doi:10.3772/j.issn.1002-0470.2019.04.001

分布式系统中的日志分析及应用①

陆 杰②[*,***] 李 丰[**] 李 炼[*,***]

([*]中国科学院计算技术研究所 计算机体系国家重点实验室 北京 100190)

([**]中国科学院信息工程研究所 北京 100193)

([***]中国科学院大学 北京 100190)

摘 要 分布式系统是支撑当前大数据时代各种大数据应用和在线服务的基础平台,分布式系统的质量是大数据应用提供良好服务的基础和前提。伴随着大规模分布式系统的广泛应用,由分布式系统缺陷带来的影响和危害日益严重。但分布式系统在设计、实现和部署方面的复杂性,导致系统的开发和维护人员很难准确地理解和掌握整个系统的行为,难以及时发现系统中存在的故障并进行修复。分布式系统日志涵盖了丰富的信息,是辅助用户理解分布式系统逻辑、剖析系统性能、检测系统异常以及诊断故障原因的重要依据。但复杂的日志结构、庞大的日志规模以及属于不同功能模块、不同用户请求的日志之间相互交错,为人工分析、挖掘日志中的有效信息带来了巨大的困难。本文对近年来针对分布式系统日志的分析和应用技术进行综述:首先总结了分布式系统日志分析与应用的通用流程,提炼出其中的3 个关键步骤,即日志的收集与解析、日志划分、以及日志特征的挖掘与应用;然后针对上述3 个关键步骤,逐一分析需要解决的技术问题,分类阐述目前主流的技术方案,对比技术特征或适用场景。文章还归纳了目前常用的3 类日志特征,并从4 个方面就该领域未来可能的研究方向提出展望。

关键词 分布式系统,日志分析,特征挖掘,异常检测,故障诊断

0 引 言

分布式系统是支撑大数据时代各种大数据应用和在线服务的基础平台,如服务于大规模数据存储的分布式数据库 BigTable[1] 和分布式文件系统 HDFS[2],服务于大规模数据处理的分布式计算框架 MapReduce[3] 和 Spark[4],服务于云计算平台的 OpenStack[5] 等。这些分布式系统的正常运行是在线服务质量的基础和前提。伴随着大规模分布式系统的广泛应用,由分布式系统缺陷(如设计缺陷、实现缺陷、硬件缺陷等)带来的影响和危害日益严重。比如2011 年,亚马逊公司的 EC2 平台的一个缺陷导致集群中的所有存储空间被耗尽,经过两天的修复,仍有0.07% 的用户数据无法复原[6]。此外,据文献[7,8]统计,微软、脸书等公司的分布式系统每年总计的宕机时间约有7.738 小时,由此带来的经济损失高达2.85 亿美元。如何确保分布式系统持续性地提供高质量服务是学术界和工业界广泛关注的问题。

为确保分布式系统持续性地提供高质量的服务,系统维护人员需要熟悉系统的整体构造和局部逻辑,从而能够及时发现系统运行时的故障、快速诊断出故障原因并进行修复。但是分布式系统通常规

① 国家重点研发计划(2017YFB0202002)和国家自然科学基金(61521092)资助项目。

② 男,1991 年生,博士生;研究方向:编译,程序分析,日志分析,分布式系统;联系人,E-mail: lujie@ict.ac.cn

(收稿日期:2018-07-10)

模庞大并且实现逻辑复杂,其行为受运行时系统中不同机器节点的软硬件环境以及所处网络环境的影响,导致系统的开发和维护人员很难准确地理解和掌握整个系统的运行,难以及时发现系统中存在的故障并进行修复。它们通常还会和其他分布式系统交互(例如 HBase[9] 可以运行在 HDFS 之上),导致开发人员很难准确地理解和掌握系统的整体结构以及各个组件间的关联。此外,由于受到自身复杂性和运行环境两方面的影响,分布式系统中的故障传播具有传播链较长、跨节点、跨组件等特点,需要维护人员深刻理解系统中各个组件的逻辑才能准确地诊断故障的根源并进行修复。比如,文献[10]描述的故障是由 6 个节点间的复杂通信导致的,而文献[11]报告的故障是由系统间的交互引起的,根本原因是 MapReduce 任务无法获取 HBase 的一些监控信息。

因此,迫切需要有效手段来辅助用户理解大规模分布式系统、剖析其性能、监测其运行时的状态,从而及时发现故障并进行诊断。目前主流的方法可以分为侵入式(intrusive)和非侵入式(non-intrusive)两类[12]。侵入式方法通过对被分析系统的源代码插桩,在运行时收集能够表征系统中关键执行行为的信息,并维护这些信息之间的时序依赖关系。代表性的技术是端到端跟踪技术[13-15]。侵入式方法需要用户参与选择插桩位置和待收集的信息,且插桩的位置和信息需要随着系统的演进持续更新,因此更适合对系统有较为深入了解的高层次用户。相比之下,基于日志的非侵入式方法,由于能够适应更广泛的用户群体,近年来逐渐得到青睐。非侵入式方法以对分布式系统日志的分析为基础。日志由分布式系统源码中自带的日志打印语句在系统执行过程中输出到日志文件。这些日志打印语句由系统开发人员书写,用于记录系统中的关键事件或状态。日志文件可以辅助开发者和系统维护者了解系统的运行实况,及时发现和诊断系统在测试或实际运行过程中存在的问题,并在必要时借助日志,重新构建出对应的执行轨迹。

为了更好地辅助用户理解分布式系统逻辑、分析其性能,并辅助检测分布式系统运行时的故障、诊断异常的原因,分布式系统的开发者通常会在源码中的关键位置输出充足的日志[16]。第 1 节将通过一个 Hadoop[17] 集群及其日志构成说明分布式系统日志的作用,以及人工分析日志面临的困难。

1 背 景

图 1 所示的 6 节点集群上运行了 Hadoop 的分布式资源管理服务、计算服务以及文件系统服务,分别由 Yarn、MapReduce 和 HDFS 三个分布式系统提供。Yarn 采用主从结构,运行在 Node1 节点上的 ResourceManger(RM)组件负责管理和分配集群资源(资源包括计算、内存、带宽等,资源分配的单位是 container);运行在其他节点上的 Nodemanger(NM)组件负责监控对应节点上的资源使用情况并汇报给 RM。MapReduce 运行在 Yarn 之上(Yarn 还支持 Spark 等其他分布式计算框架),主要包含 Map 和 Reduce 两个功能模块。HDFS 也采用主从结构,由运行在 Node6 上的 NameNode(NN)组件负责管理文件命名空间。以上组件都属于集群的后台服务进程,需要在运行用户作业前启动。

图 1 中的箭头代表了由不同用户提交的 2 个作业请求的执行过程。以黑色较粗箭头所示的用户请求 1 为例,用户通过客户端提交作业请求后,RM 首先为它创建并启动一个专门用于管理该作业所需资源的进程 ApplicationMaster(AM)。图 1 中,用户请求 1 的管理进程 AM1 启动在 Node2 上。对于 MapReduce 作业,AM 将根据作业需要处理的数据规模创建一定数量的 Map 和 Reduce 任务(这些任务对应 Hadoop 中 Map 或 Reduce 功能模块的一次执行),并向 RM 申请资源;获取资源后,在资源所在的 NM 上分别启动这些任务。比如,图 1 中,RM 分配给 AM1 的资源分别位于 Node4 和 Node5 上,AM1 在这 2 个节点上分别启动了一个进程运行对应的 Map 或 Reduce 任务。任务执行过程中将访问 HDFS 上的数据,或将结果写入 HDFS 图 1 方框中的文本即上述用户请求在执行过程中产生的日志。可以看到,这些日志分散地保存在不同节点的不同日志文件中。日志文件又分为由后台服务进程创建的日志文件

(比如图 1 中的 YARN-RM. log、Namemanger5. log)和作业本身的执行日志(比如图 1 中的 AM1. log、container1 _3. log)两类。如果日志分析的目标是理解或重构出用户请求的执行行为,就必须首先将这些分散在不同节点上的不同日志文件中隶属相同用户请求的日志提取出来。比如图中日志文件中黑体加粗的文本都是隶属用户请求 1 的日志。

