支持高并发的 Hadoop 高性能加密方法研究

金伟[1,2]，余铭洁[1,3]，李凤华[1,2]，杨正坤[1,2]，耿魁[1]

（1. 中国科学院信息工程研究所，北京 100093；2. 中国科学院大学网络空间安全学院，北京 100049；
3. 中国科学技术大学网络空间安全学院，安徽 合肥 230027）

摘 要：针对 Hadoop 平台静态存储加密方案的加密算法单一、密钥管理复杂、加解密性能低，不能高效保护 Hadoop 数据安全的问题，提出了一套基于商用密码算法的 Hadoop 高性能加密与密钥管理方案。首先，提出基于国产商密算法的 Hadoop 平台三层密钥管理体系，优化二级密钥的组织结构；其次，提出异步流水模式的高并发加解密方法，替代 Hadoop 原有的串行加解密流程，并通过密文排序确保多加密线程的密文同步。实验结果表明，所提方案的密钥存取效率、文件读写速度快于原有 Hadoop 方案，所提方法可有效提升 Hadoop 平台密钥存取与加解密速度。

关键词：Hadoop 数据安全；密钥管理；商用密码算法；高性能加密

中图分类号：TP302

文献标识码：A

doi: 10.11959/j.issn.1000−436x.2019224

High-performance and high-concurrency encryption scheme for Hadoop platform

JIN Wei[1,2]，YU Mingjie[1,3]，LI Fenghua[1,2]，YANG Zhengkun[1,2]，GENG Kui[1]

1. Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
2. School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China
3. School of Cyber Security, University of Science and Technology of China, Hefei 230027, China

Abstract: To address the problem that as preventing data leakage on Hadoop platform, the existing encryption schemes suffer from several problems (e.g., single encryption algorithm, complicated key management, low encryption performance) and they cannot protect data stored in Hadoop effectively, a high-performance encryption and key management scheme for Hadoop was proposed. Firstly, a three-level key management system was extended with the domestic commercial cipher algorithm. Then, a new data structure for encryption zone key to reduce time consumption was designed. Finally, the computing process of data stream in parallel was scheduled. The experimental results show that compared with the existing Hadoop schemes, the proposed scheme can improve the efficiency of key management, and can speed up file encryption.

Key words: Hadoop data security, key management, commercial cipher algorithm, high performance encryption

收稿日期：2019−06−20；**修回日期**：2019−09−23

通信作者：耿魁，gengkui@iie.ac.cn

基金项目：国家重点研发计划基金资助项目（No.2017YFB0802705）；中国科学院战略性先导科技专项基金资助项目（No.XDC02040400）；国家自然科学基金资助项目（No.U1836203）

Foundation Items: The National Key Research Project and Development Program of China (No.2017YFB0802705), The Strategic Priority Research Program of the Chinese Academy of Sciences (No.XDC02040400), The National Natural Science Foundation of China (No.U1836203)

1 引言

Hadoop 是一种可以对海量数据集进行分布式存储与处理的开源软件框架，该框架通过搭建单一服务器或多台服务器集群，为用户提供高可用性、可扩展性和高容错性服务。目前，Hadoop 平台在电子凭据、医疗服务等结构化和非结构化数据的服务领域均占有广泛的市场。

伴随着 Hadoop 的快速发展，其安全问题也日益凸显。据 CVE（common vulnerabilities and exposures）漏洞列表显示，从 2013 年到 2019 年 6 月，已暴露出的 Hadoop 漏洞数量共计 21 个，其中包括 6 个关于信息泄露的漏洞，这些漏洞带来了严重的安全隐患。

为确保存储在Hadoop平台中的数据不被泄露，现有的典型技术包括访问控制、数据加密等。其中，访问控制作为数据保护的第一道防线，通过开启 Kerberos 等身份认证机制，确认用户真实身份，并通过 Sentry、Ranger 等组件为用户对数据的访问提供细粒度控制。然而，访问控制具有边界性，一旦数据脱离了数据控制域，访问控制将会失效。相反，数据加密技术通过引入加密算法对大数据平台中的关键数据进行加密保护，以密文方式存储和传输，即使数据脱离访问控制域，也将为数据提供持续保护。目前，加密技术已成为确保 Hadoop 平台中数据安全的“护城河”。

目前，Hadoop 的加密包括传输加密和存储加密。在传输加密方面，Hadoop 平台的多数组件已有 SASL（simple authentication and security layer）、SSL/TLV（secure sockets layer/transport layer security）等成熟稳定的传输协议实现数据高效加密传输；在存储加密方面，Apache Hadoop 自 2.6.0 版本起增加了 KMS（key management server）密钥管理模块，并增加了透明加解密机制，允许用户或管理员设置加密区，向加密区上传/下载数据时，Hadoop 平台自动加密后进行传输和存储。然而，现有的 KMS 存在如下问题。

1) 密钥检索效率低。当前，Hadoop 发行版中的加密区密钥是按照密钥名与版本号的散列值将密钥散乱排列到 HashTable 中，相同密钥的不同版本也散乱排列到 HashTable 中，这种散乱排列导致密钥检索效率低。

2) 加解密性能低。Hadoop 平台本地文件加密过程包括以下 3 步：待加密明文准备、明文加密和密文发送准备。在当前 Hadoop 发行版中，这 3 个步骤串行执行，即在某个步骤执行过程中其他步骤处于空闲状态；在明文加密过程中，对明文的加密也是串行执行。上述 2 种原因导致加密资源利用不充分、性能低。

3) 加解密算法非国产。当前，Hadoop 发行版中主要采用的加解密算法是美国国家标准与技术研究院的 AES（advanced encryption standard）算法，这些发行版不支持密码算法动态配置。特别地，对中国来说，未采用国产密码算法，使密码应用关键环节存在重要不可控因素，这可能导致在 Hadoop 平台中“门是自家的，但钥匙在别人手中”的后果。

针对上述问题，本文提出一种面向 Hadoop 平台的高效密钥管理方案，实现国产商用密码算法补充 AES 算法，利用新的密钥存储结构提高密钥的存取效率，采用流水线方法异步优化加解密算法处理流程，提升 Hadoop 平台文件加解密速度。本文主要贡献如下。

