基于数据通道的非结构化数据多存储系统

蒋静远，鲁伟明*，王耀光，楼仁杰

（浙江大学计算机科学与技术学院 杭州 310027）

摘 要： 面对大数据大容量、高速率和多样性的特点，传统关系型数据库不再能满足处理海量非结构化数据的需求，越来越多的数据中心开始使用以 NoSQL 数据库为主，多个数据存储组件相互协同的多存储系统。为了发挥多存储系统中各个存储组件的功能，数据需要在多个存储组件之间进行同步，以 ETL 和客户端多路写为代表的传统数据同步方式不能满足以 NoSQL 为存储核心的多存储系统。本文提出的非结构化数据多存储系统以 HBase 为主数据库，使用 Coprocessor 索引信息记录和 WAL 日志文件解析两种方式捕获 HBase 中的数据变更，并将以此方法实现的变更捕获组件接入 DataBus 实现数据通道，构建以数据库变更为同步手段的非结构化数据多存储系统。实验结果表明，该系统具有较高的数据变更捕获性能和良好的可扩展能力，为非结构化数据多存储架构提供了一种可行的解决方案。

关键词： 非结构化数据；多存储；数据变更捕获；HBase；DataBus

An Unstructured Data MultiStore System Based On Data Channel

JIANG Jing-Yuan, LU Wei-Ming*, WANG Yao-Guang，LOU Ren-Jie

(College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China)

Abstract: Because of big data's large volume, high velocity and variety, traditional RDBMS cannot meet the needs of processing massive unstructured data any longer. More and more data centers begin to use a multistore system, which has one central NoSQL database and several secondary storage components. In order to make every parts of the system work efficiently, data needs to be synchronized between multiple storage components. ETL and application-driven dual writes are two common kinds of synchronization methods, but they cannot satisfy NoSQL based multistore systems. In this paper we propose an unstructured data multistore system which uses HBase as the primary store and benefits from a data channel. The data channel consists of two data change capture modules and Databus. We use coprocessor index recording and WAL log file analysis to capture data change events, and adapt it to Databus. The experimental results show that the system has high data change capture performance and good scalability, providing a feasible solution of unstructured data multistore architecture.

Key words: Unstructured Data; multistore; Data Change Capture; HBase; DataBus

1 引言

面对大数据大容量（volume）、高速率（velocity）和多样性（variety）的特点[8]，传统的关系型数据库管理系统（RDBMS）在可扩展性和性能等方面遭遇了巨大的挑战[9]。自 2006 年谷歌提出 BigTable[1]以来，一大批开源的 NoSQL 数据存储系统涌现出来[10]，旨在解决海量异构数据的存储可扩展性问题，同时提供良好的数据访问能力。越来越多的企业开始采用 NoSQL 作为其主要存储系统。在复杂的应用环境中，单一的 NoSQL 数据存储往

基金项目：国家科技重大专项 “核高基” 项目（2010ZX01042-002-003-001）；国家自然科学基金（61103099）；中央高校基本科研业务费专项（2014QNA5008）；中国工程科技知识中心项目(CKCEST)

往不能满足多样化的数据管理需求[4]，因而在典型的大数据管理系统中，除了承载用户读写任务的主存储系统之外，往往还包含有分布式文件系统、分布式数据库、关系数据库、图数据库、缓存设备、索引存储等其它数据存储组件。以这种方式将关系数据存储与大数据存储进行融合，提供多存储系统的方式得到了广泛的使用，如 Microsoft 的 PolyBase[2]，SAP HANA[3]等。欧盟第七框架 NewsReader 项目[11]就是结合 HBase 和图数据库 Virtuoso 来实现海量知识的存储和管理。在一个多存储系统中，应用不仅能够获得每个数据存储设施自有的数据访问能力，同时还可以使用不同的访问方式来操作数据，从而为用户不同的数据需求提供强有力的支撑。因此，多存储系统必然将成为未来大数据管理的自然选择[12-14]。

多存储系统中为了发挥不同存储设施的访问特点和性能，数据需要在多种设施上按照不同的组织方式进行保存。其中，一个关键问题是如何进行多种设施之间的数据同步。目前主要的解决方法有两种：ETL[15]和客户端多路写（Application-Driven Dual Writes）。ETL 是一种在数据进入一种数据存储设施（数据源）之后，人工或自动地将数据进行异步地抽取、转换和加载到目标数据仓库的过程。客户端多路写是一种同步数据传输过程，在客户端将数据写入一种存储设施的同时，向需要冗余存储该数据的其它存储组件，按照对应的组织方式写入所需的部分或全部数据。然而，ETL 方式大大延缓了数据在不同存储系统之间更新可见性，而多路写的方式尽管实现了不同存储系统的同步性，但一致性却难以保证，同时需要客户端的参与。

为了解决不同存储系统的延迟更新和一致性问题，Linkedin 提出了一种数据变化捕捉系统—DataBus[5]。DataBus 作为一种主存储系统和其他存储系统之间的中间组件，从主数据库获取数据的变更信息，以快照加增量的方式保存在中间组件中，需要获取变更数据的外围存储系统以适当的频率主动地向中间组件获取数据，并在需要时获取数据库快照以支持快速加载。但是，目前 DataBus 支持的主存储系统仅包括 Oracle 和 MySQL 这样的传统关系型数据库。这种限制的产生一方面源于 Linkedin 自身业务的需求，另一方面源于当前数据变化捕获组件的实现需要借助传统关系数据库的触发器机制和相关的协同方式。这些限制使其不适用于 NoSQL 作为主存储系统的应用场景，阻碍了它在海量数据管理系统中的广泛使用。