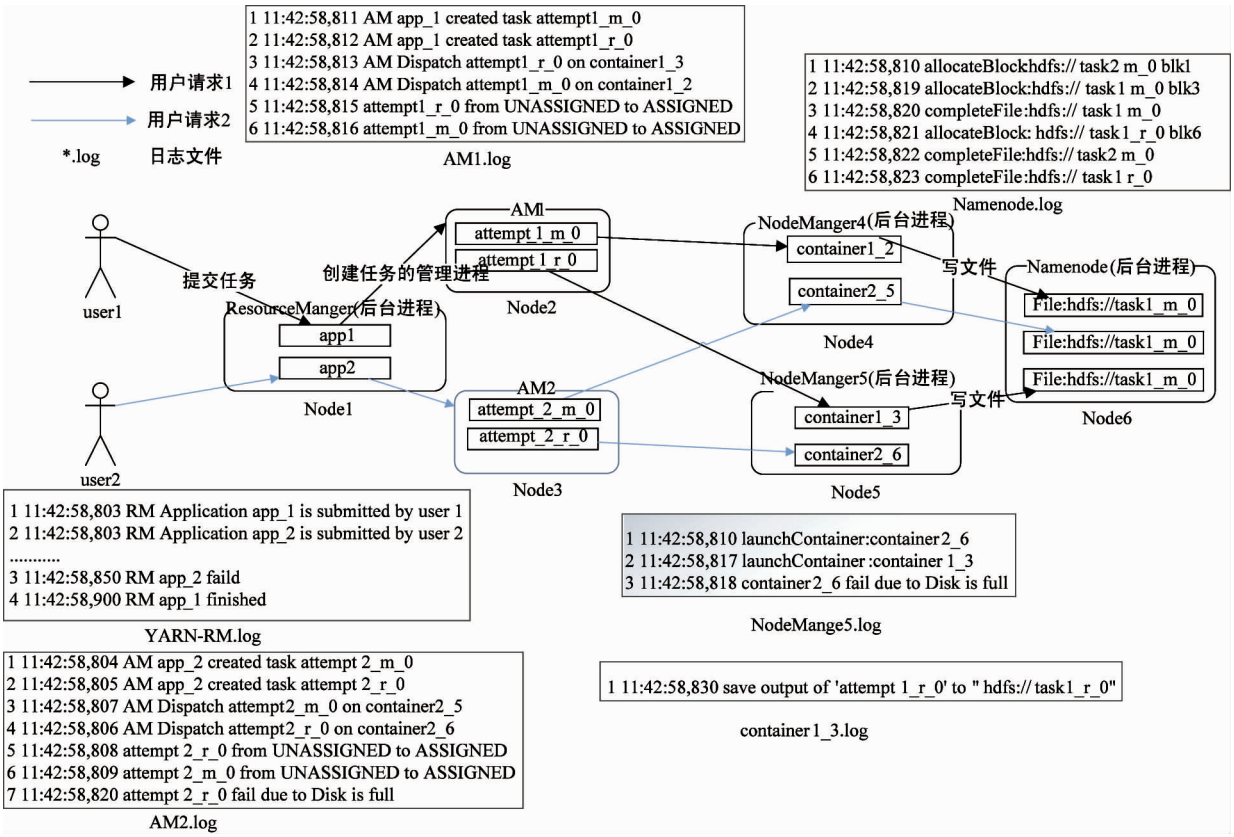

图 1　Hadoop 的用户请求执行图

日志中蕴含的信息不仅可以用来识别和区分用户请求,而且可以辅助理解分布式系统的执行行为、性能分布,发现运行过程中的功能或性能异常并提供诊断提示。仍以图 1 为例,变量值 app _1 和 app _2 可以区分不同的用户请求。对于隶属相同用户请求日志,可以进一步根据时间戳确定日志条目所代表的事件发生的先后序或时间差,对比和挖掘其中执行顺序或间隔时间有异常的事件(日志)序列。也可以通过对比一些特定的变量的值分布情况来识别异常。除了根据用户请求,也可以根据节点名将属于同一个节点的日志聚集在一起,通过分析和对比每个节点的负载、响应时间等,及时发现可能成为性能瓶颈的节点(culprit node[18])。

分布式系统日志虽然携带了丰富的信息,但系统每天需要处理大量的用户请求,产生海量的日志。这些日志大部分是如图 1 所示的非结构化日志。除了规模庞大、格式复杂之外,隶属不同功能模块、不同用户请求的日志相互交错,为人工分析和挖掘日志中的有效信息,带来了巨大的困难。针对这些问题,出现了各种自动分析日志并挖掘和应用日志特征的技术。

本文对近年来利用日志分析辅助用户理解分布式系统逻辑、剖析系统性能、检测异常以及诊断故障的技术进行综述。第 2 节基于对现有日志分析和应用技术的归纳,总结出日志分析与应用的通用流程,并提炼出其中的 3 个关键步骤:日志的收集与解析、日志划分、以及日志特征的挖掘与应用。第 3、4、5 节分别就上述 3 个关键步骤中存在的技术问题以及

解决方法进行归纳、总结和对比。其中,第 3 节介绍了日志收集的基本原理,阐述并对比了两类主流的日志解析方法的优缺点和适用场合。第 4 节将描述日志划分阶段采取的技术方案,并提炼出各个方案需要解决的关键问题。第 5 节将现有技术挖掘和使用的日志特征归纳成 3 大类,并就各类日志特征在分布式系统理解、剖析、检测和诊断方面的应用进行分类阐述,总结其中存在的问题。基于对 3 个关键步骤中主流技术的对比和总结,第 6 节就该领域面临的问题以及可能的研究方向进行了展望。

2 日志分析及应用的基本流程

本节基于近年来的代表性工作,总结了分布式系统日志分析和应用的基本流程(如图 2 所示)。流程包括 3 个主要步骤:日志的收集与解析、日志划分、日志特征的挖掘与应用。首先,收集系统中各个节点产生的日志,从中解析出各类信息,再结合不同应用需求,选取适当的信息维度对日志进行划分,并在划分得到的日志集合中进行针对性的特征挖掘,最后将挖掘结果应用于系统理解(system understanding)、性能剖析(performance profiling)、异常检测(anomaly detection)、故障诊断(problem diagnosis)等不同实际需求中。接下来将依次介绍各个步骤的作用和原理。

日志收集的目的是将分布式系统各个节点所产生的日志收集到一起。分布式系统执行过程中,各个节点产生的日志通常默认存储在该节点的本地文

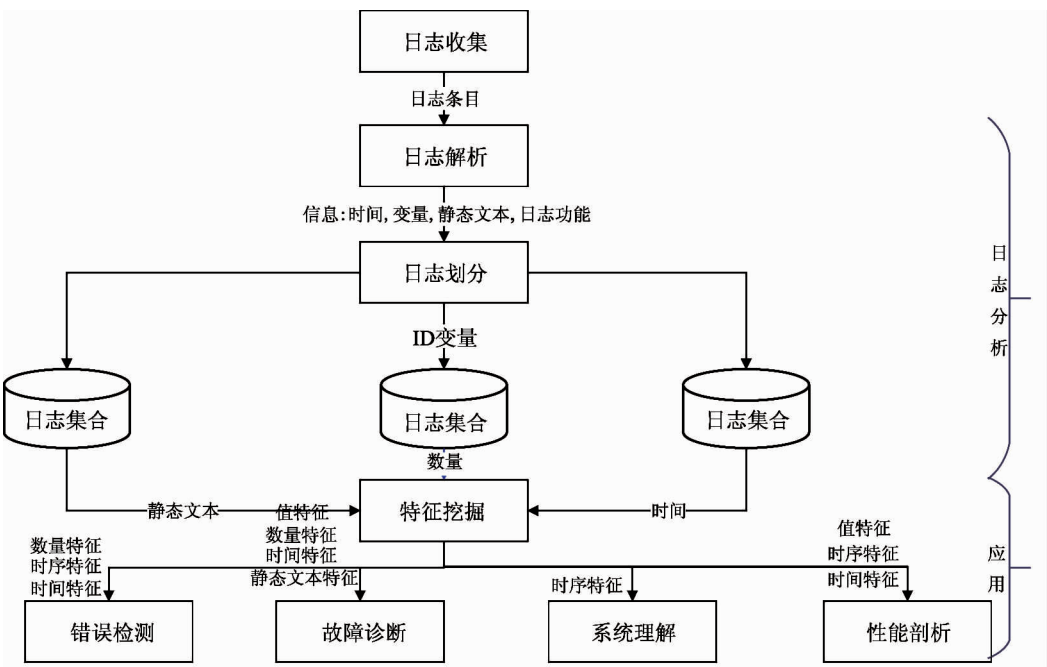

图 2 日志分析的整体流程

件系统中(比如图 1 中的 AM1. log、Namenode. log 分别默认存储在 Node2 和 Node6 上)。但由于分布式系统提供的服务往往由运行在不同节点上的服务组件协同完成(比如图 1 中的 AM1 和 container1 _3 共同完成 Reduce 模块的一次执行,产生的日志分散地存储在 Node2 和 Node5 上),只有将分散在这些节点上的日志汇集在一起,才有可能掌握或重构出该功能的执行逻辑。日志收集既可以手动完成,也可以借助在线日志收集工具自动收集。部分日志收集工具还附带按照日志等级对日志分类、用正则表达式对日志进行过滤等功能。代表性的日志收集工具有 LogStash 等[19-26],3. 1 节将介绍其中的关键技术。

日志解析的目的是从收集到的日志中提取信息。日志解析既可以与日志的在线收集同步进行,也可以单独离线进行。日志解析技术重点关注的是

如图 1 所示的非结构化日志。这些日志没有统一的格式。目前用于解析非结构化日志的方法大致可以分为两类。一类是基于对源码或二进制文件的分析,自动推导出各条日志打印语句的文法,表示成正则表达式的形式,再通过正则表达式匹配识别日志中包含的信息。另一类方法只分析日志文件本身。它们使用聚类算法将相似的(即可能由相同日志打印语句产生的)日志聚集在一起。每个聚类集合中的日志,公共部分通常对应日志中的静态文本,剩余部分对应日志中的变量值。公共文本通常也被用来代表(区分)系统中不同的事件或者不同的日志类型。3.2 节将介绍这两类方法的技术细节以及存在的问题。

日志划分是指从日志解析得到的信息中选择某个或某几个信息维度将日志聚集在一起,得到一系列日志集合。其中每个日志集合包含的日志均存在某一方面的关联性。以图 1 中的日志为例,如果用户想要了解被标记为 attempt1 _ r _ 0 的 Reduce 任务的执行情况,可以将图中所有包含变量值 attempt1 _ r _ 0 的日志归入同一个日志集合。通过日志划分步骤,可以过滤与用户需求无关的日志,将关联性较大的日志聚集在一起,减少待分析的日志规模,降低无关信息对后续分析过程的干扰,提高日志特征挖掘的效率与精度。第 4 节将详细介绍日志划分技术。

