Hadoop 技术栈的 Lambda 架构，Kappa 通过结合流处理和批处理，将所有的数据处理合并为批流一体，避免了集群内部署过多的服务，降低了系统的维护成本，也提高了处理效率，Kappa 架构注重实时性而 Lambda 架构注重精确度，因此，选择基于 Kappa 架构设计系统更合适。

2.3 系统工作流

HTDSS 系统工作流如图 2 所示。在 HTDSS 中，数据以字节码帧的格式由客户端组件持续写入数据接收层(Kafka)。在数据格式方面，为了规范测试环境，以某卫星的数据传输格式为例，每一帧数据包含了一个固定长度的帧头和若干个不定长的参数数据，其中帧头包含了数据生成时间、卫星信息、帧类型及后续紧跟的遥感参数数量等，而后续的每一段参数字节码包含了具体的参数值以及若干附加信息。实际上，该字节码已经由地面接收站解压完毕，解压前的大小约为解压后的 1/8。

在 HTDSS 的数据处理层，帧头信息被封装为可序列化对象，每个参数作为独立数据行写入表中。处理过程涉及清洗、校验和解析，生成的数据流通过 Flink 的 shuffle 策略分配至计算单元(算子)处理，保证负载均衡(采用轮询或时间戳分配)。处理完成后，数据被转换为数据库格式存储于数据存储层。

HTDSS 的数据存储层，对于已经解析出的参数数据，按照一个参数一行的方式在分布式数据表中保存下来，其中比较重要的就是参数值列和参数生成时间戳，利用 ClickHouse 的列存储特性，分布式表背后的本地表实现了高效的查询与写入性能。对于脏数据，数据处理层会下沉到 HBase 中，虽然 HBase 的查询功能有限，但其基于 HDFS 的副本策略可确保高吞吐和可靠性，便于后续通过脚本对脏数据进行清洗和解析，避免数据丢失。

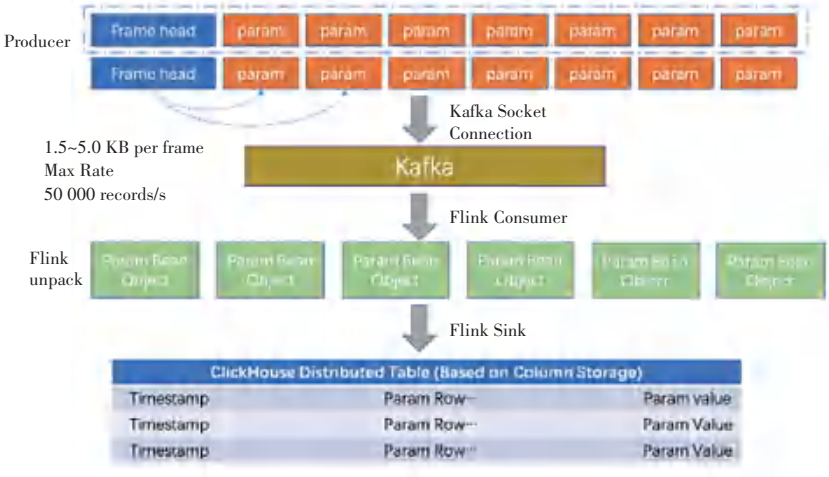


图 2 数据处理工作流

Fig. 2 Data processing workflow

3 系统实现

3.1 系统各组件设计与实现

HTDSS 主要包括四大组件，分别是基于 Java 后端技术的客户端软件包、基于 Flink 流处理引擎的流处理程序、基于开源监控平台的数据监控系统和数据库表的设计。其中，客户端组件对应于客户端层，流处理程序对应于数据处理层，数据库监控系统对应于系统可视化平台，数据库表设计对应于数据存储层。所用技术栈如图 3 所示。