本文主要解决在以 NoSQL 数据库为主存并集成有其他存储组件的多存储系统中，如何及时有效地捕获主存储系统数据变更的问题，并在此基础上提出一种以 NoSQL 数据库为存储核心的非结构化数据多存储系统。旨在设计 NoSQL 数据库变更捕获的方式和方法，重点关注如何高效地捕获数据变更，并且减少变更捕获过程对主存储系统的开销。本文以一种广泛使用的 NoSQL 存储系统 HBase 为例，设计并实现两种不同的获取主存储系统数据变更信息的策略，同时使用 DataBus 来保存和传递获取到的信息，并检验这两种策略的差异和性能。

本文第 2 节介绍 HBase、数据同步、DataBus 等相关技术研究背景；第 3 节介绍以 HBase 为主存储系统的非结构化数据多存储系统；第 4 节描述 HBase 事件的定义与两种 HBase 数据变更信息的获取方法；第 5 节通过实验对数据变更获取组件进行性能测试；第 6 节对工作进行了总结与展望。

2 研究背景

2.1 HBase

HBase[16]是根据 google 的 BigTable 设计发起的一个开源项目，是一种典型的 NoSQL 分布式数据库模型，其本质上是一个具有分布式的持久化多维排序稀疏 Map 结构。

每个 HBase 表由多个列族组成，列族（Column Family）的名称和配置必须在创建表时预先定义，而列族内的列（Qualifier）则可以在向表内插入数据时动态修改，所以它是一种面向列族存储的分布式数据库，常使用 HDFS 为其提供底层持久化存储。

HBase 以 Key-Value 键值对的形式存储数据，其中 Key 以行键值、列族名、列名等信息拼接而成，Value 为该 Key 所对应的值。存储在 HBase 中的数据最终按照 key 的顺序字典序递增排列，故所有指明行键值的读操作可以利用该有序性快速地定位数据所在位置。在 HBase 上包括删除、修改和插入的更新操作在反应到底层存储之前均以 Key-Value 的形式保存。Key-Value 数据在成功写入系统之前，均会保存在 HDFS 上一个特殊文件里，该文件叫做 WAL(write-ahead-log)文件。其目的是在 HBase 发生故障终止服务恢复后，可以根据该文件恢复在故障发生时因仅保存在内存中而丢失的数据。

HBase 在 HDFS 分布式文件系统的基础上提供

随机读写数据的能力，是一种在海量非结构化数据管理系统中常用的 NoSQL 数据库。

2.2 数据同步方式

在包含一个主数据库和多个外围数据组件的数据管理系统中，其它外围组件必然需要与主数据库中的数据进行同步。不同的数据同步方式将造成外围组件与主数据库间不同程度的耦合。

外围组件直接向主数据库读取数据是耦合程度最强的数据同步方式。外围设施的数量和数据获取的频率不仅会大大影响主数据库对外服务的能力；而且，由于其耦合程度过大，主数据库的任何变动都会引起外围组件相应的修改，这极大地提高了维护同步方式所带来的成本。

客户端多路写策略是最常见的数据同步方式。在这种同步方式下，用户层应用程序在根据其自身需要将数据写入主数据库的同时也必须同步地将全部或者部分数据写入其它外围数据组件之中。其实现方式简单，对数据管理系统的性能要求也较低。但是，这种方式要求应用层程序员也参与到数据同步的工作中来，并且其在多路写的过程中引入新的一致性问题，常需要额外的协调手段来协调（如 Paxos[17]和二阶段提交协议[18]）。

使用 ETL（Extract-Transform-Load）工具是数据仓库应用中常见的数据同步方式[7]，其功能是将数据异步地从数据源经过抽取、转换、加载到数据仓库。利用 ETL 工具周期性地按照外围组件的数据组织要求批量地将数据加载到外围组件当中，而无需用户手动编写相关功能代码。但其增加了数据更新在外围组件中的可见延迟，实时性不强。

综上所述，直接耦合、客户端多路写与 ETL 工具或是会引入新的一致性问题，或是会造成较大的更新可见延迟，均不能较好地满足以 NoSQL 数据库为存储核心多存储系统的数据同步需求。

2.3 DataBus

DataBus 是 Linkedin 自身数据处理流水线的重要组成部分[5-6]，用以实现关系数据库到其它组件的同步，功能架构如下图：

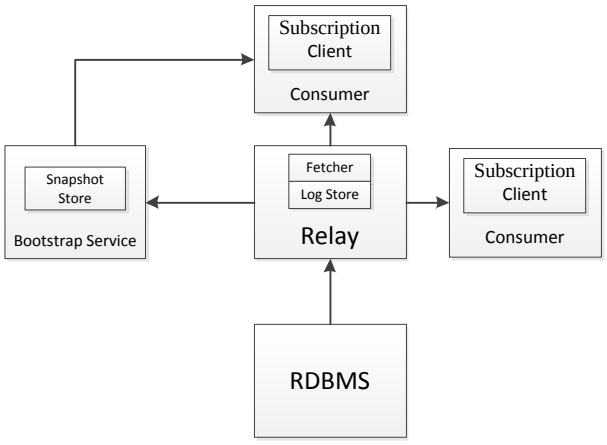

图 1 DataBus 主要架构图

Fetcher 可以从主数据库获取数据变更，并将其缓存在 Relay（中继器）的 Log Store 中。Bootstrap Service（引导服务器）用来持久化存储不同时间的数据库快照。其利用快照加增量使客户端可以快速获取任意时刻的数据库状态。客户端程序利用 Subscription Client 从 DataBus 中获取其感兴趣的数据变更信息，并将其应用于自身的业务逻辑。

