DOI:10.7544/issn1000-1239.2017.20150492
54(1):80-93,2017

基于开源生态系统的大数据平台研究

雷 军[1,2] 叶航军[2] 武泽胜[2] 张 鹏[2] 谢 龙[2] 何炎祥[1,3]
[1](武汉大学计算机学院 武汉 430072)
[2](小米科技有限责任公司 北京 100085)
[3](软件工程国家重点实验室(武汉大学) 武汉 430072)
(leijun@xiaomi.com)

Big-Data Platform Based on Open Source Ecosystem

Lei Jun[1,2], Ye Hangjun[2], Wu Zesheng[2], Zhang Peng[2], Xie Long[2], and He Yanxiang[1,3]
[1](*Computer School*, *Wuhan University*, *Wuhan* 430072)
[2](*Xiaomi Inc*, *Beijing* 100085)
[3](*State Key Laboratory of Software Engineering* (*Wuhan University*), *Wuhan* 430072)

Abstract As large-scale data collecting and processing are being widely studied in recent years, several released big data processing platforms are increasingly playing important roles in the operations of many Internet businesses. Open source ecosystems, the engine of big data innovation, have been evolving so rapidly that a number of them are successfully adopted as the components of mainstream data processing platforms. In reality, however, the open source software is still far from perfect while dealing with real large-scale data. On the basis of the industrial practice at Xiaomi Inc, this paper proposes an improved platform for collecting and processing large-scale data in face of varied business requirements. We focus on the problems in terms of the functionality, consistency and availability of the software when they are executed for data collecting, storing and processing procedures. In addition, we propose a series of optimizations aiming at load balance, failover, data compression and multi-dimensional scheduling to significantly improve the efficiency of the current system. All these designs and optimizations described in this paper have been practically implemented and deployed to support various Internet services provided by Xiaomi Inc.

Key words Hadoop; open source ecosystem; big data; data center; network virtualization

摘 要 大规模数据的收集和处理是近年的研究热点,业界已经提出了若干平台级的设计方案,大量使用了开源软件作为数据收集和处理组件.然而,要真正满足企业应用中海量数据存储、多样化业务处理、跨业务分析、跨环境部署等复杂需求,尚需设计具有完整性、通用性、支持整个数据生命周期管理的大数据平台,并且对开源软件进行大量的功能开发、定制和改进.从小米公司的行业应用和实践出发,在深入研究现有平台的基础上,提出了一种新的基于开源生态系统的大数据收集与处理平台,在负载均衡、故障恢复、数据压缩、多维调度等方面进行了大量优化,同时发现并解决了现有开源软件在数据收集、存储、处理以及软件一致性、可用性和效率等方面的缺陷.该平台已经在小米公司成功部署,为小米公司各个业务线的数据收集和处理提供支撑服务.

收稿日期:2015-06-12;**修回日期**:2016-08-08
基金项目:国家自然科学基金项目(91118003,61373039,61170022)
This work was supported by the National Natural Science Foundation of China (91118003, 61373039, 61170022).

关键词　Hadoop;开源生态系统;大数据;数据中心;网络虚拟化

中图法分类号　TP391

大规模数据的收集和处理是近年来业界和学术界的热点,被称为"大数据"问题."大数据"问题存在多种定义,现在普遍被接受的是 IBM 的 3V 定义[1],即数量(volume)、种类(variety)和速度(velocity),也就是数量巨大、种类丰富、快速生成并需要快速处理的数据.大规模数据的收集和处理有许多实际的应用.对互联网企业而言,用户在使用其产品的过程中会产生大量的业务数据,比如使用日志、交易日志和关系链等.对这些数据的分析和处理,可以深刻了解用户的需求.每一次用户对产品的使用都反映了用户的需求和对产品的反馈.对这些数据的分析和挖掘可以帮助公司改进自身产品,提升用户体验,为用户创造更大的价值.因此,公司通常有强烈的需求来分析和处理上述数据.

以小米科技有限责任公司(以下简称:小米公司)为例,公司业务数据的收集、分析和处理是一个典型的大数据问题:PB 量级的数据总量、多种数据格式(如逗号分隔值(comma separated value, CSV)、Thrift 消息[2]、文本文件、关系数据库等)、上百个数据来源、每日 TB 量级的数据增量和小时级别的处理速度要求等.大数据问题的解决,需要一套行之有效的技术架构,一般是分层次的堆栈式技术架构.对此,EMC 将大数据技术架构分成了 4 层:基础层、管理层、分析层和应用层.小米公司的内部业务在基础层和管理层上沿袭了该框架,但在数据的应用和分析上有所不同,以适应公司自身的业务特点.

1) 数据存量和增量大.PB 级别的数据总量和 TB 级别的数据日增量,对数据存储和传输的成本与效率提出很高的要求.

2) 业务线多、数据来源和格式多样化.上百个业务项目和数据来源,多种异构的数据格式,要求大数据平台有足够的灵活性和可扩展性.

3) 跨业务数据分析和挖掘的需求大.联合利用用户在多个产品上的使用数据,才能更深刻了解用户的需求,更好地改善用户体验.

4) 业务部署和大数据平台部署的情况比较复杂.多机房部署、异构的机房环境、要求集群和平台的部署、监控和报警等要足够高效.

本文从小米公司的应用和实践出发,在不失通用性的前提下,提出了一个基于开源生态系统[3]的统一的大数据收集和处理的基础平台.本文的主要贡献是将开源软件的组件与自主研发的软件组成一个完整的大数据平台,并通过一系列的技术创新和改进,使其能够胜任真实场景下大数据对系统功能、性能、一致性和可用性等各方面的需求.本文首先介绍相关研究和实践工作,然后分别描述平台的总体架构组成以及所做的改进和创新,最后展望未来的发展路线和计划.

1　相关工作

关于大数据平台,业界较为有代表性的工作是 Facebook 的实时数据收集和分析平台[3-4].该平台的目标是解决大规模(scalability)和低延迟(latency)的问题,它既使用了 Scribe[5], HDFS(Hadoop distributed file system,是 Hadoop 项目的一个核心子项目[6]),MapReduce[7],Hive[8],HBase[9]等开源系统,也自行开发了 Calligraphus,PTail,Puma 等私有系统.该平台的侧重点是数据的收集和汇聚,即实时的分类统计,而非通用的数据计算和分析服务.这个平台最终能够在 9 GBps 的写入速度下把延时控制在 10 s 之内.

学术界关于大数据平台也有大量的研究和实践,大致可以分为基于应用、基于模型以及基于平台 3 类.基于应用的研究工作主要从 Web 日志挖掘这个应用出发,考虑如何在 Hadoop 等开源的生态系统上构建分布式、可存储和挖掘大规模日志数据的平台.主要的工作在于讨论和验证分布式集群对于提高 Web 日志挖掘效率的可行性,并提出了相应的解决方案[10-12].基于模型的工作重点是讨论了更为通用的海量数据处理和计算模型,包括计算模型本身、网络模型和优化、编程模型等关键问题,也讨论了通用模型在具体应用中的实际问题和效果,比如数据清洗、容错等[13-15].此外,基于平台的工作更多是从平台自身的角度,比如数据管理、资源调度与虚拟化,并把整个系统分成多个层次[16-18].例如把系统分为数据库访问层、数据处理层和业务应用层[16];将系统分为算法层、任务层和用户层[18].

目前已有的工作主要集中研究了大数据平台中一些重要组件的设计和实现.由于小米公司的业务具有数据量大、业务需求多样化、跨业务分析的需求

大、部署环境复杂等特点,需要一个能管理海量数据整个生命周期的、完整的、通用的大数据平台.此外,还需解决现有的系统在数据收集、存储和处理、一致性、可用性和效率等关键问题上存在的缺陷.然而,现有开源软件的组合方案在数据存储、压缩、传输等性能上常常无法满足大型互联网企业的海量业务处理需求;另一方面,现有方案也无法支持多样化业务的分析和挖掘需求.此外,分布式部署环境下的可靠性尚需提升,存储、带宽、维护成本也需要进行优化.

