基于 MR 的高可靠分布式数据流统计模型

朱蔚林[1,2],木伟民[1],金宗泽[1,2],王伟平[1]

(1. 中国科学院 信息工程研究所,北京 100093;
2. 中国科学院大学,北京 100049)

摘 要:结合流数据独有的特点,以数据流上基于窗口模型的连续分组统计为应用场景,结合现今主流的流数据处理平台 Storm 和 Spark Streaming 的优点,提出了一个高吞吐、低延迟、高可扩展性的分布式数据流统计模型 Mars,解决由于流数据易失、时效性强造成的吞吐量压力大、数据延迟低等问题。在容错方面,Mars 提供了 at-least-once 语义支持以防出现重大错误。采用真实实验环境对 Mars 进行测试,与目前流行的分布式流处理平台 Spark Streaming 和 Storm 相比,Mars 对数据的实时性操作延迟介于二者之间,但就不同的集群规模而言,Mars 的吞吐率明显优于二者 1 到 2 倍,就语义准确性而言,Mars 实现了与 Storm 同级别的语义限制。

关键词:数据流;分组统计;连续查询;分布式系统;实时处理
中图分类号:TP391 **文献标识码**:A **文章编号**:1673-629X(2018)01-0006-05
doi:10.3969/j.issn.1673-629X.2018.01.002

Statistical Model of Distrubuted Data Strem with High Reliability Based on MR

ZHU Wei-lin[1,2], MU Wei-min[1], JIN Zong-ze[1,2], WANG Wei-ping[1]
(1. Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China;
2. University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: According to the unique characteristics of the data stream, with consecutive grouping statistics based on window model in the data flow as application scenarios, combined with the advantages of mainstream stream data processing platform like Storm and Spark Streaming, we propose a distributed statistical model of data stream with high throughput and scalability as well as low latency, namely Mars. It solves the problems of strong throughput and low latency due to losing data easily and strong timelessness. On the fault-tolerant, Mars provides at-least-once semantic support against major errors. It is tested in real experiment environment and made a comparison with the currently popular distributed flow processing platform Spark Streaming and Storm, which show that it is between them in real-time operation delay for data. However, in terms of the scale of the cluster, Mars' throughput rate is significantly better than that of the two, and in terms of semantic accuracy, it achieves the semantic limits of the same level as Storm.

Key words: data stream; grouping statistic; continuous query; distributed system; real-time processing

0 引 言

数据的价值随着时间推移会慢慢降低,在社会生活中,特别是商业场景中这一现象更加显著。流处理系统的出现让用户能够快速地从庞杂的数据流中提炼出数据蕴含的价值。由于数据流的持续产生且体积庞大,给系统的存储能力带来了巨大挑战。由此导致数据流统计具有 one-pass 访问、有限内存和实时等特点,也为统计带来了极大的挑战。

现如今,大数据时代涌现了如 S4[1-2]、Storm[3]、Spark Streaming[4]等一系列数据流处理平台[5-6],这些平台由于面向的应用场景不同而各有特点。S4 由 Yahoo 在 2010 年提出,S4 为了保证良好的可扩展性,采用去中心化结构,但是 S4 消息路由仅支持按照 Key 值进行分布,而且没有提供消息处理的反馈机制,使得容错成为了很大的问题。Storm 面向纯流式处理,以低延迟为核心设计目标,将数据流处理任务抽象为有

收稿日期:2017-02-15 **修回日期**:2017-06-22 **网络出版时间**:2017-10-19
基金项目:国家自然科学基金(61402473)
作者简介:朱蔚林(1992-),男,硕士研究生,研究方向为流数据处理、大数据处理;王伟平,教授,博士,CCF 会员(12474M),研究方向为数据库、大数据存储与处理、云计算。
网络出版地址:http://kns.cnki.net/kcms/detail/61.1450.TP.20171019.1625.052.html

向无环图,称作 Topology,同时在 at-least-once 语义的基础上实现了 exactly-once 语义,但是其弱中心化的结构也没有彻底解决单点的问题。Spark Streaming 构建在 Spark 的基础上,沿用了 Spark 中创新性的存储结构 RDD(弹性分布式数据集),将数据流切分为一个个 RDD,增大了处理粒度以提高吞吐量,但是也因此增加了延迟。