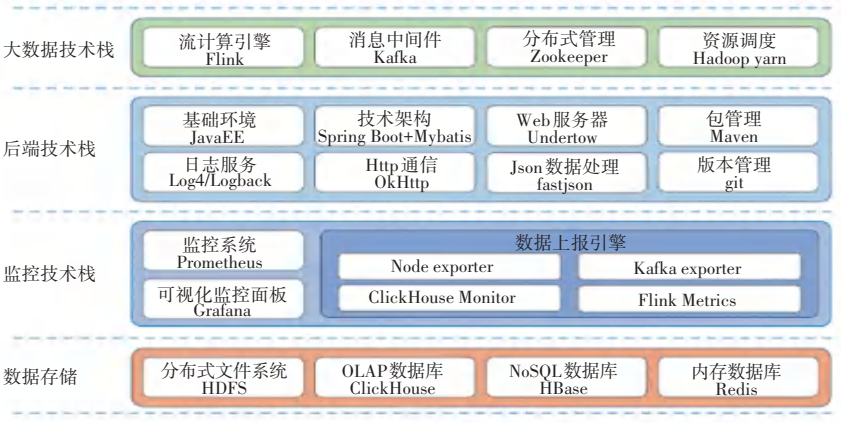


图 3 HTDSS 总体技术栈

Fig. 3 Technology stack of HTDSS system

3.1.1　客户端组件设计与实现

客户端组件整体采用主流的 Java 后端技术栈，基于 Maven 进行项目包的管理、编译和打包。得益于 Spring Boot 完善的工具类，客户端程序与轻量级 Web 服务器 Undertow 实现了无缝对接，能够快速部署和运行。客户端组件的设计由以下功能模块组成：

(1)数据转发模块。提供 Kafka 协议支持，将接收到的数据通过高效的分布式消息队列传递至 HTDSS，并且支持多种数据接入模式，包括实时数据转发和批量数据接入。

(2)数据仿真模块。可生成模拟的卫星参数数据，用于测试和验证系统的处理能力与稳定性。

(3)日志管理模块。采用 Logback 和 Log4j 结合的方式记录程序运行日志，包括类信息、时间信息、日志信息以及日志级别标记，提供详尽的运行信息用于调试和错误排查。

3.1.2　流处理组件设计与实现

HTDSS 的流处理组件是基于 Flink 流计算框架编写的流处理程序，是依赖于 Flink 流处理引擎和流处理环境的流计算程序。流处理程序的项目架构主要包含 4 个 Java 包，分别是 app 包、Java Bean 包、程序配置包和工具类包，同样基于 Maven 进行项目包管理，主要依赖 Flink 的技术栈以及一些数据库的 JDBC 连接库。Flink 流处理程序主要有 3 种部署方式，分别是基于 Hadoop Yarn 的部署方式、Flink Standalone 部署方式以及 Kubernetes 部署方式。在 HTDSS 中通过 Hadoop Yarn 资源管理器来提升 Flink 的运行效率，同时可以配置 Flink 在 Yarn 上的高可用，提升系统的可用性。

3.1.3　存储集群和分布式表设计与实现

HTDSS 的存储集群采用每节点一个 ClickHouse 实例，并隔离 ClickHouse 与 Hadoop、HBase 等组件部署，以保障资源分配。建立分布式表的流程主要有 3 步：

(1)在 ClickHouse 的 XML 配置文件中增加 ClickHouse 集群的每个节点 ip 信息和通信端口。

(2)在每个节点的 ClickHouse 中建立基于 MergeTree 引擎的本地表，定义基于时间戳的索引和存储文件切片方式，每一个 ClickHouse 实例中的本地表的定义必须完全一致。

(3)在其中一个节点的 ClickHouse 中建立分布式表，分布式表建立后，会通过 Zookeeper 被其他 ClickHouse 实例感知，此后在其他 ClickHouse 实例中也能对分布式表进行查询。