本文从通用平台设计的角度出发,主要解决下列问题:大规模数据的实时收集和存储、计算资源与作业的管理与调度、集群管理(部署、监控和报警).同时,在功能、一致性、可用性和效率等方面做了重大改进和提高.

2 总体架构

一个完整通用的大数据平台,至少要涵盖数据的收集、存储、计算和管理等方面.本平台选用了部分开源软件作为系统的主要组件,包括 ZooKeeper[19], Hadoop (HDFS/MapReduce/YARN), HBase, Hive, Scribe 等.这些开源软件相对成熟,生态系统已经比较完备,可用于快速搭建大数据平台.在此基础上,本平台增加了自主开发的 Minos 监控系统,并基于对业务特性的深入分析调整和完善平台的设计.图 1 是平台的整体架构图.出于完整性的考虑,该架构图还包含了该大数据平台正在试验支持的计算框架,包括 Storm[20], Spark[21], Impala[22] 等.

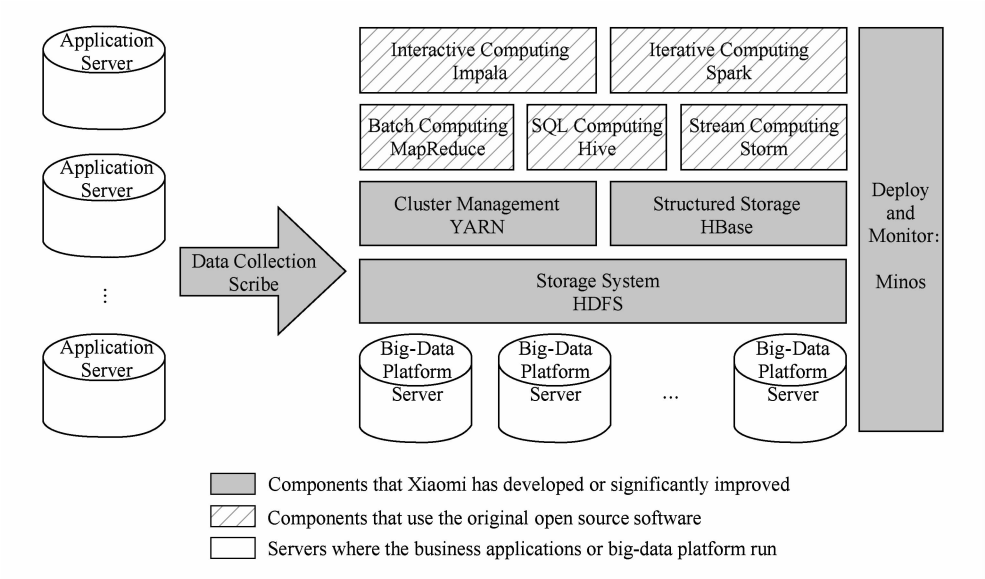


Fig. 1 Overall architecture of big-data platform

图 1 大数据平台整体架构图

3 数据收集系统

对于大部分应用场景来说,业务数据的来源和格式经常会有很多种,比如 Apache 或 Nginx 等 Web Server 的访问日志、业务自定义的 CSV 格式文件以及用 Protocol Buffer[23] 或者 Thrift 消息编码过后的消息.一个足够通用和灵活的数据收集平台,需要同时满足不同业务的多样化需求.

许多开源的数据收集系统,比如 Facebook 的 Scribe[5]、LinkedIn 的 Kafka[24]、Cloudera 的 Flume[25] 和 Apache 的 Chukwa[26],在业界都有广泛的应用.如果需要考虑到业务种类较多,数据格式和对数据的后续处理有多种方式,期望的数据收集系统需要满足下面 6 个特点(优先级由高到低):

1) 高可用.数据不会因为单节点或者少数节点的故障丢失.

2) 灵活.能够满足多种业务不同的使用方式和后续处理需求.

3) 使用简单.各业务接入系统的学习成本较低.

4) 易配置和维护.较低的运维成本.

5) 低外部依赖.较低的运维成本.

6) 架构和实现简单.多数开源系统需要一些改进来适配业务的要求.

综合考虑,Scribe 在这 6 个方面有一定优势,图 2 是本文提出的基于 Scribe 的数据收集系统架构图.

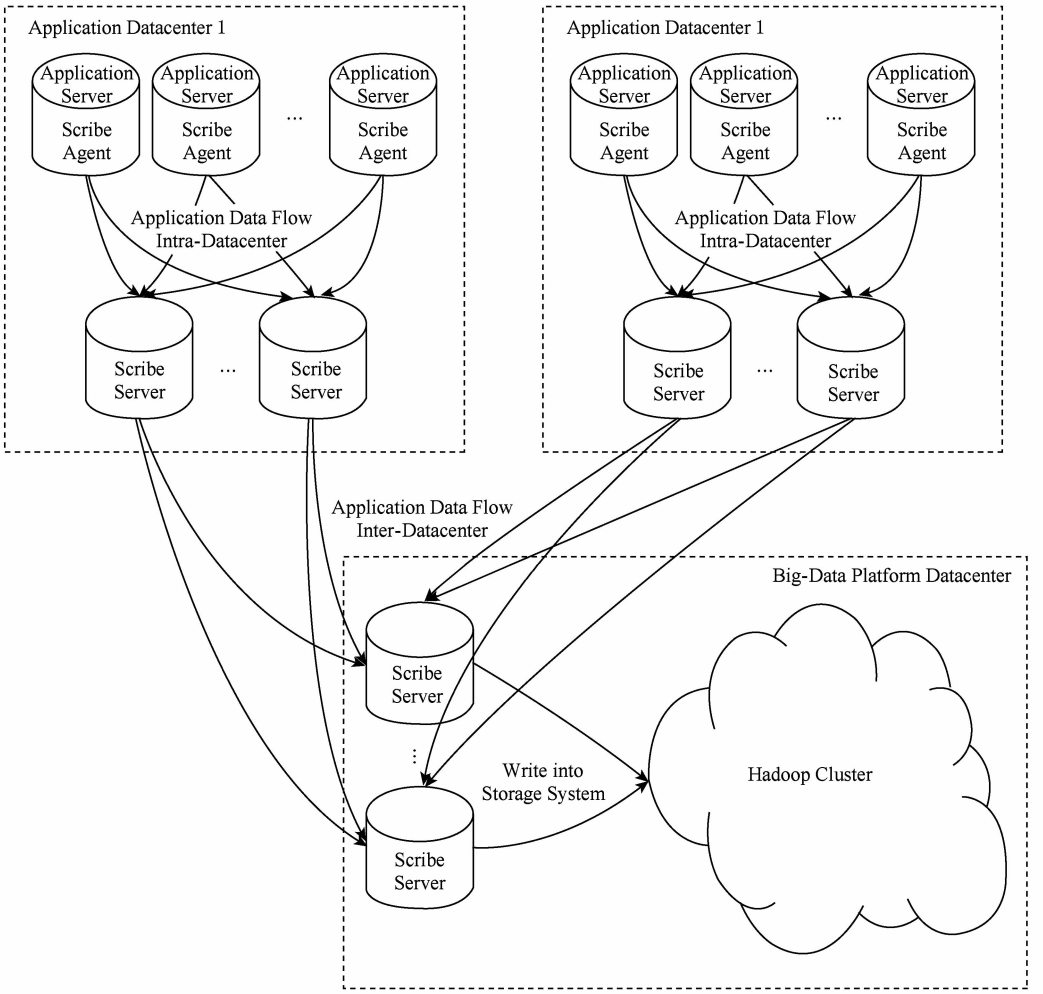


Fig. 2　Architecture of data collection system

图 2　数据收集系统架构图

3.1　数据传输的优化与改进

在设计支持跨数据中心的分布式数据收集系统时,为了统计和数据处理的方便,经常需要将所有的业务数据最终写入到同一个 Hadoop 集群里(也会在同一个数据中心),引起跨数据中心的数据传输.实践发现,大量的日志数据占据跨数据中心带宽的相当比例,浪费了宝贵的带宽资源.

