# 数据需求驱动的Storm应用辅助开发框架

周 雯 史雪菲 吴毅坚 赵文耘

(复旦大学软件学院 上海 201203) (上海市数据科学重点实验室 上海 201203)

**摘 要** Storm支持流式数据的高性能实时计算,是一种广泛使用的流式计算框架。在Storm应用的开发中,开发人员需要针对不同的流式数据需求定制开发相应的计算模块,从而导致大量的重复工作,且难以适应数据需求的变动。如何根据流式数据格式和计算方式等数据需求,快速开发Storm应用并配置相应的环境,是提升大部分流式计算应用开发效率的重要问题。提出了流式数据需求描述方法,设计并实现了一种基于Storm的、由数据需求驱动的流式数据实时处理应用辅助开发框架,其根据业务人员描述的领域数据需求自动生成符合数据处理需求的Storm实时数据处理应用。实验表明,该框架能帮助不具备Storm开发能力甚至非软件开发人员快速配置常见的基于Storm的流式计算应用,对于常见的流式数据的实时处理需求具有一定的适应性。

**关键词** 流式计算,开发框架,数据需求,Storm

**中图法分类号** TP311.5 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.09.012

## Framework Assisting Storm Application Development Driven by Data Requirements

ZHOU Wen SHI Xue-fei WU Yi-jian ZHAO Wen-yun

(Software School, Fudan University, Shanghai 201203, China)

(Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 201203, China)

**Abstract** Storm, a widely used stream calculation framework, supports high efficient real-time calculation for stream data. In the development of Storm applications, developers have to write modules for various stream data requirements, causing repetitive work and difficulties in adapting to changes in data requirements. How to develop Storm applications and configure corresponding environment rapidly based on data requirements such as stream data format and calculations is an important research question for improving the efficiency of stream-oriented application development. An approach for describing stream data requirements was proposed in this paper. A framework assisting Storm application development was designed and implemented for business people to describe domain-specific data requirements and generate Storm applications automatically. Experiments show that the framework is able to help non-developers configure and deploy common Storm-based stream calculation applications. The framework is adaptive to common requirements in real-time stream data calculations.

**Keywords** Stream calculation, Development framework, Data requirements, Storm

## 1 引言

随着互联网、物联网和大数据的发展,流式数据处理应用日益广泛。流式数据具有无限性、实时性、易失性和无序性等特征[1]。流式数据由数据源实时产生,数据价值的时效性短,处理的实时性要求高,一般数据到达后直接在内存中进行计算。流式数据实时处理适用于计算精度要求相对宽松、响应时间要求较高的场景,如监测数据的在线收集、社交网站的主题推荐、网站页面的访问统计等。

传统上,为这些业务场景提供实时数据处理服务需要在并发传输、实时性等方面耗费大量的开发投入。而采用流式计算框架后,软件开发的工作主要集中在收集和解析数据、分组计算等业务需求方面,而数据的并发、分布式处理、容错等基础性的功能则由所选的流式计算框架来完成,从而提升了流式数据实时处理应用的开发效率。

然而在实际应用中,流式数据实时处理应用的开发往往基于相应的开发框架和编程规范,由开发人员根据业务需求手工编写相应的数据处理程序单元。即使简单的数据计算模

到稿日期:2017-10-05 返修日期:2017-12-29 本文受上海市科技发展基金项目(16JC1400801)资助。

**周 雯**(1992-),女,硕士,主要研究方向为流式大数据和软件开发平台;**史雪菲**(1992-),女,硕士,主要研究方向为流式大数据和软件开发平台、内存计算系统软件;**吴毅坚**(1979-),男,博士,副教授,主要研究方向为软件维护与演化、大数据应用开发平台,E-mail:wuyijian@fudan.edu.cn(通信作者);**赵文耘**(1964-),男,博士,教授,博士生导师,主要研究方向为软件工程、企业应用集成、软件开发平台、大数据等。

块也需要由专门的软件开发人员手工编写并部署到流式计算环境中。这对于一些需要将遗留应用迁移到新型流式计算环境下的用户而言,往往难以实施。虽然现有的 Storm[2],S4[3]等流式计算框架已经提供了丰富的开发支持,但非专业开发人员在开发和部署过程中仍有较高的学习成本,无法快速搭建流式数据实时处理应用。