1) 提出基于国产商用密码算法的 Hadoop 平台三层密钥管理体系，对密钥库口令、加密区密钥、文件密钥这三层密钥逐级加密，确保密钥安全。

2) 针对密钥检索效率低的问题，设计“密钥串”存储结构，该结构能够紧凑组织二级密钥的存储，提高密钥存取效率。

3) 针对加解密性能低的问题，提出了异步流水模式的并发加解密方案，替代 Hadoop 原有的串行加解密流程，提高 Hadoop 文件加解密效率。

4) 实现了基于商用密码算法的 Hadoop 平台三层密钥管理，实验结果表明，本文方法可有效提升 Hadoop 平台数据加解密效率。

2 相关工作

2.1 面向 Hadoop 平台的加解密

如前文所述，面向 Hadoop 平台的加解密主要聚焦于传输加密、存储加密及其高性能实现。

在传输加密方面，Hadoop 主要使用 SASL 认证框架，从认证、消息完整性、机密性等多方面保护客户端与服务器之间交换的数据[1]。在 Apache Hadoop 2.9.2 版本中，Hadoop 服务可开启 RPC（remote procedure call）数据加密。Apache Hadoop 自 2.5.2 版本始支持 SSL/TLS v1，Apache Hive 自 1.0 版本起添加 SSLv2 协议。HDFS（Hadoop dis-

tributed file system）、YARN（yet another resource negotiator）、MapReduce、Oozie 等组件均支持 SSL 网络传输，并且不影响 KMS 和 HttpFS 的安全使用。

在存储加密方面，HBase 支持 HFile v3 单数据项级的数据库加密，使用两级密钥，并使用 SASL 协议和 Kerberos 协议分别为 RPC 和 ZooKeeper 进行认证。Apache Hadoop 自 2.6.0 版本始添加了端到端的透明加解密机制，用户可设置加密区，向加密区上传/下载文件时，数据在客户端自动加解密，实现传输安全和存储安全，支持 AES、3DES（triple data encryption algorithm）和 RC4（rivest cipher 4）等多种算法，其中 AES 算法速度最快，但文件加密模式仅支持“AES/CTR/NoPadding”（CTR 为 counter，表示计数器模式；NoPadding 表示无补丁）。梁胜昔等[2]提出一种 HDFS 混合加密保护方案，支持 AES、RC4 混合加密模式，可用于实现云数据安全共享。文献[3]在 Hadoop 平台中建立基于双线性椭圆曲线的认证和加密机制，并对数据访问进行审计，保护 Hadoop 平台数据安全，但基于非对称密钥的运算开销较大，配置复杂。Wang 等[4]借助 Hadoop 平台和 Sqoop 组件，分布式处理数据库的加密运算，保护海量数据安全。

在加密算法替换与补充方面，Song 等[5]在 HDFS 透明加密机制含有 AES 算法的基础上，添加了 ARIA 分组加密算法，使 HDFS 加密系统双算法可选。Lin 等[6]提出了 HDFS-RSA 算法和 HDFS-Pairing 算法 2 种实现补充，使用混合加密机制保护 Hadoop 平台数据机密性，并进行仿真验证。上述工作主要聚焦于 Hadoop 平台下传输与存储中加解密的透明性，忽略了 Hadoop 环境下对加解密的性能需求。

在高性能实现方面，Bhatotia 等[7]使用 GPU 提升 Hadoop 的计算和存储速度。Cloudera 与 Intel 合作的 Rhino 项目为 HBase 0.98 贡献了关键的安全特性。它提供了数据单元（cell）级别的加密和细粒度访问权限控制的功能。同时，Intel 研制的 AES-NI 可为 HDFS 提供硬件加速。Cohen 等[8]使用可信平台模块（TPM, trusted platform module）进行密钥保护，应用 AES-NI 指令集进行硬件加速，最高计算速度可达 1.84 GB/s，是同数量级软件加密速度的 18 倍。

但是，CPU 层面的硬件加速仅适用于 X86 处理器，对硬件品牌要求高，操作系统支持的通用性较差，无法广泛适配 Hadoop 平台的各种服务器机型，并且，AES-NI 主要针对 AES 算法，对于其他算法的速度暂无提升。

2.2　密钥管理机制

目前，针对 Hadoop 系统的密钥管理的研究相对较少，但从云环境密钥管理出发，已有很多研究可供借鉴。本文从云环境密钥管理入手，对相关研究进行论述。

Zhou 等[9]针对面向群组的应用中层次化访问控制的密钥管理问题，从密钥管理拓扑模型、密钥更新方法与策略等方面分析了安全分发和密钥更新的典型需求，枚举了密钥管理方案所需机制。Kandah 等[10]针对大数据环境下无线传感网动态环境低耗能密钥分发需求，提出了中心状态链接（CSC, centralized stateful connection）方案，动态地管理传感网密钥，并通过公钥加密对称密钥，然后使用对称密钥加密传输数据，从而减少开销。Albakri 等[11]提出了一种基于多项式的轻量级层次密钥机制，管理雾节点（fog node）与其他用户设备间的密钥共享，有效保证物联网设备安全接入的同时，不增加用户设备的存储空间占用。

3　Hadoop 平台密钥管理架构

3.1　基于商密算法的三层密钥管理体系

如图 1 所示，本文所提架构采用三层密钥管理体系对 Hadoop 平台中的海量密钥进行保护和管理。其中，一级密钥为加密区密钥库口令，将该口令采用 SM3 单向函数进行散列处理后用于加密二级密钥，该口令存储在本地文件中；二级密钥为加密区密钥，该密钥与加密区绑定（即加密区与二级密钥存在一一映射关系），并用散列化后的一级密钥加密，将加密后的二级密钥存储在加密区密钥库中；三级密钥为文件密钥，用于加解密存储在加密区中的文件，文件密钥使用二级密钥加密，以密文方式存储在 NameNode 文件目录的扩展属性中。

1) 一级密钥管理

一级密钥即密钥库口令，在 KMS 部署时，由管理员进行配置，以明文的形式存储于本地文件中。当 KMS 启动时，从文件中读取口令，以字节数组的形式存储在内存中。当 KMS 加载或存储加密区密钥时，对该口令使用 SM3 单向函数散列处理，用散列化的口令对二级密钥进行加密或解密。

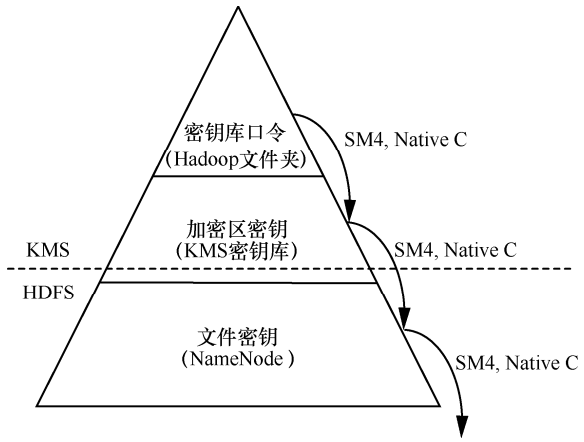


图 1 三层密钥管理结构

2) 二级密钥管理

二级密钥即加密区密钥，用来加密指定加密区内所有文件的密钥，二级密钥包含 8 个部分：密钥名、密钥长度、密钥、所采用的加密算法、生成时间、当前密钥版本号、密钥描述信息以及用户自定义属性，二级密钥由散列化后的密钥库口令加密，并以文件等方式存储在密钥库中。