在数据流上有一种普遍的应用场景是在数据流上进行数据统计[7],例如对于电信网络、社交网络数据流等等。在这些数据流上的实时统计无论对于舆情控制,还是提供更高质量的服务都有重要意义。然而上述平台并不能完美契合此类应用场景。

数据流统计场景下一个很典型的处理是分组统计。以社交网络数据流为例,假设数据流 S 中每个元组包含 uid,topic,time 三个字段,uid 唯一标识一个用户个体,topic 为用户参与的话题,time 为该条信息产生的时间分片,现在需要统计各个时间分片的热点话题。如果将数据流类比为传统的关系数据库,会提交以下 SQL 语句:

```
SELECT topic, time, count(1)
FROM S GROUP BY topic, time
```

而对于数据流而言,数据流具有快速流动、易失的特性,绝不可能像数据库一样在一个时刻看到整个数据流的全部数据,而且热点话题具有很强的时效性,统计结果延迟要尽量做到最小。同时,电信网络、社交网络由于用户众多,数据量往往很大,因此对流统计平台的性能要求很高,如流统计的吞吐率和延迟等。

面对这样的应用场景,系统主要的需求在于高吞吐率和低延迟。目前流行的通用分布式流处理平台如 Spark Streaming 和 Storm 等,系统结构较复杂,在这种高吞吐量的实时连续统计场景上性能有所局限。对此,面向基于窗口的连续查询需求,提出了一个高吞吐、低延迟的分布式数据流统计模型 Mars,同时提供了强大的容错性。

1　关键技术

数据流可以理解为一个不断增长的数据集合[8-9],将其定义如下:

定义 1:设 t 表示任意时间戳,t 时刻到来的数据集为 d_t,称$\{\cdots d_{t-1}, d_t, d_{t+1}\}$为数据 d 的数据流。

数据流具有实时性、易失性、突发性、无序性、无限性等特征。在数据流上的统计查询处理由于这些特征,划分为不同的模型。

1.1　数据流统计时序模型

按照统计的时序范围来划分,数据流上的统计处理分为界标模型[10]、滑动窗口模型[11]以及跳动窗口模型[12]。

1.1.1　界标模型

设当前时间戳为 t,s 为一个指定的在 t 之前的时间戳,称作界标,界标模型统计的是从界标时间戳 s 开始一直到当前时间戳 t 范围内数据流$\{d_s \cdots d_t\}$上的统计结果。

1.1.2　滑动窗口模型

设当前时间戳为 t,滑动窗口模型统计的是当前时间戳之前一定时间范围内的数据流,设滑动窗口大小为 n,即时间范围为 n,滑动窗口模型所处理的数据流为$\{d_{t-n} \cdots d_t\}$。

除了上述基于时间戳的滑动窗口模型,另外还有基于元组数量的滑动窗口,即在内存中保存一定数量的元组。

1.1.3　跳动窗口模型

跳动窗口是滑动窗口的一个延伸。在滑动窗口模型中,窗口以一定时间范围或者元组数量为单位向前滑动,两次连续的处理的数据集有重叠,而跳动窗口模型相邻两次处理数据集无重叠,这一次处理时间范围的起点是上一次处理时间范围的终点(以基于时间的窗口为例)。因此,每个跳动窗口的处理结果之间无交集,同时它们的并集就是整个数据集上全量的处理结果。

1.2　数据流统计查询类型

根据统计查询提交形式的不同,查询类型分为 Ad-hoc 查询[13]和连续查询[14]。

1.2.1　Ad-hoc 查询

Ad-hoc 查询也称即席查询,Ad-hoc 查询请求可以在统计系统运行时的任一时刻提交,统计系统接收到查询请求后立即处理该查询并在产生结果后立即返回查询结果。目前处理 Ad-hoc 查询的数据流处理平台较少。

1.2.2　连续查询

连续查询请求在查询统计系统启动时或某一特定时间戳提交,在这一时间戳之后,直到用户主动取消查询请求,系统一直处理该查询,每隔一定时间间隔输出查询结果。Mars 面向的查询类型即是连续查询。

1.3　数据流统计语义模型

对数据流做统计处理时,最小的单位是元组。理想状态下,数据流中的每一个元组都应该被处理一次,且仅仅处理一次。但是在系统层面,由于分布式系统本身的复杂性和不确定性,以及在某些应用场景下对系统需求的不同,往往会对精度做一定程度上的牺牲。根据对每一元组处理次数的保证,统计模型的语义分为三种。

