GuardSpark: Spark 访问控制增强机制

宁方潇[1,2], 文　雨[1], 史　岗[1]

[1] 中国科学院信息工程研究所, 北京 中国 100093
[2] 中国科学院大学 网络空间安全学院, 北京 中国 100049

摘要　作为最流行的大数据分析工具之一, Spark 的安全性却未得到足够重视。访问控制作为实现数据安全共享的重要手段, 尚未在 Spark 上得以部署。为实现隐私或敏感数据的安全访问, 本文尝试提出一种面向 Spark 的访问控制解决方案。由于 Spark 架构具有混合分析的特点, 设计和实现一个可扩展支持不同数据源的细粒度访问控制机制具有挑战性。本文提出了一种基于声明式编程和 Catalyst 可扩展优化器的统一、集中式访问控制方法 GuardSpark。GuardSpark 可支持复杂的访问控制策略和细粒度访问控制实施。文章实验部分对所提访问控制方法在 Spark 上进行了原型实现, 并对其有效性和性能开销进行了实验验证和评价。实验结果表明, GuardSpark 可实现细粒度、支持复杂策略的访问控制机制。同时, 该方法带来的性能开销可忽略, 并且系统具有可扩展性。

关键词　Spark SQL；访问控制；安全优化；大数据
中图法分类号　TP309　DOI 号　10.19363/j.cnki.cn10-1380/tn.2017.10.006

GuardSpark: Access Control Enforcement in Spark

NING Fangxiao[1,2], WEN Yu[1], SHI Gang[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

Abstract　As one of the most popular big data analysis tools, the security of Spark has not raised sufficient concern. Access control is an important means of safe data sharing, which was not deployed on Spark. In order to safely access privacy or sensitive data, this paper attempts to propose an access control solution for Spark. Due to the unification of Spark framework, it is very challenging to design and implement a scalable and fine-grained access control schemes which support variety of data sources. We proposed GuardSpark, a unified, centralized access control method based on declarative programming and Catalyst extensible optimizer. GuardSpark supports complex access control policies and fine-grained access control enforcement. The experimental part of this paper implemented the proposed prototype on Spark to verify the correctness of the function of AC enforcement. We also evaluated the system overhead introduced by AC enforcement. The experimental results show that GuardSpark can achieve fine-grained access control and support complex AC policies. At the same time, the performance overhead of this approach is negligible with good scalability.

Key words　Spark SQL；access control；　security optimization；big data

1　引言

随着 2016 年全球全年互联网流量数据超过 1ZB[1], 世界正式进入 ZB 级时代。大数据于人类社会的重要性是其蕴含的极大价值, 而挖掘这些价值的大数据工具则成为驾驭大数据的关键。自 2009 年作为 Apache 基金会的顶级项目以来, Spark[2]已成为当今最流行的大数据工具之一[3]。截止 2016 年底, 该开源项目已有来自 250 多个组织的超过 1000 名贡献者。在工业界, Spark 项目成功吸引了包括 IBM, Yahoo, Facebook 等 IT 巨头在内的众多企业的大力投入和支持。甚至, 作为 Hadoop 商业版本厂商之一的 Cloudera 宣布其核心产品将从 Hadoop 上迁移到 Spark 上。在实际生产环境应用中, Yahoo、eBay 和 Netflix 等互联网巨头内部部署了大规模 Spark 节点, 可在超过 8000 个节点的集群上处理 PB 级数据[4]。

虽然大数据技术得到了各界的极大关注, 并取得了快速发展, 但仍面临诸多问题。其中, 大数据安全是不容忽视的重要问题之一[5]。首先, 大数据的巨大价值使其极易成为攻击者的目标。例如, 2015 年全

通讯作者: 文雨, 高级工程师, Email: wenyu@iie.ac.cn。
收稿日期: 2017-03-20; 修改日期: 2017-06-02; 定稿日期: 2017-08-23

美第二大的医疗保险公司 Anthem 遭受黑客攻击, 泄露了近 8 千万客户和雇员的个人信息, 包括姓名、生日、社会保险号、住址、邮件地址和薪资等信息[39]。同年, 美国联邦人事管理局 2 千 2 百万历史和现任联邦雇员的个人数据被泄露, 其中包括 5 百万的指纹信息[38]。其次, 为了使数据价值被最大化的利用, 数据通常需要被开放共享以满足各方的分析决策的使用需求, 又进一步加大了数据保护的难度。然而, 当前大数据技术的发展主要关注大数据的大规模(volume)、高速(velocity)和多样性(variety)[6] 等特点带来的挑战, 而数据安全则较少考量, 因此不能满足大数据安全共享需求。

虽然访问控制被公认为确保数据安全共享的重要方法之一[7], 但在大数据技术发展中没有受到足够的重视。访问控制通常作为抵御内部威胁和攻击的一种基本方法。2007 年, 美国 FBI 的计算机犯罪和安全调查结果显示, 59%的被调查机构遭受过内部攻击, 并且 25%的被调查机构超过 40%的安全事故损失是由内部攻击造成。著名的美军基地“曼宁泄密事件”和斯诺登“棱镜门”事件, 则是典型的内部攻击案例。因此, 研究面向大数据的访问控制对于大数据安全共享有重要意义和实际价值。然而, 在 Spark 中实现访问控制需要应对很大挑战。

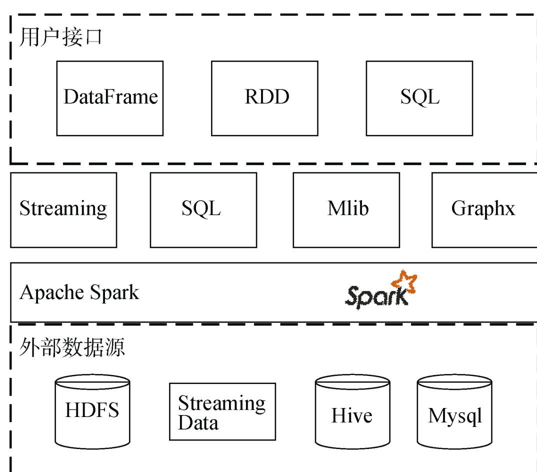


图 1 Spark 系统框架

图 1 展示了 Spark 应用场景的典型架构。该架构具有混合分析特点: 系统可接入不同类型数据源, 既有分布式文件系统的持久化文件、关系型数据库的关系表、实时流数据, 甚至有其他大数据系统的输出数据。现有访问控制是一种特定化(ad hoc)或补丁式的措施, 而不是一种完整的解决方案。首先, 各类系统自身对访问控制的支持参差不齐。其中, 一些系统不支持访问控制, 如 Apache Flume[8]。而有一些系统仅支持粗粒度的访问控制。如 Hadoop HDFS 的访问控制机制是一种全或无的控制手段, 只要获取了读写权限即可对整个文件进行访问。Apache Kafka[9] 虽然具备认证和加密机制, 支持客户身份认证和模块间数据传输加密, 但不能实现话题内数据流的细粒度控制。其次, 中间件式的访问控制增强系统, 如 Apache Sentry[10]和 Apache accumulo[11] , 虽然提供细粒度访问控制, 但尚不能支持所有数据源, 如 Apache Kafka、Apache Flume 等流数据。由于需要屏蔽各种异构数据源的差异, 中间式系统复杂度高, 并且开发周期长, 需要不断支持新的数据源。因此, Spark 数据访问控制的挑战是需要一个可扩展支持各种数据源的细粒度访问控制机制。