当某空文件目录被用户指定为加密区后，它会与用户所指定的二级密钥绑定，该密钥即成为该目录的加密区密钥，其密钥名存储在该目录的扩展属性中。二级密钥可以更新，在二级密钥更新后，KMS 采用新版本密钥对加密区内新文件的密钥进行加密。注意：旧文件的密钥仍以旧二级密钥加密。

3) 三级密钥管理

三级密钥即文件密钥，用于加解密上传至加密区的文件。每个文件拥有独立的文件密钥，经二级密钥加密后，密态存储在 NameNode 文件系统的文件扩展属性中，与文件名的文件属性绑定。

文件的密钥由 KMS 随机生成，并由该文件所在加密区的二级密钥加密。NameNode 将二级密钥名、版本号和密态的三级密钥存储在文件 INode 节点的扩展属性中，并与文件永久绑定。加解密文件时，NameNode 将该文件所在加密区的二级密钥名、版本号和密态的文件密钥发送给客户端，客户端再将这三部分数据发送给 KMS。KMS 根据加密区密钥名和版本号查询存储在密钥库中的加密区密钥，然后对密态文件密钥进行解密，得到明文的文件密钥，并以 HTTPS 协议封装后发送给客户端，客户端收到文件密钥后即可对文件执行加解密操作。

3.2 二级密钥的快速存取

如上所述，客户端对文件加解密前，需等待 KMS 返回明文的文件密钥。而 KMS 在解密密态的文件密钥并将明文的文件密钥返回给客户端前，需要根据二级密钥名与版本号从密钥库中读取该文件所在加密区的二级密钥，因此二级密钥的读取速度是制约文件密钥的解密性能的关键因素之一。现有方案中二级密钥的元数据、不同密钥、相同密钥名的不同版本均散落在 HashTable 中，并使 HashTable 空间占用率高，导致需要对 HashTable 频繁扩容与复制，二级密钥的存取效率低下。针对上述问题，本文设计了更为便捷高效的二级密钥存取结构，以降低文件加解密时延。

3.2.1 二级密钥的数据结构

为了使密钥的组织不松散，本文设计了“密钥串”新方案，其数据结构如图 2 所示。首先，将同一密钥名对应的各版本密钥依版本号组织在 ArrayList 中（即 ArrayList 的下标为该密钥的版本号）；然后，将 ArrayList 与该密钥的元数据节点组成一个 KeyChain，这样同一密钥名对应的元数据信息和密钥信息均有序地组织在一起，通过密钥名即可获取该密钥相关的全部信息。不同密钥名对应的 KeyChain 以<keyName, keyChain>的形式存储在一个 HashTable 中，以 keyName 的 hash 值排列。

生成或更新密钥时，在对应密钥名的 ArrayList 中顺序添加新生成的密钥，并更新元数据节点即可。查找密钥时，先由密钥名找到当前密钥名的 KeyChain 节点，再根据版本号直接读取 ArrayList 下标，获取对应版本的密钥，操作简捷、直接。

删除密钥时，不需要遍历同一密钥名的各个版本，只需将一个密钥的<keyName, keyChain>节点移出 HashTable 即可。在密钥使用的过程中，会对 HashTable 同步上锁，以保证密钥在使用过程中不会被误删导致数据错误。

3.2.2 二级密钥存取效率分析

以“密钥串”新方案组织内存中的二级密钥，可达到减少时间、降低空间、提升效率的效果，简析如下。

1) 查询/删除效率提升，新增/更新效率稳定

在密钥查询方面，原有 Hadoop 方案的查询需先经过密钥名或密钥别名的 hash 值定位，随后在同 hash 值的链表中依次匹配，效率为 $O(1)[+O(m)]$。“密钥串”新方案则是先通过密钥名获取 ArrayList，再使用版本号作为 ArrayList 的下标直接获取密钥，效率为 $O(1)[+O(n)]+O(1)$。其中，$O(m)$ 和 $O(n)$ 分别

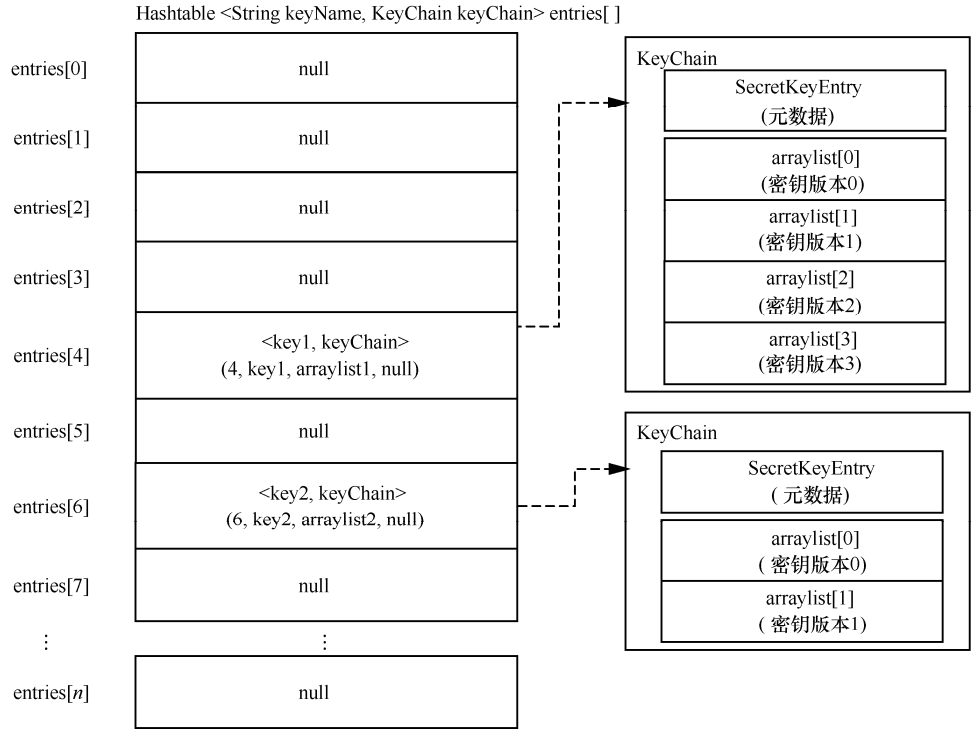


图 2　二级密钥数据结构

为 2 种结构的 HashTable 中发生冲突时遍历链表节点所需时间。在总密钥数量相同时，“密钥串”新方案的冲突概率远小于原有 Hadoop 方案，因此 $O(n)<O(m)$。同时，在相同空间情况下，“密钥串”新方案的效率只与加密区密钥的数量有关，与版本个数无关。在密钥名少、版本多的时候，“密钥串”新方案有明显优势。若密钥名多、版本少，则“密钥串”新方案会退化为与原有 Hadoop 方案类似。

在密钥删除方面，原有 Hadoop 方案需先获取待删除密钥的当前版本号 n，随后逐个移除 HashTable 中的密钥节点，最后移除元数据节点，故删除效率为 $O(n+1)$。“密钥串”新方案则是根据待删除的密钥名直接移除 KeyChain 节点，删除效率为 $O(1)$，与密钥版本数量无关。