1.3.1　至多一次(at most once)

至多一次语义是最松的约束,该语义模型尽力使得每一元组被处理到,但不保证对任意元组处理的必然性。由于约束宽松,实现这样的模型系统开销往往是最小的,由于这样的特性,该语义适合对吞吐量要求较高、但不要求统计精确度的应用场景。

1.3.2　至少一次(at least once)

至少一次语义保证数据流中每一元组都至少被处理一次,但是在一些特殊情况发生时,有可能会造成重复处理某些元组。在这样的语义约束下,统计模型实现时需要设计严格的容错机制,确保在任何可控故障发生时,每个元组都会被处理,但是容错性会带来不菲的系统开销。

1.3.3　精确一次(exactly once)

精确一次语义是理想状态下的语义,也是最严格的语义约束。这一语义一般是在至少一次语义的模型基础上,对统计结果进行去重而得到的。因此,系统实现时,精确一次语义比至少一次语义又多了一重开销,只有在精确度要求极其苛刻的应用场景下会使用这一语义约束,例如银行系统或证券系统。

2　Mars 设计与实现

2.1　系统架构

Mars 的系统架构如图 1 所示。

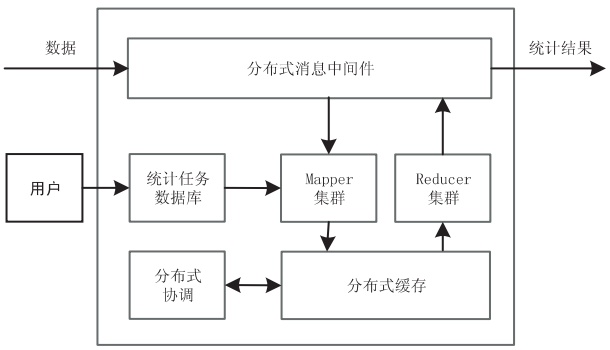

图 1　Mars 系统架构

由图 1 可见,Mars 依赖的外部组件有四个:分布式消息中间件提供消息服务,Mars 的输入和输出都通过分布式消息中间件完成,统计任务数据库储存用户的统计需求,分布式协调系统提供对分布式集群运行时的状态管理等等,分布式缓存负责异步解耦和临时缓存中间数据。

Mapper 集群、Reducer 集群是 Mars 的核心组件。Mapper 集群从消息中间件拉取消息并处理,将中间结果顺序缓存在分布式缓存中;Reducer 集群从分布式缓存中顺序读取分布式缓存中的中间数据,处理后将最终的统计结果再发送回消息队列。

Mapper 和 Reducer 的灵感来自于 MapReduce 编程模型,尽管 MapReduce 编程模型是为了批处理场景而提出的,但是它将大规模数据处理过程抽象为 Map 和 Reduce 两个阶段,对于数据流统计问题同样具有重要的指导意义。Mars 将 MapReduce 模型扩展到了集群概念上,每一个 Mapper 或者 Reducer 计算单元都是分布式集群中的一个节点,分别称作 Mapper 或 Reducer。所有的 Mapper 节点组成 Mapper 集群,所有的 Reducer 节点组成 Reducer 集群。

Mapper 集群采用去中心化结构,集群内各个节点是对等关系,使用去中心化结构的核心目标是将计算粒度切分,在进行较大窗口下的统计时,如果使用集中式结构,缓存窗口内的全部数据将造成极大的内存开销和时间开销。而 Mars 将这样的大窗口切分为小窗口分布到集群上并行处理,有效解决了该问题。同时这一设计保证了 Mapper 集群良好的可扩展性,使得 Mars 的计算能力随着集群规模的扩大可以得到近似于线性的增加。

Reducer 集群的统计功能是将经过 Mapper 切分的细粒度统计结果合并为任务需求窗口大小的结果,因此 Reducer 集群采用主从架构。集群启动时,各个节点首先借助分布式协调系统选举出一个领导者,该领导者负责给各个节点分配任务,并监听节点状态,当集群规模发生改变时重新分配任务。同时,各个从节点也监听主节点的状态,当主节点发生故障时重新选举领导者。

2.2　数据传输协议

Mapper 和 Reducer 通过分布式缓存传递数据[15]的协议设计是 Mars 的一个关键点。Mars 使用了一种特殊的序号机制保证 Mapper 和 Reducer 协作步进,同时保证两个阶段异步运行。