本文首次提出一种面向 Spark 系统的统一、集中式的访问控制机制 GuardSpark。GuardSpark 基于 Spark 的声明式(declarative)编程接口和 Catalyst[12]可扩展优化器, 实现了细粒度的访问对象(即数据)的识别和标识、访问操作识别和访问策略实施。通过统一的声明式编程接口与基于规则的树状图变换, 实现了访问控制与数据源和用户应用代码的解耦合, 能有效地对各种数据源和用户行为施加集中化控制。通过对具有标准表达形式的声明式编程进行安全检查和变换, GuardSpark 控制的对象实际是高级运算子的执行流, 而无需干预数据源和用户程序代码, 从而为 Spark 提供了一种集中、统一可扩展支持各种数据源的细粒度数据访问控制机制。

本文的主要贡献如下:

- 首次提出了 Spark 混合分析场景的数据访问控制问题。
- 设计和实现了一种面向 Spark 的统一、集中式访问控制机制 GuardSpark。
- 基于 Spark 的 Catalyst 可扩展优化器实现 GuardSpark, 符合 Spark 结构化数据处理技术的发展趋势, 具有前瞻性。同时, 使得 GuardSpark 对数据源和用户程序透明。
- GuardSpark 实现了结构化数据键值对粒度的访问管理能够支持复杂访问控制策略。
- 本文已在 Spark 系统中实现了 GuardSpark 原型, 并使用基准测试 TPC-DS 评价了本文方案的有效性和性能开销。

本文结构如下: 第二节介绍相关工作进展, 第三节是本文的研究背景介绍, 第四节系统概述简要介绍 GuardSpark 的访问控制的框架, 具体的工程实现位于第五节, 第六节的实验评价验证了所提机制

的有效性并对其引入性能开销进行了分析, 最后一部分是全文的结论与总结。

2 相关工作

本文的相关工作包括三个方面: Spark 系统中访问控制相关研究, 现有大数据系统中访问控制技术, 以及传统数据库领域基于查询重写技术的访问控制增强方法。

2.1 Spark 系统访问控制相关研究

Apache Spark 的发布版本中并未提供基于策略的细粒度访问控制能力。当前仅支持共享口令的认证机制，而认证依赖于访问控制列表 ACLs，经过认证的用户被授权访问所有数据或者进行特定操作，但无法实现数据细粒度的访问控制。由于 Spark 出现时间较晚，关注 Spark 访问控制的研究尚处于起步阶段，SparkXS 是仅有的现在可查的研究。

SparkXS[13]首次提出了针对 Spark 系统及其流处理扩展 Streaming 的基于属性的访问控制(Attributed-based access control, ABAC)解决方案。SparkXS 提供了对流式隐含数据定义访问控制策略的能力, 文章重点举例介绍了如何表达访问控制策略, 但并未详细说明具体的部署方式。用户的属性被用于用户身份的认证, 经过认证的用户向策略决策单元 PDP 发起请求, PDP 根据预定义的策略集判定是否赋予用户执行权限。

SparkXS 属于特定化的解决机制，它仅针对流数据的处理场景，不能为 Spark 其他组件提供统一的解决方案。在其实现方式上也存在不足：需要对用户所提交的具体操作进行识别, 只有在用户身份, 访问对象, 访问目的与访问控制策略完全一致时, 才应赋予用户对数据对象的访问权限。不仅要保证单个RDD变换满足访问控制需求, 还要保证RDD操作链作为一个整体也满足用户对数据的访问权限, 如此才可能有效地辨识出真正符合访问策略要求的 RDD 操作链。而 RDD 函数式编程与命令式编程的特点, 使得识别 RDD 的动作变得异常困难。

2.2 其他大数据系统中访问控制技术

由于 Spark 系统出现较晚, 与其相关的安全研究尚十分有限, 故只能借鉴 MapReduce 系统及其 Hadoop 实现上相关的工作。Hadoop 生态中已出现了补丁式的中间件对现有系统进行访问控制的增强, 典型代表有 Apache Accumulo 与 Apache Sentry。Accumulo 扩展了 Google BigTable 的数据模型, 针对的是 HDFS 中的结构化存储, 每个键值对都有各自的安全标签, 安全标签记录当前关键字的可见范围, 基于此实现了键值对单元级的访问控制。Apache Sentry 是面向 Hadoop 的基于角色授权的细粒度的访问控制模块, 它能为经过授权认证的用户和应用程序提供数据访问权限的精确控制, 支持 Hive, Impala, HDFS 等。Accumulo 与 Sentry 都试图提供一种中间层的检查机制, 它们的共同之处都是针对存储在分布式文件系统中持久化文件, 目标是构建一种可插拔的控制组件, 有效地认证对 Hadoop 资源的请求。但是它们均不支持 MapReduce 编程模型中复杂的访问控制, 也不适用于流数据的实时处理场景。

GuardMR[14]在 MapReduce 系统中实现了细粒度的访问控制, 它依赖关键组件过滤器完成对受保护数据源的授权视图构建。GuardMR 是独立于 MapReduce 系统的模块化框架, 用户与 MapReduce 系统所有的外部通信会被拦截, 访问控制模块会将的安全管理员定制的访问控制策略转换成过滤器筛选条件, 输入数据经过过滤后的才能进行 MR 作业。GuardMR 通过数据的预过滤限制用户可访问数据在其授权范围之内, 不能支持复杂的访问控制策略。

Hu 等人[15]认为访问控制策略的失配比加密组件或协议更易造成隐私或安全问题, 提出了一种面向分布式大数据处理集群的通用访问控制机制。研究主要关注的是集群环境中主从系统上访问控制能力的协同一致问题。Airavat[16]针对的是分布式计算中敏感数据的安全隐私保护, 将强制访问控制与差分隐私技术相结合, 克服了仅依赖访问控制无法避免隐私泄露的弊端。

2.3 基于查询重写的访问控制方法

本文所提方案受到了文献[17]的启发, 区别在于 GuardSpark 的访问控制实施过程不直接对查询代码本身进行重写, 而是通过对节点对象的树状图的进行变换实现。与 Catalyst 优化器的深度整合可最大程度的利用已有的优化流程, 兼顾访问控制与执行计划优化。文献[17]致力于在数据库系统中使用查询重写技术实现细粒度的访问控制, 提出了基于授权视图的访问控制模型和查询重写过程中的条件有效的概念, 扩展了查询重写使其具备了访问控制的能力。最后基于一组强有力的推断规则, 可确保查询结果满足用户的授权视图。文献[18]提出了集成与 Aurora 数据流系统的访问控制增强原型, 基本原理是用安全运算符替换非安全运算符, 安全运算符会过滤掉非授权访问的元组。在此基础上, Carminati 等[19]提出了基于查询重写的数据流管理系统访问控制增强框架。该框架会以一种遵循访问控制策略的方式重写用户查询, 使得查询结果中不包含策略中不允许访问的属性或元组。为此, 作者提出了安全运算子以及

查询重写算法, 安全运算子在普通算子的基础上增加了访问控制策略的执行功能会过滤掉禁止访问的数据域, 也就是说安全运算子的返回结果中只包含访问主体所授权访问的访问对象。而查询重写算法则负责遍历原始查询执行流在特定节点使用安全运算子替换对应的运算子以实现查询图的安全重写。