在 LinkedIn 中，DataBus 开源实现使用 Oracle 触发器作为其记录额外信息的手段，Fetcher 通过额外信息表查询结果确定所要查询的数据位置，再通过 SQL 语句查询实际数据信息，从而同步 Oracle 中的数据。或者通过 MySQL 底层数据存储引擎原生提供的数据迁移工具，完成对 MySQL 数据变化信息抓取的工作。

通过 DataBus 作为中间组件完成主数据库与外围组件的数据同步有效降低了外围组件与主数据库的耦合程度。同时具备高吞吐、可扩展等特点。但是，其开源版本仅被设计用来提取 Oracle、MySQL 等传统关系型数据库中的数据变化信息，逻辑实现上大量使用诸如触发器、事务、SQL 查询等关系型数据库的概念和工具。要扩展其应用场景，使其能应用于以 HBase 为主数据库的非结构化数据管理系统，获取并传播 HBase 中的数据变更信息，包含如下几个重点和难点工作：

1.实现 HBase 数据变更捕获组件，设计 HBase 变更事件的模型定义及其记录方式。HBase 数据变更捕获组件要能够实时地从 HBase 数据表中获取数据变更信息。并且，捕获过程不应对 HBase 造成较大的性能开销，从而影响其对外提供服务的能力。

2.为获取到的 HBase 变更事件实现序列化和反序列化方法。开源版本中的 DataBus 利用 Avro[19]为其数据库各字段数据进行序列化和反序列化，并

通过数据库表结构的查询自动产生 Avro Schema 文件。在其应用于 HBase 以后，要根据 HBase 的变更事件模型定义为其定制序列化和反序列化方法。

3.将数据变更捕获组件实现为一个 DataBus Fetcher 实例，其应能被 Relay 正确调用，实现数据通道。获取的变更事件经序列化后存于 log store 之中，供 DataBus 其它组件或客户端程序查询。

3 总体架构

本文所提出的非结构化数据多存储系统以 NoSQL 数据库为其存储核心，承载主要的用户读写请求。同时，为了满足多样化的数据查询管理需求，系统中还部署有图数据库、索引服务器、关系型数据库等其它外围服务组件。用户通过统一的服务接口直接向主 NoSQL 数据库写入数据，数据通道抓取主 NoSQL 数据库中的数据变更信息，其它外围组件通过数据通道获取用户的数据变更信息，并根据变更信息更新其自身数据，完成数据同步过程。一旦数据同步完成，系统就可以提供如索引查询、图查询、关系查询、全文查询等多样化的数据服务。其架构如图 2 所示：

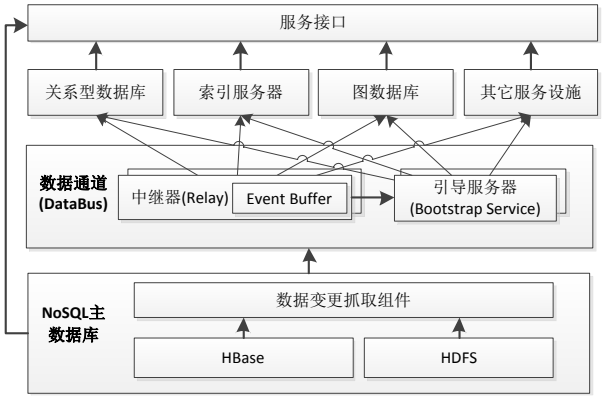

图 2 非结构化数据多存储系统架构

外围组件通过数据通道与 NoSQL 数据库进行数据同步，这样的设计降低了外围组件与主 NoSQL 数据库之间的耦合程度，外围组件不必知晓主 NoSQL 数据库的类型及其工作方式。同时，数据通道提高了多存储框架的可扩展性，外围服务组件可以根据需要动态地在数据数据通道上挂载与卸载。

在这样的多存储系统中，数据通道是其能否高效运行的关键。一个符合要求的数据通道在提供其扩展性和解耦合的同时不应对主 NoSQL 数据库产生较大的性能损耗，其挂载的组件数量不应明显地影响主数据库的读写性能。