初始状态时,首先针对每个处理任务在分布式缓存中为 Mapper 集群和 Reducer 集群初始化一个序号,称为 SEQ。当 Mapper 节点处理完输入的原始数据集后,将缓存中的 SEQ 自增 1,使用自增操作的主要目的是使得多节点并行处理统一统计任务时不会得到统一 SEQ,造成数据覆盖。

而 Reducer 集群出于容错性考虑,使用了延迟更新 SEQ 的策略。

2.3　容错设计

容错是所有分布式流处理系统应当关注的问题,虽然集群中每个节点发生故障的概率很小,但是一旦发生,由于数据流不断流动的特点,丢失的数据便很难找回。大多数现有的故障恢复策略都是通过冗余备份策略实现的[16],Mars 所采用的策略也类似。

在 Mapper 端,每一个节点拉取数据后都首先将数据存一份本地文件,同时将文件名与需要该数据的统

计任务 id 列表的对应关系注册到分布式缓存中,每当某一任务处理完该数据时,从列表中删去该任务 id,直到列表为空时删除本地文件,最后向分布式消息中间件反馈 ack 消息。假如在 ack 之前该节点发生了宕机,由于消息中间件未接收到 ack 消息,当发生超时后,消息中间件会向其他节点重发该消息。这一机制保证了每一条元组都会被处理至少一次。

在 Reducer 端,采用延迟更新 SEQ 的方式来保证容错性。以一个统计任务的处理过程为例,如图 2 所示。Reducer 集群有两个节点,R_1 正在处理数据,初始 R_1 从缓存中得到序号 1,于是从缓存中得到序号为 1 的数据集并处理,处理完成后在内存中序号加 1 变为 2,并不更新到缓存中。接着在缓存中读取序号为 2 的数据集,假如在处理过程中,归并后的统计结果尚未输出之前 R_1 节点发生宕机,经过新一轮的领导者选举以及任务分配,该任务迁移到 R_2 节点。这时 R_2 节点从缓存中获取到的序号依然为 1,不会造成数据丢失。

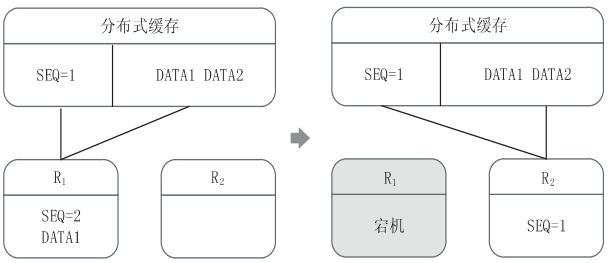

图 2　Reducer 容错机制示意图

综上,Mars 强大的容错机制保证了 Mars 的 at-least-once 语义。

3　性能评估与分析

使用 Java 语言实现了 Mars,并在典型的分组统计使用场景下对 Mars 进行了测试,同时与 Storm 和 Spark Streaming 进行对比。

3.1　实验环境

实验采用由 30 台服务器组成的集群。其中,消息中间件拥有 3 个服务节点和 30 个数据节点,分布式缓存为 15 节点的主备集群,分布式协调服务拥有 7 个服务节点。

3.2　数据流

实验的数据源选用了模拟的网络数据流 S,数据流的每个元组包含 20 个字段,其中关键字段有 timestamp、type、sip、dip、port、location 等。

每个字段的内容根据一个已知的集合中以均匀概率随机生成,生成后每个元组的平均大小为 150 字节。共生成元组 10 亿条,整个数据集大小 140 G。在每个单元实验前,提前将数据集加载到分布式消息中间件中。

3.3　统计需求

在上述数据流上,构建了如下的统计场景:

```
SELECT sip, type, timestamp, count(*)
FROM S
GROUP BY sip, type, timestamp
WINDOWING 60 s
```

其中,WINDOWING 关键字表示以 60 s 大小的跳动窗口进行统计。需要特别说明的是,对时间戳进行分组统计是为了使统计结果具有应用价值,Mapper 会对时间戳以窗口大小,即 60 s 为单位进行归一化。

3.4　性能实验

性能一般以吞吐量为表征,吞吐量计算公式如下:

$$T = \text{tps} \times \text{bs} \times \text{ts}$$

其中,T 表示吞吐量;tps 表示消息中间件每秒处理的事务数,实验中 Mapper 集群作为消息中间件的消费者,tps 相当于每秒消费的数据集个数;bs 表示每个数据集包含的元组数;ts 表示每个元组的大小(平均)。