挖掘日志特征的最终目的是服务于应用。目前的研究工作关注的应用大致可以分为以下 4 类:系统理解、性能剖析、异常检测和故障诊断。不同的应用挖掘和使用的日志特征不尽相同,也各有侧重。常见的日志特征有:时序特征[27-29]、数量特征[16,30,31]、时间特征[12,32]等(见第 5 节)。以图 1 为例,如果日志划分阶段将包含变量值 attempt1 _ r _ 0 的日志归入同一个日志集合,在特征挖掘阶段,就可以通过挖掘集合内日志之间的因果关系(实际应用中常以时序关系代替因果关系),为 attempt1 _ r _ 0 代表的 Reduce 任务构造执行流(request flow),再从执行流中提取诸如关键路径等信息服务于性能剖析,辅助用户发现性能瓶颈。如果特征挖掘的目的是辅助用户理解系统行为,则特征挖掘阶段更倾向于分析和提取在各个日志集合内都成立的时序特征。比如,图 1 中包含变量值 attempt1 _ m _ 0 和 attempt2 _ m _ 0 的日志分别对应 Map 模块的两个执行实例,即两个 Map 任务(这两个任务分别隶属用户请求 1 和用户请求 2)。假设日志划分阶段将包含变量值 attempt1 _ m _ 0 和 attempt2 _ m _ 0 的日志分别归入两个不同的日志。

"AM created task" →"AM Dispatch on"　(1)

集合,则这两个集合都满足表达式(1)所示的时序不变式。表达式(1)以日志中的静态文本代表日志类型,以"→"代表 happened-before 关系[33],表达的含义是:只有当一个任务被创建后,系统才能为它分配资源。如果还存在另外一个 Map 任务的日志集合,从中挖掘出的时序违反了表达式(1),则该集合中的日志对应的程序行为将被报告为异常或突变(mutation)。在发现包含异常的日志集合后,故障诊断应用可以进一步分析和对比正常集合和异常集合之间的日志特征差异,为用户提供更多更详细的有关异常原因的线索。第 5 节将详细介绍对日志特征的挖掘及应用。

3　日志的收集和解析

如前所述,日志的收集与解析既可以在线进行,也可以离线完成。在线收集和解析能够尽早发现系统中可能存在的功能或性能异常、诊断它们的原因,但缺点是可能占据系统资源,影响系统正常对外提供服务。离线收集与解析虽然不影响系统性能和资源,但在异常发现和故障诊断方面存在滞后性。本文将着重描述目前比较流行的在线方法。

3.1　日志的在线收集

为了应对在线收集分布式系统日志所面临的问题,许多公司以及开源组织纷纷开发出了具有高可用性、高可靠性和高可扩展性的日志收集平台[19,24-32]。这些平台自身也是分布式部署,分为代理层、接收层和存储层。为方便管理,日志分析平台通常选用主-从式结构,即使用一个主节点(master)统一管理所有的发送/接收节点。代理层由部署在被监控系统各个节点上的代理进程构成。代理进程

负责监控日志文件或者指定的数据流,将新收集到的数据缓存在本地,并实时发往接收层。仅当接收层收到数据后,代理进程才将对应的数据从缓冲区删除。为了减轻带宽压力,有些日志系统支持用户书写正则表达式过滤无用的日志,或者只传输日志中被认为是关键的数据。比如 Stitch[32] 不只传输变量值和时间戳,还对被传输的数据进行压缩,以进一步降低对被监控系统带宽的占用。

接收层开启多个接收进程接收数据,并使用负载均衡工具(如 zookeeper[34])均衡接收节点的负载。接收进程数量随代理进程数量的增加而增加,但前者的数量应小于后者。日志收集平台会监控每个代理进程和接收进程的存活情况,如果发现有代理进程死掉,立刻重启。如果一个接收程序死掉,代理进程会将数据发往另一个接收进程。

接收层将收到的数据存储在存储层。存储层通常选择扩展性较好的数据库或者文件系统,比如 HDFS。如果数据库或文件系统宕机,接收端会将数据存储在本地磁盘,并在数据库或文件系统恢复工作后重新传输。

为了方便用户对收集到的日志进行更系统地解析、分析和挖掘,日志收集平台端通常还会提供查询接口和可视化界面。用户既可以使用人工书写正则表达式或 SQL 查询语句的方式手工查找所关心的日志,也可以集成 MapReduce 等分布式计算框架对所收集的日志进行实时自动化地分析处理[22]。

3.2 日志的解析

如前所述,日志解析的目的是提取日志中包含的信息(如图 3 所示),作为后续日志划分和特征挖掘的输入。日志解析的重点是对非结构化日志的处理。目前自动日志解析技术可以分为基于文法的方法和基于聚类的方法两类。

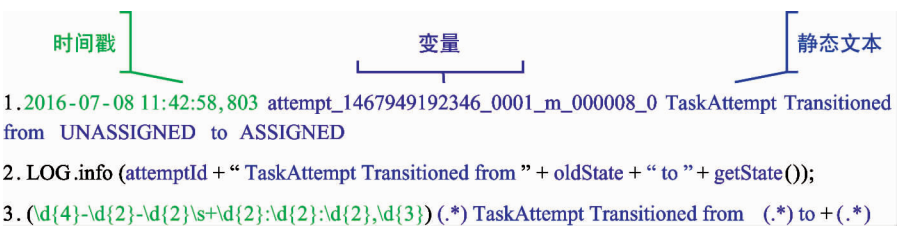

图 3 日志解析

基于文法的日志解析方法 此类方法将日志打印语句的文法表示成正则表达式的形式(图 3 第 3 行),再用正则表达式匹配日志,识别其中包含的时间戳、静态文本、变量值等信息(图 3 第 1 行)。

部分研究工作,如 CSight[27]、Ivan[35] 和 Logstash[20],在解析日志时,要求用户提供正则表达式。这类方法一方面要求用户具备一定的专家知识,另一方面不容易推广。

Xu[16]、lprof[12]、ELT[36] 基于源码分析日志打印语句的文法,形成正则表达式。由于目前大部分分布式系统都使用通用的日志库(如:Log4j[37]、Java Commom Log)函数接口打印日志,通过在源码或中间代码中搜索这些日志库接口函数(如:Log4j 的 fatal、error、warn、info、debug、trace)的调用点,就可以识别出日志打印语句。在解析日志打印语句的文法构成、推导正则表达式的过程中,ELT[36] 将其中被双引号包裹的字符串识别成静态文本,而将剩余的部分识别成变量。Xu[16] 则根据抽象语法树上所记录的变量类型区分日志打印语句中的变量和静态文本。为了更精确地识别面向对象程序中的变量类型信息,Xu 递归地解析日志打印语句中出现的变量类型及其子类的 toString 函数,直到所分析的类型为基本类型。通过这个方法,为每个日志打印语句生成了所有可能用来匹配日志的正则表达式。在匹配日志条目时,根据最长匹配原则,选出匹配度最高的正则表达式。lprof[12] 在解析日志时采取了和 Xu 相同的方法。

与上述基于源码分析日志打印语句文法并形成正则表达式的工作不同,Stitch[32] 仅分析程序的可执行文件。它首先通过 linux 系统/proc 目录下的文件找出正在运行的进程,再根据每个进程的文件描述符找到进程打开的文件,目的是找出打开了名称

以 log 结尾的文件的进程,然后在该进程的可执行文件(比如:java bytecode、python bytecode 或 ELF 文件)中找出所有的静态文本,使用这些静态文本和动态规划算法最大化地匹配每条日志,最后根据匹配结果,将日志中未匹配的部分识别成变量值,再根据静态文本和变量值生成正则表达式。

基于聚类的日志解析方法 此类方法通常不需要分析源代码,而是基于聚类算法,将相似的(即可能由相同日志打印语句产生的)日志聚集在一起。目前的工作多使用编辑距离及其变种作为度量日志间相似度的基础。理想的聚类结果是每个聚类集合中的日志都由同一条日志打印语句产生;对于每个聚类集合,日志间公共的部分对应日志中的静态文本,剩余部分对应日志包含的变量值。对于此类技术,设计合理的日志相似度度量公式是决定解析效率和精度的关键。其中,Jiang[38] 和 Fu[39] 为了提高聚类精度,在聚类前先根据经验规则识别日志中的变量值。Jiang 识别变量值的目的是在聚类时同时考虑了静态文本和变量数量两方面的相似度,Fu 的目的则是排除变量值对相似度计算的干扰。但 Jiang 的经验规则过于单一,并不适合图 1 所示的日志;Fu 也只支持对常见变量值(如 IP 地址、URL、数字等)的识别,如果处理对象是图 1 中的日志,则需要用户手工补充规则。LogSig[40] 和 LogTree[41] 不需要提前识别日志中的变量值,LogSig 提出利用单词之间的序来提高聚类算法的精度;LogTree 则要求由用户为被分析系统提供日志的上下文无关文法,作为分析的基础。上述工作对日志的聚类都是离线进行的,Barash[42] 提出了一种线上聚类的日志解析技术。