本文提出了一种改进方法,可以在传输时对收集的数据进行压缩.实践证明这可以有效地减少数据传输量,很大地节约运营成本.

Scribe 是通过 Thrift 的 RPC 接口对外提供服务,Thrift 本身不提供传输数据压缩的功能.Thrift 本身也是一个分层设计的结构,加上 Scribe 又是搭建在 Thrift 之上的应用,所以有多个地方可以选择来实现压缩,比如 Thrift Protocol 层、Thrift Transport 层或者在 Scribe 本身.由于其他 Thrift Server 也可能有数据传输压缩的需求,本文提出了一种通用的解决方案,在 Thrift Transport 层来实现 Compressed 的传输协议,使得各类 Thrift Server 都能与之兼容.

Thrift 本身提供了良好的扩展性.Thrift Server 缺省使用了内置的 TFramedTransport 传输协议,这是一个直接基于系统底层传输协议(在 Thrift Server 里就是 TCP 协议)之上的简单的非压缩传输协议.同时 Thrift Server 在构造的时候允许传入一个 TTransportFactory 的传输层工厂类,通过传输层的串联模式,可以在内置传输协议的基础上实现更复杂的协议.

本文提出了一种新压缩传输协议 TSnappyTransport 和它的工厂类 TSnappyTransportFactory.图 3 是原始的非压缩的传输协议和本文提出的压缩传输协议的对比.由于本文提出的协议使用了传输层的串联模式,所以可以认为在原始的传输协议

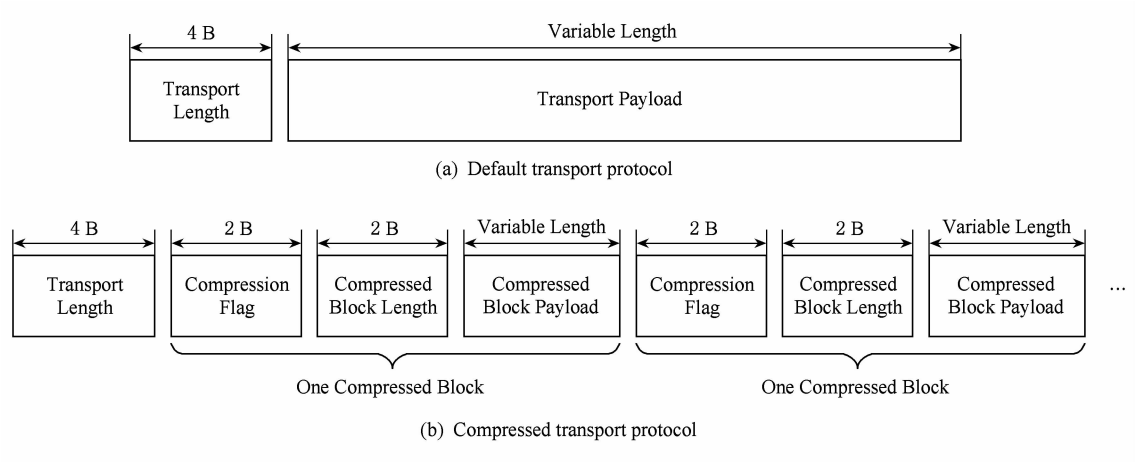


Fig. 3 Default transport protocol and compressed transport protocol

图 3 缺省传输协议与压缩传输协议

基础上,对它的有效载荷(payload)又进行了一次分块压缩与编码.

本文提出的压缩传输协议使用了 Snappy 压缩算法,它是 Google 提出并开源的一个压缩算法和代码库[27]. 和其他常用的压缩算法相比,它的最大特点是在压缩率可接受的情况下,压缩和解压缩的速度非常快. 例如与 zlib 的快速模式相比,对于大部分输入 Snappy 能够快 10 倍以上,但其压缩率会有 20%~50%的损失. 所以该算法特别适用于在线传输数据的压缩,不会给 CPU 造成严重负担或明显增加延迟.

根据 Google 的官方数据,使用 64 位 Intel Core i7 CPU,单核模式下 Snappy 的压缩速度超过 250 MBps,解压速度超过 500 MBps. 线上服务器一般是 8~24 核的配置,所以它引起的 CPU 开销基本可以忽略不计.

目前的实现仅支持了一种压缩算法,所以本文提出的压缩传输层协议直接命名为 Snappy Transport. 理论上该协议可以扩展支持任意的块压缩算法,以便于业务根据实际需求进行选择,留给将来的工作做扩展.

表 1 是从 3 种典型的业务日志数据中分别抽取一段,分别用未压缩和压缩 2 种模式传输日志消耗的网络带宽以及压缩率.

Table 1 Compression Ratio of Data Transportation

表 1 数据传输的压缩率

Category	Data Size/B	Trans Size/B (Uncompressed)	Trans Size/B (Compressed)	Compression Ratio/%
Text	58 404 985	63 984 525	18 124 071	28.33
Base64	52 547 823	55 270 043	30 176 765	54.60
Short Log	65 838 014	79 685 597	22 901 854	28.74

在真实业务场景下,压缩传输只使用了原来 30%左右的网络带宽,并且 CPU 没有成为新的瓶颈,因此也不需要部署新的 Scribe Server 来分担负载. 该项改进明显降低了日志数据在网络传输上的成本.

3.2 负载均衡和故障处理的优化与改进

数据收集系统很重要的一个要求是高可用性. Scribe 在这方面有独特设计,比如 Buffer Store 可以在下游的主通道不可用的时候,先把数据写到本地文件(也可以配置为写到其他 Store 中),待下游主通道可用时再把本地缓存的数据发送过去.

在本文提出的数据收集系统中,需要有一套中心服务器负责接受所有业务的数据,再把数据写入到统一的 HDFS 集群中. 为了避免该服务器成为系统的故障点,需要用一主一备 2 个服务器来提高可用性,用 Buffer Store 配置成主服务器不可用时写入备服务器. 这在应对服务器的偶然宕机或者运维操作时将起关键作用,显著提升可用性.

然而,在这种配置下的单个服务器需要承担系统的所有负载(主和备同时只有一个在提供服务). 随着业务数据流量的增加,在业务峰值时,流量经常超过单个服务器的处理能力. 如果主服务器因为超载变得不可用,所有数据又都会写到备服务器,由于这些服务器的配置相同,备服务器也常常超载,导致整个系统的不可用或者抖动. 实际上并不需要关心具体是哪个 Scribe 服务器把数据写入到 HDFS,所有服务器的角色是对等的,所以需要一个完备的负载均衡方案. Scribe 有一种 Bucket Store 的配置,具有负载均衡的能力,但对 Scribe 服务器的故障处理(failover)支持差,单个服务器故障也会导致整个系

统不可用.本文对此提出了4点改进以提高可用性:

1)跟踪所有服务器的状态,未能成功应答的服务器会被标志成"不可用".

2)只有处于"可用"状态的服务器才会成为日志数据下发的候选.

3)定义了一种"round_robin"的Bucket新类型,在所有"可用"服务器中循环选择候选下发数据,直到有一个服务器成功应答(即发送成功).

4)定期重连"不可用"的服务器,如果连接成功,把它放到"可用"服务器集合里.