3.1.4　系统监控平台设计与实现

在 HTDSS 中，主要的系统设计目标是完成高通量的数据存储，但为了感知系统的运行状态和性能，需要建立可视化监控平台来方便测试、管理和运维。HTDSS 总体技术栈见图 3，系统监控可视化主要采用 Prometheus 开源监控上报组件和 Grafana 可视化监控面板、单独部署的基于 Prometheus 数据上报规则的 Node exporter 和 Kafka exporter 组件，以及 ClickHouse 和 Flink 自带的接入 Prometheus 的组件。Prometheus 统一收集多源的监控指标并存储为时间序列数据，同时通过其定义的查询语法进行时间序列查询。Grafana 支持特定数据源的可视化面板系统，能够提供自定义的数据监控面板和清晰的可视化图表，与 Prometheus 结合，可以提供成熟的告警和监控功能。

3.2　系统运行环境

系统运行环境具体要求为：

(1)服务器集群数量大于等于 3 台，并且数量需要为奇数，每台机器至少保证 8 GB 内存，保证同时装载固态硬盘和机械硬盘，其容量可根据需求而定，其中保证至少有一台节点内存在 32 GB 以上，作为中央节点部署高性能需求的服务。

(2)操作系统为 CentOS 7.5 及以上，JDK 版本为 1.8，Scala 版本为 2.12，存储目录所用文件系统应采用 ext4。

(3)系统使用软件框架名称及相关版本见表 1。

表 1　系统使用软件框架及版本

Table 1　System software framework and version

框架名称	版本号
Zookeeper	3.5.9
Kafka	2.12-2.63
Flink	1.13.6
ClickHouse	22.2.2.1
Hadoop	3.1.3
HBase	2.3.x

4　实验优化

为了验证系统设计是否达到设计优化要求，设计 1∶10 的仿真实验，即模拟了实际系统 10 倍的数据和吞吐量。在仿真验证的基础上进行优化。

4.1　客户端组件优化

客户端组件的开发包括 2 方面的优化：数据通信方式和 Kafka 多实例配置。

在仿真实验中,初版的存储系统暴露出了数据丢失,以及吞吐波动。其主要原因在于,客户端和数据接入层建立的是 Http 连接,在数据传输高峰时会导致大量的并发,超过 Spring Boot 实例处理的上限,特别是其中量级可观的小数据传输,导致大量建立连接和释放连接产生不小的开销。一种弥补的方式是将 http 连接设置为长连接,并且在数据发送前在客户端缓存一定的批量再发送。但 http 本身的特性并不适合作为数据存储协议,在压测过程中,时间一长,仍然会出现 http 连接集中释放高峰,从而使得大量数据堆积在 socket 缓存中,进一步导致内存溢出以及客户端崩溃。通过数据缓存批量再发送的方式,还会在接下来的计算中额外多加了一步解析计算,增加后续的计算负担。

针对 Spring Boot 和 Nginx 组成的数据接入层产生的 http 请求连接资源浪费、吞吐瓶颈的问题,重新设计了整体架构,去除了原本的数据接入层。新的架构将客户端直接对接 Kafka,通过 Kafka 生产者优化后的通信机制,可以大幅度节省通信成本,并且在新的架构中可以通过建立多个生产者实例池来提升并发和吞吐,进一步提升 20%~30%的吞吐性能,并且增加了连接的稳定性和可靠性。在实际测试中,在仿真环境里可以达到的 Kafka 写入吞吐为 150~220 MB/s,可以达到该层的设计目标。

4.2 Kafka 优化

Kafka 作为数据接收层缓存临时数据的重要组件,需要进行配置优化提升吞吐。

首先,Kafka 的数据存储时长保持为 3~7 天之间,这个时间足够运维人员进行容灾处理。由于 Kafka 并不需要长时间存储数据,只需要关注读写吞吐,所以可以采用固态硬盘存储的方式来提升数据 I/O 速度。通过 Linux 的 time 命令和 dd 命令,可以测出固态硬盘的稳定读写速度(稳定读写速度指全盘刷写过 3 次以上)均为 3 GB/s 以上。同时,在配置项上,设置 Kafka 的磁盘 I/O 线程数量为 CPU 核心数两倍,网络通信线程数量与 CPU 核心数一致。在上述配置下,通过客户端可以测出吞吐速度稳定在 130 MB/s。