目前,基于文法的日志解析技术在解析精度上优于基于聚类的技术,对于源码可见或二进制码未经加固的分布式系统的日志解析方面更具优势;但基于聚类的方法对于使用多种语言编写的分布式系统,以及源码不可见或二进制文件经过加固的商用分布式系统,具有更强的适应性。此外,基于文法的方法更适合于在线日志解析,且可以与日志收集同时进行,基于聚类的方法则通常是在日志收集完毕后离线进行的。

4　日志划分

分布式系统每天产生大量的日志。这些日志虽然包含丰富的信息,但并非所有信息都与应用需求相关。且由于不同用户请求的日志混杂在一起,人工分析很难高效的识别出其中有价值的日志特征。为了更好地服务于日志特征的挖掘与应用,目前日志分析工作在挖掘日志特征之前,通常会对日志进行划分,将属于同一用户或时间段的日志聚集在一起(归入同一日志集合),提高后续分析的效率与精度。

4.1　基于用户视角划分

基于用户视角划分是为了将同一用户请求产生的日志聚集在一起,这样后续的工作可以针对某个用户某次请求进行具体的分析。目前基于用户视角的划分方法首先会找到可以代表用户某次请求的变量,称之为 ID 变量,然后依据变量的值划分日志。

Mystery Machine[28] 使用 Facebook 内部的插桩工具为每个用户请求赋予全局唯一的 ID,并对已有的日志打印语句进行改写,使输出的日志总是包含这个 ID,这样不仅可以区分不同的用户请求,又可以将不同分布式系统产生的日志聚集在一起。SALSA[43] 和 Tan[44] 基于对 HADOOP[17] 系统的理解,人为指定系统中的哪些变量可以作为区分不同用户请求的 ID,并进一步指定哪些类型的日志属于同一个用户请求。

与需要人工辅助的方法不同,Stitch[32] 基于流重构原则(flow reconstruction principle)聚集同一用户请求的日志。流重构原则是指开发者为了能够根据日志重构出用户请求在系统中的执行路径,会在日志中打印足以区分不同请求、不同模块乃至不同执行实例的一系列 ID 变量。比如,如果能够从日志中识别出这一系列 ID 的变量以及它们之间的对应关系,就可以区分属于不同用户请求的日志,并将隶属该请求但跨越了不同系统的日志关联起来。考虑到不同的分布式系统可能使用不同的编程语言书写,且可能存在外用库源码不可得的情况,使用以下 2 条启发式规则筛选 ID 变量:(1)排除所有非名词

类型的变量。(2)排除所有前后出现了某些特定名词(如 size、usage、progress)的变量。在筛选出 ID 变量后,Stitch 根据变量类型判断两条日志是否打印了相同的变量。变量的类型由变量值中包含的静态文本决定。比如,图 1 中 attempt2 _ m _ 0 和 attempt2 _ r _ 0 这两个变量值包含的静态文本分别是"attempt _ m _"和"attempt _ r _",因此,被认为是代表不同的 ID 变量(事实上,二者分别对应 Map 任务的 ID 和 Reduce 任务的 ID)。Stitch 还进一步根据打印了多个 ID 变量的日志,推导日志之间的对应关系。比如,根据图 1 中 AM1. log 的前两条日志,推导出"app _"代表的 ID 变量与"attempt _ m/r _"代表的 ID 变量之间的 1 对多关系,从而将包含"app _ 1"和"attempt1 _ m/r _"的日志划入同一个日志集合(即隶属用户请求 1 的日志集合)。同理,根据 AM1. log 中的 3、4 两条日志,可以进一步将包含了"container1 _"的日志归入这个集合。

lprof[12] 基于对分布式系统源码的静态分析,在识别 ID 变量的过程中,找到可以作为一个相对独立的功能模块入口的顶层方法(top method)。lprof 采用基于概要[45] 的变量全局使用/修改分析识别 ID 变量和顶层方法。该技术为函数调用图中的每个函数维护两个集合 ID 变量集合(request identifier candidate, RIC)和被修改变量集合(modified variable set, MV)。分析过程中,沿被调用路径自底向上的统计每个函数中被日志打印的变量(加入 RIC)、打印次数以及被修改的变量(加入 MV 并从 RIC 删除),并合并对被调用函数的统计结果(累加各个变量被打印的次数,合并 MV,并将 MV 中的元素从当前函数的 RIC 中删除)。一旦出现合并后 RIC 中元素被打印的次数骤降的情况,则将被调用函数识别成顶层方法,将顶层方法 RIC 中的元素标记成 ID 变量。由于同一个功能模块输出的日志并不总是包含了作为 ID 的变量,为了将这些日志划分到正确的日志集合中,lprof 通过对以顶层方法为顶点的函数调用子图以及子图中各函数的控制流的分析,构造出日志打印语句间的控制关系图。lprof 还进一步借助对进程间通信代码以及节点间通信代码特征的识别,合并 ID 变量不冲突的日志集合。但是与前面的工作不同,由于其使用静态分析方法,lprof 只能划分单一系统的日志,不能将分散在不同系统的用户日志聚合在一起。

4.2 基于时间视角的划分

由于程序的执行具有时间局部性,出现在一定时间窗口内的日志之间存在关联性的概率高于这些日志与窗口之外的日志间的关联性。基于时间视角的日志划分要解决的主要问题是对时间窗口规模的选择。时间窗口的大小决定了划分得到的日志集合的规模。窗口越大,日志集合的规模也越大,日志分析的效率和精度也相应降低[46]。但是,如果时间窗口设置的过小,漏掉有用日志的概率就会提高。以图 1 中 AM2. log 为例,如果将日志 1 到 5 划归一个时间窗口,那么日志 3 和 6 之间的时序关系就会被遗漏,由此得到的不完整的日志时序关系可能会影响用户对系统逻辑的理解,甚至可能在后续的应用中影响异常检测或故障诊断的精度。

LogMaster[46] 将初始的时间窗口大小设定为 1 小时,然后在分析过程中持续地精化和调整窗口大小,以权衡分析精度和效率。为了避免出现因窗口规模过小而遗漏了有效日志的情况,LogMaster 还设计了滑动的时间窗口来优化对日志的划分结果,即每完成对一个时间窗口内的日志的分析后,将时间窗口向前滑动一段距离,使前后两个窗口部分重叠,对重叠部分的分析就有可能识别出一些原先没有被完整覆盖的日志对或日志序列之间的关联。

Xu[16] 选择时间窗口的依据是确保每个时间窗口中打印了状态变量的日志的出现次数不少于 10 次,这样既可以保证计数统计的有效性[47],又能够将时间窗口维持在一个尽可能小的规模。

Lim[48]、GAUL[49] 在确定时间窗口大小时结合了应用需求。GAUL 面向的应用需求是识别分布式系统中的故障是否是之前已经出现过的(recurring problems)。为了确保划分得到的日志集合能够代表已发生的故障的特点,GAUL 将故障发生前 3 天的日志收集在一起。用户也可以自己配置时间窗口大小。Lim 的应用需求是错误/故障的预测。为了找到能够代表错误特点的日志集合,Lim 将错误发生前后的日志收集在一起。默认的时间窗口的大小

是4小时,包括错误发生前3小时以及错误发生后1小时。用户同样可以配置时间窗口的大小。

5　日志特征的挖掘与应用

如第2节所述,不同应用关注的日志特征以及对这些特征的使用方法各不相同。目前的研究工作关注的应用主要分为系统理解、性能剖析、异常检测和故障诊断,也有部分工作关注错误预测等其他应用。本节将目前研究工作中挖掘的日志特征归纳成5类:时序特征[27-29]、数量特征[16]、时间特征[12,32]。在特征使用上,有的应用,如系统理解、性能剖析,更倾向于关注各个日志集合之间的共性,以期从中找到系统固有的一些性质,如时序逻辑、关键路径等,而其他应用,如异常检测、故障诊断,则更关注存在明显差异的特征。本节将分别介绍各项研究工作如何根据应用需求对所关注的日志特征进行抽象、挖掘以及使用。

5.1　数量特征的挖掘与应用

对日志数量特征的挖掘常用于异常检测和故障诊断。这是因为日志的数量可以在一定程度上反映系统中某个用户请求的执行行为。本小节将介绍利用日志数量特征检测分布式系统异常或诊断异常原因的技术。

5.1.1　异常检测

Xu[16]使用ID变量划分日志,然后利用日志向量描述一个日志集合中的日志数量特征。日志向量的长度是日志类型的个数,日志向量的每个元素是日志集合中每个日志类型(用日志中包含的静态文本表示)的日志数量。以图1为例,如果以功能视角对日志进行划分,以日志中的静态文本代表日志类型,则以形如"attempt[1-9][0-9]*_r/m_[1-9][0-9]*"的ID变量值划分得到的日志集合最多包含5种日志类型:"AM created task"、"AM Dispatch On"、" from to"、"fail due to Disk is full"和"save output of"。因此,从这些集合中提取的日志向量长度是5。以ID变量值为attempt1_m_0的日志集合为例,对应的日志向量是[1,1,1,0,0]。Xu[16]将所有的日志向量组合在一起形成向量矩阵,然后找出在日志的数量构成上明显有异于其他日志集合的异常集合,并将不同类型日志的数量与异常集合之间的关系以决策树的形式返回给用户,用户可以根据决策树人工分析导致异常的日志和代码。