下面通过模拟实验来比较改进前后日志收集系统的总体可用性.假设单个Scribe服务器的可用性为p,总共有n台Scribe服务器,将n台Scribe服务器配成n个bucket.假设各个服务器的可用性是独立的,可以推导出总体可用性为

$$p_{\text{final}}=p^n. \tag{1}$$

在改进之后,同样假设各个服务器的可用性是独立的,但至少要有m个服务器可用总体系统才可用(考虑到服务器的处理能力),可以推导出总体可用性为

$$p_{\text{final}}=\sum_{i=m}^{n}C_n^i p^i(1-p)^{n-i}. \tag{2}$$

表2比较了改进前后日志收集系统的总体可用性.假设单个Scribe服务器的可用性$p=0.99$,同时至少有一半的服务器可用总体系统才可用($m=n/2$).这个改进彻底解决了Scribe在负载均衡和故障处理上的缺陷.在业务中的实践也表明进行上述改进后,可用性和系统的可扩展性有明显提高,没有再出现因为超载或者单机故障造成的系统不可用.

Table 2　Comparison of Log Collection System Overall Availability Before/After Improvement

表2　改进前后日志收集系统总体可用性的比较($p=0.99$,$m=n/2$)

n	Before Improvement	After Improvement
2	0.980 1	0.999 9
4	0.960 596 01	0.999 996 03
6	0.941 480 149 401	0.999 999 852 39

4　数据存储系统

数据规模较大的存储会超出单机的存储能力,需要一个分布式的存储系统.传统的技术包括存储区域网络(storage area network, SAN)、网络附加存储(network attached storage, NAS)、网络文件系统(network file system, NFS)等.这些存储技术都需要高端或专用存储设备,成本通常较高.

近年来随着低成本存储设备的可靠性提高,软件冗余和纠错技术的发展,也逐渐出现了基于廉价和通用存储设备的分布式文件系统.尤其是Google发表了内部设计和使用的分布式文件系统(Google file system, GFS)[28],验证了这种技术在提供类似可靠性的前提下,性价比和可扩展性有很大的提高.

此后出现了大量的开源实现.其中HDFS是使用比较广泛、也比较成熟的一种开源实现.本文提出的大数据平台也是以HDFS为核心的存储系统.

作为一个分布式存储系统,最重要的衡量指标是一致性(可靠性)、可用性和性能.尤其是一致性和可用性,往往是选择一个分布式存储系统时的关键因素.在部署和使用开源的HDFS版本时,我们发现HDFS在一致性和可用性上的一些严重缺陷.本文提出了相应的改进和优化方案并在业务系统中部署了改进后的版本.

4.1　一致性的优化与改进

存储系统由于各种原因(新特性、修复缺陷等),会对软件版本进行发布和升级.为了尽量避免对业务的影响和提高可用性,更好的实践是在持续提供服务的情况下,对集群中的各个节点进行逐台滚动升级.

在实施过程中发现在这种升级方式下,HDFS上的文件很小概率下有损坏的情况.对于一个存储系统而言,文件损坏是很严重的缺陷,所以也是本文必须要解决的问题.由于该现象是偶发的,深入分析后确认是在HDFS写数据的流水线中间节点宕机后恢复的过程中,由于HDFS本身逻辑的缺陷,导致Checksum文件多出一个Checksum,从而导致HDFS校验Checksum失败,进而认为数据被损坏.这已经相当于出现了丢失数据的现象,之前已经成功写入的数据无法再正确读出,从而破坏了一致性的约定.

在Hadoop2.0版本时我们已向社区汇报了该问题,并提交了补丁代码[29].该问题被社区确认为严重的数据损坏问题,并在Hadoop2.7版本中得到了解决.对此缺陷进行了修正之后,再未出现集群逐台滚动升级时的文件损坏.

除了需要对存储系统的软件版本进行升级外,经常也会有需求添加或移除一些存储节点(DataNode).添加存储节点的过程比较简单,只需要在新的节点

上配置好软件环境并启动相应的服务即可,将来的数据写入就会依据一定的概率和规则分配到新节点上.但移除旧节点会复杂一些,为了防止数据丢失或者可靠性下降,需要先将旧节点所服务的数据移到还将提供服务的节点之后才能下线.同样的,需要存储集群在整个移除过程仍能正常服务.

HDFS 提供了从集群优雅地卸下存储节点的机制(decommission).在集群迁移的过程中,需要同时卸下(decommission)多个节点.实施过程中发现,当 Decommission 进行到最后的时候,有部分节点无法结束 Decommission,强制把这些节点关闭服务发现会有数据丢失.经过调查发现,在移除节点的过程中,如果某个数据块的 3 个副本都在需要移除的节点上,而且这个数据块在移除时正在被打开写的话,这里 HDFS 自身的处理逻辑有缺陷,会导致这样的数据块无法被正常复制到能够提供正常服务的节点上去.

针对该缺陷,本文调整了文件完成的判断条件:只要活跃节点和待移除节点上的块复本数满足最小复本数,则正常结束文件.之后由 Decommision 流程将数据块从待移除节点复制到活跃节点,完成全部数据块复制后再移除节点,实现了无数据损失的节点退出.

下面通过模拟实验计算在改进之前出现异常(移除节点时无法正常结束或者丢失数据)的概率.假设集群有 n 个存储节点,同时移除 m 个存储节点,当时有 k 个文件同时被写入数据.根据前面的分析,只要任何一个正在被写入的文件的 3 个副本都在这 m 个存储节点,就会出现异常.假设副本在数据节点上的分配是均匀分布且独立的,可以推导出出现异常的概率为

$$p_{\text{abnormal}}=1-\left(1-\left(\frac{m}{n}\right)^3\right)^k. \tag{3}$$

表 3 给出了 5 种典型配置下出现异常的概率.可以看出在对此缺陷进行了修正之前,出现异常导致移除节点时无法正常结束或者丢失数据的概率较大.对此缺陷进行了修正之后,再未出现集群移除节点时无法正常结束或者丢失数据的情况.

Table 3 Probability of Abnormity for Representative Configurations

表 3 典型配置下出现异常的概率

N	m	k	p_{abnormal}
10	3	1	0.027
10	3	10	0.239 4
10	3	100	0.935 4
100	10	100	0.095 21
100	10	1 000	0.632 3

4.2 可用性的优化与改进

当前 HDFS 的实现中,在客户端有一个数据节点(DataNode)的黑名单,在用户使用客户端操作 HDFS 的过程中,如果发现某个数据节点出现故障,都会被加入到这个黑名单,后续该客户端就不再从该数据节点读写数据.这样是一种优化,目的是避免从故障或者繁忙的节点读写数据.

在集群规模较小时,由于集群上的计算任务繁重,高负载的情况时有发生,导致客户端偶尔发生数据节点读、写超时的情况.这类数据节点将被加入到上述黑名单.在本文的数据收集系统中,中央 Scribe Server 写 HDFS 的模式是:打开一个文件持续写,直到达到一定的大小,或者到第 2 天再切换文件.在实际的生产环境中,有些业务数据量不大但持续会有,一天的日志总大小达不到切换文件的条件,因此,一整天都在持续地写同一个文件.在这样的情况下,当所有的数据节点都进入到黑名单后,Scribe Server 对 HDFS 就不能写了.由于这个黑名单是文件流级别的,所以后续除非重新创建文件流,否则该文件流涉及到数据节点的操作都会失败.这时已经写入的数据不会丢失,而且能够正确读出,但从 Scribe Server 的角度,HDFS 集群已经处于不可用的状态.

下面通过模拟实验来计算在改进之前 HDFS 集群出现不可用的概率.假设集群有 n 个存储节点,每个存储节点在这个时间周期内(这里是 1 d)出现不可用(主要是读写超时)的概率为 p,这个时间周期内有 k 个文件被写入数据且未出现文件切换.假设副本在数据节点上的分配是均匀分布且独立的,存储节点出现不可用是独立事件,可以推导出 HDFS 集群出现不可用的概率为

$$p_{\text{unavailable}}=1-(1-p^n)^k. \tag{4}$$

表 4 给出了 6 种典型配置下出现不可用(某个文件无法写入)的概率.