在密钥增加方面，原有 Hadoop 方案直接向 HashTable 中添加 2 个键值对（一个密钥描述信息的 <keyName, keyMetadata> 和一个密钥数据 <keyName@version, keyMaterial>），添加速度均为 $O(1)$。而“密钥串”结构在添加时也添加 2 个键值对，时间与原有 Hadoop 方案相同。

在密钥更新方面，更新流程为：首先，查询该密钥名的当前元数据信息，获取版本号并自加 1，更新元数据；其次，添加新版本的密钥信息键值对。原有 Hadoop 方案获取元数据节点、更新元数据节点、新增密钥节点，效率均为 $O(1)[+O(m)]$；“密钥串”新方案在相同步骤下的效率均为 $O(1)[+O(n)]+O(1)$。同理，在密钥名多、版本少的情况下，“密钥串”新方案的效率会退化为与原有 Hadoop 方案类似。

2) 扩容次数减少，内存组织速度加快

采用 HashTable 作为存储加密区密钥的数据结构，在加密区密钥数量达到阈值时会触发扩容。HashTable 的最小数组元素数量为 $n=11$，阈值为数组长度的 $\frac{3}{4}$，即最初数组元素数量达到 $11\times\frac{3}{4}\approx 8$ 时即触发扩容，扩容后长度为 $n'=2n+1$。当加密区密钥数量很大时，频繁触发 HashTable 扩容会耗费大量时间空间。

原有 Hadoop 方案中，新建一个二级密钥会在 HashTable 中添加 2 个元素，更新一次再添加一个元素；而新方案中，新建一个密钥只会在 HashTable 中添加一个元素，更新密钥不添加元素。因此，原有 Hadoop 方案中 HashTable 的元素数量会频繁到达阈值，触发扩容；而新方案的元素数量相对稳定，不会频繁触发扩容。

3) 存储空间降低

输出至文件存储时，不再采用 Java 自身的序列化机制，而是采用 TLV（tag-length-value）格式紧凑输出。存储一个相同内容的密钥时，Java 序列化将输出 2.48 KB 大小，而 TLV 格式的输出仅需 1.64 KB，节省 $\frac{1}{3}$ 的存储空间。

3.3 三级密钥异步调度加解密文件

如图 3 所示，当前，Hadoop 发行版的写入文件过程包括 4 个阶段。1) 客户端准备文件数据阶段。客户端从数据源中读取数据，将该数据切分成多个固定长度的数据分组，计算校验值，并将数据分组和校验值添加至发送队列，消耗时间为 T_1。当发送队列已满时，该阶段会进入阻塞状态直到发送队列有空闲位置。2) 客户端从发送队列中获取数据分组，将其通过网络传输给数据节点 DataNode，并添加该数据分组至待确认队列，消耗时间为 T_2。3) DataNode 接收到数据分组后，校验数据完整性，检查后将数据写入文件块中，并向客户端返回确认信息 ACK，消耗时间为 T_3。4) 客户端收到 DataNode 返回的确认信息后，从待确认队列中删除数据分组，消耗时间为 T_4。在需要发送多数据的情况下，这 4 个阶段并发执行，但每阶段内部的执行是串行的，每个阶段各由一个线程完成。

特别地，当需要向加密区写入文件时，需要在客户端对文件进行加密，其加密过程在上述第一阶段完成，即客户端待发送数据分组为密文。具体地，第一阶段可细化为 3 个步骤。步骤 1，从明文数据源（例如客户端的明文文件）中获取明文，消耗时间为 t_1；步骤 2，将明文加密为密文，消耗时间为 t_2；步骤 3，将密文打包为固定长度的数据分组，并计算校验和，添加至发送队列，消耗时间为 t_3。当前，Hadoop 版本存在如下问题使加解密效率低：1) 这 3 个步骤串行执行，不能并发执行，存在加密空闲期，导致资源利用不充分；2) 同步骤 1 和步骤 3 相比，步骤 2 耗时较长，即使这 3 个步骤并发执行，也会使步骤 1 和步骤 3 需要等待步骤 2 完成，导致长时间阻塞。针对上述问题，本文提出了多线程流水线式并发加解密方案。

3.3.1 文件加密并发调度

为了解决加密空闲导致的低效率问题，必须尽可能确保加密过程不间断。为此，本文提出基于流水线的并发加密方案，如图 4 所示。

每条流水线至少包括 3 个线程：明文读取线程、加密线程和密文处理线程。其中，明文读取线程从明文数据源中读取固定长度的明文，将其写入明文缓冲队列；加密线程从明文缓冲队列中读取固定长度的明文进行加密，将得到的密文写入密文缓冲队列，在实际中可设置多个加密该线程（具体见 3.3.2 节）；密文处理线程从密文缓冲队列中取出密文计算校验和，封装并添加至发送队列。从图 4 可以看出，该流水线式调度方案中加密过程几乎不间断，加密资源被充分利用。