Storm 作为一种广泛应用的流式计算框架,提供了灵活的编程结构,便于开发者开发。与各类应用的软件开发类似,Storm 应用的开发也需要具有相应开发经验的软件开发人员和业务人员的密切配合。而在运行过程中,任何数据的格式变化、计算要求的变化或者数据异常情况,都有可能需要开发人员对相应的计算模块进行修改,降低了开发和部署效率。在很多实际应用场景中,数据需求虽然千变万化,但实时处理的总体流程和计算模式却很少发生变化。因此,在常见的典型应用场景下,如何能不依赖于软件开发人员,而仅仅根据描述数据格式和计算要求等数据需求,快速配置出可用的流式数据实时处理环境,对于提升实时数据处理应用的开发效率具有重要的意义。

本文提出一种基于 Storm 的、由数据需求驱动的流式数据实时处理应用辅助开发框架,其面向业务人员而非专业开发人员提供领域数据需求描述工具,进而自动生成符合数据处理需求的 Storm 实时数据处理应用。该开发框架简化了实时流式数据环境的配置过程和应用开发过程,降低了应用开发的技术要求,提升了基于 Storm 的流式数据实时处理应用的开发效率,对于常见的数据实时处理需求具有较好的适应性。

本文第 2 节介绍了流式数据处理和围绕 Storm 的相关研究工作;第 3 节给出了两种典型的快速开发 Storm 应用的需求场景;第 4 节给出了流式数据需求定义和辅助开发框架的总体设计;第 5 节阐述了框架的实现并对其运行效果进行了评价分析;最后总结全文并展望下一步的工作方向。

## 2 相关工作

随着大数据技术的发展和实时处理需求的增加,业界研发了多种流式数据处理框架,包括 Twitter 的 Storm、Yahoo 的 S4、LinkedIn 的 Samza、Twitter 的 Heron[4]、Google 的 MillWheel[5]和 Microsoft 的 Timestream[6]等。此外,一些流处理和批处理混合的计算框架受到越来越多的重视,如 Berkeley 的 Spark Streaming[7]和 Apache Flink 等。这些计算框架在数据处理的实时性、可扩展性、容错性等方面具有各自的特点。其中,Storm 是一个免费开源、分布式、高容错的实时计算系统,常用于实时分析、持续计算、分布式远程调用和 ETL 等领域,是各大企业流式数据实时处理应用的常用框架之一。

随着 Storm 在流式数据实时处理方面的广泛应用,有关 Storm 的研究也越来越多,但主要集中在性能优化、任务调度等运行方面。Papageorgiou 等[8]基于 Topology 与外部物联网实体之间的交互,提出了边缘计算感知部署优化算法,减少了 Topology 边缘的处理延迟和带宽消耗。Aniello 等[9]提出了 Topology 的部署优化策略,即通过持续监测系统负载,调度组件在可用的计算设施上执行,以减少 Storm 的处理延迟。Xin 等[10]为 Storm 流处理架构设计了分布式的 QoS 感知调度器,提高了 Storm 运行时的自适应能力。Xiong 等[11]提出了 Topology 热边的调度算法,将与热边关联的执行器(Executor)分配到同一个节点执行,减少了节点间的通信。Li 等[12]根据拓扑感知方法的性能预测结果,设计了预测性的 Storm 调度框架,减少了数据元组(Tuple)的平均处理时间。

虽然 Storm 在架构优化和效率优化方面有大量研究,但是针对 Storm 应用开发技术的研究相对较少,特别是没有针对面向数据需求的开发框架的研究。Santurkar 等[13]提出通过 DSL 自动生成 Storm Topology 代码,但是 DSL 的定义仍要求开发人员具备一定的 Storm 开发经验。孙朝华[14]设计并实现了一种基于 Storm 的数据分析系统,在系统中引入了 Spring 编程模型和并行化的聚类方法。龙少杭[15]提出了 Storm 实时数据分析系统,通过设计相应的优化机制和集成开发库来提升系统的性能和可用性。但这些系统仍然侧重于流式数据应用场景和运行效率的提升,并非致力于简化常见流式数据实时处理应用的开发过程。如何简化 Storm 应用的开发过程,降低 Storm 应用开发的技术门槛,并面向业务人员快速搭建基于 Storm 的常见实时数据处理平台,是本文关注的核心问题。