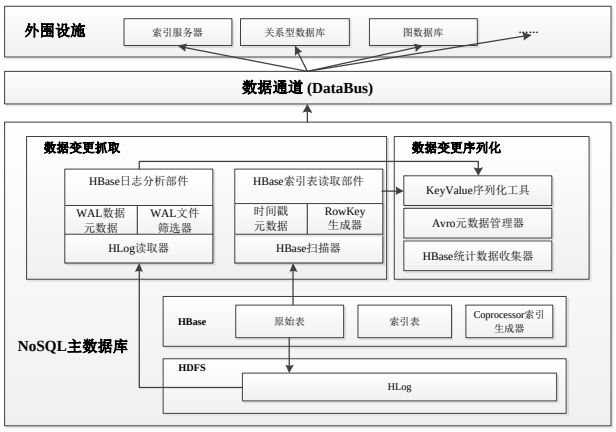

图 3 数据捕获系统

在本文提出的多存储系统中，以 HBase 作为主 NoSQL 数据库为用户提供数据的主要读写服务，HBase 数据变更抓取组件和经过扩展的 DataBus 共同组成数据通道。

数据通道具体架构如图 3。架构中包括两种类型的数据变更抓取组件。其一利用 HBase Coprocessor 在数据插入 HBase 时同步地产生索引数据，抓取组件依据待捕获时间区间和索引数据拼接出查询原始数据所需的 Row-Key 等位置信息，最后利用位置信息对原表进行扫描获取变更数据。其二直接利用保存于 HDFS 上的 WAL 日志文件，筛选含有被监测 HBase 表数据的 WAL 日志文件，从中读取以二进制形式保存的 Key-Value 数据，获取待捕获时间区间内的数据变更信息。捕获到的数据变更信息由 Key-Value 序列化工具按照统一的变更事件模型序列化为多个数据变更事件，并以二进制数据流的形式保存于 DataBus 的 Log Store 中，由 DataBus Relay 实例负责变更事件的复制与传播。挂载在数据通道上的外围数据组件，通过向 DataBus Relay 发起变更查询请求的方式查询其所需时间区间内 HBase 内发生的数据变更事件，并在自身服务数据集上应用查询到的变更事件完成数据同步过程。

4 数据变更抓取与序列化

为了实现数据变更捕获组件，获取 HBase 中数据变更信息，使其能够与 DataBus 组合，充当以 NoSQL 数据库为主存储的多存储系统数据通道，我们在本节定义 HBase 变更事件，提出 HBase 中数据变更信息的提取方法，并为其设计序列化手段。

4.1 HBase事件变更定义

把 HBase 中数据发生变化的时间、位置和值定义为一个 HBase 变更事件。

定义 1. HBase 变更事件

Event = （TableName，TimeStamp，RowKey，ColumnFamily，Column，Value，Type）

其中 TableName 为发生变更事件的表表名；TimeStamp 为事件发生的时间，数值上与 HBase 中该事件对应 Key-Value 键值对时间戳相同；RowKey 为变更事件所在行的行键值；ColumnFamily 与 Column 表示变更事件所在的列族和列；Value 为值，若该变更为数据插入或修改则值对应插入或修改的实际值，若为删除操作其值为空；Type 为数据事件，包含 Put、Delete、DeleteFamily 等，与数据插入、删除等操作一一对应。HBase 事件是记录 HBase 数据变更的最小单位。这些 HBase 变更事件应以其发生的先后顺序组织，这样外围组件才能按照事件发生的时间顺序依次读取变更信息，使其不会因为变更应用的先后顺序颠倒而产生混乱。

易知，每一个存储于 HBase 的 Key-Value 均与一个 HBase 变更事件一一对应，其中除值以外的其它部分被连接起来组成 Key 部分，而 Value 则直接由值来表示。为了从 HBase 中读取 HBase 变更，并以时间顺序组织，我们提出了两种可行的解决方案：一，利用 HBase Coprocessor 仿造关系型数据库中触发器的功能和作用，在用户使用 API 对 HBase 中数据进行操作的同时，记录额外信息，使事件抓取组件可以利用这些额外信息在原表中获取相关的变更数据；二，利用 HBase 在完成用户操作之前均会将相关数据写入 WAL 日志文件作为备份，变更捕获组件可以直接访问 HDFS 中的 WAL 文件，从中筛选出所需的 HBase 事件。

4.2 Coprocessor变更获取

数据在插入 HBase 的过程中，HBase 自身功能逻辑保证所有表内数据按照其行键值的字典序有序排列，所以在通过行键值对数据进行读取或者扫描时，HBase 因这种有序性而具有较高的性能。而普通数据表的行键值往往与其插入时间无关，所以，在普通表中获取特定时间区间内的变更数据，只能依靠全表扫描并根据其时间戳进行筛选。为了避免全表扫描带来的大量性能损耗，我们利用其行键值有序排列的特性，在 HBase 中新建一张与原表关联的索引数据表。

表 1. 与原表关联的索引表

TableName_SCN = （index）

其中 TableName 为与其关联的原始表表名。这样，数据变更捕获组件能从原始表的表名中直接推断出其索引表的表名。索引表仅有一个列族 index，该列族中的各列数据用于记录每个原表变更事件的相关属性。因为索引表的各列常在一次读取中全部读出，所以将其安排在同一个列族内是合理的。

定义每个原始数据表变更事件在索引表中的索引形式。

定义 2. 原始数据表变更事件表示形式

(TimeStamp ， index:(ColumnFamily_R ， ColumnFamily_C),(RowKey,Column))