在 Hadoop 原加密流程的第一阶段中，第 k 个数据段从明文数据源到进入发送队列总耗时 $T_1(k)$ 为

$$T_1(k) = t_1(k) + t_2(k) + t_3(k) \tag{1}$$

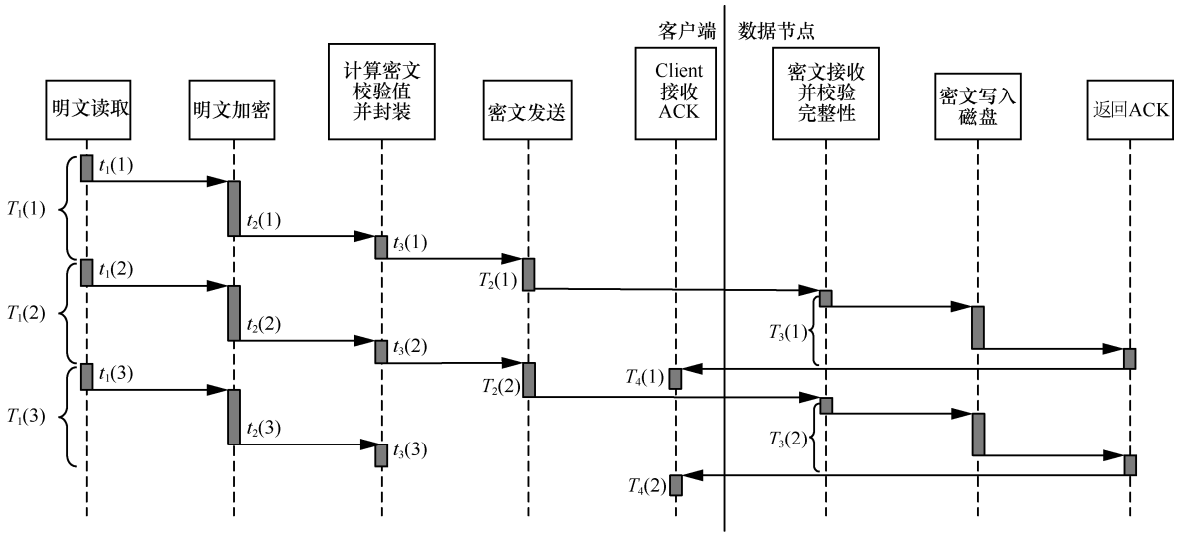


图 3 当前 Hadoop 发行版的写入文件过程

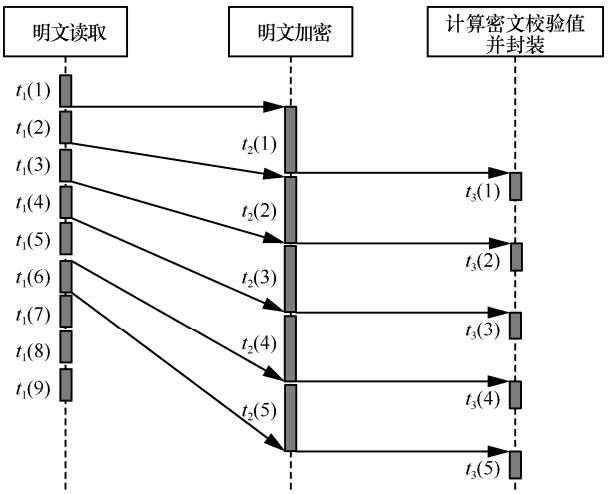


图 4　基于流水线的并发加密方案

若明文被切分为 m 个数据段，并且加密速度比文件读写速度慢，那么发送队列不会满，即第一阶段不会进入阻塞状态，因此第一阶段的总耗时为

$$T_1 = \sum_{k=1}^{m}(t_1(k)+t_2(k)+t_3(k)) \tag{2}$$

由于加密时间较长，因此原始加密方式和流水线式加密方式具有如下特征

$$t_2(k) > t_1(k) \quad \text{and} \quad t_2(k) > t_3(k) \tag{3}$$

在式(3)情况下，流水线加密方式的第一阶段总耗时为

$$T_1^{'} = t_1(1) + \sum_{k=1}^{m} t_2(k) + t_3(m) \tag{4}$$

由式(2)和式(4)可以得到

$$\Delta T_1 = T_1 - T_1^{'} = \sum_{k=2}^{m} t_1(k) + \sum_{k=1}^{m-1} t_3(k) \tag{5}$$

由于时间始终为正数，因此有

$$\Delta T_1 > 0 \tag{6}$$

即流水线式的加密过程能缩短明文从数据源读取经过加密后被写入密文缓冲队列的总时间。

此外，式(5)可以说明流水线方案比原 Hadoop 方案缩短的时间包含 2 个部分，一是明文读取线程与加密线程的并发时间，二是加密线程与密文处理线程的并发时间，这两部分分别对应式(5)最右侧表达式中的第一项与第二项。

3.3.2　数据分段加密与同步

为了解决明文加密时间长问题，本文在上述流水线中设置多个加密线程，每个加密线程均可从明文缓冲队列里取明文数据，并将加密得到的密文放入密文缓冲队列，实现多段明文数据的并发加密，从而缩短整个明文的加密时间。

在分段加密中每个加密线程加密时间不相同，可能导致同步问题。例如，加密线程 A_1 和 A_2 顺序取明文段 P_1 和 P_2，并独立加密，若 A_2 对 P_2 的加密时间小于 A_1 对 P_1 的加密时间，则可能导致 P_2 的密文先写入密文缓冲队列，从而导致乱序密文。为此，本文提出基于密文排序的多加密线程密文同步方案。

该方案为每一个明文段/密文段依照其先后顺序依次分配唯一的递增整数序列号 seq，相邻数据段序列号之差为 1。为了对密文进行重组排序，采用临时链表 tmpList 来按照 seq 序列号顺序存储密文段。当密文缓冲队列接收到来自临时链表中序列号为 n 的密文段后，通过设置与临时链表共享的等待序列号 ackSeq 为 n+1 来告知临时链表：下一个需要发送的密文段序列号为 n+1；当临时链表中第一个密文段序列号为 ackSeq 时，它将该密文段发送给密文缓冲队列，并从临时链表中移除。

图 5 给出了一个例子，在该例中，3 个加密线程（记为 $encrypt_1$、$encrypt_2$ 和 $encrypt_3$）分别加密第 6、4、8 段明文（即 $encrypt_1$= 6， $encrypt_2$= 4，$encrypt_3$= 8）。临时链表 tmpList 中按序存储第 5、7 段密文，记为 tmpList = {5, 7}。密文缓冲队列中已有密文 2、3，正在等待第 4 段密文的写入，记为 outQueue = {2, 3}，ackSeq=4。图 5 的编号①～⑩为第 4 段明文与第 8 段明文先后加密完成时本文算法所执行的 10 个步骤。

步骤 1　加密线程 2 加密完第 4 段明文后，将第 4 段密文按序插入临时链表中，此时 tmpList= {4, 5, 7}。

步骤 2　临时链表判断首元素序列号 4 是否与 ackSeq 相等，判断结果为“是”，然后将第 4 段密文从临时链表中移除，添加至密文缓冲队列中，此时 tmpList = {5, 7}，outQueue = {2, 3,4}。