## 3 Storm 应用开发的典型案例分析

本节通过两个典型的应用案例来展示基于 Storm 的流式数据实时处理应用在开发过程中的相似性。

案例 1:某市对环境数据的实时监测(见图 1(a))。在全市部署了上千个监测点,每个监测点由多组传感器对不同的环境因素(如噪音、粉尘等)进行实时监测,监测数据通过移动网络按一定的时间间隔(如 1 min)实时发送给中心服务器。用户除查看实时数据外,还需要查看各种环境数据在过去 1 h,2 h,4 h 的动态均值的变化情况。当出现数据丢失或数据异常时,需要进行相应的异常处理。典型的实时环境数据由多层结构的键值对组成,例如 s182-p0,2017-06-13 14:12:10,{Pudong:{Noise:55,Dust:0.1}}。开发人员需要开发相应的数据解析程序,将数据按键值拆分,并进行实时计算,最后将计算结果(滚动时间窗口的测量值的均值)以折线图的方式进行可视化。

案例 2:微博热词的实时统计(见图 1(b))。在某微博系统中,每个用户发布的博文是由单词组成的文本数据,形如 uid,13/06/2017 14:12:10,The 2016 Olympics was held in Rio.。平台需要在任意时刻都能实时计算过去 2 h 内用户发布的微博中出现频率较高的词,并将这些词作为热搜关键词推荐给微博用户。这需要开发人员开发相应的微博文本解析程序,对每个词分别统计过去 2 h 出现的次数,最后将这些词按出现次数由高到低排列,选出前 10 个作为热搜关键词,并展示在界面上。

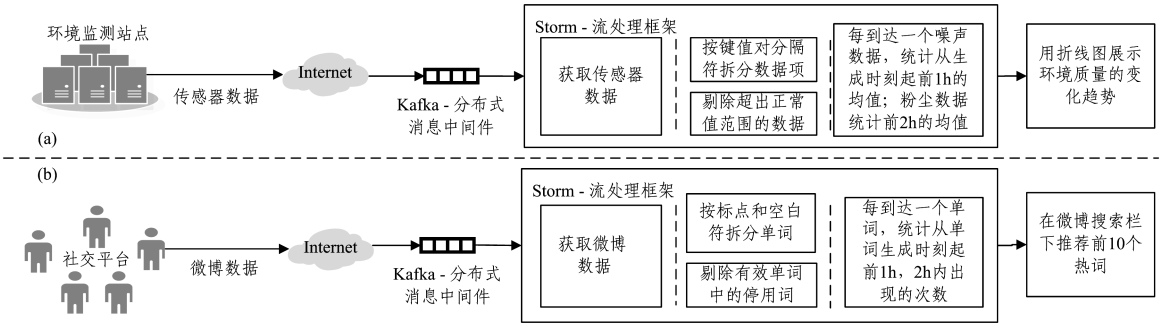


图 1　两种应用场景下的流式数据实时处理过程

Fig. 1　Real-time processing of stream data in two application scenarios

上述两个人工开发 Storm 流式计算应用的案例具有类似的过程:1)使用 KafkaSpout API 创建 Spout,用于获取数据;2)根据业务需求创建两个 Bolt,分别用于数据解析和计算,Bolt 需要实现 IRichBolt 接口并重写 prepare,declareOutputFields 和 execute 等方法;3)将 Spout 和 Bolt 组装成 Topology。

从这两个典型案例的业务需求和开发过程可以看出,不同场景下的流式数据实时处理应用,在处理步骤和实现框架上非常相似,但每个步骤的具体实现又存在源于具体业务中的数据需求的多种细微差异。因此,准确地描述流式数据处理过程和处理方式等数据需求,是提高流式数据实时处理应用的开发效率的关键。