在优化和改进之前,*HDFS* 集群有较高的概率出现某个文件不可写入.在本平台中,存储与计算共享同一个集群,而且集群上的计算任务大,单个机器在 1 *d* 的时间周期里,出现(对某个客户端至少一次)读写超时的概率非常高.另外计算任务是批处理提交的,机器出现读写超时并不是独立的,所以会经常遇到某个文件不可写入的情况.

Table 4　Probability of Unavailability for Representative Configurations

表 4　典型配置下出现不可用的概率

N	p	k	$p_{\text{unavailable}}$
10	0.5	10	0.009 723
10	0.5	100	0.093 08
10	0.9	10	0.986 2
10	0.9	100	$1-2.396\times10^{-19}$
100	0.5	10 000	7.889×10^{-27}
100	0.9	10 000	0.233 3

对此本文做了优化和改进,对于进入黑名单的数据节点,当它进入黑名单超过一定的时间,给与它一定的机会让其复活.从上线后的效果来看,对可用性有很明显的提高,再未出现由于数据节点负载高造成的偶尔超时,导致某个文件不可写入的情况.

5　计算系统

和分布式的数据存储系统相类似,对规模较大的数据进行处理和计算,往往也会超出单机的处理能力,需要一个并行计算的系统和框架,传统的技术包括 MPI 和分布式数据库等.

Google 近些年陆续发表了内部设计和使用的计算框架,包括 MapReduce,Sawzall[30],Dremel[31]等,为大规模数据的计算框架带来了一些新思路.其中 MapReduce 是把所有的并行计算都分解为 Map,Shuffle 和 Reduce 这 3 个阶段进行并行化,能够满足一大类并行计算的需求;而 Dremel 则是用 SQL 语句来表示计算任务,由后台的计算系统把 SQL 语句翻译成执行计划,在多个节点上并行执行.这 2 种框架非常适合大规划数据的批次处理.

在开源生态系统里,Hadoop 的 MapReduce(也是 Hadoop 项目的一个核心子项目)和 Hive 是对应的 2 个实现,也是目前使用广泛、成熟度较高的实现.本文提出的大数据平台,也是以开源的 MapReduce 和 Hive 为核心的计算系统.

在具体的 MapReduce 版本方面本文选用了最新的 Hadoop MapReduce 2.0,该版本引入了通用的资源调度系统 YARN,整体架构也代表了下一代计算和资源管理的发展方向,也得到了业界的广泛认可和支持.在 2.0 的架构中,资源调度和作业调度逻辑分离,有效地减轻了中央节点的压力,以提供更好的集群可扩展性.各个 MapReduce 作业之间是独立的流程,由各自的 Job Master 进行管理,单个作业的失败不会影响到其他作业,因此作业的容错方面较 1.0 的架构也有了大幅改进.另外相比先前架构中以槽位(slot)作为单一调度维度,新架构中引入了内存、CPU 等多个调度维度,用户可以更准确地对任务所需要的资源进行描述,有利于集群资源的有效利用.此外,2.0 架构中的通用资源系统还支持在其上运行多种非 MapReduce 的作业,这也为不同业务的集群复用提供了可能.

5.1　计算资源的配额管理

在本平台的 Hadoop 应用中,离线集群存储了多种业务的数据,各业务通常都有各自的计算处理需求.除了 HDFS 存储配额管理之外,还需要为各业务的计算需求合理地分配计算资源.

Hadoop 的 YARN 延续了之前 MapReduce 的调度器的模型,包括先入先出调度器(FifoScheduler)、容量调度器(CapacityScheduler)以及公平调度器(FairScheduler).先入先出调度器是系统的默认调度器,它不考虑作业间的优先级差异,简单地按先到先服务的策略进行作业调度,在前面的作业没有执行完前,后续的作业只能排队等待,因此它并不适合本文所讨论的企业级需求场景;容量调度器和公平调度器在演化的过程中相互取长补短,功能特性具有一定的相似性,它们相比默认的调度器支持作业的优先级设置,支持多级调度队列的配置,支持作业抢占等,适用于企业级集群的资源分配场景.考虑到公平调度器还在开发和完善阶段,本文选用了更成熟的容量调度器作为资源配额管理的方案.

在实践中,面对不同业务的计算需求,本平台为各主要业务建立作业队列,为每个队列配置一定的计算资源底限以保证基本运算需求,同时为每个队列设置允许在集群空闲时最多使用的资源量,以提高集群整体的利用率.考虑到业务的层次化结构,本平台还在一级作业队列下建立二级队列,以满足一个业务内部的细分计算需求.通过队列的合理配额配置,在对各业务的资源需求进行隔离的同时,也能够充分复用集群,最大化集群的资源利用率.

5.2　多维度资源调度

在 Hadoop 1.0 中,计算资源使用槽位作为表示方式.一个计算节点上的 CPU、内存等资源被等分为若干个槽位,每个任务则描述需求多少个槽位的资源.这种方式将多维度的资源抽象为一种“资源”,简化了资源调度问题,但这种方式也有很多不足:槽位是预先静态划分的,无法最佳地适应动态

变化的作业,通常导致由于划分粒度过大而造成资源的浪费;其次,单一维度的资源描述不利于对CPU或内存需求多样化的任务共享资源,降低了集群的资源利用率;另外,以槽位作为资源描述单位也不方便对任务进行使用资源的隔离.

针对基于槽位调度的不足,Hadoop 2.0的YARN引入了多维度的资源调度,目前支持CPU和内存2个维度.例如,在新框架下,一个偏内存型的任务可以描述它需要4 GB的内存和1个CPU核,而偏CPU型的任务可以描述它需要1 GB内存和4个CPU核,这样的2个任务在不同维度上的需求互补性,可以最大化地发挥计算节点的资源利用率.除了充分提高资源利用率的同时,多维度的资源调度也有利于控制一个节点的并发任务,避免让节点负载过高.假设在集群中节点的内存较大(如64 GB),而CPU核数较少(如8核),在只有内存一个维度调度的情况下,要求1 GB内存的任务会在一个节点上运行几十个,任务彼此间会形成对CPU资源的强烈竞争,导致机器负载高,作业执行速率也大幅下降.引入CPU维度后,任务默认指定需求一个CPU核,调度时会因在这一维度达到上限而不再下发任务,从而控制机器的负载,保证作业的计算性能.

多维度调度的引入大大优化了资源的描述和资源调度功能,但由于它是Hadoop 2.0中较新的特性,所以也有一些潜在的问题.例如在使用过程中发现它在调度时计算下发任务量时存在缺陷,可能会而导致MapReduce作业的调度死锁.针对这一较严重的问题,本文对容量调度器进行了修改,在下发时综合多维度资源计算下发任务量,从而避免了调度死锁的发生.

5.3 容量调度器的负载均衡

容量调度器的功能满足了本平台的大部分需求,但它也存在不完善的地方.在实践中,调度器会在计算节点心跳汇报时,尽可能多地下发任务.这一策略不利于计算任务在集群中的均匀分布:在集群整体空闲时,任务集中分布在少量的节点上,并没有充分利用集群中节点的并发计算能力.针对这一问题,本文修改了调度下发策略,限制单节点单次下发的任务上限.修改后虽然会降低平均下发的速率,但由于任务在集群中的分布更新均匀,有效地利用了节点间的并发,因此整体上缩短了作业级的执行时间:在集群空闲时单作业执行时间能缩短30%~50%.另外引入单次下发的上限,在一定程度上也避免了内存或CPU需求密集性的任务集中分布在单个节点,有利于使一个节点上的任务需求多样化,提高单节点上可运行的任务数和节点资源的利用率.