步骤 3　密文缓冲队列将等待序列号 ackSeq 增加 1，此时 ackSeq = 5。

步骤 4　临时链表判断首元素序列号 5 是否与 ackSeq 相等，判断结果为“是”，然后将第 5 段密

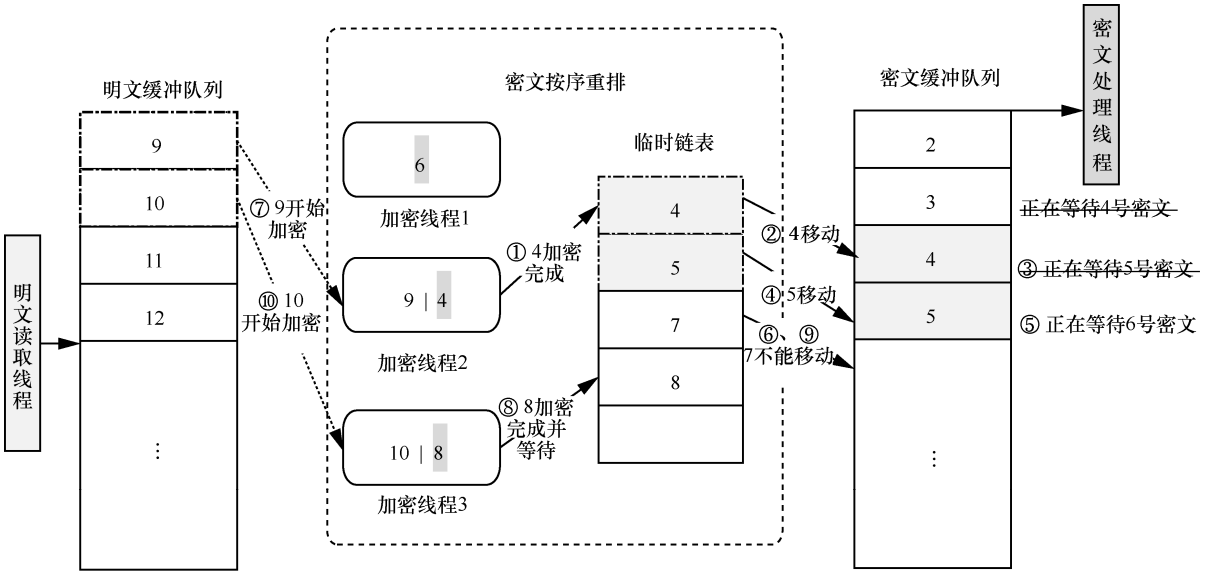


图 5　并发加密的密文重排序算法

文从临时链表中移除，添加至密文缓冲队列中，此时 tmpList = {7}，outQueue = {2, 3,4, 5}。

步骤 5　密文缓冲队列将等待序列号 ackSeq 增加 1，此时 ackSeq = 6。

步骤 6　临时链表判断首元素序列号 7 是否与 ackSeq 相等，判断结果为“否”，不能向密文缓冲队列中写入第 7 段密文。

步骤 7　加密线程 2 从明文缓冲队列中读取明文段 9 并开始加密，此时 $encrypt_2 = 9$。

步骤 8　加密线程 3 加密完第 8 段明文后，将第 8 段密文按序插入临时链表中，此时 tmpList= {7, 8}。

步骤 9　临时链表判断首元素序列号 7 是否与 ackSeq 相等，判断结果为“否”，不能向密文缓冲队列中写入第 7 段密文。

步骤 10　加密线程 3 从明文缓冲队列中读取明文段 10 并开始加密，此时 $encrypt_3 = 10$。

从上述例子可看出，尽管明文段 4、5、7、8 的加密不是按序完成的，但它们的密文会在临时链表内排序重组，从而确保将原本乱序的 4 个密文段顺序写入密文缓冲队列，所以本文方案能够有效地解决多加密线程密文同步问题。

基于上述分析，本文给出多线程流水线式并发加解密算法流程，该流程包括 3 个部分的算法：明文读取线程算法、加密线程算法、密文处理线程算法，如算法 1~算法 3 所示。其中，inQ 表示明文缓冲队列，outQ 表示密文缓冲队列，list 表示临时链表，len 表示数据源 dataSource 内的待加密明文长度，seq 表示下一个明文段应分配的序列号，ack 表示密文缓冲队列正在等待的密文段序列号，buffSize 为每个明文段/密文段的长度，stop 表示明文是否读取完毕。

全局初始化

ack ← 0，stop ← false, 清空 inQ、outQ、list

算法 1　明文读取线程

初始化　len ← dataSource.size, seq ← 1

1) while len > 0 do
2)　data ← 读取明文 buffSize 字节
3)　plain ← (data, seq)
4)　if inQ 已满
5)　　阻塞等待，直到 inQ 不满
6)　end if
7)　将二元组 plain 放入 inQ 尾部
8)　seq ← seq + 1
9)　len ← len – buffSize
10) end while
11) stop ← true

算法 2　加密线程

初始化　无

1) while stop 为假 or inQ 非空 do
2)　if inQ 为空
3)　　阻塞等待，直到 inQ 非空
4)　end if
5)　plain ← 取出 inQ 中的第一个二元组

6)　cData ← 加密 plain.data
7)　cSeq ← plain.seq
8)　index ← 根据 cSeq 查找 plain 在 list 中的顺序插入位置
9)　cipher ← (cData, cSeq)
10)　将 cipher 插入到 list 的第 index 个位置
11)　while list(0).seq == ackSeq do
12)　　cipher ← 取出 list 中的第一个密文元组
13)　　if outQ 已满
14)　　　阻塞等待，直到 outQ 不满
15)　　end if
16)　　将 cipher 放入 outQ 尾部
17)　　ack ← ack + 1
18)　end while
19) end While

算法 3 密文处理线程

初始化　无

1) while !(stop && outQ 为空) do
2)　if outQ 为空
3)　　阻塞等待，直到 outQ 非空
4)　end if
5)　cipher ← 取出 outQ 中的第一个密文元组
6)　data ← cipher.cData
7)　checksum ←计算 data 校验和
8)　sendPacket ← (data, checksum)
9)　将 sendPacket 放入 sendQueue 尾部
10)end while

4　实验分析

4.1　实验环境

基于本文架构在局域网中搭建实验环境，实验环境分别包括一个 NameNode 管理节点 node 0、3 个 DataNode 数据节点 $node_1$~$node_3$、一个 KMS 服务器节点 $node_4$，以及若干 Client 客户端节点模拟用户访问。根据 Hadoop 大数据平台架构，NameNode 节点 $node_0$ 存储文件密钥节点，并管理 3 个 DataNode 数据节点，$node_2$ 和 $node_3$ 为 $node_1$ 的备份节点，KMS 服务器节点 node 4 管理和存储一级密钥（密钥库口令）和二级密钥。拓扑如图 6 所示。

4.2　效果分析

4.2.1　二级密钥的存取效率性能

本文所提密钥串管理方案的关键在于存储结构的变化。为避免 JDK（Java SE development kit）调用带来的影响，本次实验共设置 6 组对比实验。1) 使用 Hadoop 原有方案的结构，在本地实现 HashTable 和 3DES 加密算法。2) 使用本文的“密钥串”新方案的数据结构，在本地实现 HashTable 和 3DES 加密算法。3) 使用本文的“密钥串”新方案的数据结构，在本地实现 HashTable 和 SM4 加密算法，数据存储过程无序列化。4) 仅实现本文“密钥串”新方案的数据结构，不实现加密过程，作为空白对比。5) 仅实现本文的“密钥串”新方案的数据结构，不实现加密过程，数据存储过程无序列化，作为空白对比。6) 仅实现本文“密钥串”新方案的数据结构，不实现加密过程，数据存储过程无序列化，且删除过程简化，作为空白对比。

密钥管理包括密钥的生成、更新和查询等，Hadoop 平台的二级密钥均在 KMS 本地使用，不涉及分发的问题。因此依次测试各类结构在以下 4 个生命周期的密钥管理中所需的时间：创建 n 个密钥，每个密钥更新 100 个版本的密钥，随机查找 n 个密钥，删除所有密钥。