## 4　流式数据的处理过程和框架设计

### 4.1　流式数据处理过程

上述案例分析揭示了流式数据实时处理总体过程的一致性。该过程主要分为以下 4 个步骤(见图 2)。

1)数据接入。数据通过统一接口接入消息中间件,并作为原始数据流进入流处理系统。数据源可以是现有生产系统中产生的数据,也可以是根据用户配置模拟生成的数据。

2)数据预处理。根据用户定义的数据格式和范围约束,将多层次结构的原始数据解析成若干系统可以处理的单层次数据项,检查数据项的取值范围,处理异常值。

3)数据处理。按一定规则对预处理后的数据进行分组,针对同一组中的数据流,按用户指定的计算需求进行实时计算,得出统计结果。

4)数据输出。将结果持久化到数据库或文件中,以供后续进一步分析和挖掘;也可将计算结果以可视化的方式展示给用户。

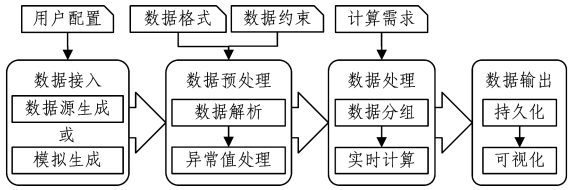


图 2　流式数据实时处理过程

Fig. 2　Real-time stream data processing

### 4.2　基本开发框架的设计

#### 4.2.1　Topology 基础结构的设计

基于上述常见流式数据处理过程的分析,基于 Storm 框架开发的基础连接拓扑结构(Topology)的设计如图 3 所示。虽然数据处理的场景不同,但处理的过程相似,Topology 的基本处理组件也是相同的。

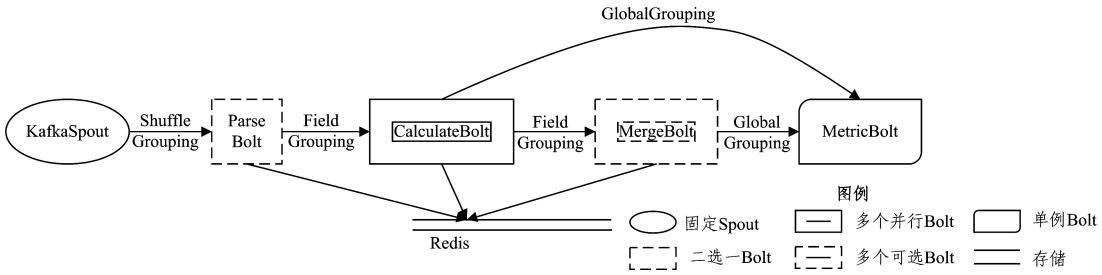


图 3　基于 Storm 的基础拓扑结构设计

Fig. 3　Basic topology design based on Storm

数据读取模块 KafkaSpout,是 Storm 框架提供的数据生成单元,可以不断地从 Kafka 消息队列中拉取数据,并向后续的 Bolt 实例随机发送这些数据流元组(Tuple)。

数据解析模块 ParseBolt,根据数据格式有两种实现。ParseBolt 解析数据后检查数据项的有效性,分发数据流时使用计算需求中该 Tuple 对应的算法名来标识 stream-id,使其流向对应算法的 CalculateBolt 实例。

数据计算模块 CalculateBolt 和 MergeBolt。计算需求中的每种算法都需要实例化为一个 CalculateBolt,每个实例只执行一种算法。MergeBolt 是可选的计算 Bolt,如果存在一组数据项有合并计算的需求,则在对应算法的 CalculateBolt 后加入 MergeBolt,实现该组数据的综合计算。CalculateBolt 实例分发数据流时,若没有合并计算需求,则会直接流向最后一个 Bolt;否则,会流向对应算法的 MergeBolt。

数据量统计模块 MetricBolt。所有成功计算的数据流都汇聚到单一的 MetricBolt 实例中,用于周期性地统计 Topology 成功处理的数据量,一般用作性能评价。