4.3 流计算程序优化

在仿真测试中,流处理部分主要出现的问题是反压。流处理上下游反压示意如图 4 所示。反压是流计算中会出现的典型性能问题,在流计算中会将数据发送端称为上游,数据接收端称为下游,当下游处理能力不足时,会来不及处理上游发送的数据,为了协调下游的速度,上游就会将要发送的数据缓存下来,导致上游缓存膨胀,处理性能也随之下降。在 HTDSS 中,主要是由于过早地对字节码数据进行解析,导致数据流大小膨胀了 3 倍,使得上下游网络带宽不足以及 I/O 性能的下降。主要采用的优化方案如下:

(1)将数据解析处理下放到下游,避免数据膨胀带来的带宽占用。

(2)通过配置 Yarn 的资源分配策略,将 Flink Task Slot 分配到 ClickHouse 实例本地,并且每个实例的数据库下沉算子并行度设置为 2。

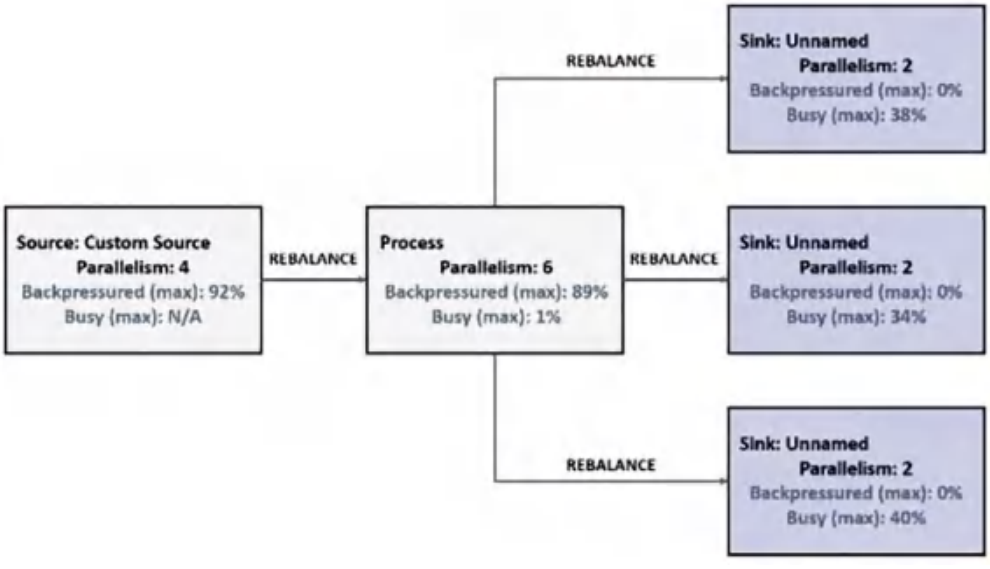


图 4 流处理上下游反压示意图

Fig. 4 Schematic diagram of upstream and downstream backpressure in flow processing

4.4　ClickHouse 优化

在数据存储上，主要通过优化 ClickHouse 的配置和写入方式来提升吞吐。主要的优化包括：

（1）流处理程序将数据写入 ClickHouse 的本地表而不是分布式表。因为分布式表会有一个 leader 节点，写入过程会先写入 leader 的本地磁盘后再分发给其他节点，这样就会导致单机瓶颈。在使用本地表时，数据直接写入各分片的本地磁盘，从而避免了 leader 中转的额外开销。

（2）采用批量插入，避免 ClickHouse 产生大量小文件以及过重的 Merge 操作（每次插入新数据 ClickHouse 会先写入新的文件中，经过一段时间后再进行合并）。同时写入频率控制在 2 次/s 左右，每次写入数据条数控制在 3 万~5 万。

（3）配置后台线程池大小为 CPU 核心数两倍，分布式处理后台线程数为 CPU 核心数两倍，内存资源为总内存的 70%~80%。

（4）通过挂载硬盘虚拟卷组可以配置多块机械硬盘，提升 ClickHouse 的 I/O 效率，但是由于 ClickHouse 保存的是长时间存储的业务数据，不建议使用固态硬盘。因为固态硬盘长时间不通电会出现数据丢失的情况，出现数据丢失后也不易找回。