无论是查询语句改写还是运算符替换，都属于用户编程层次的控制，虽然能针对性解决各自领域内的访问控制问题，但无法满足 Spark 统一控制的需求。上述工作均未考虑性能方面的问题，大数据系统的 3V 特点决定必须统筹考虑访问控制与性能优化。作为探索性工作, 它们共同的不足是查询重写未考虑到性能优化的问题。

3 研究背景与威胁模型

3.1 Apache Spark

3.1.1 Spark

Apache Spark[20]致力成为大数据处理的统一引擎, 是一种高效通用的集群计算系统。Spark 系统具有类似于 MapReduce 编程模型, 但在其基础上进行了扩展——弹性分布式数据集(Resilient Distributed Datasets, RDD)[21]。RDD 是 Spark 特有的数据共享抽象, 使用这种抽象 Spark 能够承担之前需要由多种不同引擎才能完成的许多不同工作量。RDD 的两大特征为内存计算以及粗粒度的变换, 正是基于此 Spark 获得计算的高效性以及强容错性。一方面 RDD 允许将中间结果持久化存储在内存中, 很好地满足了迭代算法和交互式数据挖掘对数据重用的要求。另一方面基于粗粒度的变换而非细粒度状态更新, RDD 只需记录构造数据集的变换操作链而不必存储实际数据, 当 RDD 的某个分区发生缺失, RDD 拥有足够的信息关于如何从其他的 RDD 重新计算此分区, 具有很高的容错性。

Spark 不仅继承了 Hadoop MapReduce 的所有优点, 而且还具备三大优势。一是计算速度更快, 内存计算时速度高达 Hadoop 的 100 倍。二是易用性, 提供多种运算符且允许自定义运算符。三是通用性, 在此之前出现的系统只关注特定的计算领域, 比如 MapReduce[22], Storm[23], Dryad[24], GraphLab[25]等, 而 Spark 致力于建立统一的计算框架, 能在同一平台上处理批处理、SQL 查询、流数据以及机器学习算法等多种负载。

3.1.2 Spark SQL

早期的系统, 比如 MapReduce 为用户提供了功能强大但是低层级的过程式编程接口。使用此类系统编程需要用户对算法和系统内部实现有很深的理解, 才能手动的对运算链进行优化以获取高性能。后来, 出现了许多新系统通过提供关系接口给大数据以期实现更加高效的用户体验。这些系统利用声明式查询提供丰富的自动优化能力, Spark SQL 就是其中之一。

Spark SQL 是 Spark 的一个扩展组件, 设计之初是为处理关系型数据的查询任务, 现在的 Spark SQL 远超出了 sql 查询的范畴, 它能高效地进行结构化数据的处理。Spark SQL 的设计目标是更简洁地编写 Spark 应用程序, 更高效地执行分析任务, 为实现用更少的代码, 读取更少的数据的目标, 需借助优化器完成最繁重的性能优化任务。Spark SQL 及其前辈 Shark[26] 使用了类似分析型数据库的技术在 Spark 上实现了关系查询。如其他关系数据库一样, Spark SQL 支持列存储、基于代价的优化以及用于查询执行的代码生成。Spark SQL 允许使用声明式查询语句处理结构化关系型数据集, 其内部包含了高度可扩展的查询优化器 Catalyst。Catalyst 优化器基于 Scala[27]的函数式编程范式, 能够会对 SQL-Like 查询语句进行解析、基于规则的逻辑优化、基于代价的物理优化, 并最终将查询图转化为 RDD 变换流执行。使用声明式查询用户不必具体描述如何使用 RDD 完成特定作业, 只需告诉引擎想要得到哪些数据, 从而可以避免了用户由于对 RDD 理解不深刻, 编写出低效的 RDD 变换操作代码。

Spark SQL 的核心是 Catalyst 优化器, 下面介绍 Catalyst 的具体实现, 其流程图如图 2 所示:

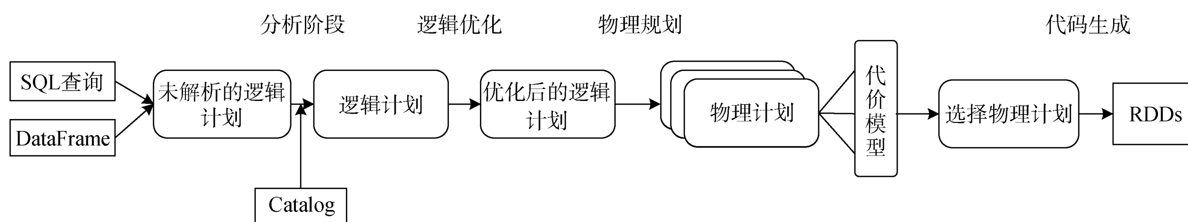


图 2 Spark SQL 查询执行流程图

圆角矩形代表 **Catalyst Trees**

Catalyst 树变换通用框架会在 4 个阶段被使用:

1. 逻辑计划的解析;
2. 逻辑计划的优化;
3. 物理计划的制订;
4. 将部分查询编译为 Java 字节码的代码生成。

Catalyst 中最主要的数据结构就是由节点对象组成的树。每个节点拥有一个节点类型以及零个或多个子节点。节点对象是不可更改的但支持通过函数变换操作。操作树的具体方式称为规则(Rules)。规则是从一种树结构到另一个种树结构的函数映射。

最普遍的方式是使用一组模式匹配函数发现并使用特定结构来替换子树。模式匹配允许从潜在的嵌套代数数据类型中抽取数值。Catalyst 能够遍历树中的所有节点并对其适用模式匹配函数, 每个与之匹配的模式会被变换成对应的结构。

3.2 威胁模型和假设

GuardSpark 假设用户可能怀有恶意的目的, 因此必须对用户提交的作业进行控制。同时我们假设 Spark 用户不能直接访问 Hadoop 文件系统等外接数据源, 一旦用户拥有数据源端系统的访问权限, 他们完全可以不受 Spark 访问控制的限制直接检索全部数据文件。比如, 用户可以通过 HDFS 命令行直接读文件。本文亦假设敌手不能获取集群节点操作系统的 root 权限, 也就是说 OS 和 Spark 运行时环境是安全的。

在 Spark 的使用模型中, 终端用户提交 Spark 作业对输入数据源进行信息。由于 Spark 编程模型中的数据抽象 RDD 是不可更改的只读数据集, 故任何操作都不会改变原数据集本身, 因此只需控制读权限。最后, GuardSpark 基于现有安全认证基础之上, 通过认证机制获取访问主体的身份信息。当前的 Spark 系统中提供了基于共享口令的认证功能。创建 Spark 应用的用户通过配置 ACL, 对其他用户身份进行认证, 口令比对成功的用户可通过认证, 认证用户被授权访问创建者包括中间 RDD 的全部数据。