分别在不同的数据集大小和不同的集群规模下对上述统计需求进行实验,结果如下所述。

3.4.1　集群规模固定,数据集大小不同

实验在 20 个节点的集群上完成,吞吐量取统计过程中整个集群吞吐量的平均值。由图 3 可见,随着每个数据集所包含的元组数量的增长,一开始吞吐量上升很快,当数据及大小达到 5 000 时,吞吐量达到峰值,随着数据及大小继续增加,吞吐量呈缓慢下降的趋势。不难分析出,当数据集大小为 1 这种极端情况时,每次网络开销只传输一个元组,效率极低;当数据集大小增大到 5 000 个元组时,网络开销和解包开销达到一个平衡点,故性能达到最优;当数据集大小继续增大时,虽然每次网络开销得到的数量足够大,但是反序列化数据流会占用大量的 CPU 资源,导致用于统计的系统资源减少,从而吞吐量下降。

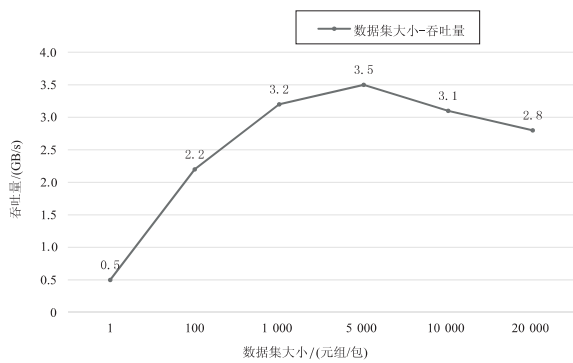

图 3　不同数据集大小时的吞吐量变化曲线

3.4.2　数据集大小固定,集群规模变化

在上述实验得出的最优数据集大小下,吞吐量取统计过程中整个集群吞吐量的平均值。由图 4 可见,

吞吐量随着集群规模的逐步增大,几乎呈线性增长,当集群规模增大到 20 个节点时,性能达到最优,此时整个集群吞吐量达到 3.5 GB/s,可以计算得单节点吞吐量为 179 MB/s。

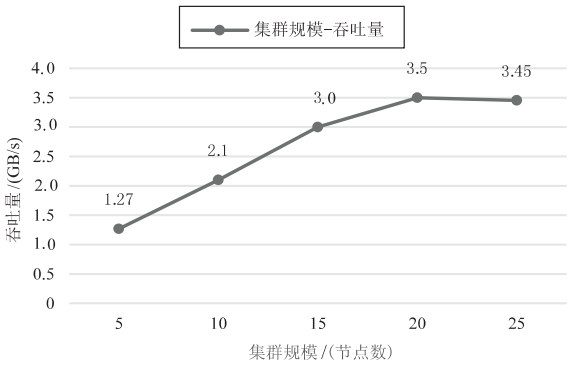

图 4 不同集群规模时的吞吐量变化曲线

当集群规模继续扩大时,吞吐量并未继续增加,这是由于从分布式消息中间件消费数据时是读盘操作。

测试数据中使用的数据是提前发送并缓存在消息队列上的。系统从消息队列中消费数据时,消息队列会从磁盘上读取数据。由于消息队列特性,每个消息队列节点只会从一块磁盘上读取数据。一共有 35 个消息队列节点,磁盘读取速度上限大约是 100 Mbps,因此整个消息队列所能提供的最大消费速率约为 3.5 G。当集群规模大于 20 时,由于消息队列磁盘读取速度已达上限,速度无法继续增加。

3.5 对比实验

同时,将上述的统计需求分别使用 Spark Streaming 和 Storm 的编程接口进行了实现,二者同样使用 Mars 的分布式消息中间件作为输入和输出。

3.5.1 性能对比

由于 Spark Streaming 和 Storm 本身已经对数据集进行了抽象,故无需在不同数据集大小的情况下进行对比。在不同的集群规模下,实验结果如图 5 所示。