每一条索引数据的行键值为其对应的变更事件发生时间，仅有的一个列族下记录两列数据：它们的列名均与原表变更数据位置的列族名有关，但使用不同的后缀标识不同的数据。后缀为 R 的列其值为该次事件所对应的原表数据行键值；另一列以 C 为列族名的后缀其值为该条数据在原表的列列名。这样，在需要对特定时间区间内发生的变更事件进行获取时，抓取组件仅需使用 Scan 操作扫描该索引表，由于其行键值已经是原表中变更事件的时间戳，故其效率相比在原表中直接进行查找得到显著的提高。在扫描得到的索引表数据中，每一行就包括了该事件对应的原表行键值、列族名、列族，利用这些数据就可以快速地定位到该条数据在原表中的位置，方便地利用 Get 操作将其读出。

为了构建索引表数据，在插入数据时多存储系统需要在原数据表和索引表中同时插入数据。若将这一工作交由客户端程序处理，不仅一致性很难得到保证，同时也提高了客户端程序与系统的耦合程度。

为了解决这一问题，我们使用 Coprocessor。HBase Coprocessor 给予用户在服务端运行自订代码的能力。它可以以每一个 Region 为单位，在恰当的时机触发运行用户代码，这一特性与关系型数据库中的触发器相似。通过定制 Coprocessor 基类方法 PostPut，HBase Coprocessor 框架会在每一次 Put 操作完成之后同步调用该方法，完成索引数据写入工作。在该方法中，代码逻辑从当前执行的 Put 实例中提取出数据变更所在的行键值、列族、列等信息，并使用运行框架提供的运行时环境获取对索引表的连接，将相关信息同步地写入到索引表当中。

由于使用了 Coprocessor 完成从插入数据中提

取数据到建立索引信息的工作，客户端程序将不再参与这一工作。这不仅在一定程度上解决了客户端多路写入引入的一致性问题，也降低了客户端程序的编写难度，它将按照往常的方式向 HBase 插入数据。

依据索引表和原数据表中的数据，我们实现了变更抓取组件并将其接入 DataBus 适配器，完成从 HBase 读取变更数据的工作。其主要操作包括两部分：

第一部分，根据给定的待查询变更时间起始值和当前时间获得待查询时间区间。利用这个区间在索引表中进行 Scan 操作，获取行键值落在这一时间区间内的索引数据。

第二部分，利用第一部分中获取的索引数据，拼装成源数据表中包含行键值、列族名、列名的 Get 操作列表。使用该 Get 操作列表对原始数据表进行批量的 Get 操作，获得这部分数据的 Key-Value 值。

DataBus Fetcher 进程周期性地调用数据变更捕获组件，完成 HBase 变更事件的抓取，经序列化后放入 log store 供客户端读取查询。

4.3 WAL日志文件解析

Coprocessor 变更抓取组件在一定程度上满足了数据同步的需求，但其自身仍存在一定的缺陷。首先，其不能很好地支持删除事件的记录，由于在利用 Coprocessor 完成变更数据抓取的过程中，变更数据的抓取需要由相应的 Get 操作从原始数据表中读出，这限定了可被获取的变更数据必须是新插入表中的数据或者经过修改仍然存储在 HBase 表中的数据。在这样的前提下，若发生的事件恰好为删除事件，Coprocessor 变更捕获组件将无法捕捉这一事件。其次，使用 Coprocessor 对 HBase 的 Put 操作的性能产生了一定的影响，一定程度上降低了其对外服务的能力。由于 Coprocessor 程序在客户端向数据表插入数据的同时会同步地执行额外的 Put 操作，导致 put 操作所需的耗时有所增加，这一问题在第 6 节实验中将得到验证。再次，若 HBase 中的数据变化速度过快，导致在变更抓取组件根据索引信息向原表进行查询时，原值已经被后序操作修改或者删除，这时变更抓取组件就无法获取正确的数据。以上问题在一定程度上制约了多存储系统的一般可用性。

为了解决 Coprocessor 变更抓取中的问题，本文提出的多存储框架还使用了一种 WAL 日志文件解析方案。利用每次 Key-Value 键值对进入 HBase 之前均会由 HBase 写入 WAL 文件作为备份这一特性。我们在数据变更抓取组件中增加一个日志分析部件，通过分析存储在 HDFS 上的 WAL 文件，查找我们所需的 Key-Value 键值对，从中获取 HBase 变更数据。

HBase WAL 文件存于 HBase 位于 HDFS 上的数据根目录，按照其所属 Region Sever 的不同，存储于不同的子目录下。写入 HBase 的 Key-Value 数据被各个 Region Server 实例接收后，追加在各个 WAL 文件末尾。同时 logroll 进程定期地检查这些 WAL 文件，一旦确认某一个 WAL 文件中的所有 Key-Value 信息都已经持久化到 HStoreFile 中时，logroll 进程将该 WAL 文件从其所属目录下移除。

WAL 文件内按时间顺序保存了写入同一个 Region Server 的所有数据，每条数据由 HLogKey 和 WALEdit 两部分组成。HLogKey 中保存了其所对应的数据表名和 Region 名称，WALEdit 保存了对应的 Key-Value。通过对两部分数据的读取，变更捕获组件即能获取事件所对应的 Key-Value 信息。