4 系统概述

决定采用何种策略实施访问控制是实现访问控制的关键, 存在三种可能的解决方案: 预处理(preprocessing), 后处理(postprocessing)以及查询重写(query rewriting)[19]。预处理顾名思义就是在查询之前对要分析的数据进行过滤清洗。预处理的手段主要是传统的抽取-转换-加载(Extract Transform and Load, ETL)方式。ETL 依赖于可信的处理单元在数据分析之前根据用户的安全分级滤除数据集中的敏感数据。预处理实现简单, 但存在两方面的不足, 一是可能会造成数据爆炸。不同的角色对应不同的安全分级。对于每个安全级别, 都要生成单独的数据文件, 角色的增长意味着数据副本数目的指数级增长, 势必会造成数据量的爆增, 这在大数据应用中尤为严重。另一方面, 预处理范式无法支持复杂的访问控制策略, 是一种允许访问与拒绝访问的二元对立。假如数据集中存在允许间接访问的数据项, 这些数据不允许以直接的形式输出, 但是可作为过滤或排序条件, 采用预过滤方式要么在前端过滤掉指定数据项要么允许其进入分析系统, 总之无法实现间接访问的控制要求。后处理方式在得到查询结果之后再对结果进行二次处理, 其主要的问题会产生大量的不必要运算, 会造成运行时开销。相较而言, 查询重写的方式在查询定义阶段重新定义授权限制下的查询, 既能满足复杂的访问控制要求, 又不引入运行时开销。

本文借鉴了查询重写机制的思想, 但是在 Spark 计划优化阶段实施控制, 不直接控制数据源和用户应用程序代码, 实现了访问控制与数据源和用户程序代码的解耦。本方案工作原理可抽象为图 3 所示, 为实现集中控制的目的, 用户只能通过统一的用户交互接口提交声明式分析作业, 声明式编程既简化了编程复杂度也能实现执行过程的统一的抽象表达。作业进入 Spark 后会被解析成高级操作符的树状表达, 逻辑树中的节点代表具体操作而层次关系表示输入输出关系。本文的工作的重点就是扩展现有性能优化机制, 提出满足 Spark 细粒度访问控制需求的变换规则。对原始树状图适用这些安全控制规则, 原有逻辑树会变换为满足访问控制规则的逻辑树, 然后对其进行代码生成, 分布式执行最后得到执行结果, 以实现访问控制的增强。GuardSpark 控制的实质是对具有标准表达形式的声明式编程进行安全检查并变换的过程, 控制的对象是高级运算子的执行流, 避免了直接控制底层的基本运算子。

集中式访问控制并不影响数据分析过程的分布式执行, 而是指访问控制有共同的入口, 用户提交的所有作业都会先经过 Spark 内部集成访问控制单元的强制重写过程。执行流的访问控制变换作用在执行流生成阶段, 不会增加系统的运行时开销。所得执行流图仍照常调度至分布式集群上计算处理。建立树状图的目的是允许对作业进行整体地分析和优化, 优化之后得到的最优执行计划会分解成底层的 RDD 操作流。

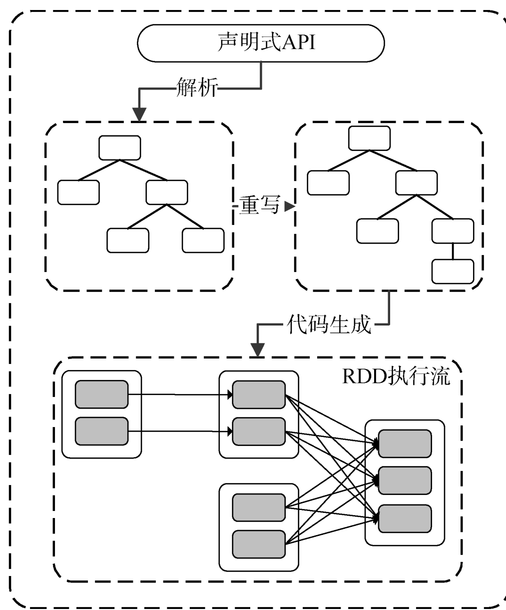


图 3　基于执行流重写的访问控制设计

Spark 最新的发布版本尚未提供大一统的声明式 API, 但是已有团队在做这方面的尝试[27]。但在 Spark SQL 结构化处理工具有着类似的实现机制, 为验证本文所提机制的可行性, 本文根据 Spark SQL 的特点设计出基于逻辑优化的细粒度的访问控制机制。借助结构化数据抽象, 可使用声明式编程编写分析作业, 避免了访问控制实施中识别作业中具体操作或整个操作链操作目的的困难, 可定制的逻辑优化过程又为部署访问控制规则提供了灵活的支撑。Spark SQL 的分析作业逻辑上可视作树状的逻辑计划的执行流, 使用逻辑计划表示的优势在于操作数与运算子都是抽象为逻辑计划的节点, 从而能从整体上分析作业中所涉及的访问对象以及访问的目的。基于规则的优化能确保满足模式匹配的树结构变换为满足要求的新的树结构。通过树的遍历并适用规则就能实现对逻辑计划的优化与控制。基于执行流重写的访问控制的另外一个优势在于支持复杂的访问控制策略, 这是由于规则作用在可作用逻辑树的任意级别上, 既可以控制数据又可以控制操作本身, 而不仅限于控制输入和输出。

GuardSpark 是对 Spark 及其结构化数据处理模块 Spark SQL 的扩展, 由两大主部分组成, 策略管理模块和策略实施监视模块, 见图 4。策略管理模块的用途是: 安全管理员通过策略管理模块为不同安全级别的所有用户制定不同的授权权限。使用策略管理模块管理员可以增加, 删除, 更新控制规则。

首先, 需要根据 Spark 的分析特点制定满足条件的访问控制策略。文献[28]中定义了复杂的访问策略, 从四个维度表征复杂的访问需求, 访问的间接性、访问的组合性、聚合权限以及联合访问权限。在本模型中暂只考虑访问的间接性: 直接访问权限和间接访问权限。被授权进行间接访问的数据, 在查询中可被用于过滤, 联合以及排序, 但不应直接投影到输出结果之中。而被授权直接访问的数据, 允许用户所有类型的访问, 既可作为谓词表达式中操作数, 也可被选择并投影到查询结果集合当中。

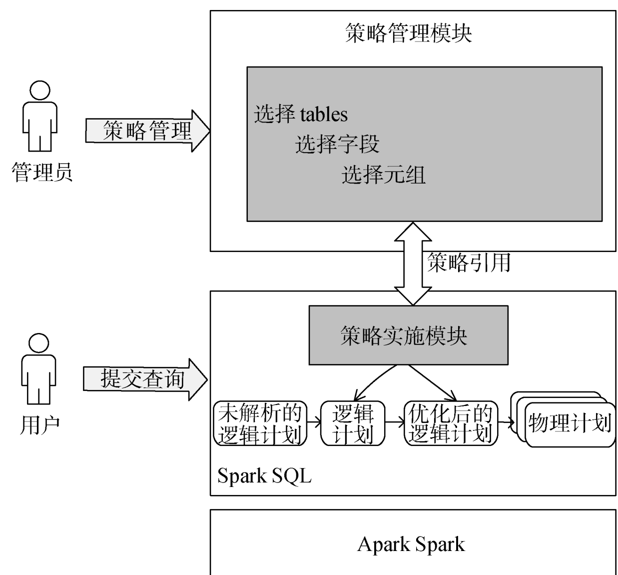


图 4　GuardSpark 系统组成框图

访问控制策略是访问控制规则的集合, 而访问控制规则描述的访问主体对访问对象的访问权限。访问主体和访问权限易于明确, 下面给出访问对象的形式化表示。访问对象由 3 部分组成, 定义 4.1 是其表示格式。