Topology 基础结构中,KafkaSpout 和 MetricBolt 都是固

定的处理组件,不因业务需求的变化而变化。而 ParseBolt 组件根据数据格式选择一个模板实例化,CalculateBolt 和 MergeBolt 组件根据计算需求实例化为不同的计算 Bolt,最终影响到生成的 Topology 结构。因此,数据解析和数据计算模块是代码生成的核心。

#### 4.2.2 数据规范需求

数据解析是基于数据格式和相关约束实现的。因此,需要描述数据基本语义信息和规范性需求,以便开发框架适应不同业务数据的自动解析、过滤和分析。

数据规范需求描述的内容主要包括数据的组成元素、数据格式、取值约束,以及它们之间的关系,如图 4 所示。其中,Data 表示原始数据,由数据源标识(sourceId)、生成时间(dateTime)和数据体(Content) 3 个基本元素组成。若干键值对形式的数据项(Item)组成了数据体内容,Item 的 Value 可以是基本数据类型,也可以是多个 Item 组成的下一层数据。这种 Item 之间的组合嵌套关系形成了数据的层次结构,可以用数据格式(Data Format,DF)来描述。

DF 定义了数据的基本格式(format)和每个层次的格式(Layer Format,LF)。其中,format 描述了数据的 3 个基本元素的顺序和分隔符;LF 描述了当前层次的起止符号、Item 之间的分隔符、键值的分隔符,以及键值的存在情况。

数据取值约束(Constraint,DC)是对 Item 有效取值范围的描述,包括数据项的最大最小值和异常值列表。异常值一般为超出正常值范围的特殊码,表示该时刻无数据项或者指明故障原因。

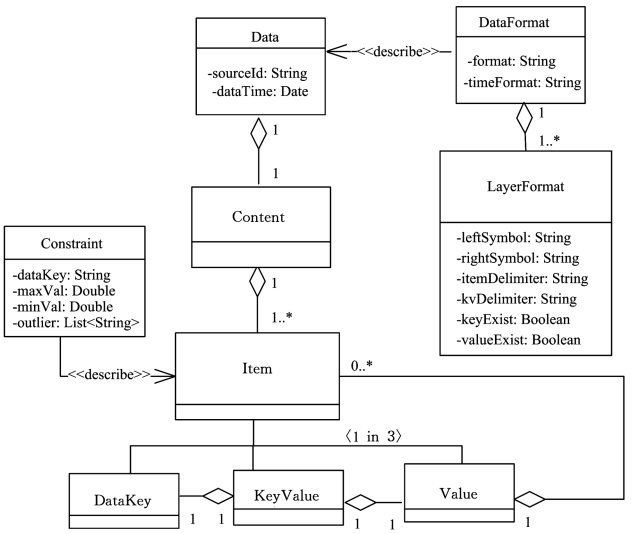


图 4 数据规范需求的设计模型

Fig. 4 Design model for data specification requirements

上述设计为了满足无 Storm 开发经验的业务人员快速描述数据需求,采用了业务人员常用的术语,并且采用示例标注和辅助工具的方法帮助业务人员定义数据需求,从而提升了定义数据需求的易用性。以环境监测数据为例,各站点的数据格式形如:[s182-p0],[2017-06-13 14:12:10],{Pudong,{Noise,55,Dust,0.1}},相应的 DF 和 DC 的 XML 描述为:

```
〈dataFormat〉
  〈format〉[sourceId],[datetime],content〈/format〉
  〈contentFormat〉
    〈layerFormat leftSymbol="{" rightSymbol="}"
      itemDelimiter="," kvDelimiter=","
      keyExist="true" valueExist="true"〉
      〈layerFormat leftSymbol="{" rightSymbol="}"
        itemDelimiter="," kvDelimiter=","
        keyExist="true" valueExist="true"/〉
    〈/layerFormat〉
  〈/contentFormat〉
〈/dataFormat〉
〈dataConstraint〉
  〈constraint dataKey="Noise" minVal="20" maxVal="120"
   outlier="[-1000,9999]"/〉
  〈constraint dataKey="Dust" minVal="0.01" maxVal="0.5"
   outlier="[-1000]"/〉
〈/dataConstraint〉
```