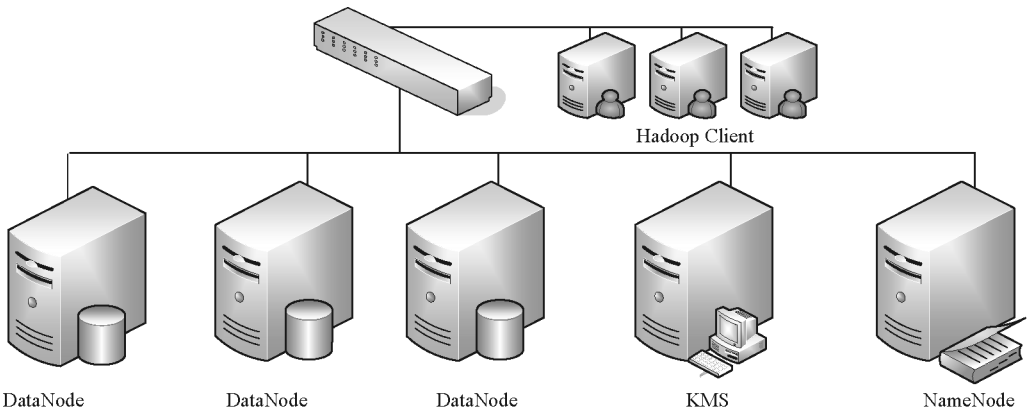


图 6　所用实验部署环境

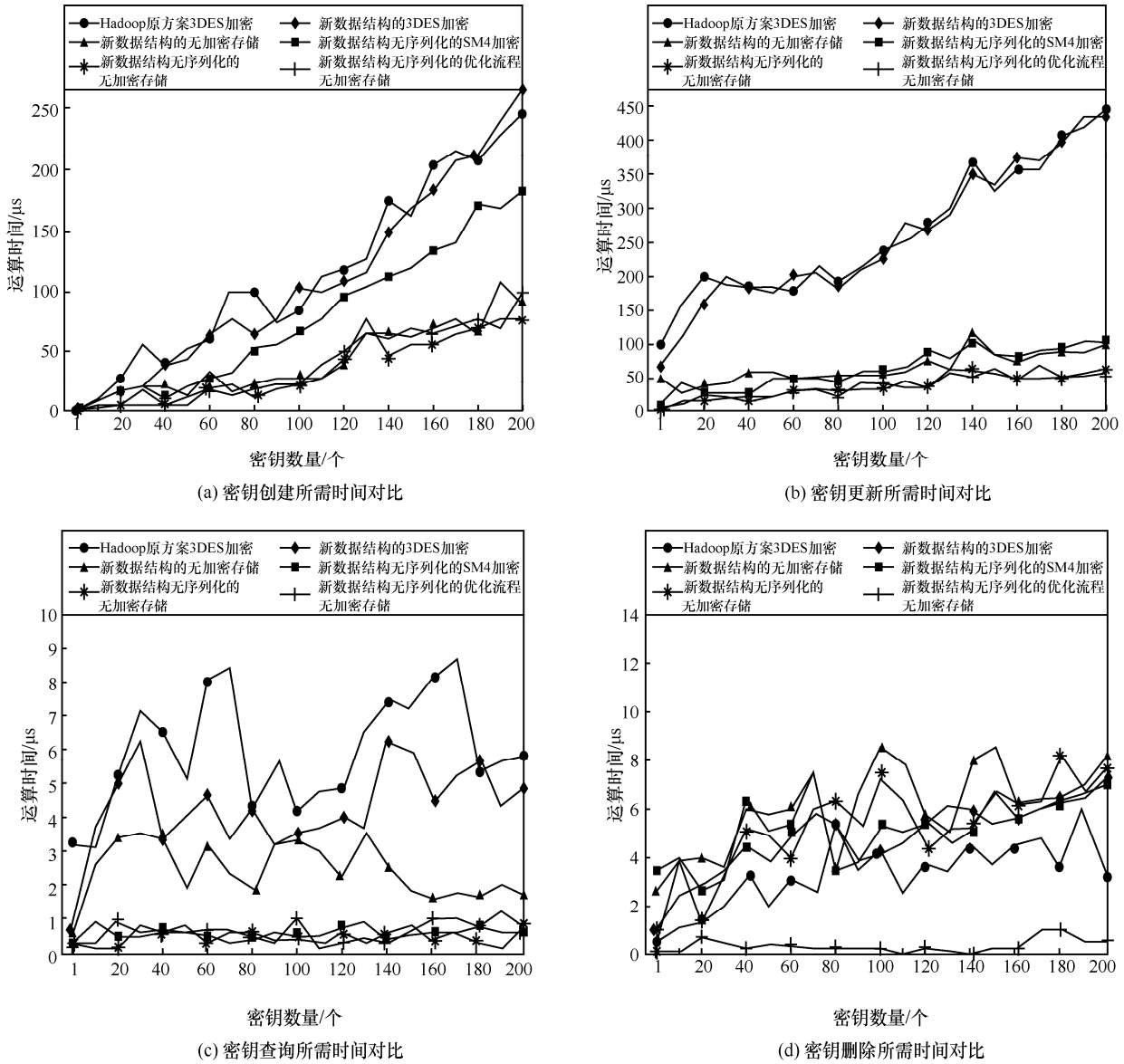
因每一次的操作耗时在纳秒级，小量测试不足以区分各方案之间的优劣。本文采用逐级递增的大数据量进行测试，即 n 的范围为 1～200（以 10 为单位递增），以全部运算之后的总时长进行比较。

实验结果如图 7 所示。由图 7 可以得出多种方案中密钥创建、更新、查询、删除的效率对比。图 7 的共有特点是所有不加密的方案比同结构的加密方案效率高，所有不使用序列化的方案比使用序列化的方案效率高，与已知常识一致。

在密钥创建方面，“密钥串”新方案比原有 Hadoop 方案速度快，使用无序列化的 SM4 加密方案比 3DES 加密方案速度快。在密钥更新方面，当前为少密钥名、多版本号的情况，“密钥串”新方案比原有 Hadoop 方案速度快，使用无序列化的 SM4 加密方案几乎可与序列化的无加密方案速度持平。在密钥查询方面，“密钥串”新方案比原有 Hadoop 方案速度快，可以提升 Hadoop 密钥使用过程中的查询效率。在密钥删除方面，只有第 6 组实验优化了删除流程，只需按密钥名删除整个节点、不需要逐个删除密钥，效率明显优于其他 5 组实验。与 3.2.2 节效率分析一致。

4.2.2　多线程流水线式并发加解密

本文所提出的流水线式并发加解密方案中，



(a) 密钥创建所需时间对比　　(b) 密钥更新所需时间对比

(c) 密钥查询所需时间对比　　(d) 密钥删除所需时间对比

图 7　多种方案的密钥操作效率对比

提高加解密速度的关键因素在于两点：一是串行加解密改为流水线式加解密，将明文读取、密文处理与加解密 3 个步骤并发执行，从而缩短明文从数据源中被读出经过加密并打包添加至发送队列的总时间；二是增加加解密线程的个数，使不同的明文段能够同时加密，从而缩短整个明文的加密时间。