Lou[30]也使用日志向量描述日志集合中日志的数量特征。但与Xu的方法不同,Lou采用监督学习的方法获得正常执行时向量矩阵应满足的不变式:如果将向量矩阵记作$\boldsymbol{X}$,则可以令等式$\boldsymbol{X}\times Y=0$成立的系数Y就是正常执行时应满足的不变式。如果其他某个日志集合对应的日志向量违反了这个不变式,该日志集合对应的执行行为就被认为是异常的。和Xu[16]的非监督学习方法相比,Lou的优势在于能够处理待检测的日志集合中包含大量异常集合的情况。对于这种情况,Xu的方法可能会出现大量漏报,甚至误报。但是监督学习要求训练集足够全面,否则也会出现误报。

ELT[36]对日志向量进行聚类,在将规模较小的聚类集合识别为异常后,为规模较大的聚类集合中的每个日志集合构造一个MFG(message flow graph)图。MFG图描述了不同类型日志之间的转移关系。图中的顶点代表日志类型,时间相邻的两个日志类型之间有一条边。ELT使用图编辑距离描述两个MFG之间的差异,计算每个MFG和其他MFG之间的最短距离。如果某个MFG的最短距离大于指定阈值,则将其标记为异常。针对不同分布式系统的实验结果显示,由于MFG的引入,相对于Xu[16],ELT的检测精度最高提高了80%(95% vs 15%)。

5.1.2　故障诊断

故障诊断是指在已知系统执行出现异常(包括功能异常和性能异常)后,分析正常执行和异常执行之间的差异,找到可能有助于用户诊断故障原因的信息或提示。本小节将介绍目前研究工作中根据正常执行和异常执行之间的日志数量特征差异来辅助用户诊断异常原因的技术。

由于大规模分布式系统及应用的开发者通常会先在伪分布式集群或者小规模集群上对新开发的系统或应用进行测试,测试正确后才部署到大集群上[31],因此,分析测试环境下正确测试用例生成的日志的数量特征和生产环境下的日志数量特征之间

的差异,有可能获得有助于诊断异常原因的信息。Shang[50] 和 LogCluster[31] 的方法都采用了这个思路。

Shang[50] 使用 Jiang[38] 的方法分别解析从测试环境和生产环境中收集到的日志,再根据用户指定的日志变量(ID 变量)将上述日志分别划分成多个集合;在提取日志向量时,不区分日志集合中同类日志的多次出现以及日志间的顺序;最后将生产环境下日志向量聚类结果与测试环境下日志向量聚类结果之间的差集作为关键诊断信息报告给用户。

LogCluster[31] 认为不同日志在故障诊断中的重要性不同,比如在所有日志集合中都出现的日志对故障原因的区分度低于只在部分日志集合中出现的日志,生产环境下产生的日志对故障原因的区分度高于测试环境下产生的日志。为了描述日志对故障原因的区分度,LogCluster 在提取日志向量时,为每个日志赋予一个权重,再分别对测试环境和生产环境下的日志向量进行聚类,并从聚类结果中选取代表性的日志向量,将生产环境与测试环境下代表性日志向量的差集反馈给用户,作为诊断错误的依据。LogCluster 还可以自动诊断重复发生的故障(recurring problem),对于每个新生成的差集,如果已经保存了一个对应的历史诊断策略,则将该策略直接反馈给开发者,否则,保存本次人工诊断的结果。实验表明,开发者使用 LogCluster 辅助诊断时需要查看的日志数量,比使用 Shang[50] 的方法减少 24.38%。

ELT[36] 在识别出异常的聚类集合或日志集合后(见 5.1.1 节),进一步分析这些异常是否由相同的原因导致,并提供可以辅助诊断故障原因的关键日志。分析的方法是先找出每个异常 MFG 图中未在正常集合的 MFG 图中出现的边,表示成日志向量的形式(日志向量的元素是边的起点和终点),然后对这些日志向量进行聚类,得到的每个聚类集合对应一个异常类型,每个聚类集合中被所有日志向量共享的日志类型将作为诊断该异常类型原因的关键日志返回给用户。

5.2 时序特征的挖掘与应用

日志的时序特征侧重于日志所代表的事件或状态之间的关联性(correlation)、偏序关系(partial)、时序依赖关系(temporal depenendce)等。基于日志间时序特征构造的有限状态自动机或工作流图(workflow)既可以帮助用户理解系统的实现逻辑,也可以帮助用户检查系统的实现是否与设计相符。在所有日志集合中都成立的时序特征通常被看做被分析系统固有的时序逻辑,可以辅助用户理解系统行为、剖析系统性能。如果某个日志集合中的日志时序违反了上述固有逻辑,则该集合对应的执行行为可能包含异常。本节将分别介绍日志时序特征的抽象表示形式、挖掘策略以及在分布式系统理解、异常检测和性能剖析方面的应用。

由于分布式系统不同节点的时间可能不同步,日志中包含的时间戳未必能准确地反映不同节点上日志间的时序关系。目前常用的确定日志时序关系的方法有两种:一种是使用向量时间戳[51],比如 CSight[27] 为每个日志添加向量时间戳;另外一种是通过时钟同步[52] 算法使各个节点上时钟保持一致,比如 Mystery Machine[28]。

5.2.1 系统理解

CSight[27] 基于从日志中挖掘出的 5 类时序不变式,为被分析的分布式系统构建自动机,辅助用户理解系统的整体逻辑以及状态间的迁移关系。在使用向量时间戳确定了各个日志集合内部不同日志间的时序后,CSight 复用 Ivan[53] 提出的时序不变式挖掘技术挖掘不同日志类型之间满足的 5 类时序关系。如果用 a、b 代表 2 个不同的日志类型,这 5 类日志时序关系可以表示成:a 之后总跟着 b,a 总是在 b 之前,a 之后从不跟着 b,a 和 b 总是并发,a 和 b 从不并发。为了构造能够刻画整个分布式系统执行逻辑的状态机,CSight 先为每个日志集合构造一个有限状态机,状态机上的边代表日志类型,节点代表某个类型的日志执行前后对应的状态;然后在不违反时序不变式的前提下将这些状态机合并成一个抽象状态机,并进一步合并抽象状态机内部的状态。

Lou[29] 认为之前的研究工作在挖掘时序特征时,并没有考虑分布式系统中不同线程或进程间日志交错的情况。针对这个问题,Lou 定义并挖掘日志(或日志对应的事件)之间的 4 种时序依赖关系。前向依赖:一个或多个事件 A 发生,最终会导致事件 B 的发生;后向依赖:一个或多个事件 B 发生,事

件 A 必然已经发生;严格前向依赖:每次事件 A 发生,至少有一个事件 B 随后发生;严格后向依赖:每次事件 B 发生,事件 A 都已经发生。根据以上 4 类关系,Lou 为被分析的分布式系统构建一个工作流图,图中的顶点代表日志类型,边代表以上任意一种时序关系。如果图中两个顶点之间的边可以通过传递得到,则删除这条边。

Fu[39]在根据用户指定的 ID 变量完成对日志的划分后,按照日志时间戳将各个日志集合转化成日志序列,从日志序列中提取有限状态机。状态机的节点代表日志类型。对于首次读入的日志序列,如果两个日志类型对应的日志条目在日志序列中相邻,则在代表这两个日志类型的节点之间建一条迁移边,得到初始的状态机。行为获取和测试(behavior capture and test, BCT)将后续读入的日志序列与初始的状态机匹配,合并匹配的状态,对于不匹配的状态,在原始状态机上加入新的状态以及迁移边。

Lou[54]认为先前的工作只关注了分布式系统中一个组件内部的依赖关系,没有考虑来自不同组件的日志之间的关系。Lou[54]通过对 Hadoop 系统日志的分析,总结了存在依赖关系的日志所具有的特征,即发生在较短时间间隔(如 3 秒)内的日志以及打印了相同变量值的日志之间存在依赖关系的可能性较大。根据这个发现,Lou[54]先将各个组件的日志单独收集起来,然后在来自不同组件的日志之间挖掘满足上述特征的日志。实验表明,该方法可以识别出 MapReduce 和 HDFS 日志之间的依赖关系。

CloudSeer[55]先假定任意两个日志类型之间都存在 happened-before[33]关系,形成 happened-before 关系图。然后,根据每个日志集合内的各个日志实例的时间戳,删除一定不存在的 happened-before 边,进一步删除可以通过传递关系推出的边,最终得到工作流图。

SALSA[43]只将人工指定的 Hadoop 中比较关键的日志作为有限状态机的顶点,然后分别挖掘状态间的控制流和数据流两类关系,得到两类独立的有限状态机。挖掘过程也需要专家知识辅助。得益于后续的 Hadoop 版本对日志质量的改进,Tan[44]利用打印了多个(人工指定的)ID 变量的日志,将 SALSA 中的两类有限状态机合并成一个有限状态机。

5.2.2　异常检测

CloudSeer[55]使用离线构造出的工作流图在线分析 OpenStack[5]的运行日志,实时检测系统中的异常。检测方法如下:CloudSeer 为每个执行流维护一个工作流图的实例图,不同执行流的实例图之间通过 ID 变量值区分;每当一个新日志产生时,根据日志中的 ID 变量值找到其所归属的执行流,并在对应的工作流实例图上执行状态迁移,如果触发状态迁移需要的日志在一定时间内没有产生或者产生的日志与所需的日志不匹配,则认为对应的执行流存在异常。

5.2.3　性能剖析

Mystery Machine[28]在为被分析系统构造出工作流图后,对于隶属同一用户请求的日志构造出的请求执行流,基于工作流图识别其中的关键路径,辅助用户剖析性能分布、发现性能瓶颈。