定义 4.1　访问对象 object 可定义为三元组(files, columns, predicate), 此处:

files 是一组存储于外部持久化存储中的结构化文件的文件名或者外部输入流数据标识的集合 $\{f_1, f_2, \cdots, f_n\}$;

columns 表示一组属性 $C_1, C_2, \cdots, C_m$, 对于任意 $i \in \{1, \cdots, m\}$, C_i 属于文件集合的笛卡儿积 $\left(f_1 \times f_2 \times \cdots \times f_n\right)$。

predicate 是返回值为布尔类型的表达式, 谓词表达式作为筛选条件, 可以是某个属性与具体值之间的比较也可以是两个属性之间的比较。关系运算符的左右操作数满足类型兼容的要求。

定义 4.2　Spark 中结构化查询访问控制策略是一个三元组: (sub, obj, priv)。

访问主体 sub 可以是用户身份标识, 也可以用户的角色信息, 访问对象 obj 的定义如前所述, 访问权限 $priv \in \{read, indirect\,read, deny\}$。Read 读权限允

许用户对访问对象进行包括选择, 投影在内的所有读访问, 而间接访问权限则限制数据的直接输出, 只允许用于过滤, 连接以及排序等操作, deny 权限则拒绝任何形式的数据访问。Read 权限是记录项的默认访问权限, 不明确限制的情况下数据项是可以直接访问的。

集成于 Catalyst 优化器中的策略实施模块的作用是策略的解析和适用, 它负责完成: 1)识别用户的身份信息并去策略库中找到所有与其相关的规则; 2)当逻辑树中任意结构与访问规则之中定义模式相匹配时, 访问控制规则就会适用到对应子结构上, 实现执行流图的安全优化。新结构的变换过程可能是原节点的条件增强, 也可能在原节点前后增加新节点, 或者直接删除原节点等等, 总之访问控制规则适用的过程就是生成新的逻辑计划的过程。策略实施模块允许安全管理员根据需要定制访问控制规则, 具备很强的灵活性和扩展性。基于执行流重写的访问控制还能实现传统访问控制手段所不具备的控制能力。比如, 关系 R 中的两个属性 A, B 最多同时只允许投影 1 个, 就不能借助简单的过滤的方式实现。但是在本文的设计中, 通过检查逻辑计划中 Project 列表, 只要 A, B 同时出现在列表中就选择性的移除一个, 即可实现此类控制。

5 具体组成

5.1 策略管理模块

在 GuardSpark 的系统组成中, 策略管理模块主要实现访问规则的添加、更新、删除功能。策略库是独立于 Spark 系统的外部持久存储中固定格式的文件, 在本实验中为保存在 HDFS 中的文件。只有安全管理员能够对此文件进行写操作。策略管理模块负责策略的引用, 解决策略集中不同规则间的重复与冲突等问题, 将与用户直接相关的规则集合下发至策略实施模块。这方面的工作已有很多可以借鉴的研究[29-31], 而本文中研究的重点是如何在大数据分析系统中实施访问控制策略, 故不展开详细论述。

5.2 策略实施模块

策略实施模块基于 Catalyst 优化器实现, Catalyst 具备高度可扩展的模式匹配能力, 可实现执行树的启发式变换。为了最大程度的利用 Catalyst 的优化能力, 安全优化应该处于逻辑优化阶段之前, 这样安全规则执行完成后所得到的新的逻辑计划树能正常按照逻辑优化规则进行优化。

策略实施模块集成在 Catalyst 优化器的逻辑优化阶段, 当用户启动一个 SparkSession 时, 用户的认证信息会作为访问主体传递到外部策略管理模块中, 所有与该主体相关的访问规则都会在访问实施模块内被解析成安全规则。安全规则要添加到 Catalyst 优化流程中才能起作用, 实际上添加这些安全优化规则的作用原理类似于 Spark 提供的用户定制优化, 区别在于 Spark 现阶段为用户提供的定制优化都位于优化阶段的末端, 而考虑到安全优化规则可能对性能产生的影响, 需要在分析或者优化阶段的特定位置部署安全优化, 确保执行安全规则转化后的逻辑计划仍能进行性能优化。由于安全优化不同其在整个优化阶段中的作用位置也不尽相同, 优化策略也不一致, 需要为不同的安全规则定制不同的接口, 参照现有用户定制规则接口, 对 SparkOptimizer 类进行重写改造实现, 最后将安全定制类置于优化 batches 中适当位置。

为了从源头对数据的控制, 需要对 DataFrame 或者逻辑计划进行标记, 标记的方法是对创建 DataFrame 或者 Table 的操作符进行改造, 使得外部数据通过 API 接口进入 Spark 时, 其生成的 Catalog 元数据中包含对象身份信息, 比如外部文件的存放路径或者是数据流的 ID 等, 这些唯一不重复的对象信息会作为对象的身份标识伴随着逻辑计划的优化执行的全过程。对于持久化存储的文件, 无论是本地文件系统还是 HDFS 分布式文件系统中的数据, 在数据输入 Spark 时, 既可以使用 sql 语句创建, 也可以调用 DataFrame 运算符实现 DataFrame 的新建。而数据流导入 Spark 系统, 也可由 RDD 转换成 DataFrame, 方便的参与查询。

以 HDFS 中持久化文件为例, 文件加载入 Spark 系统可调用数据定义操作符 createExternalTable 方法, createExternalTable 会根据数据源, 数据结构和一组选项从指定路径创建外部表, 返回相应的 DataFrame。在 GuardSpark 的实现中, 将输入的 path 参数保存在 CatalogTable 中 properties 成员字段内。在分析器解析逻辑计划时, 通过检查 properties 映射集中是否包含 path 键值对判定访问对象的身份。

安全优化规则与 Catalyst 自带的优化规则的区别在于, 安全优化不再强调保持变换前后等价性, 是一种条件等价变换, 实际上安全规则会缩小执行树的授权视图或者说访问范围, 常用的安全优化规则可以是在关系扫描输入结点后增加一个过滤结点, 缩减投影结点的可投影列表集合中的元素等。下面是本文功能验证过程所用到的两个安全规则:

安全规则 1: 过滤器添加规则。适用对象: 不可

访问的对象。检查所有叶子节点是否包含受保护对象中的 files, 当满足条件时在叶子结点上添加一个以 object.predicate 为过滤条件的 Filter 结点。

安全规则 2: 投影节点投影列表清洗规则。适用对象: 间接访问对象。检查投影结点的输出列表是否包含间接访问权限的数据域, 假如存在任意间接访问字段, 则将其从投影列表中移除, 用新的投影结点替换原来的结点。

安全优化规则将访问控制的强制执行转化为具体的执行计划树的变换过程。

5.3 安全性分析

GuardSpark 的基本假设是访问主体暨用户的权限是经过合法认证的, 在此假设之上, 对访问对象从外部源头开始施加的控制能确保访问主体能遵守访问控制原则访问受保护对象。安全管理员权限高于普通用户, 负责访问控制策略库的创建、更新、删除等操作, 而策略库本身对普通用户并不可见, 用户也就无法对策略库进行修改。用户登录 Spark 之后其认证信息会被用于控制规则的引用, 与其相关的控制规则会在其提交作业之前完成对优化器的定制, 由于访问控制规则会注入到 Catalyst 优化器的核心代码中, 用户提交的所有作业都会严格执行访问控制规则, 不存在被绕过的可能。

由于安全增强位于系统应用层, GuardSpark 无法抵御硬件旁路攻击, 如通过内存直接访问读取文件内容。也不能在 OS 被攻破条件下实现预期的访问控制, 一旦用户通过系统漏洞获取了安全管理权限即可任意改动安全策略库。