#### 4.2.3 数据计算需求

在 Storm 流处理框架中,数据计算需求(CR)定义了 Storm 拓扑结构的各计算单元(Bolt)之间交换数据以及消费数据的方式。

Storm 中数据交换方式又称为数据流分组(grouping)。CR 描述了指定数据源(sourceId)和数据项名称(dataKey)的处理方式,这两个域控制了数据流的分组情况:相同域组合的数据归集起来发送给同一个 Bolt 的任务进行处理。在某些需求下,还需要对部分数据源的数据进行归并计算,因此允许使用模糊匹配的方式来描述数据源,并设计 combineCal 布尔参数来决定是否对相应数据项进行合并计算。

数据消费方式表现为实时计算的实现逻辑。CR 定义了计算方法(algorithm)、统计时间段(interval),以及统计执行的最少数据量(threshold)。计算方法的统一公式表示如下:

$$index_{i,t}=func_{\text{algorithm}}(\{Item_{i,t_1},\cdots,Item_{i,t_n}\},threshold)$$

其中,$Item_{i,t_k}$ 表示数据项 $i$ 在 $t_k$ 时刻的取值,$t_k\in(t-interval,t]$,algorithm 为系统内置算法或用户自定义算法。当数据缺失导致数据量 $n<threshold$ 时,暂不执行统计。

仍然以环境监测数据的实时统计为例,用户提供的计算需求为:对于所有名称包含 s182-p0 的设备,计算噪声(Noise)数据在过去 1h 内的最大值;对于每个名称包含 s184-p1 的设备,分别计算粉尘(Dust)数据在过去 2h 内的算术平均值。

相应计算需求(CR)的 XML 定义为:

```
〈processReqs〉
  〈processReq sourceId="s182-p0" dataKey="Noise"
    fuzzyMatch="true" combineCal="true"
    algorithm="Maximum" interval="1" threshold="100"/〉
    〈processReq sourceId="s184-p1" dataKey="Dust"
      fuzzyMatch="true" combineCal="false"
      algorithm="Mean" interval="2" threshold="80"/〉
〈/processReqs〉
```

## 5 Storm 应用辅助开发框架的实现

### 5.1 总体实现思路

首先,用户通过配置界面提供数据规范需求和数据计算需求,生成 DF,DC 和 CR 3 类配置文件。其次,由配置文件生成数据解析模块和数据计算模块,并选择性生成合并计算模块。最后,通过合适的数据流分组策略,将以上 3 个模块组装到 Topology 的基础结构中,形成可执行的 Java 项目,打包部署到 Storm 集群中运行。需求配置和代码生成的过程如图 5 所示。

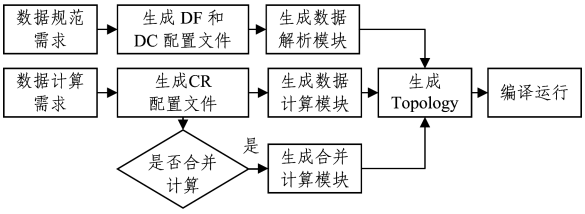


图 5　需求配置和代码生成流程图

Fig. 5　Flowchart of requirements configuration and code generation

### 5.2　生成数据解析代码

为了支持数据的多层结构解析,在辅助开发框架中引入了模板化的数据解析模块(ParseBolt);同时,考虑到数据解析的性能,采用了基于 DF 定义自动生成硬编码的数据解析代码。

#### 5.2.1　数据解析模板

针对键值对形式的数据和文本类型的数据,ParseBolt 有两种实现模板,即 KeyValueParseBolt 和 TextParseBolt,其核心都是根据 DF 将多层结构的原始数据分解为单层结构的数据项。区别在于,KeyValueParseBolt 拆分得到 dataKey 和 value,而 TextParseBolt 拆分只得到 dataKey,value 则默认为 1。

任意格式的数据经 ParseBolt 都被解析成数据项,表示为〈sourceId,dataKey,time,value〉四元组的形式。ParseBolt 根据 DC 和 CR 对数据项的有效性进行检查,过滤掉异常值和无需处理的数据项,并记录在日志中。清理后的数据项以 CR 中指定的算法名作为 stream-id,分发给对应的计算 Bolt。