Tan[44]构造的有限状态机的节点代表 Hadoop 在执行用户请求时的关键阶段,比如 Map 阶段、Reduce 阶段、数据块的读写等。Tan 基于有限状态机统计了每个关键阶段的耗时,以及 Map 阶段和 Reduce 阶段并行执行的情况,从而确定其中导致任务阻塞的阶段。

5.3　时间特征的挖掘与应用

日志的时间特征包括日志间的时间间隔以及执行时间的分布。与时序特征相比,目前的工作主要利用日志的时间间隔检测性能异常[12,18,32,43,44],并不关系两个事件之间是否具有先后序。分布式系统中的一个功能模块在正常执行时的耗时通常是趋于稳定的。如果某次执行耗时出现较大的波动,说明执行过程中可能有异常。同理,配置相似的节点在处理相同任务时的耗时也是趋于稳定的,如果某个节点和其他节点耗时相比出现较大波动,则这个节点出现问题的几率很大。

5.3.1　异常检测

利用日志的时间特征既可以检测异常的执行实例,也能够发现系统中有问题的节点。

Fu[39]基于工作流图检测以下两种性能异常:一种是实际执行时两条日志间的时间间隔不满足训练

得到的工作流图上(正常的)时间间隔的正态分布;另一种是实际执行时某些日志序列被循环执行的次数不满足工作流上对应的日志序列被循环执行的(正常)次数的正态分布。

Tan[44] 在 5.2.1 节生成的有限状态机上用深度优先搜索找到耗时最长的路径,将该路径的耗时记为执行流的耗时,再根据所有执行流的平均耗时和标准差来检测有异常的执行流。

5.3.2 性能剖析

lprof[12] 采用第 4 节介绍的方法对日志进行划分后,得到的每个日志集合对应一个相对完整的功能模块的一个执行实例。lprof 还在划分日志集合的过程中,额外识别出了部分日志之间的序关系。在此基础上,可以根据各日志集合中起止日志的时间戳,计算出日志集合对应的局部执行流的持续时间。为了更直观地辅助用户剖析模块性能,lprof 将同一类(顶层方法、ID 变量都相同)执行流的持续时间、各个节点上的耗时以及网络延迟等分析结果以图形化的方式呈现给用户。

5.4 小结

本节归纳了目前研究工作关注的三类日志特征,并介绍了挖掘和使用这些特征的代表性技术。表 1 从应用的角度归纳本节所介绍的技术,小结了各项技术挖掘日志特征时使用到的日志中的信息,以及挖掘和使用日志特征时的技术特点。对于可以服务于多项不同应用的技术,表 1 就这些技术在面向不同应用时所使用的日志信息以及技术特点进行了分项罗列。

从中可以看出,除服务于系统理解的日志分析技术以挖掘和使用时序特征为主之外,面向其他 3 项应用的技术都使用了多项特征,只是每项技术通常只关注日志的某一类特征,并未考虑不同特征之间的结合或相互影响。DISTALYZER[56] 在诊断导致分布式系统性能差异的原因时分别使用了日志的数量特征和时间特征,但这项工作对不同日志特征的挖掘和使用是分别独立进行的,提供给用户的分析结果也是分开的。此外,日志特征挖掘阶段使用的日志信息与日志划分阶段有显著区别,使用较多的是日志时间戳以及日志的数量(日志划分阶段使用最频繁的日志中的变量信息)。

目前的研究工作对分布式系统异常诊断以及系统理解的关注度高于对性能剖析和错误诊断的关注度,这一方面是由于后两类应用需要以前两类应用为基础,另一方面也与后两类应用需要更精确的程序依赖关系(program dependence)或者日志间的因果关系(causal relationship)作为分析的基础,而目前挖掘日志时序特征的技术还不足以精确、全面刻画出上述关系。此外,导致分布式系统出现功能或性能异常的原因可能来自系统设计、硬件、网络等多个方面,完全相同的异常也可能由完全不同的原因导致,目前使用日志分析辅助分布式系统问题诊断的技术只能尽量缩小与问题有关的日志范围,为用户提供诊断方向,自动化程度有限。而异常检测虽然自动化程度高,但也存在因训练集不完备而导致误报或漏报的情况。由此可见,日志分析在辅助分布式系统异常检测和故障诊断方法都还存在提升空间。

6 结 论

本文归纳了分布式系统日志分析与应用的通用流程,并就流程中 3 个主要步骤:日志的收集与分析、日志划分以及日志特征的挖掘与应用中的关键技术进行了分类阐述与小结。第 3 节介绍了日志在线收集平台的原理。第 4 节对代表性的划分日志技术进行归纳整理。从中可以看出,目前的研究工作倾向将隶属相同请求的日志划分在一起,也使后续的日志特征挖掘技术更有针对性。在划分过程中,日志中的 ID 变量是被使用的最多的信息。第 5 节将现有技术重点关注的日志特征分为数量特征、时序特征、时间特征等 3 类,并结合具体应用,阐述对比了如何挖掘和使用不同类型的日志特征。从中可以看出,目前日志分析结果主要应用于辅助分布式系统理解、异常检测(包括错误预测)、故障诊断和性能剖析 4 个方面,但都在一定程序上需要用户的参与。

基于上述分析,目前在分布式系统日志分析及应用领域,仍然存在的问题或者下一阶段的研究方向大致包括以下 4 个方面。

表 1　日志特征的挖掘与应用小结

应用	代表性工作	挖掘的特征	挖掘使用的信息	关键技术
异常检测	Xu[16]	数量特征	日志数量	用 PCA 识别由日志向量构成的矩阵中的异常日志向量
	Lou[30]		日志数量	求解可令 $\boldsymbol{X}\times Y=0$ 成立的系数不变式 Y($\boldsymbol{X}$ 是量矩阵),检测违反 Y 的日志向量
	ELT[36]		日志向量	先根据日志向量的相似性聚类,对聚类结果中较大的集合,根据由日志序列转化得到的 MFG 图间的距离识别异常
	CloudSeer[55]	时序特征	时间戳 静态文本	根据日志类型间的 happened-before 关系建立工作流图,在线检测时,根据新产生的日志是否与工作流匹配来检测异常
	Fu[39]	时间特征	时间戳	用 BCT 算法提取工作流图后,分别计算流图上两个节点间的正常执行的时间和循环次数的正态分布,作为检测标准
	Tan[44]			在人工辅助构建的有限状态机上,根据关键路径的平均耗时和标准方差检测异常
	Kahuna[18]			在人工辅助的为每个节点建立的有限状态机上,计算每个状态的耗时,建立直方图。通过比较不同节点的直方图距离识别异常节点
故障诊断	Shang[50]	数量特征	日志向量	根据日志向量的相似性分别对正常执行和异常执行聚类,将二者的差集反馈给用户
	LogCluster[31]			在 Shang 的基础上考虑日志权重以及对重复发生的故障的识别
系统理解	CloudSeer[55]	时序特征	时间戳(或向量时间戳)、静态文本	假设不同日志类型间都存在某种时序关系,然后删除日志序列中不存在的时序关系以及可以通过传递得到的时序关系,形成工作流图
	Lou[29]			基于 4 种时序关系推导工作流图
	Ivan[53]			基于 5 种时序关系,将每个日志序列表示成有向无环图并挖掘时序不变式
	Csight[27]			基于 Ivan 的方法推导有限状态机,并在不违背时序不变式的前提下合并有限状态机
	Lou[54]			归纳组件间具有时序关系的两条日志应满足的特征
	SALSA[43]		时间戳	人工确定 hadoop 中的关键日志及其之间的时序关系,形成代表控制流和数据流两个有限状态机
	Mystery Machine[28]			同 CloudSeer
	Tan[44]			利用打印了两个 ID 变量的日志合并 SALSA 的两个有限状态机
性能剖析	Stitch[32]	时间特征	ID 变量,时间戳	根据经验规则筛选出 ID 变量后,根据变量值间的 3 种关系挖掘 ID 变量间的可替换关系或层次关系,并可视化
	Mystery Machine[28]		时间戳	基于工作流图和日志序列识别执行流上的关键路径以及路径上的性能分布
	Tan[44]			将每个执行流中各个阶段的执行时间累加到工作流上
	Iprof[12]		时间戳 机器节点	将同类执行流(顶层方法及 ID 变量相同)的时间分布可视化

(1) 日志质量的评价标准

日志质量是决定日志分析精度以及应用效果的关键因素之一。现有的日志分析方法大多要求或假定日志中包含了所有必须的信息。比如,Stitch 假设日志满足流重建原则(见第 4 节),即程序员总是令日志输出充足的可以用于重新构造出运行时执行流的信息,换而言之,对于每个重要的事件,程序员总是会插入一条日志打印语句,该语句打印的内容中总是包含一组能够唯一标识该用户请求中该事件的变量。然而,在实际开发过程中,程序员由于疏忽或者出于对效率因素的考虑,并不总能遵循上述原则。实际日志质量不能完全满足日志分析方法所假定的日志质量标准的问题,不仅在分布式系统程序中存在,在传统应用中也不例外。针对这个问题,文献[57-60]提出了日志的自动优化技术。其中,LogEnhancer[59] 基于数据流分析为日志打印语句添加能够