6 实验评价

本文在 Spark 上扩展实现了集中式访问控制原型, 实验部分首先对 GuardSpark 访问控制的功能进行验证, 然后评价其带来的性能开销。

6.1 有效性验证

作为原型验证, Spark 运行在 Standalone 模式下, Spark 的版本为 Spark 2.1.1。有效性测试主要测试的是 AC 增强机制是否能够完成预定的访问控制效果。作为原型验证, 现阶段的实现直接对 sql catalyst 优化器进行源码修改, 尚未实现外部可定制的访问规则接口。简单起见, 考虑单用户的场景, 并选择 Spark 自带的 SparkHiveExample 示例进行验证, 相关数据与查询在 Spark 的发布版本中可见。

首先给定访问控制策略, 按照定义 4.2 的格式:

{*user*, (*examples/src/main/resources/kv1.txt, key, key>70*), *indirect*}

受保护对象 kv1.txt 中的 key 字段, 在满足 key 的值大于 70 的情况下, 可被访问主体间接访问, 即可被用于过滤, 和排序, 但无法直接投影到输出结果之中。根据上述规则, 修改 Spark sql 中 catalyst 的分析器和优化器代码, 将安全优化规则集成到查询工作流执行中。编译打包得到满足控制规则的二进制安装文件, 然后在集群上重新部署修改后的 GuardSpark, 并在其上运行查询并检查查询结果是否满足控制要求。

现将 GuardSpark 上的执行结果, 与 Spark 原始发布版本上执行的重写过了的查询的结果作对比。重写查询需要移除原查询 SELECT 语句中所包含的 key 字段, 并在 WHERE 表达式中增加过滤条件 key>70。

实验结果表明, GuardSpark 上部署的访问控制规则能够实现预期的控制目标, 在保证 key 字段不被直接输出的情况下, 仍能够作为 filter/join 的条件, 且 key 允许访问的范围也受到了相应的控制。

表 1 有效性实验结果

Spark	GuardSpark	结果	是否一致
SELECT value FROM src WHERE key>70	SELECT * FROM src	443 项 Value 字段	是
SELECT COUNT(*) FROM src WHERE key>70	SELECT COUNT(*) FROM src	443	是
SELECT value FROM src WHERE key < 10 AND key>70 ORDER BY key	SELECT key, value FROM src WHERE key < 10 ORDER BY key	空集	是
SELECT r.value s.value FROM records r JOIN src s ON r.key = s.key WHERE s.key>70	SELECT * FROM records r JOIN src s ON r.key = s.key	29 项, 两个 Value 字段	是

为了进一步验证所提机制的有效性, 本实验有采用复杂的数据集进行了功能验证。此处使用的是 TPC-DS[32]数据集, TPC-DS 自带有数据集与查询生成工具, 允许使用者根据需要生成可变规模的数据集。选择 q7-derived 中的查询作为测试样例, 首先制定访问控制策略, 为了证明 GuardSpark 的访问控制能力, 根据每个查询的不同特点, 随机指定 2 组访问控制规则。

在 Spark 集群上运行的改造过的查询, 需要将查询中的 indirect 权限的字段从 select 的输出列表中移

除, 而deny权限的字段不仅要从select的输出列表中移除, 还不应出现在 filter, join, sort 的表达式中。不同的访问规则下的GuardSpark上查询结果, 与Spark上执行的相对应的查询修改版本的结果比较如下表2。实验结果表明 q7-derived 相关的查询都受到了预设控制规则的约束, 得到了预期结果。

6.2 性能评价

性能评价部分主要测试引入 AC 机制后带来的计算开销。首先介绍实验平台的相关配置参数。

6.2.1 实验环境

测试环境介绍: 一台安装有两个 Intel Xeon E5-2650v4多核处理器曙光620服务器作为实验平台, 处理器主频为 2.2GHz, 服务器的内存为 128G。使用 virtualbox 创建 3 个虚拟机, 由 1 个 master 节点, 2 个 slave 节点组成 Spark 虚拟集群, 为每个节点的分配 4 核, 16G 内存, 200G 虚拟磁盘的物理资源。每个节点上安装的 Spark 的版本为 2.1.1, Hadoop 版本为 2.6.4。GuardSpark 也是在 Spark 2.1.1 基础上进行二次开发。

6.2.2 测试工具与数据集

性能测试使用的是 databricks 公司推荐的 Spark-sql-perf[33]测试工具集, Spark-sql-perf 是针对 Spark SQL 的性能测试框架。Spark-sql-perf 选用的正是 TPC-DS 测试基。由于 GuardSpark 需要根据查询对象定制访问控制规则, 需要首先选定某个查询作为实验目标。参考加州大学伯克利分校 AMPLAB 的大数据测试基准[34]的选择标准, 本实验仍选择 tpcds 包中的 q7-derived 测试用例作为标准查询。该查询涉及到了 scan, join, group 等操作, 相对简单且具有代表性。

为了尽可能消除偶然性误差, 采用多次测试取均值的方法作为实验结果进行比较, 实验中每个查询重复的次数为 20 次, 取其均值作为实验结果。由于实验原型中并不包含策略的解析单元, 故本实验只对安全规则优化部分带来的性能开销进行评价。

表 2 基于 TPC-DS 测试基准的 AC 有效性验证

<table>
<tr><th>查询</th><th>访问控制规则</th><th>是否与预期结果一致</th></tr>
<tr><td rowspan="2">q7-simpleScan</td><td>Case 1: store_sales.ss_sold_date_sk > 2450915 | read;</td><td>是</td></tr>
<tr><td>Case 2: store_sales.ss_sold_date_sk < 2450950 | read; store_sales.ss_item.sk | indirect;</td><td>是</td></tr>
<tr><td rowspan="2">q7-twoMapJoins</td><td>Case 1: customer_demographics.cd_demo_sk | indirect;</td><td>是</td></tr>
<tr><td>Case 2: item.i_item_sk | read;</td><td>是</td></tr>
<tr><td rowspan="2">q7-fourMapJoins</td><td>Case 1: store_sales.ss_sales_price | indirect;</td><td>是</td></tr>
<tr><td>Case 2: customer_demographics. cd_education_status | deny;</td><td>是</td></tr>
<tr><td rowspan="2">q7-noOrderBy</td><td>Case 1: store_sales.ss_promo_sk | indirect;</td><td>是</td></tr>
<tr><td>Case 2: store_sales.ss_sold_date_sk < 2450950 | read;</td><td>是</td></tr>
<tr><td rowspan="2">q7</td><td>Case 1: item.i_item_id | indirect;</td><td>是</td></tr>
<tr><td>Case 2: store_sales. ss_sales_price | indirect;</td><td>是</td></tr>
<tr><td rowspan="2">store_sales-selfjoin-1</td><td>Case 1: store_sales.ss_sold_date_sk > 2450950 | read;</td><td>是</td></tr>
<tr><td>Case 2: store_sales. ss_list_price | deny;</td><td>是</td></tr>
<tr><td rowspan="2">store_sales-selfjoin-2</td><td>Case 1: store_sales.ss_sold_date_sk < 2451000 | read;</td><td>是</td></tr>
<tr><td>Case 2: store_slaes.ss_item_sk | indirect;</td><td>是</td></tr>
</table>