同 Coprocessor 解决方案一样，我们实现了另一个数据变更抓取组件，使其能够通过解析 WAL 文件获取 HBase 变更数据。其主要操作包括：周期性地遍历每一个 Region Server 所对应的日志文件夹，读取其中 WAL 文件中的 Key-Value 键值对，分析读取到的 Key-Value 键值对，捕获落在待查询时间区间内的数据变更事件。由于被同一个 RegionServer 所服务的多个数据表的数据均保存在同一个 WAL 文件当中，所以在读取到 Key-Value 数据之后我们需要进行额外的筛选，不仅需要判断该条 Key-Value 数据是否落在我们所感兴趣的时间区间之内，同时还要判断其所属的数据表、列族是否是需要捕获的数据表范围。这些额外操作在 Coprocessor 解决方案中是不存在的。

为了提高变更获取效率，在读取 WAL 文件的过程中，我们记录每个 WAL 文件已经被读取并筛选过的 Key-Value 键值对在文件中的偏移量，这样，在下次读取到该文件时就可以直接从这个偏移量继续读取。这样的操作是有理可依的。首先，Key-Value 键值对以时间递增的顺序追加到 WAL 文件末尾，所以新出现的 Key-Value 并不会出现在已经读取过的 WAL 文件片段当中；其次，WAL 文件在本质上是一个 HDFS 上的 SequenceFile，专门为

在 HDFS 上存储二进制 Key-Value 数据设计，支持随机读写，所以我们在其上的 seek 操作是有性能保证的。这样就避免了在每次读取 WAL 文件时都遍历整个 WAL 文件，而只需从已经读取过的位置开始，继续读取尚未筛选的 Key-Value 键值对即可。

利用 WAL 日志文件解析实现的数据变更抓取组件解决了采用 Coprocessor 方案时，抓取组件所存在的问题。但其也有一些不足，当需抓取变更数据的数据表仅是一个较大的 HBase 集群的一小部分时，由于各个 WAL 文件内属于该数据表的 Key-Value 键值对比重较小，对其筛选的过程将耗费大量的额外时间。所以，两种变更数据抓取组件互为补充，在需要时可相互切换。

4.4 Avro Schema

为了使读取到的 HBase 数据变更事件能够正确地被序列化和反序列化，与 DataBus 一样，我们选用 Avro 完成这一功能。为此，我们需要为其准备 Avro Schema 文件。

在此，每一个 HBase 变更事件将由一个 Key-Value 键值对来表示，其 schema 是固定不变的，仅仅由 Key 和 Value 两部分组成，且它们都由一连串的 Byte 串组成，故其对应的 Avro Schema 结构简单，且不会随着 HBase 数据表结构的变化而发生变化。所以这里不再需要一个可以根据其表结构自动生成 Schema 文件的工具，而仅需要一个通用的 Schema 文件就可供所有的 HBase 表使用。

一个可用的 Avro Schema 文件片段示例如下：

```
"fields" : [ {
    "name" : "key",
    "type" : [ "bytes", "null" ],
    "meta"                                    :
"dbFieldName=key;dbFieldPosition=0;"
  }, {
    "name" : "value",
    "type" : [ "bytes", "null" ],
    "meta"                                    :
"dbFieldName=value;dbFieldPosition=1;"
  } ]
```

5 实验结果与分析

5.1 实验环境

a) 实验集群配置如下：

HBase 集群配置：

Inter Xeon CPU 2.8GH * 2，内存 12G，操作系统 Ubuntu 12.04，共九个节点

数据中继器 Relay 所在服务器配置：

Inter i5-2400 CPU 3.10GHz，内存 4G，操作系统 Ubuntu 12.04。

客户端所在节点配置：

Inter i5-2400 CPU 3.10GHz，内存 4G，操作系统 Ubuntu 12.04。

b) 源数据表：

在 HBase 中建立三张数据表作为我们将要抓取数据变更信息和空白对照的源数据表，它们均由 CF1、CF2 两个 ColumnFamily 组成。第一张表为空白基准表，没有定义 coprocessor 操作以及额外变更抓取操作。第二张表（CP 表）配置了相应的 Coprocessor 代码，使其能够在数据插入时向索引表中插入相关的索引信息，并使用变更捕获组件和 DataBus 进行变更抓取。第三张表上定义了日志分析组件，使用带有读取 WAL 文件实现的变更捕获组件和 DataBus 抓取数据变更信息，记为 WAL 表。

c) 数据插入

实验进程按照一定的速率分别向三张 HBase 实验表中插入数据，插入的数据为每个列族一列，其值均为 512B 的字符串，故每次插入数据总大小为 1KB，对应为两个 Key-Value 键值对。

对变更抓取组件的配置使得其会同时抓取两个 ColumnFamily 上的数据变更，并通过 DataBus 将其传递给客户端程序。

5.2 Coprocessor对HBase效率的影响

图 4 显示了对三张 HBase 实验表插入数据的用时比较。

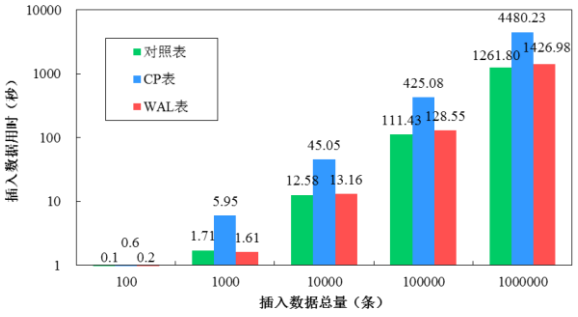

图 4 Coprocessor 对 HBase 效率的影响

图中横轴为插入到 HBase 中的数据条数，纵轴为插入特定条数的数据所需的耗时。从图中可以看到，CP 表插入相同数据所需要的时间高于基准表和 WAL 表，数值上在 3 到 4 倍之间。这是由于一