帮助用户追踪错误传播路径的变量。Errlog[58] 通过对 5 个开源系统中失效特征的分析,归纳了书写日志打印语句可以帮助开发者诊断故障的 7 种代码模式,并开发了一个自动为上述代码模式插入日志打印语句的工具。LogAdvisor[57] 根据已有的日志打印语句所在代码模块包含的错误类型(如异常类型“FileNotFoundException”)、文本、函数名等特征学习出预测模型(predictive model),为其他代码模块提供日志优化建议。但这些日志优化技术都没有考虑分布式系统的日志质量需求。为了支持流重建原则,针对分布式系统日志的优化不仅要考虑诸如日志级别、日志内容这些传统日志优化技术关注的问题,还必须额外确保日志中包含的变量能够唯一地标识日志对应的事件。除此之外,日志优化须以不影响分布式系统的服务质量为前提。对于频繁被调用的关键功能模块,输出充足的日志固然可以辅助用户及时监测到系统异常和诊断异常原因,但同样可能影响模块的效率。如何制定日志质量评价标准是分布式系统日志分析及应用的研究者可以继续深入探索的一个方向。

(2) 进一步提高日志分析的效率

尽管现有的工作已经能够对分布式系统日志进行在线收集以及高效地解析、划分和挖掘,但随着分布式系统应用的推广,日志规模也在持续、急速地增加(比如,Facebook 在 2010 年底累积存储的日志规模是 20 PB,到 2011 年 3 月就急速增加到 30 PB[61])。这一方面为日志收集工具带来巨大的带宽压力和存储压力,另一方面也对后续的日志解析、划分以及挖掘技术的可扩展性带来重大考验。虽然日志中包含丰富的信息,但从第 4、5 节对在日志划分阶段和日志特征挖掘阶段使用到的信息的归纳结果来看,现有日志分析技术关注的只是日志中的少部分内容。如果能够结合最终的应用需求,在日志收集或解析阶段提前过滤那些与需求无关的日志或日志中与需求无关的部分,无疑能够缓解传输带宽、存储空间以及分析技术的可扩展性等多方面的压力。一些研究工作已经在这方面进行了尝试,比如 Stitch 只传输日志中的变量值和时间戳。

(3) 进一步提高日志分析在辅助分布式系异常检测和故障诊断方面的精度

如第 5 节所述,检测分布式系统中的异常是目前日志分析技术的一项重要应用。以监督或半监督学习为基础的异常检测方法在检测精度上优于非监督的方法,但也存在由于训练集不完备而引起的漏报或误报。如何在挖掘过程中考虑描述模块间的关联,并通过对模块的上下文的分析提高异常检测中的精度是一个值得研究的方向。

与异常检测相比,目前利用日志分析诊断分布式系统中故障(例如功能异常、性能异常、性能差异等)的原因的方法在自动化程度和精度上都存在进一步提升的空间。对程序依赖(包括数据依赖和控制依赖)的刻画和分析是传统诊断技术定位错误原因的重要基础。但正如第 5 节所述,目前挖掘日志时序特征的技术还不足以精确、全面的刻画出上述关系。此外,由于导致分布式系统出现功能或性能异常的原因可能来自系统设计、硬件、网络等多个方面,单纯的关注日志的某一项特征,可能忽略其他与错误源以及传播有关的信息。如何提取能够反映程序依赖关系的日志特征,以及如何综合日志的多项特征提高诊断的效率都是可以进一步深入的研究方向。

此外,无论是使用日志分析技术服务于分布式系统的功能理解、性能剖析,还是异常检测、故障诊断,都需要与用户交互,由用户确认。如何提供更友好、直观的用户交互平台和结果展现形式也是日志分析技术的研究者需要关注的一个重要问题。

(4) 与其他技术的结合

受到日志质量以及技术自身存在的不足的影响,目前的日志分析技术虽然能辅助用户理解、剖析、检测和诊断分布式系统,但所提取出的工作流图和用户请求执行流与基于端到端追踪得到的结果在精度上还存在一定差距。比如,Stitch 虽然能基于日志重构出全局执行流,但无法还原出日志之间的时序依赖。针对这个问题,一些研究工作开始尝试将日志分析与轻量级的插装追踪结合。一旦能够获得与端到端追踪精度相当的执行流,就能复用针对后者开发的检测和诊断技术[62,63]。

一些使用插装控制执行轨迹的领域,如分布式

模型检测[64-68]、分布式测试[69]等,也可以考虑结合对日志的分析来识别已探索的路径或评价测试的完备性。

随着深度学习的流行,目前已经有相关工作结合日志分析和深度学习自动化的挖掘日志特征。如DeepLog[70]首先利用3.2节描述的方法解析日志,获得变量值和时间,然后利用长短期记忆((long short-term memory, LSTM),一种时间递归神经网络)[71]学习正常日志序列的时序特征,构成工作流图,最后部署在线上监控由于系统受到网络攻击而产生的异常日志序列。相信如何结合深度学习、强化学习更好地识别特征是日志分析未来的研究方向。

参考文献

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

[2] Borthakur D. HDFS architecture guide[J]. *Hadoop Apache Project*, 2008, 53:1-13

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

[4] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets[J]. *Hot Cloud*, 2010, 10(10): 1-7

[5] Sefraoui O, Aissaoui M, Eleuldj M. OpenStack: toward an open-source solution for cloud computing[J]. *International Journal of Computer Applications*, 2012, 55(3): 38-42

[6] The AWS Team. Summary of the Amazon EC2 and Amazon RDS service disruption in the US east region[EB/OL]. http://aws.amazon.com/message/65648: AWS, 2018

[7] Dylan T. 5-minute outage costs Google MYM545,000 in revenue[EB/OL]. http://venturebeat.com/2013/08/16/3-minute-outage-costs-google-545000-in-revenue: Venturebeat, 2013

[8] Iwgcr. Downtime costs per hour[EB/OL]. http://iwgcr.org/? p=404: Iwgcr, 2012

[9] George L. HBase: The Definitive Guide: Random Access to Your Planet-Size Data[M]. O'Reilly Media, Inc, 2011. 1-552

[10] Sylvain L. CAS should distinguish promised and accepted ballots [EB/OL]. https://issues.apache.org/jira/browse/CASSANDRA-6023: Apache, 2013

[11] Ming M. Provide metrics for hbase client[EB/OL]. https://issues.apache.org/jira/browse/HBASE-4145: Apache, 2011

[12] Zhao X, Zhang Y, Lion D, et al. lprof: a non-intrusive request flow profiler for distributed systems[C]. In: Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, Broomfield, USA, 2014. 629-644

[13] Sambasivan R R, Fonseca R, Shafer I, et al. So, you want to trace your distributed system? Key design insights from years of practical experience[J]. *Parallel Data Lab, Carnegie Mellon Univ, Pittsburgh, PA, USA, Tech. Rep.* 2014,14(102):1-23

[14] Sigelman B H, Barroso L A, Burrows M, et al. Dapper, a large-scale distributed systems tracing infrastructure, Google Technical Report dapper-2010-1 [R]. California: Google, 2010

[15] Fonseca R, Porter G, Katz R H, et al. X-trace: a pervasive network tracing framework[C]. In: Proceedings of the 4th USENIX Conference on Networked Systems Design and Implementation, Cambridge, USA, 2007. 20-20

[16] Xu W, Huang L, Fox A, et al. Detecting large-scale system problems by mining console logs[C]. In: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, Big Sky, USA,2009. 117-132

[17] Hadoop. Welcome to Apache? Hadoop[EB/OL]. https://hadoop.apache.org: Apache, 2018

[18] Tan J, Pan X, Marinelli E, et al. Kahuna: problem diagnosis for mapreduce-based cloud computing environments[C]. In: Proceedings of the 2010 IEEE Network Operations and Management Symposium, Osaka Station, Japan, 2010. 112-119

[19] Rabkin A, Katz R. Chukwa: a system for reliable large-scale log collection[C]. In: Proceedings of the 24th Large Installation System Administration Conference, San Jose, USA, 2010. 1-15

[20] Logstash. Centralize, transform & stash your data[EB/OL]. http://www.elasticsearch.org/overview/logstash: Elastic, 2018

[21] Scribe. Scribe is a server for aggregating log data streamed in real time from a large number of servers[EB/OL]. https://github.com/facebookarchive/scribe: Face-

book, 2018

[22] Kafla. A distributed streaming platform[EB/OL]. https://kafka.apache.org:Apache, 2018

[23] Flume. Tool for streaming log and event data into Hadoop [EB/OL]. http://www.cloudera.com/products/apache-hadoop/apache-flume.html: Apache, 2018

[24] Btraceio. Amazon CloudWatch[EB/OL]. https://aws.amazon.com/cloudwatch:AWS, 2018

[25] Btrace. A safe, dynamic tracing tool for the Java platform [EB/OL]. https://github.com/btraceio/btrace: Github, 2018

[26] Oldmanpushcart. Java 诊断工具[EB/OL]. https://github.com/oldmanpushcart/greys-anatomy: Github, 2018

[27] Beschastnikh I, Brun Y, Ernst M D, et al. Inferring models of concurrent systems from logs of their behavior with CSight[C]. In: Proceedings of the 36th International Conference on Software Engineering, Hyderabad, Italy, 2014. 468-479

[28] Chow M, Meisner D, Flinn J, et al. The mystery machine: end-to-end performance analysis of large-scale Internet services[C]. In: Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, Broomfield, USA,2014. 217-231

[29] Lou J G, Fu Q, Yang S, et al. Mining program workflow from interleaved traces[C]. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, USA. 613-622

[30] Lou J G, Fu Q, Yang S, et al. Mining invariants from console logs for system problem detection[C]. In: USENIX Annual Technical Conference, Boston, USA,2010. 23-25