5　系统测试

5.1　系统测试环境

为了对高通量卫星遥测数据存储系统 HTDSS 的功能和性能进行完整测试，本文按照如下配置搭建测试环境：

（1）物理节点集群包括 3 台高性能物理机。物理机的配置为 16 核心 CPU、64 GB 内存和万兆网卡。在存储容量方面均搭载一块 512 GB 固态硬盘和一块 4 TB 容量机械硬盘。

（2）在操作系统方面，安装 CentOS 7.6 系统，所有系统内置依赖均更新到最新版本，用于存储数据的文件系统为 ext4。

（3）HTDSS 依赖的软件均按照表 1 的版本标准进行安装。

（4）网络配置为千兆带宽，物理机直接为星形拓扑连接，中央交换机为千兆交换机。

5.2　功能与性能测试方法

由于千兆带宽的局限性，实际上无法在测试环境下精准测试出系统的吞吐上限，带宽导致的反压见图 4，上游的计算资源只利用了 1%，并且下游的下沉算子的计算资源利用率也不高，由此推测应该是网络带宽导致的数据吞吐限制，使得上游数据无法及时传到下游，下游也没法获得足够的数据，在 4.3 节也提到了反压的概念和原因。由于系统的实际运行环境是在万兆带宽下的，所以并不会受到网络带宽的局限。为了测试，采用单机性能压测，然后通过分布式线性性能提升的特性来对系统吞吐进行估算。

通过客户端的仿真数据来对系统进行功能性和稳定性测试。首先，清空数据库中的所有数据，利用客户端组件生成固定条数的遥测数据，并封装为若干个数据包，然后查询数据库中的数据条数是否和生成的一致。

对于吞吐测试，首先停止流处理程序，然后通过客户端生成仿真数据，向 Kafka 积压 20~30 GB 的数据，接着运行流处理程序，观测处理完成所花费的时间。

5.3　测试结果分析

功能性测试结果见表 2。全表查询 SQL 语句为 select count(*) from satellite. param_detail_all。整体测试 5 次，测试数据量从 5 000 条左右一直增加到 30 万条左右，所有测试均通过，全表查询出的数据数量和生成数据条数一致，并且查询间隔保持在 20 s 内，这意味着单机情况下 30 万条数据在 20 s 内也能完全写入。

表 2　功能性测试结果

Table 2　Functional test results

测试轮次	生成数据条数/条	全表查询条数/条
1	4 890	4 890
2	57 623	57 623
3	130 423	130 423
4	342 337	342 337
5	322 336	322 336

吞吐量测试结果见表 3。5 次测试中，在单机情况下，吞吐率能够稳定在 39 MB/s，在系统组件中并不存在中心化架构，所以多个节点合作的集群性能会呈现线性上升。由此推测，当带宽满足万兆的情况下，3 台物理机组成的集群是可以达到 100 MB/s 的吞吐，符合系统设计要求。

对于系统稳定性，系统持续运行，客户端保持 30 MB/s 写入速度，经过 72 h 运行后未发现异常和宕机现象，可以认为 HTDSS 满足系统目标的稳定性要求。

表 3 单机吞吐量测试结果

Table 3 Single machine throughput test results

测试轮次	生成数据量/MB	系统处理时间/s	平均吞吐速率/(MB·s^{-1})
1	3 523.5	91	38.7
2	8 452.8	215	39.3
3	14 742.3	382	38.6
4	22 348.9	572	39.1
5	31 482.6	799	39.4