5.4 MapReduce开发流程优化

在离线处理集群的运营过程中,除了积累Hadoop系统的应用和改进经验之外,对于优化MapReduce开发流程本文也进行了探索和尝试.分布式环境中,当程序出现问题时,快速准确地定位问题是一个巨大挑战.通常情况下,MapReduce程序的开发者在编写完程序后会在集群上直接运行测试,当出现异常时,很多时候需要查看作业日志,甚至到远程计算节点上分析问题.这种方式的问题定位成本非常高,既耗费了开发者的大量时间,也浪费了宝贵的集群计算资源.在协助用户定位问题的过程中发现,很多问题并不需要在集群上运行作业才能暴露出来,通过单元测试或本地模式运行就可以有效地排查.因此本文提出一个优化的开发流程如下:

1) 开发程序时,利用MR Unit测试框架为Mapper和Reducer等编写单元测试.通过单元测试覆盖主要场景,保证程序的基本正确性.

2) 取部分真实输入数据,利用MapReduce的本地模式运行作业,排查真实数据中的边界情况.如果遇到错误,则可以利用Eclipse等集成开发环境单机调试,分析定位问题.之后可以把新的场景补充到单元测试之中.

3) 上述2个阶段运行成功之后,再在集群上对更多的数据进行测试.在这一过程中重点关注作业的运算性能和资源使用情况,可以利用MapReduce的计数器功能查看系统及用户自定义的计数器,从而优化作业配置.

上述开发流程将问题以最小代价暴露出来,充分利用单机调试的便利性,尽量减少集群调试的需要,整体上降低了开发者定位问题的难度,有效地提高了开发效率.

5.5 MapReduce作业调优

MapReduce程序开发者除了要保证数据处理逻辑的正确性之外,还需要关注作业在集群中的运行性能和资源消耗.后者要求开发者对数据处理逻辑以及MapReduce和YARN系统的细节有深入的了解,能够根据实际情况调优作业参数,这无疑增加了MapReduce用户的使用成本.在协助用户进行作业性能分析和参数优化的过程中,发现常见的问题可以按处理阶段概括为以下3类:

1) Map阶段.内存配置不合理导致内存数据频繁落地磁盘,磁盘IO开销大.

2) Shuffle 阶段. Map 输出未压缩导致 Shuffle 数据量过大,带宽开销大;Reduce 端的 Shuffle 内存及并发参数的配置不合理导致磁盘 IO 开销大或数据拉取慢.

3) Reduce 阶段. 任务并发数不足导致单任务处理数据量过大;Reduce 的输出数据过大和 HDFS 多副本导致带宽开销大等.

上述这些问题覆盖了实际应用中大部分的性能调优的场景. 为了减少用户的使用门槛,可以利用 Hadoop 系统为每个作业记录历史文件,分析其中的任务数和各种系统计数器,判断可能的参数优化点,再提醒用户去关注相关问题. 这种自动化的流程也有效地降低了集群的运营成本. 例如在实践中曾遇到某一作业,虽然能够正常运行,但整体运行比同规模作业时间长很多. 通过自动化分析,发现问题在于其 Map 阶段 Java GC 时间占比很大(用户的 Map 算法频繁利用内存进行数据缓存),因此本平台调大了 Map 阶段的内存需求量,从而使单 Map 任务时间减少为原来的 1/5,作业整体时间也大幅缩短. 表 5 是优化前后的 CPU 耗时对比.

Table 5　Comparison of MapReduce Job Before/After Optimization

表 5　MapReduce 作业调优前后对比

Category	CPU Time/ms	GC Time/ms	GC Time over CPU Time/%
Before Optimization	746 860	472 015	63.20
After Optimization	129 562	1 027	0.79

6　集群管理

随着接入业务数量的增加和集群规模的增长,集群的布署、升级、监控以及管理成为了一个挑战,亟需一套能够方便布署、升级集群,同时能够直观查看集群运行状态的系统. 希望能够通过这样的系统,一方面可以降低集群维护成本,减轻维护集群的压力;另一方面可以实时查看集群的运行状态,让团队成员和用户了解集群的健康状况,同时也可以及时把集群的故障反馈给团队成员,能够让团队成员在第一时间发现问题、解决问题,把对业务的影响降到最小.

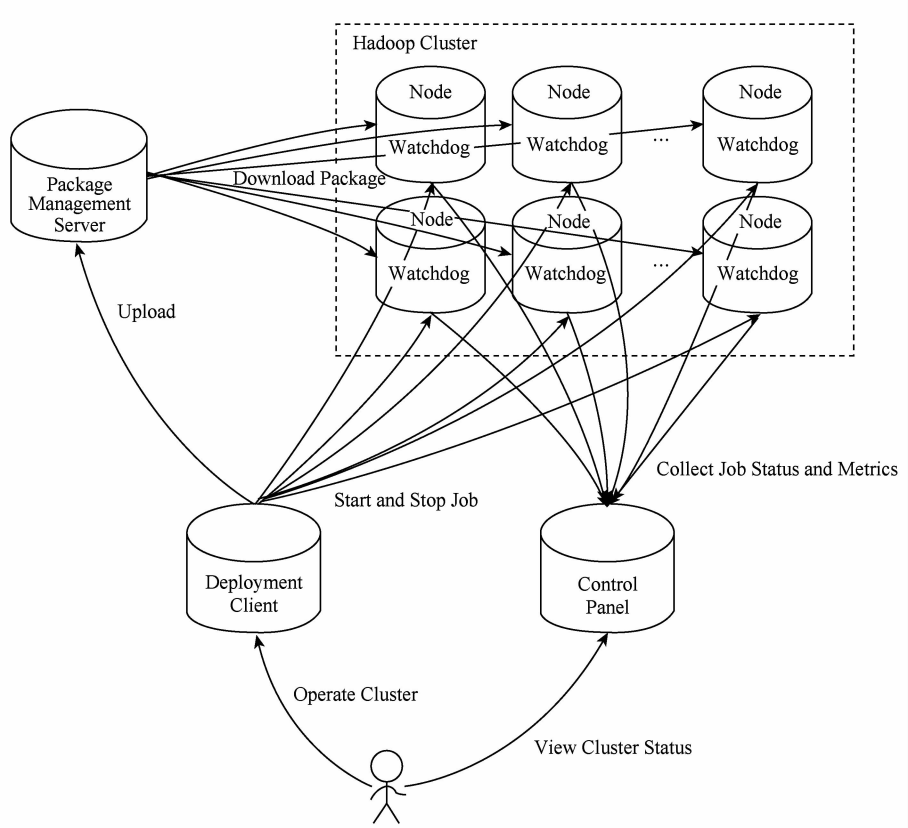


Fig. 4　Architecture of Minos deployment system

图 4　Minos 部署系统架构图

业内已有的解决方案,包括 Hadoop 原生的布署脚本、Cloudera Manager[32] 和 Apache Ambari[33] 等尽管有各自的优点与缺点,但都与本文要研究的目标系统有一些距离. 因此本文提出了一套自主设计和实现的 Hadoop 布署和监控系统 Minos,目前该系统已经开源[34]. 图 4 是 Minos 部署系统的架构图,整体系统主要由 4 个组件组成.

1) 客户端(client). 直接提供给用户使用的命令行工具. 用户可以用来部署和管理多种系统的集群服务与进程,包括安装、启停、清除等.

2) 监控面板(owl). 展示集群服务和进程状态的网站. 它通过 JMX[35] 接口从它管理的各个进程收集内部数据和状态,并根据集群的配置,按照服务、作业、任务(Service/Job/Task)3 个级别汇总和展示.