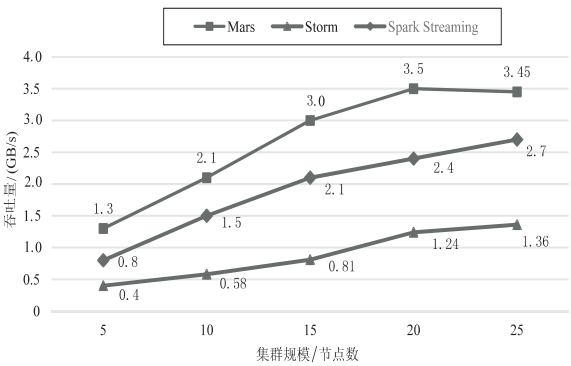

图 5 Mars,Spark Streaming,Storm 性能对比曲线

由图 5 可见,与 Storm 和 Spark Streaming 相比,Mars 在分组统计需求下具有较明显的性能优势。在集群规模为 20 时,Mars 的吞吐量是 Spark Streaming 的 1.46 倍,是 Storm 的 2.82 倍。

3.5.2 实时性对比

实时性方面,实验计算了部分处理日志中记录的每个元组的平均延迟。Storm 专门为了流处理场景设计,平均延迟最小,为 653 ms;Spark Streaming 由于需要“攒”数据,平均延迟达到了 2 383 ms;Mars 介于两者之间,平均延迟为 1 372 ms。

3.5.3 语义准确性对比

由于统计需求是在连续跳动窗口上的分组统计,没有过滤对数据量产生变化的计算,因此如果统计过程保证了 exactly-once 语义,那么统计结果中分组统计量的和应与原始记录数量保持一致。

该实验使用上述实验数据中一个 4 000 万数据量的子集完成,实验集群在实验过程中分别将其中三个节点断网以模拟故障,实验结果如表 1 所示。

表 1 Mars Spark Streaming,Storm 语义准确性对比

平台	原始数据	统计结果
Mars	40 000 000	40 000 000
Spark Streaming	40 000 000	40 000 272
Storm	40 000 000	40 000 000

由表 1 可见,Mars 虽然仅实现了 at-least-once 语义,但由于其良好的容错性,在发生节点故障时并没有造成数据丢失或重复,实现了与 Storm 相同级别的语义限制。

4 结束语

提出了一个面向基于窗口的连续查询需求的分布式数据流统计模型。该模型在保证 at-least-once 语义的前提下,实现了优异的性能,尤其在基于窗口的分组统计这一统计场景下,相比目前流行的分布式数据流处理平台具有较明显的优势。同时,Mars 具有良好的可扩展性,使其在面对不同规模的数据场景时有良好的适应性。

与此同时,由于该统计模型是面向大规模实时流统计场景的,因此对于需要进行迭代计算、复杂计算的流处理场景支持并不完善。该模型的下一步发展和完善应当是面对更多流处理场景,进行通用化拓展。

参考文献:

[1] CHAUHAN J, CHOWDHURY S A, MAKAROFF D. Performance evaluation of Yahoo! S4: a first look[C]//Seventh international conference on P2P, parallel, grid, cloud and internet computing. [s. l.]: IEEE, 2012: 58-65.

[2] NEUMEYER L, ROBBINS B, NAIR A, et al. S4: distributed stream computing platform[C]//Proceedings of the 10th IEEE international conference on data mining workshops. Sydney: IEEE Press, 2010: 170-177.

(下转第 16 页)

码质量检查需要 2 个小时。

工作实践表明,在虚拟云环境中可以根据软件代码量的变化随时改变虚拟机的数量和规格,灵活开展持续集成工作。

6 结束语

云构建环境中开展持续集成工作,从设备层面来讲,可以对私有云的虚拟机进行统一部署、管理与监控,提高资源利用效率。集中管理可以提高管理人员技术水平,提高服务质量,节约人力和物力。在软件产品开发层面,各产品可以根据软件源代码规模灵活申请云虚拟机的数量和规格,提高软件持续集成的速度与效率,从而节约软硬件资源,提高软件开发效率,降低软件开发成本。

参考文献:

[1] 陈国良,明 仲,冯禹洪. 云计算工程[M]. 北京:人民邮电出版社,2016.

[2] 魏志华,赵强强. 构建企业私有云软件测试平台[J]. 电子技术与软件工程,2015(9):63-64.

[3] JUN W,MENG Fanpeng. Software testing based on cloud computing[C]//International conference on internet computing and information services. [s. l.]:[s. n.],2011:176-178.

[4] 孟祥超. 云计算环境下的软件测试服务研究[D]. 大连:大连海事大学,2013.