6 结束语

针对目前航天领域卫星数据实时存储的丢包率高、数据存储系统的吞吐量不足的问题,本文基于现有的开源大数据技术和实时数仓解决方案,提出了一套面向卫星遥测数据的高通量大数据存储系统 HTDSS。并制定测试方案对系统的功能和吞吐率进行了测试,在 3 节点集群下吞吐率能够达到 100 MB/s,能够符合实际应用的目标。本文不足之处在于 HTDSS 没有在安全性方面做特别的设计,虽然目前航天领域运行环境本身的安全性较高,数据本身的安全性的要求势必会影响系统的吞吐性能。可以从安全性和吞吐率的角度出发,探讨两者的权衡方式、相互影响的数学模型以及优化方案则是亟待后续深入探讨和研究的重点方向。

参考文献

[1] Viktor M S, Kenneth C. Big Data: A revolution that will transform how we live, work and think[J]. American Journal of Epidemiology, 2014, 179(9): 1143-1144.

[2] 林子雨. 大数据技术原理与应用[M]. 北京:人民邮电出版社, 2015.

[3] 罗敬宁, 刘立葳. 遥感大数据分布式技术研究与实现[J]. 应用气象学报, 2017, 28(5):621-631.

[4] 罗中凯. 基于机器学习的卫星在轨运行状态判别方法研究[D]. 北京:中国科学院大学(中国科学院国家空间科学中心), 2019.

[5] 李向前. 复杂装备故障预测与健康管理关键技术研究[D]. 北京:北京理工大学, 2014.

[6] 王怡婷, 陈曦, 鞠兴旺, 等. 基于 GRU 网络的气象要素预测算法[J]. 计算机仿真, 2021, 38(7):419-423.

[7] Zhang R, Chen Z Y, Xu L J, et al. Meteorological drought forecasting based on a statistical model with machine learning techniques in Shaanxi province, China[J]. Science of the Total Environment, 2019, 665: 338-346.

[8] 崔星灿, 禹晓辉, 刘洋, 等. 分布式流处理技术综述[J]. 计算机研究与发展, 2015, 52(2):318-332.

[9] 彭喜元, 庞景月, 空彭宇, 等. 航天器遥测数据异常检测综述[J]. 仪器仪表学报, 2016, 37(9):1929-1945.

[10] 彭宇, 庞景月, 刘大同, 等. 大数据:内涵、技术体系与展望[J]. 电子测量与仪器学报, 2015, 29(4):469-482.

[11] 马福利, 石涛, 陈玲, 等. 面向多星多任务的大数据处理系统设计[J]. 大数据, 2021, 7(5):3-16.

[12] Williams J F. NASA's Nebula cloud computing initiative: Cloud innovation at NASA[R]. California: NASA Ames Research Center, 2012.

[13] 苏丽, 张博为, 谌明, 等. 大数据技术在航天领域发展与应用[J]. 遥测遥控, 2015, 36(2):1-9.

[14] Yu X Y, Lu S F, Wang T Y, et al. Towards higher efficiency in a distributed memory storage system using data compression[J]. International Journal of Bio-inspired Computation, 2022, 20(4): 232-240.

[15] Gil M S, Moon Y S. SPinDP: A high-speed distributed processing platform for sampling and filtering data streams[J]. Applied Sciences, 2023, 13(24):12998.

[16] Lupi G, Accorsi R, Battarra I, et al. Space efficiency and throughput performance in AVS/RS under variant lane depths[J]. International Journal of Advanced Manufacturing Technology, 2024, 131(3-4):1449-1466.

[17] Puranik S, Barve M, Rodi S, et al. FPGA-based high-throughput key-value store using hashing and B-Tree for securities trading system[J]. Electronics, 2023, 12(1):183.

[18] 李勇, 唐国琴, 罗琅腾, 等. 云环境下海量 GPS、视频监控数据存储检索研究[J]. 电力大数据, 2022, 25(5):85-92.

[19] Kim S, Sung D, Son Y. IFLustre: Towards interference-free and efficient storage allocation in distributed file system [C]// Proceedings of the 30th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). Piscataway, NJ: IEEE, 2022:105-112.

[20] Wu Haibo, Liu Hui, Li Jun. FabricETP: A high-throughput blockchain optimization solution for resolving concurrent conflicting transactions[J]. Peer-to-Peer Networking and Applications, 2023, 16(2):858-875.