#### 5.2.2　数据解析模块生成

数据解析模块的生成主要是根据最里层的 Item 的键值存在情况选择合适的 ParseBolt 模板,再根据 CR 对模板中的 stream-id 声明进行修改,图 6 为修改部分对应关系的示意图。例如,环境数据的 DF 定义中最里层的 Item 同时存在键值(即环境因素及其取值),使用 KeyValueParseBolt 模板;而微博数据最里层 Item 只有键(即单词),使用 TextParseBolt 模板。

```
/* KeyValueParseBolt Template */
public class <KeyValueParseBoltTemplate>
implements IRichBolt {
    ......
    public void execute(Tuple input) {
        /* Implements data parser */
        ......
        collector.emit(algorithm, new
Values(sourceId,name,time,value));
        ......
    }
    public void declareOutputFields (
OutputFieldsDeclarer declarer) {
        /* Declare stream-id: all algorithm in
CR*/
        declarer.declareStream("default", new
Fields("sourceId", "dataKey", "time",
"value"));
    }
    ......
}
```

(a)模板代码

```
/* Auto-generated bolt name */
public class ParseBolt implements IRichBolt {
    ......
    public void execute(Tuple input) {
        /* Implements data parser */
        ......
        collector.emit(algorithm, new
Values(sourceId,name,time,value));
        ......
    }
    public void declareOutputFields (
OutputFieldsDeclarer declarer) {
        /* Auto-generated stream-id declare*/
        declarer.declareStream("Maximum", new
Fields("sourceId", "dataName", "time",
"value"));

        declarer.declareStream("Mean", new
Fields("sourceId", "dataKey", "time", "value"));
    }
    ......
}
```

(b)替换和展开后的实例代码

图 6　ParseBolt 的模板代码替换示意

Fig. 6　Code replacement of ParseBolt template

### 5.3　生成数据计算代码

实际的业务需求对不同数据有不同的计算要求,但常见的计算需求具有一定的共性(如统计一段时间内某些数据项的度量指标),涉及的算法也都相对简单(如累计值、均值等)。因此,首先需要定义常见的基础算法;其次设计对不同计算需求通用的数据计算模板;最后基于上述计算模板和计算需求,生成数据计算模块(CalculateBolt)。

#### 5.3.1　基础算法

在调研了常见需求的基础上,辅助开发框架内置了累加和、均值、方差等 5 种基础算法,且都实现了 Algorithm 接口(见图 7)。基于该接口还可自行开发其他算法,扩展 Storm 应用的处理能力。