[31] Lin Q, Zhang H, Lou J G, et al. Log clustering based problem identification for online service systems[C]. In: Proceedings of the 38th International Conference on Software Engineering Companion, Austin, USA, 2016. 102-111

[32] Zhao X, Rodrigues K, Luo Y, et al. Non-intrusive performance profiling for entire software stacks based on the flow reconstruction principle[C]. In: Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, Savannah, USA,2016. 603-618

[33] Lamport L. Time, clocks, and the ordering of events in a distributed system[J]. *Communications of the ACM*, 1978, 21(7): 558-565

[34] Hunt P, Konar M, Junqueira F P, et al. ZooKeeper: wait-free coordination for internet-scale systems[C]. In: Proceedings of the USENIX Annual Technical Conference, Boston, USA,2010. 1-14

[35] Beschastnikh I, Brun Y, Schneider S, et al. Leveraging existing instrumentation to automatically infer invariant-constrained models[C]. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European Conference on Foundations of Software Engineering, Szeged, Hungary,2011. 267-277

[36] Kc K, Gu X. ELT: Efficient log-based troubleshooting system for cloud computing infrastructures[C]. In: Proceedings of the 30th IEEE Symposium on Reliable Distributed Systems, Madrid, Spain,2011. 11-20

[37] Log4j. Apache Log4j 2[EB/OL]. http://logging.apache.org/log4j:Apache, 2018

[38] Jiang Z M, Hassan A E, Hamann G, et al. An automated approach for abstracting execution logs to execution events[J]. *Journal of Software Maintenance and Evolution: Research and Practice*, 2008, 20(4): 249-267

[39] Fu Q, Lou J G, Wang Y, et al. Execution anomaly detection in distributed systems through unstructured log analysis[C]. In: Proceedings of the 2009 edition of the IEEE International Conference on Data Mining series, Miami, USA, 2009. 149-158

[40] Tang L, Li T, Perng C S. LogSig: generating system events from raw textual logs[C]. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, Glasgow, UK, 2011. 785-794

[41] Tang L, Li T. LogTree: a framework for generating system events from raw textual logs[C]. In: Proceedings of the 10th IEEE International Conference on Data Mining series, Sydney, Australia,2010. 491-500

[42] Aharon M, Barash G, Cohen I, et al. One graph is worth a thousand logs: uncovering hidden structures in massive system event logs[J]. *Machine Learning and Knowledge Discovery in Databases*, 2009, 5781:227-243

[43] Tan J, Pan X, Kavulya S, et al. SALSA: analyzing logs as state machines[J]. *WASL*, 2008, 8: 6-6

[44] Tan J, Kavulya S, Gandhi R, et al. Visual, log-based causal tracing for performance debugging of mapreduce systems. In: Proceedings of the 30th International Conference on Distributed Computing Systems, Genoa, Italy, 2010. 795-806

[45] Sharir M, Pnueli A. Two Approaches to Interprocedural Data Flow Analysis[M]. New York: New York University, 1978. 189-234

[46] Fu X, Ren R, Zhan J, et al. LogMaster: mining event correlations in logs of large-scale cluster systems[C]. In: Proceedings of the 31st Symposium on Reliable Distributed Systems, Irvine, USA, 2012. 71-80

[47] DeGroot M H, Schervish M J. Probability and Statistics [M]. Pearson Education, 2012

[48] Lim C, Singh N, Yajnik S. A log mining approach to failure analysis of enterprise telephony systems[C]. In: Proceedings of the 38th IEEE International Conference on Dependable Systems and Networks with FTCS and DCC, Anchorage, USA, 2008. 398-403

[49] Zhou P, Gill B, Belluomini W, et al. Gaul: gestalt analysis of unstructured logs for diagnosing recurring problems in large enterprise storage systems[C]. In: Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems, Delhi, India, 2010. 148-159

[50] Shang W, Jiang Z M, Hemmati H, et al. Assisting developers of big data analytics applications when deploying on hadoop clouds[C]. In: Proceedings of the 2013 International Conference on Software Engineering, San Francisco, USA, 2013. 402-411

[51] Fidge C J. Timestamps in message-passing systems that preserve the partial ordering[J]. *Australian Computer Science Communications*, 1988, 10(1): 56-66

[52] Kopetz H, Ochsenreiter W. Clock synchronization in distributed real-time systems[J]. *IEEE Transactions on Computers*, 1987, 100(8): 933-940

[53] Beschastnikh I, Brun Y, Ernst M D, et al. Mining temporal invariants from partially ordered logs[J]. *ACM SIGOPS Operating Systems Review*, 2012, 45(3): 39-46

[54] Lou J G, Fu Q, Wang Y, et al. Mining dependency in distributed systems through unstructured logs analysis [J]. *ACM SIGOPS Operating Systems Review*, 2010, 44 (1): 91-96

[55] Yu X, Joshi P, Xu J, et al. Cloudseer: workflow monitoring of cloud infrastructures via interleaved logs[J]. *ACM SIGOPS Operating Systems Review*, 2016, 50(2): 489-502

[56] Nagaraj K, Killian C, Neville J. Structured comparative analysis of systems logs to diagnose performance problems [C]. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, San Jose, USA, 2012. 26-26

[57] Zhu J, He P, Fu Q, et al. Learning to log: helping developers make informed logging decisions[C]. In: Proceedings of the 37th IEEE International Conference on Software Engineering, Firenze, Italy, 2015. 415-425

[58] Yuan D, Park S, Huang P, et al. Be conservative: enhancing failure diagnosis with proactive logging[C]. In: Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation, Hollywood, USA, 2012. 293-306

[59] Yuan D, Zheng J, Park S, et al. Improving software diagnosability via log enhancement[J]. *ACM Transactions on Computer Systems*, 2012, 30(1): 4

[60] Ding R, Zhou H, Lou J G, et al. Log2: a cost-aware logging mechanism for performance diagnosis[C]. In: Proceedings of USENIX Annual Technical Conference, Santa, USA, 2015. 139-150

[61] Paul Y. Moving an elephant large scale hadoop data migration at facebook[EB/OL]. https://www.facebook.com/notes/paul-yang/moving-an-elephantlarge-scale-hadoop-data-migration-atfacebook/ 10150246275318920: Facebook, 2011

[62] Reynolds P, Killian C E, Wiener J L, et al. Pip: detecting the unexpected in distributed systems[C]. In: Proceedings of the 3th USENIX Conference on Networked Systems Design and Implementation, San Jose, USA, 2006. 115-128

[63] Sambasivan R R, Zheng A X, De Rosa M, et al. Diagnosing performance changes by comparing request flows [C]. In: Proceedings of the 8th USENIX conference on Networked Systems Design and Implementation, Boston, USA, 2011. 43-56

[64] Lin H, Yang M, Long F, et al. MODIST: transparent model checking of unmodified distributed systems[C]. In: Proceedings of the 6th USENIX conference on Networked Systems Design and Implementation, Boston, USA, 2009. 213-228

[65] Leesatapornwongsa T, Hao M, Joshi P, et al. SAMC: semantic-aware model checking for fast discovery of deep bugs in cloud systems[C]. In: Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, Broomfield, USA, 2014. 399-414

[66] Leesatapornwongsa T, Gunawi H S. SAMC: a fast model checker for finding heisenbugs in distributed systems [C]. In: Proceedings of the 2015 International Symposium on Software Testing and Analysis, Baltimore, USA, 2015. 423-427

[67] Guo H, Wu M, Zhou L, et al. Practical software model checking via dynamic interface reduction[C]. In: Proceedings of the 23rd ACM Symposium on Operating Sys-

tems Principles, Cascais, Portugal, 2011. 265-278

[68] Guerraoui R, Yabandeh M. Model checking a networked system without the network[C]. In: Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation, Soston, USA, 2011. 225-238

[69] Ju X, Soares L, Shin K G, et al. On fault resilience of openstack[C]. In: Proceedings of the 4th Annual Symposium on Cloud Computing, Santa Clara, USA, 2013. 1-16

[70] Du M, Li F, Zheng G, et al. Deeplog: Anomaly detection and diagnosis from system logs through deep learning [C]. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, USA, 2017. 1285-1298

[71] Hochreiter S, Schmidhuber J. Long short-term memory [J]. *Neural Computation*, 1997, 9(8):1735-1780

Log analysis for distributed systems and its application

Lu Jie[*,***], Li Feng[**], Li Lian[*,***]

([*] State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100193)

([***] University of Chinese Academy of Sciences, Beijing 100190)

Abstract

Distributed systems are the key infrastructure in the big data era, it is vital to ensure their reliability since faults in the infrastructure can often lead to catastrophic failures. However, it is also very challenging, due to the complexity in its design, implementation and deployment. Log analysis is an effective way to help address the problem. Distributed systems often output logs with various runtime information. Such information can help users to understand system behaviour, profile system performance, detect anomaly and diagnose faults. However, log messages are unstructured and the unstructured logs from different user requests and different running threads are often intertwined with each other, all located in a large-sized log file. Therefore, it is very difficult for users to manually inspect the large log files and mine useful information in these files. Log analysis automatically processes the vast log data, providing user-friendly and useful information to the users, to help them understand system behavior, monitor system execution, and detect anomalies. This paper surveys various log analysis techniques. We summarize the basic process of log analysis into three main steps: log parsing and collection, log partition and log mining. We elaborate the key problems in each step and summarise their corresponding solutions. We describe three kinds of different features in logs which can be mined for different use cases. At last, we highlight the future research directions in this field.

Key words: distributed system, log analysis, feature mining, anomalies detection, fault diagnosis