[5] 严宇平,王学文,陆 璐. 基于云平台的软件自动持续集成研究[J]. 信息通信技术,2014,8(1):50-54.

[6] DUVALL P M,MATYAS S,GLOVER A. 持续集成软件质量改进和风险降低之道[M]. 北京:电子工业出版社,2012.

[7] 兰 洋,温迎福. 持续集成实践[M]. 北京:电子工业出版社,2015.

[8] 姜 文,刘立康. 基于 ClearCase 的软件配置管理与持续集成[J]. 计算机技术与发展,2016,26(1):10-17.

[9] 姜 文,刘立康. 基于 SVN 的软件配置管理和持续集成[J]. 电子设计工程,2016,24(2):1-5.

[10] COLLINS-SUSSMAN B C,FITZPATRICK B W,PILATO C M. Version control with subversion for subversion 1. 5[M]. [s. l.]:[s. n.],2005.

[11] KUNG S,ONKEN L,LARGE S. Tortoise SVN version 1. 5. 2[M]. [s. l.]:[s. n.],2005.

[12] 姜 文,刘立康. 软件配置管理中的基线问题研究[J]. 计算机技术与发展,2016,26(6):6-10.

[13] LOELIGER J,MCCULLOUGH M. Git 版本控制管理[M]. 王迪,丁 彦,译. 第 2 版. 北京:人民邮电出版社,2015.

[14] 王春海. VMware 虚拟化与云计算应用案例详解[M]. 北京:中国铁道出版社,2013.

[15] 张小斌. OpenStack 企业云平台架构与实践[M]. 北京:电子工业出版社,2015.

[16] Amazonvirtual private cloud user guide[R]. [s. l.]:Amazon Web Services,Inc. ,2013.

(上接第 10 页)

[3] SIMONCELLI D,DUSI M,GRINGOLI F,et al. Scaling out the performance of service monitoring applications with BlockMon[C]//Proceedings of the 14th international conference on passive and active measurement. Hong Kong: IEEE Press,2013:253-255.

[4] ZAHARIA M,DAS T,LI H,et al. Discretized streams:an efficient and fault-tolerant model for stream processing on large clusters[C]//USENIX conference on hot topics in cloud computing. [s. l.]:USENIX,2012.

[5] 张 鹏,李鹏霄,任 彦,等. 面向大数据的分布式流处理技术综述[J]. 计算机研究与发展,2014,51:1-9.

[6] 崔星灿,禹晓辉,刘 洋,等. 分布式流处理技术综述[J]. 计算机研究与发展,2015,52(2):318-332.

[7] GÜNDÜZ S,ÖZSU M T. A web page prediction model based on click-stream tree representation of user behavior[C]//Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining. [s. l.]:ACM,2003:535-540.

[8] BABCOCK B,BABU S,DATAR M,et al. Models and issues in data stream systems[C]//Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems. [s. l.]:ACM,2002:1-16.

[9] 孙大为,张广艳,郑纬民. 大数据流式计算:关键技术及系统实例[J]. 软件学报,2014,25(4):839-862.

[10] PERNG C S,WANG H,ZHANG S R,et al. Landmarks:a new model for similarity-based pattern querying in time series databases[C]//16th international conference on data engineering. [s. l.]:IEEE,2000:33-42.

[11] BABCOCK B,DATAR M,MOTWANI R. Sampling from a moving window over streaming data[C]//Proceedings of the thirteenth annual ACM-SIAM symposium on discrete algorithms. [s. l.]:[s. n.],2002:633-634.

[12] ZHU Y,SHASHA D. Statstream:statistical monitoring of thousands of data streams in real time[C]//Proceedings of the 28th international conference on very large data bases. [s. l.]:[s. n.],2002:358-369.

[13] 熊全洪,魏 娟,刘 武. 即席查询研究[J]. 现代商贸工业,2008,20(12):345-346.

[14] CHANDRASEKARAN S,FRANKLIN M J. Streaming queries over streaming data[C]//Proceedings of the 28th international conference on very large data bases. [s. l.]:[s. n.],2002:203-214.

[15] 何小东,尹海波. 基于共享缓冲区的数据流处理框架设计与实现[J]. 计算机工程与设计,2012,33(11):4398-4401.

[16] 陈晗鸣,罗 威,李明辉. 分布式系统中基于主/副版本的实时容错调度综述[J]. 计算机应用研究,2012,29(11):4017-4022.