次 put 操作需要其它额外的同步操作来记录其索引信息，而造成了额外的时间消耗。WAL 表的插入用时和基准表相比耗时没有明显的增加，这也证明了采用分析日志的方式不会对原 NoSQL 数据库产生明显的性能损耗。

虽然利用 Coprocessor 实现的变更抓取组件会对 HBase 的插入性能产生一定的影响。但其抓取变更事件过程中不会因为读取无需抓取的数据而造成额外开销，且其在 HBase 版本迭代中实现方法稳定，不会因为主数据库的 HBase 版本迭代而造成较大的修改成本。当 HBase 插入数据频率较低或者 HBase 集群规模较大但所需抓取的数据表又相对较小时，使用记录索引信息的方式实现的变更抓取组件依旧是一种较合适的方法。

5.3 HBase事件抓取速率

以使用解析 WAL 文件方式获取其数据变更事件的 HBase 数据表为例。按照不同的速率向原数据表插入数据,同时利用一个充当外围组件的客户端程序不断地通过数据通道获取 HBase 变更数据，比较客户端所读取到的变更速率与发生在原始 HBase 数据表中数据插入速率的大小。以原始解决方案中使用 ETL 工具进行数据迁移作为对照，以使用 HBase Scanner API 实现的简单 ETL 工具获取同样时间区间内的数据插入结果。为保证对照的有效，插入 HBase 中的数据以其插入时间戳为其行键值，从而使 Scan 操作能尽可能快的获取落在其时间区间内的数据，尽量减少其时间损耗。结果如图 5。

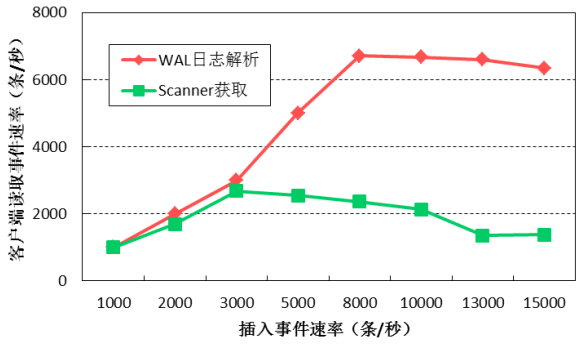

图 5 HBase 表数据变化速率对客户端的影响

其中横轴为实验程序插入数据的速率，纵轴为客户端程序通过数据通道件获取的 HBase 变更速率，单位均为条每秒。

客户端读取 HBase 事件的速率是外围设施能否实时反应原 NoSQL 数据库的关键所在。如图所示，当原 HBase 数据表中的数据插入速率大于 3000 条每秒时，用 Scanner 实现的 ETL 进程将不能提供与其一致速率的 HBase 变更数据流，也就是说从由 Scanner 实现的 ETL 工具中获取数据的客户端将随着插入速率的提高出现明显的滞后，而且该滞后程度随着更高的插入速率对 HBase 性能的损耗愈发的明显。以解析 WAL 文件实现的数据变更捕获组件获取 HBase 变更的速率直到插入速率到达 7000 条每秒时均能与原数据库保持一致，且当变化速率进一步增大时不会出现明显的速率损失。

5.4 客户端数量对数据变更抓取的影响

在该实验中，我们在系统中启动不同数量客户端来查询 HBase 的数据变更事件，我们按照特定的速度向 HBase 中插入数据，观察挂载在数据通道上不同数量的客户端获取变更事件的速率。

图 6 展示了在不同的数据插入速率下，不同数量的客户端抓取数据变更事件的速度。

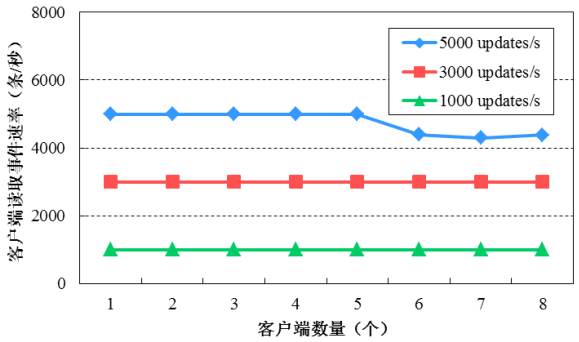

图 6 客户端数量对客户端的影响

其中横轴为系统中客户端的个数，纵轴为客户端获取 HBase 变更的速率。

由实验结果可知，当原表操作 HBase 速率保持在低于 3000 条每秒时，客户端数量在一定程度内的增加不会增加客户端中获取 HBase 变更的速率，即外围组件在一定程度内的增加不会致使其滞后主 NoSQL 数据库的程度增加。验证了以获取数据库变更信息为同步方式的多存储系统，具有较高的可扩展能力。