3) 监视进程(supervisor). 部署在集群的所有机器上,负责管理和监控服务的所有进程. Supervisor 原本是一个开源项目[36],提供了一套让用户在类 UNIX 操作系统上远程监控和控制进程的方法. 本文根据 Minos 的需要进行了扩展和改进,主要增加了一套 RPC 接口供 Minos Client 调用.

4) 包管理服务器(tank). 集群运行所使用的软件包集中管理和存放的服务器. Minos 以包名和版本号来唯一表示一个软件包.

使用 Minos 系统部署和管理一个集群服务的典型流程如下:

1) 安装 Minos 系统(所有集群服务仅需要做一次),安装集群服务所需要的软件包到 Tank;

2) 编写集群配置文件,通过 Minos Client 初始化集群;

3) 查看集群运行状态,根据需求启停、更新、清除集群服务.

Minos 系统已经成为内部部署和管理大数据平台各个组件服务的标准工具,目前支持了在使用的主流开源系统,包括 Hadoop(HDFS/YARN),ZooKeeper,HBase,Impala,Storm 等. 它大大降低了管理和维护这些大规模分布式系统的成本,提升了业务团队的生产效率. 根据实际使用的经验,Minos 系统主要具有 6 个特点:

1) 提供了直观的 Web 界面来查看集群的运行状态,提供了命令行工具来管理集群,方便快速定位错误.

2) 放宽了布署服务必须是系统级服务的约束,支持同机运行多个实例. 这个特性主要的应用场景是在大内存的机器上通过布署多个 RegionServer 来提高机器内存的使用率,同时能避免单个 RegionServer 的堆太大而导致的 GC 时间过长引起的一系列问题.

3) 灵活的包管理功能,对开发团队更加友好. 这个特性主要的好处有:①对于同一个系统特定的版本,团队内部只要有一位成员构建,其他成员便可以方便地复用编译好的软件包;②对于同一个系统不同版本的软件包都有明确的标识,互相不影响;③所有软件包都集中管理,有直观的 Web 界面进行操作.

4) 在集群中抽象出了 Service/Job/Task 的概念,能够通过配置文件直观、简洁地描述集群.

5) 对集群的管理既支持集群级别的管理,也支持 Job/Task 级别的管理. 这个特性可以灵活地支持操作整个集群,或者是集群中的某些 Job/Task.

6) 监控指标的收集与展示采用了 OpenTSDB[37],具有强大的线型扩展性. 由于 Hadoop 系统的监控指标较多,需要存储的时间较长,在前期采用 MySQL 来存储这些指标时,随着集群规模的增长,很快 MySQL 就成为了瓶颈. 后来经过调研,本平台把 MySQL 换成了 OpenTSDB,由于 OpenTSDB 底层的存储是基于 HBase 的,HBase 本身具有强大的线型扩展性,因此 Minos 中指标存储的问题便得到了很好的解决.

很多业务已经接入或正在接入本平台的存储与计算集群. 目前,整体数据存储量已达到 PB 级规模,每天运行计算作业 2 000 多个,吞吐量在 50TB 左右. 图 5 展示了 2013 年 8 月至 11 月的每日作业数情况.

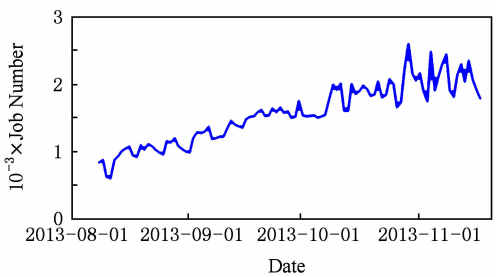


Fig. 5 Daily running jobs of MapReduce

图 5 MapReduce 每日作业数

7 未来工作

7.1 计算系统

Hadoop YARN 平台在支持现有 MapReduce 计算的同时,也为未来更多的扩展成为可能. 目前很

多开源项目支持在 YARN 平台上运行或部署,包括 Storm[20],Spark[21],Tez[38],Impala[22]. 这些项目扩展了分布式计算模型,对特定领域有更好的支持. 本文也尝试将这些项目应用到计算集群上,在复用集群的同时为用户提供更多的选择. 此外,YARN 也有发展成为通用部署平台的潜力,目前已经有将 HBase 部署在 YARN 上的开源项目,我们也会在这一领域继续探索和尝试.

7.2　存储系统

HDFS 目前已经基本能够满足大部分业务的需求,但是随着业务规模的增长,也凸显出一些新的需求. 此外 HDFS 本身的易用性方面也有很大的提高空间,未来的 5 个主要发展方向如下:

1) 名字服务. 支持通过名字访问 HDFS 集群.

2) HDFS Raid. 希望在减少备份数的同时不损失数据的可靠性,从而达到节约成本的目的.

3) HDFS QoS. 希望能够对用户提供的服务有基本的网络延迟和吞吐量的保证,同时保障数据的可靠.

4) 冷热数据分离. 希望对冷热数据使用不用的策略和备份数,进一步降低存储成本.

5) 跨数据中心同步.

7.3　集群管理

本文的数据存储与计算平台主要基于开源系统. 在受益于开源系统提供便利的同时,也希望能做一些事情来回馈开源社区,这是把 Minos 开源出去的主要目的. 另外也希望能够借助社区的力量,一起来完善 Minos. 当前已经规划要做或者正在做的一些特性主要有:

1) 同机多实例布署的支持;

2) 异构机型的支持;

3) 易用性的提升,包括相关文档完善、安装过程自动化等.

7.4　公有云

目前为止,数据的收集、存储、处理、计算平台都是面向公司内部用户的,属于私有云的概念. 小米公司有提供开放平台的计划,把自己拥有的平台与数据开放出去,便于各种应用的开发;同时也会开放数据处理的能力,让更多的用户收益. 在这个场景下会有 3 个新的挑战:

1) 多租户. 多个用户之间是不可见和不相互影响的,需要良好的数据和资源隔离来达到这点;同时在多用户情况下也要达到和用户约定的服务等级协议(service-level agreement, SLA).

2) 安全. 因为用户的数据和计算任务会托管在小米公司提供的环境里,安全是用户最为关心的问题之一.

3) 弹性. 用户的需求是动态变化的,平台需要根据用户的实际需求来分配资源,以降低用户的使用成本.

8　总　　结

随着互联网和移动互联网的快速发展和普及,人类所创造的数据量和产生的速度都在迅速膨胀,比如用户访问日志、用户生成内容(user generated content, UGC)等,客观上推动了大数据问题的研究. 大数据的一个特点是价值密度较低,但在数量庞大的数据背后,隐藏着深刻的规律和洞见. 对这些规律的挖掘和发现,一方面可以为企业带来巨大的商业价值,获得超越其他竞争对手的优势;另一方面也能丰富用户服务,提供更稳定、更优异的使用体验. 因此,如何从这些庞大、分散的数据中去粗存精,沙里淘金,是大数据要解决的问题和面临的挑战.

本文从小米公司的行业应用和实践出发,在深入研究现有平台的基础上,提出了一种基于开源生态系统的大数据收集与处理平台的设计方案. 同时针对现有开源软件在功能、一致性、可用性和效率等关键问题上的缺陷,提出了相应的优化和改进方案,并在业务系统中得以实施和验证.

当然,本文提出的大数据平台还有需要改进和完善的地方,比如计算模型较为单一、存储尚未支持冷热数据分离、尚未提供跨数据中心的同步功能等. 下一步研究工作将集中在全面的计算模型、低成本存储、跨数据中心同步、多租户等问题上.