6.2.3 GuardSpark vs. Spark

在访问控制策略的作用下, GuardSpark 实际允许用户访问的数据集是原始的数据集的子集, 由于 Spark 采用的 lazy 执行的模式, 数据的读入操作只在 action 触发执行流的时刻进行, 故数据的可访问范围实际上决定了计算量的多少。当某个访问控制策略允许用户可访问数据集缩小很大时, 对于同一个查询, 查询的执行时间会明显缩短。这样就无法分析安全优化部分对整个系统性能的影响。为排除数据集大小引入的干扰, 实现单一变量条件下的对比分析, 实验中将访问控制策略设计成用户仍能访问整个数据集, 意味着在指定的约束条件下整个数据集仍会参与计算。事先通过对数据集查询获取数据的最小值和最大值, 然后定义用户的可访问范围包含区间[min, max]。

Spark SQL 作为结构化分析工具, 结构信息对于数据的分析过程显得尤为重要。parquet 列式存储文件是自描述性的且保留了 schema 信息, 允许直接访问单独某列而避免了读入整个记录项, 能够加速查

询和减少磁盘空间的占用。在本实验中, 数据集的生成过程中设置格式为 parquet。

性能评价实验中 GuardSpark 和 Spark 同时部署在集群中, GuardSpark 上定制的访问控制策略限定访问主体对整个数据集具有读权限, 如此设计的目的就是要观察安全优化给运行时执行引入的系统开销。

图 5 展示的是 4 种不同查询下, GuardSpark 与 Spark 的执行时间的比较, 4 个查询依次为 simpleScan, twoMapJoins, noOrderBy 以及 selfJoin-1。实验测试了不同数据集大小的情况, 数据集的大小依次为 2G, 4G, 8G, 用 a, b, c 标识。

从图 5 可见, 两者的执行时间基本一致, GuardSpark 的访问控制机制对 Spark 的性能影响微乎其微。

由于本设计中安全控制作用在 Catalyst 的优化阶段, 主要的影响的是优化时间, 即由解析的逻辑计划得到优化的逻辑计划的时间。现将 Spark 与 GuardSpark 中的优化时间进行比较, 可以看出在增加了安全控制之后优化时间会出现一定程度的增长, 见图 6。

虽然优化时间会出现增加, 但是由于优化时长在整个执行过程中占得比例非常小, 而且优化时间是相对固定的并不随数据集规模的增长而线性的增加, 所以对整个查询的执行而言, 引入访问控制后系统的性能并不会有明显的改变, 证明 GuardSpark 访问控制的机制具有很好的性能和可扩展性。

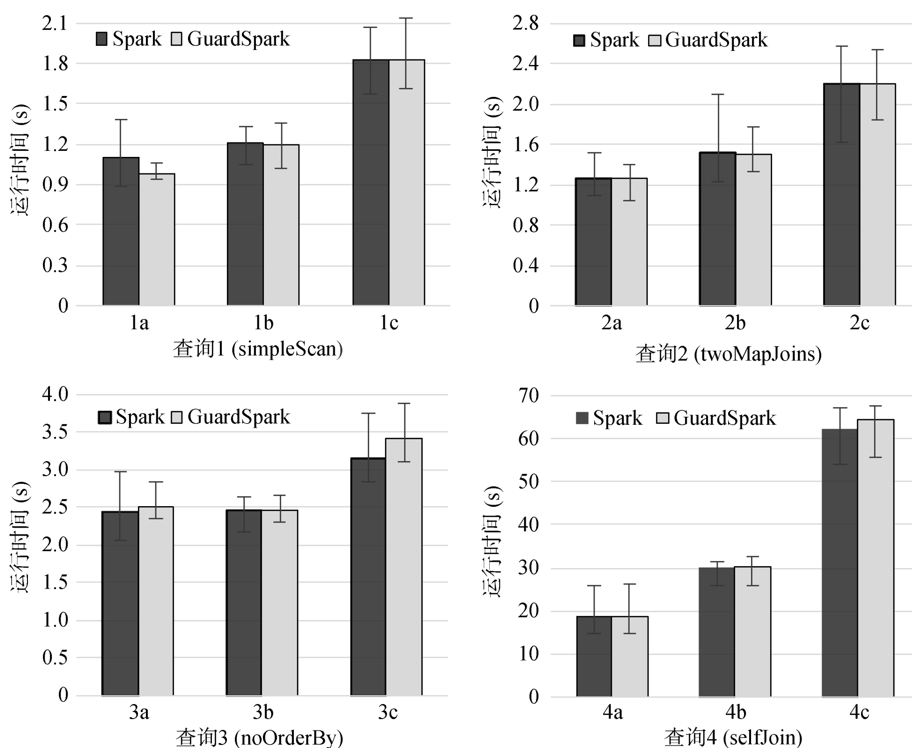
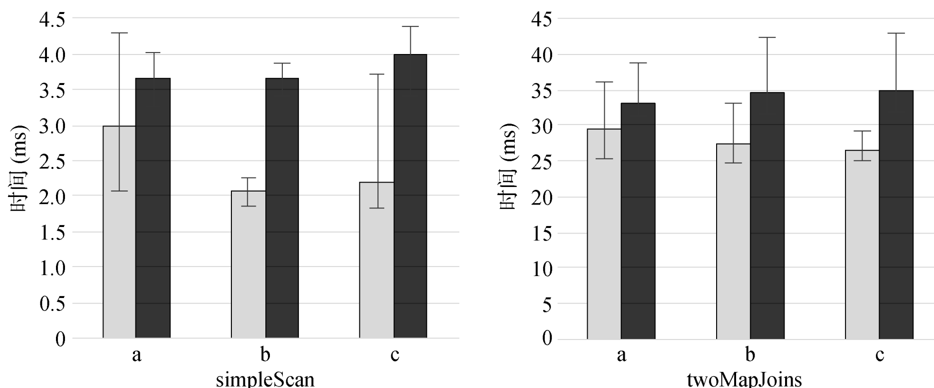


图 5 基于 TPCDS 查询的性能对比



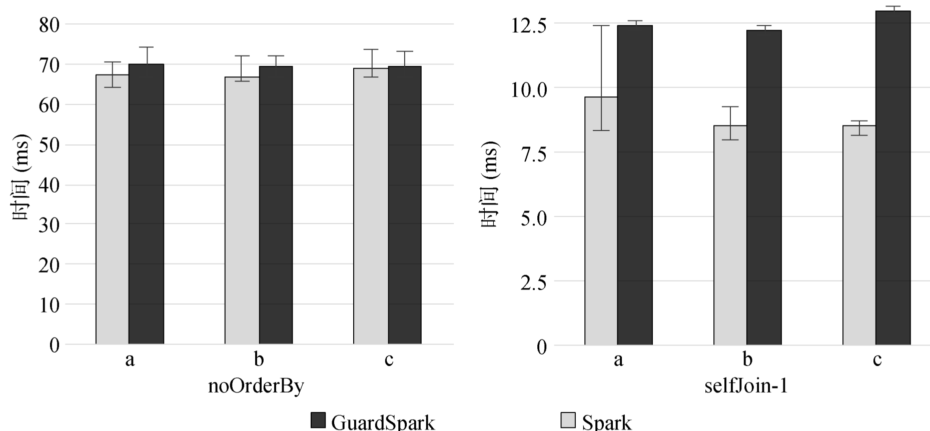


图6 Spark与GuardSpark优化时间对比

7 结论

本文提出了一种面向Spark的统一、集中式的访问控制机制，将统一的声明式编程接口与基于安全规则的执行树状图变换相结合，实现了访问控制与数据源和用户程序代码的解耦合，能集中有效地对各种数据源和用户数据访问施加控制。

文章对所提机制在Spark系统中进行了原型实现和集群部署，通过实验对系统进行了功能验证和性能评价，实验结果表明GuardSpark正确的实施细粒度的复杂访问控制，且引入的系统运行时开销可忽略，同时不受数据规模的影响，具有可扩展性。

致谢 感谢审稿人对本文提出的宝贵修改意见。感谢Spark以及Spark SQL开源项目的贡献者。特别感谢DataBricks公司的连城工程师在Spark系统开发实现上对本文的帮助。

参考文献

[1] “The Zettabyte Era - Trends and Analysis,” Cisco, http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html, 2016.