6 结束语

本文提出了一种基于数据通道的非结构化数据多存储系统。系统将 HBase 作为非结构化数据的主存储系统，设计并实现了两种不同感知主存储系统数据变更的策略，并采用 DataBus 数据通道来存储和传递数据变更事件，从而实现融合 NoSQL 数据库、索引系统、关系数据库和图数据库等系统的

多存储系统。通过实验证明，我们的系统具有较高的数据变更捕获性能和良好的可扩展性。

在系统中，当多客户端集群对同一个变更数据流协同处理时，需要对数据流进行相应的划分处理，从而使得集群中每个客户端都能获取其中一部分 HBase 数据变更事件。本文定义的 HBase 事件经序列化后仅仅包含了 Key 和 Value 两类信息，其中列族、列等信息都隐含在 Key 当中。这使得系统只能基于 Key 进行变更事件的划分，来提高客户端获取变更事件的性能。但尚不能实现基于列名、列族名等信息的事件划分。因此，在未来，系统将对 HBase 事件进行重新设计，使其字段更细化、粒度更细。

然而，细粒度的 HBase 数据变更事件将带来另一个问题。在基于 Key-Value 的事件设计中，其模式对于任意的表均保持不变，其序列化仅需要一个通用的 schema 文件即可完成。而细粒度的 HBase 数据变更事件需要对具有不同结构的每张表单独考虑，为其设计相应的 schema 文件，供其序列化所用。这将涉及到 schema 自动生成等组件的设计，使其为具有不同结构的 HBase 表生成相应的 schema 文件。这些改进都将带来新的研究和开发工作。

参 考 文 献

[1] Chang F, Dean J, Ghemawat S, Hsieh C, et al. Bigtable: a distributed storage system for structured data, ACM Transactions on Computer Systems (TOCS), 2008, 26(2): 4

[2] DeWitt D,Halverson A, Nehme R, et al. Split query processing in polybase//Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data , New York, USA,2013:1255-1266

[3] Färber F, Cha S K, Primsch J, et al. SAP HANA database: data management for modern business applications, ACM SIGMOD Record, 2011,40(4):45-51

[4] Stonebraker M, Madden S, Abadi D J, et al. The end of an architectural era:(it's time for a complete rewrite//Proceedings of the 33rd international conference on Very large data bases, Vienna, Austria 2007:23-27

[5] Das S, Botev C, Surlaker K, et al. All aboard the Databus!: Linkedin's scalable consistent change data capture platform//Proceedings of the Third ACM Symposium on Cloud Computing, San Jose, USA,2012: 18

[6] Auradkar A, Botev C, Das S, et al. Data infrastructure at linkedin// Proceedings of the 2012 IEEE 28th International Conference on Data Engineering, Washington, DC, USA ,2012: 1370-1381

[7] Abouzied A, Abadi D J, Silberschatz A. Invisible loading: access-driven data transfer from raw Files into database systems//Proceedings of the 16th International Conference on Extending Database Technology, Genoa,Italy, 2013:1-10

[8] Zikopoulos P, Eaton C. Understanding big data: Analytics for enterprise class hadoop and streaming data. New York,McGraw-Hill Osborne Media, 2011

[9] Stonebraker M. SQL databases v. NoSQL databases, Communications of the ACM, 2010,53(4) :10-11

[10] Pokorny J. NoSQL databases: a step to database scalability in web environment.International Journal of Web Information Systems, 2013,9(1): 69-82.

[11] Vossen P, Rigau G, Serafini L, et al. NewsReader: recording history from daily news streams//Proceedings of the 9th Language Resources and Evaluation Conference (LREC2014). Reykjavik,Iceland, 2014: 26-31.

[12] Hacígümüş H, Sankaranarayanan J, Tatemura J, et al. Odyssey: a multistore system for evolutionary analytics.Proceedings of the VLDB Endowment, 2013, 6(11): 1180-1181

[13] LeFevre J, Sankaranarayanan J, Hacigumus H, et al. MISO: Souping up big data query processing with a multistore system//Proceedings of the 2014 ACM SIGMOD international conference on Management of data. Snowbird, USA,2014: 1591-1602

[14] LeFevre J, Sankaranarayanan J, Hacigumus H, et al. Opportunistic physical design for big data analytics[C] //Proceedings of the 2014 ACM SIGMOD international conference on Management of data. Snowbird, USA,2014: 851-862

[15] Vassiliadis P. A survey of Extract–transform–Load technology. International Journal of Data Warehousing and Mining (IJDWM), 2009, 5(3): 1-27.

[16] Apache HBase, http://hbase.apache.org/.

[17] Lamport L. The part-time parliament. ACM Transactions on Computer Systems (TOCS), 1998, 16(2): 133-169.

[18] Flynn M J, Gray J, Jones A K, et al. Operating systems, an advanced course[M]. Springer-Verlag, 1978.

[19] Apache Avro,http://avro.apache.org/.

作者简介

蒋静远 男，1989 年生，浙江大学计算机科学与技术硕士研究生，主要研究领域为存储管理、NoSQL 多存储系统

E-mail:jiangjingyuan@zju.edu.cn

鲁伟明（通信作者） 男， 1980 年生，浙江大学计算机科学与技术学院博士，主要研究领域为非结构化数据管理、数字图书馆等

E-mail: luwm@zju.edu.cn