参　考　文　献

[1] Zikopoulos P, Eaton C. Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data [M]. New York: McGraw-Hill, 2011

[2] Slee M, Agarwal A, Kwiatkowski M. Thrift: Scalable cross-language services implementation [R/OL]. Palo Alto: Facebook, 2007 [2015-06-08]. https://thrift.apache.org/static/files/thrift-20070401.pdf

[3] Shao Z. Real-time analytics at Facebook [C] //Proc of the 5th Extremely Large Databases Conf. Menlo Park: SLAC National Accelerator Laboratory, 2011: 21-33

[4] Shao Z. Real-time analytics at Facebook: Data freeway and puma [C/OL] //Proc of 2011 Hadoop in China. [2015-04-18]. http://hic2011. hadooper. cn/dct/attach/Y2xiOmNsYjpwZGY6MTQxMzY=

[5] Facebook. Scribe [CP/OL]. [2015-06-08]. https://github.com/facebook/scribe

[6] Apache. Hadoop [CP/OL]. [2015-06-08]. http://hadoop.apache.org

[7] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1): 107-113

[8] Apache. Hive [CP/OL]. [2015-06-08]. http://hive.apache.org

[9] Apache. HBase [CP/OL]. [2015-06-08]. http://hbase.apache.org

[10] Cheng Miao, Chen Huaping. Weblog mining based on Hadoop [J]. Computer Engineering, 2011, 37(11): 37-39 (in Chinese)
(程苗, 陈华平. 基于 Hadoop 的 Web 日志挖掘[J]. 计算机工程, 2011, 37(11): 37-39)

[11] Song Ying, Shen Qiwei, Wang Jing. Design and implementation of Web log pre-processing based on Hadoop [J]. Telecom Engineering Technics and Standardization, 2011, 24(11): 84-89 (in Chinese)
(宋莹, 沈奇威, 王晶. 基于 Hadoop 的 Web 日志预处理的设计与实现[J]. 电信工程技术与标准化, 2011, 24(11): 84-89)

[12] Liu Yongzeng, Zhang Xiaojing, Li Xianyi. Design of Web log analysis system based on Hadoop/Hive [J]. Journal of Guangxi University: Natural Science Edition, 2011, 36 (Suppl1): 314-317 (in Chinese)
(刘永增, 张晓景, 李先毅. 基于 Hadoop/Hive 的 Web 日志分析系统的设计[J]. 广西大学学报: 自然科学版, 2011, 36 (增刊 1): 314-317)

[13] Zhu Zhu. Research and application of massive data processing model based on Hadoop [D]. Beijing: Beijing University of Posts and Telecommunications, 2008 (in Chinese)
(朱珠. 基于 Hadoop 的海量数据处理模型研究和应用[D]. 北京: 北京邮电大学, 2008)

[14] Li Jun. Exploration on the cloud computing model based on Hadoop [J]. Information Security and Technology, 2011 (6): 30-32 (in Chinese)
(李珺. 基于 Hadoop 云计算模型探究[J]. 信息安全与技术, 2011 (6): 30-32)

[15] Wan Zhizhen. Design and implementation of parallel computing platform based on MapReduce model [D]. Hangzhou: Zhejiang University, 2008 (in Chinese)
(万至臻. 基于 MapReduce 模型的并行计算平台的设计与实现[D]. 杭州: 浙江大学, 2008)

[16] Cui Jie, Li Taoshen, Lan Hongxing. Design and development of the mass data storage platform based on Hadoop [J]. Journal of Computer Research and Development, 2012, 49(Suppl1): 12-18 (in Chinese)
(崔杰, 李陶深, 兰红星. 基于 Hadoop 的海量数据存储平台设计与开发[J]. 计算机研究与发展, 2012, 49(增刊 1): 12-18)

[17] Dong He, Xu Lingyu. SaaS-Flow system structure based on cloud platform [J]. Journal of Shanghai University: Natural Science Edition, 2013, 19(1): 14-20 (in Chinese)
(董贺, 徐凌宇. 基于云平台的软件服务流体系结构[J]. 上海大学学报: 自然科学版, 2013, 19(1): 14-20)

[18] Ji Jun. Design and implementation of a data mining platform architecture based on cloud computing [D]. Qingdao: Qingdao University, 2009 (in Chinese)
(纪俊. 一种基于云计算的数据挖掘平台架构设计与实现[D]. 青岛: 青岛大学, 2009)

[19] Hunt P, Konar M, Junqueira F P, et al. ZooKeeper: Wait-free coordination for Internet-scale systems [C] //Proc of the 2010 USENIX Annual Technical Conf. Berkeley: USENIX Association, 2010: 11-18

[20] Apache. Storm [CP/OL]. [2015-06-08]. http://storm.apache.org

[21] Apache. Spark [CP/OL]. [2015-06-08]. http://spark.incubator.apache.org

[22] Cloudera. Impala [CP/OL]. [2015-06-08]. http://impala.io

[23] Google. Protocol Buffer [CP/OL]. [2015-06-08]. https://code.google.com/p/protobuf

[24] Apache. Kafka [CP/OL]. [2015-06-08]. https://kafka.apache.org

[25] Apache. Flume [CP/OL]. [2015-06-08]. http://flume.apache.org

[26] Apache. Chukwa [CP/OL]. [2015-06-08]. http://chukwa.apache.org

[27] Google. Snappy [CP/OL]. [2015-06-08]. http://google.github.io/snappy

[28] Ghemawat S, Gobioff H, Leung S T. The Google file system [C] //Proc of the 19th ACM Symp on Operating Systems Principles. New York: ACM, 2003: 29-43

[29] Apache. HDFS-4660 [CP/OL]. [2015-06-08]. https://issues.apache.org/jira/browse/HDFS-4660

[30] Pike R, Dorward S, Griesemer R, et al. Interpreting the data: Parallel analysis with Sawzall [J]. Scientific Programming, 2005, 13(4): 277-298

[31] Melnik S, Gubarev A, Long J J, et al. Dremel: Interactive analysis of Web-scale datasets [J]. Proceedings of the VLDB Endowment, 2010, 3(1/2): 330-339

[32] Cloudera. Cloudera Manager [CP/OL]. [2015-06-08]. https://www.cloudera.com/products/cloudera-manager.html

[33] Apache. Ambari [CP/OL]. [2015-06-08]. http://ambari.apache.org

[34] Xiaomi. Minos [CP/OL]. [2015-06-08]. https://github.com/XiaoMi/minos

[35] Oracle. JMX:[CP/OL]. [2015-06-08]. http://www.oracle.com/technetwork/articles/java/javamanagement-140525.html

[36] Agendaless Consulting and Contributors. Supervisor [CP/OL]. [2015-06-08]. http://supervisord.org

[37] StumbleUpon. OpenTSDB [CP/OL]. [2015-06-08]. http://opentsdb.net

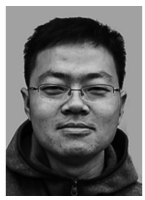
[38] Apache. Tez [CP/OL]. [2015-06-08]. http://tez.incubator.apache.org

Lei Jun, born in 1969. PhD candidate. Founder, board chairman and CEO of Xiaomi Inc. His main research interests include software engineering, distributed system, storage system, big data and high performance computing.

Ye Hangjun, born in 1976. PhD. Software engineer of Xiaomi Inc. His main research interests include distributed system, storage system and cloud computing (yehangjun@xiaomi.com).

Wu Zesheng, born in 1986. Bachelor. Former software engineer of Xiaomi Inc and co-founder of Hangzhou Bongmi Technology Co, Ltd. His main research interests include distributed system and cloud computing (wuzesheng@bongmi.com).

Zhang Peng, born in 1984. Master. Software engineer of Xiaomi Inc. His main research interests include distributed computing system and resource management system (peng.zhang@xiaomi.com).



Xie Long, born in 1984. Master. Software engineer of Xiaomi Inc. His main research interests include high availability and high performance in distributed system (xielong.me@gmail.com).

He Yanxiang, born in 1952. PhD, professor and PhD supervisor. Member of China Computer Federation. His main research interests include trusted software, distributed parallel processing and high performance computing.