根据上述两点因素，本实验测试了不加密、串行加密、流水线式加密、2 线程并发加密、3 线程并发加密以及 4 线程并发加密共 6 种条件下的文件写入速度。由于同一时间内，用户可能会同时向加密区内写入多个文件，因此实验在上述各条件下分别测定了同时向加密区内写入 1～5 个文件所花费的时间，根据写入文件的总大小计算得出文件的写入速度。实验结果如图 8 所示，在明文数据段大小为 8 KB 下测得的写入速度。

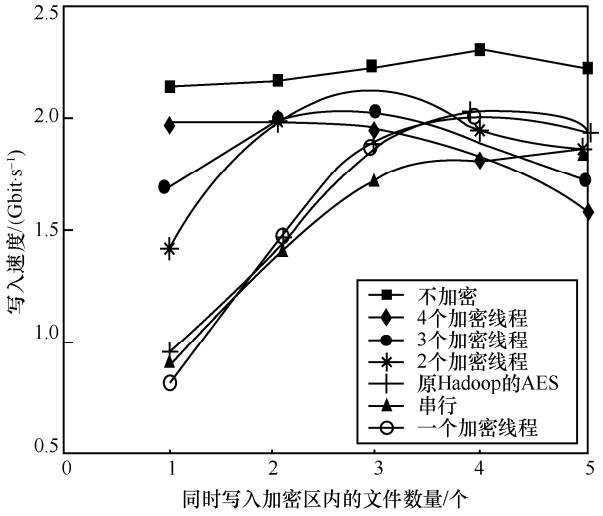


图 8　文件分段大小为 8 KB 时的加密写入速度

从图 8 中可以看出，不加密时文件写入速度基本稳定在 2.2 Gbit/s 左右，而 HDFS 原本的串行加密方式在同时写入一个文件时，速度只有 0.8 Gbit/s，与不加密时相比性能大打折扣。随着同时写入文件数的增大时，各文件的并发加密使加密资源逐渐被充分利用，从而写入速度逐渐提升。

采用流水线式加密方案后，文件写入速度相对于串行加密方式有小幅提升，与前面的分析一致，这部分速度的提升来自式(5)所计算的时间差。

采用多线程并发加解密方案后，可以看出，随着加密线程数量的增多，文件写入速度大幅提升。当采用 4 个线程进行并发加密时，即使同时只写入一个文件，写入速度也几乎接近不加密的写入速度。

与原有 Hadoop 方案的 AES 算法相比，替换 SM4 算法在串行方案中并不占优，而在流水线式加密方案和多线程并发加解密方案中，快于原有 Hadoop 方案的 AES 算法。

由此可见，本文的多线程并发加解密方案确实能够高效地完成文件加密，能够使用户在通过文件加密的方式保障安全性的同时，不损失文件读写的效率。

5　结束语

本文针对 Hadoop 平台密钥管理复杂、数据加解密性能低的问题，提出一种面向 Hadoop 平台的高效密钥管理方案，该方案通过 Hadoop 平台三层密钥管理体系，结合国产商用系列密码算法，实现多级密钥系统管理与保护，优化二级密钥组织方式提高密钥的存取效率，流水线作业的异步调度加密资源，优化文件加密的流程，提升加密性能，并通过密文重排机制，恢复异步加密数据的原有顺序。模拟实验结果表明，该方案可以有效提升 Hadoop 静态加密速度，同时减少空间占用。

参考文献：

[1] 陈丽, 黄晋, 王锐. Hadoop 大数据平台安全问题和解决方案的综述[J]. 计算机系统应用, 2018, 27(1): 1-9.
CHEN L, HUANG J, WANG R. Overview on security issues and solutions of Hadoop big data platform[J]. Computer Systems & Applications, 2018, 27(1): 1-9.

[2] 梁胜昔, 秦军, 宋蕾. HDFS 混合加密保护方案的设计[J]. 计算机时代, 2014(7): 17-19.
LIANG S X, QIN J, SONG L. Design for protection scheme of hybrid encryption in HDFS[J]. Computer Era, 2014(7): 17-19.

[3] DÍAZ A F, BLOKHIN I, ORTEGA J. Secure data access in Hadoop using elliptic curve cryptography[C]// International Conference on Algorithms and Architectures for Parallel Processing. 2016: 136-145.

[4] WANG F, KOHLER M, SCHAAD A. Initial encryption of large searchable data sets using Hadoop[C]//The 20th ACM Symposium on Access Control Models and Technologies. SACMAT, 2015: 165-168.

[5] SONG Y, SHIN Y S, JANG M. Design and implementation of HDFS data encryption scheme using ARIA algorithm on Hadoop[C]// 2017 IEEE International Conference on Big Data and Smart Computing. IEEE, 2017: 84-90.

[6] LIN H Y , SHEN S T , TZENG W G , et al. Toward data confidential-

ity via integrating hybrid encryption schemes and Hadoop distributed file system[C]// 2012 IEEE 26th International Conference on Advanced Information Networking and Applications (AINA). IEEE, 2012: 740-747.

[7] BHATOTIA P, RODRIGUES R, VERMA A. Shredder: GPU-accelerated incremental storage and computation[C]//The 10th USENIX conference on File and Storage Technologies. 2012.

[8] COHEN J, ACHARYA S. Towards a trusted Hadoop storage platform: design considerations of an AES based encryption scheme with TPM rooted key protections[C]//The 2013 IEEE 10th International Conference on Ubiquitous Intelligence & Computing and 2013 IEEE 10th International Conference on Autonomic & Trusted Computing. IEEE, 2013: 444-451.

[9] ZHOU W, XU Y, WANG G. Research on key management for multi-privileged group communications[C]// IEEE Trustcom/BigDataSE/ISPA. IEEE Computer Society, 2015: 151-158.

[10] KANDAH F I, NICHOLS O, YANG L. Efficient key management for big data gathering in dynamic sensor networks[C]// International Conference on Computing. IEEE, 2017: 667-671.

[11] ALBAKRI A, MADDUMALA M, HARN L. Hierarchical polynomial-based key management scheme in fog computing[C]// IEEE TrustCom/BigDataSE. IEEE, 2018: 1593-1597.

[作者简介]

金伟（1994– ），女，北京人，中国科学院信息工程研究所博士生，主要研究方向为大数据访问控制与密钥管理。

余铭洁（1998– ），男，江西景德镇人，中国科学技术大学博士生，主要研究方向为大数据访问控制与密钥管理。

李凤华（1966– ），男，湖北浠水人，博士，中国科学院信息工程研究所研究员、博士生导师，主要研究方向为网络与系统安全、大数据安全与隐私保护、密码工程。

杨正坤（1994– ），男，重庆人，中国科学院信息工程研究所硕士生，主要研究方向为入侵响应。

耿魁（1989– ），男，湖北红安人，博士，中国科学院信息工程研究所助理研究员，主要研究方向为网络安全。