[2] Apache Spark, http://spark.apache.org/.

[3] “Apache Spark rises to become most active open source project in big data,” TechRepublic, http://www.techrepublic.com/article/apache-spark-rises-to-become-most-active-open-source-project-in-big-data/, 2016.

[4] “Spark Summit 2017”, databricks, https://spark-summit.org/, 2017.

[5] V. Mayer-Schönberger and K. Cukier, “Big data: A revolution that will transform how we live, work, and think,” Houghton Mifflin Harcourt, 2013.

[6] “Gartner Says Solving 'Big Data' Challenge Involves More Than Just Managing Volumes of Data,” Gartner, http://www.gartner.com/newsroom/id/1731916, 2011.

[7] H. Li, M. Zhang, D.G. Feng, and Z. Hui, “Research on Access Control of Big Data”, Chinese Journal of Computers, vol. 40, no. 1, pp. 72-91, 2017.
(李昊, 张敏, 冯登国, 惠榛, “大数据访问控制研究”, 计算机学报, 2017, 40(1): 72-91。)

[8] Apache Flume. https://flume.apache.org/

[9] Apache Kafka. http://kafka.apache.org/.

[10] Apache Sentry, http://sentry.apache.org/.

[11] Apache Accumulo, http://accumulo.apache.org/.

[12] M. Armbrust, R.S. Xin, C. Lian, Y. Huai, D. Liu, J.K. Bradley, X. Meng, T. Kaftan, M.J. Franklin, A. Ghodsi, and M. Zaharia, “Spark SQL: Relational Data Processing in Spark,” in Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD'15), pp. 1383-1394, 2015.

[13] D. Preuveneers and W. Joosen, “SparkXS: Efficient Access Control for Intelligent and Large-Scale Streaming Data Applications,” in International Conference on Intelligent Environments (IE'15), pp. 96-103, 2015.

[14] H. Ulusoy, P. Colombo, E. Ferrari, M. Kantarcioglu, and E. Pattuk, “GuardMR: Fine-grained security policy enforcement for MapReduce systems,” in Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (ASIA CCS'15), pp. 285-296, 2015.

[15] V.C. Hu, T. Grance, D.F. Ferraiolo, and D.R. Kuhn, “An Access Control scheme for Big Data processing,” in 10th IEEE International Conference on Collaborative Computing Networking, Applications and Worksharing (CollaborateCom'14), pp. 1-7, 2014.

[16] I. Roy, S.T. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and Privacy for MapReduce. in Proceedings of the 7th USENIX conference on Networked systems design and implementation (NSDI'10), pp. 20-20, 2010.

[17] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy, “Extending query rewriting techniques for fine-grained access control,” in Proceedings of the 2004 ACM SIGMOD international conference on Management of data (SIGMOD'04), pp. 551-562, 2004.

[18] B. Carminati, E. Ferrari, and K.L. Tan, “Enforcing access control over data streams,” in Proceedings of the 12th ACM symposium on Access control models and technologies (SACMAT'07), pp. 21-30, 2007.

[19] B. Carminati, E. Ferrari, J. Cao, and K.L. Tan, “A framework to

enforce access control over data streams," ACM Transaction on Information System Security, vol. 13, no. 3, pp. 1-31, 2010.

[20] M. Zaharia, R.S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M.J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache Spark: a unified engine for big data processing," Commun. ACM, vol. 59, no. 11, pp. 56-65, Nov. 2016.

[21] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI'12), pp. 15-28, 2012.

[22] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in Conference on Symposium on Opearting Systems Design & Implementation (OSDI'04), pp. 137-150, 2004.

[23] Apache Storm project, http://storm.apache.org/.

[24] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems, pp. 59-72, 2007.

[25] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J.M. Hellerstein, "Distributed GraphLab: a framework for machine learning and data mining in the cloud," Proceedings of the VLDB Endowment, vlo. 5, no. 8, pp. 716-727, 2012.

[26] R.S. Xin, J. Rosen, M. Zaharia, M.J. Franklin, S. Shenker, and I. Stoica, "Shark: SQL and rich analytics at scale," in ACM SIGMOD International Conference on Management of Data (SIGMOD'12), pp. 13-24, 2012.

[27] The Scala Programming Language, https://www.scala-lang.org/.

[28] P. Colombo and E. Ferrari, "Efficient Enforcement of Action-Aware Purpose-Based Access Control within Relational Database Management Systems," IEEE Transactions on Knowledge and Data Engineering, vol. 27, no. 8, pp. 2134-2147, 2015.

[29] E. C. Lupu and M. Sloman, "Conflicts in policy-based distributed systems management," IEEE Transactions on Software Engineering, vol. 25, no. 6, pp. 852-869, 1999.

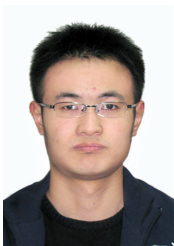
[30] M. Y. Becker and P. Sewell, "Cassandra: distributed access control policies with tunable expressiveness," in Proceedings. Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, pp. 159-168, 2004.

[31] T. Jaeger, X. Zhang, and A. Edwards, Policy management using access control spaces. ACM Trans. Inf. Syst. Secur., 2003. 6(3): p. 327-364.

[32] "TPC BENCHMARK DS Standard Specification Version 2.3.0," Transaction Processing Performance Council (TPC), http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.3.0.pdf.

[33] "Spark SQL Performance Tests," Databricks, https://github.com/databricks/spark-sql-perf.

[34] "Big Data Benchmark," University of California, Berkeley, AMPLAB, https://amplab.cs.berkeley.edu/benchmark/.



宁方潇 于 2012 年在中国科学院微电子研究所集成电路工程专业获得硕士学位。现在中国科学院信息工程研究所计算机系统结构专业攻读博士。研究领域为计算机系统安全。研究兴趣包括: 访问控制, 数据完整性保护等。Email: ningfangxiao@iie.ac.cn

史岗 于 2004 年在中国科学院计算技术研究所计算机体系结构专业获得博士学位。现任中国科学院信息工程研究所第五研究室正高级工程师。研究领域为计算机系统安全。研究兴趣包括计算机架构, 计算机芯片安全等。Email: shigang@iie.ac.cn

文雨 于 2011 年在中国科学院计算技术研究所计算机体系结构专业获得博士学位。现任中国科学院信息工程研究所第五研究室高级工程师。研究领域为大数据与系统安全。研究兴趣包括: 威胁检测、数据挖掘等。Email: wenyu@iie.ac.cn