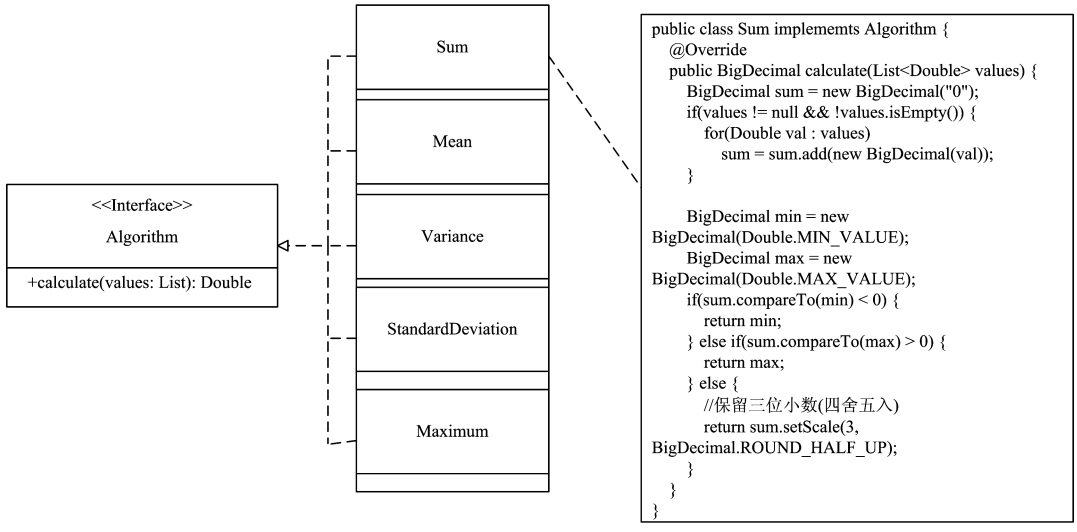


图 7　基础算法和接口实现

Fig. 7　Basic algorithm and interface implementation

#### 5.3.2　数据计算模板

数据计算模板有:CalculateBolt 和 MergeBolt。由于在实际业务需求中,每个数据项都有固定的统计时间段(interval),CalculateBolt 的实现采用了滑动窗口的方法,将 interval 作为固定的窗口长度,每接收到一个数据项(Tuple),就将其放入当前窗口中,同时将窗口向前滑动,并清除窗口尾部超出 interval 范围的 Tuple。每个 CalculateBolt 实例只实现一种算法,并且对每个需要处理的数据项都维护一个窗口。因此,对于同一个数据项在不同统计时间段内的计算,需要为该数据项维护一个最长的窗口(maxInterval),例如计算 s182-p0 设备的 Noise 数据项在 1h,2h 和 4h 内的均值,需要为 s182-p0-Noise 维护 4h 的窗口长度,即最近 4h 内该数据项的集合。在

实际计算中，CalculateBolt 按 CR 从数据项对应的窗口中取出时间长度为 interval 的数据集，并按指定的算法对数据集进行计算。CalculateBolt 核心伪代码的实现如算法 1 所示。

**算法 1** CalculateBolt 算法

输入：CR，tuple(表示数据项，为〈sourceId，dataKey，time，value〉四元组的形式)

输出：新的 tuple(表示数据项的度量指标，为〈srcIdInCR，dataKey，time，index〉四元组的形式)

t←$time_{tuple}$；
获取 tuple 的最大窗口长度(maxInterval)；
将 tuple 添加到对应的 window 中；
for $tuple_k$∈window do
  if $time_{tuple_k}$≤t−maxInterval then
    remove $tuple_k$；
for $CR_{tuple}$〈interval，threshold〉 do
  valueList←window 中选取生成时间在(t−interval，t]内的 tuple 值集；
  if count(valueList)≥threshold then
    index←algorithm. calculate(valueList)；
    发送新的 tuple 给下一个 Bolt。

对于有合并计算需求的数据项，首先需要由 CalculateBolt 处理后得到各自的统计指标，再经过 MergeBolt 进一步对这些指标进行合并计算，得到多个数据项的综合统计结果。例如，计算浦东新区所有监测点的 Noise 均值，首先需要计算张江镇、北蔡镇等多个监测点的 Noise 均值，再将这些站点的均值进行合并计算，得到整个浦东新区的 Noise 均值。MergeBolt 为每个合并计算数据项组都维护一个窗口，其核心是将窗口中的数据项时间对齐，其余实现与 CalculateBolt 类似，核心部分的伪代码实现如算法 2 所示。

**算法 2** MergeBolt 算法

输入：CR，tuple(表示数据项的度量指标，为〈srcIdInCR，dataKey，time，index〉四元组的形式)

输出：新的 tuple(表示合并计算数据项组的综合度量指标，为〈srcIdInCR，dataKey，time，comIndex〉四元组的形式)

t←$time_{tuple}$；
if CR 中定义了该数据项的合并计算需求 then
  将 tuple 添加到对应的 window；
else 结束；
for $tuple_k$∈window do
  if $time_{tuple_k}$<t then
    remove $tuple_k$；
for $CR_{tuple}$〈interval〉 do
  valueList←window 所有 tuple 的值集；
  comIndex←algorithm. calculate(valueList)；
  发送新的 tuple 给下一个 Bolt.

5.3.3 数据计算模块生成

数据计算模块生成的主要任务是为 CR 中的每种 Algorithm 都生成一个 CalculateBolt 实例，再根据合并计算需求选择性生成对应算法的 MergeBolt 实例，具体实现只要实例化模板中的 Algorithm 即可。对于有合并计算需求的 CalculateBolt 实例，还需修改模板中的 stream-id 声明，使输出流分发给相应的 MergeBolt，修改部分如图 8 所示。

```
/* CalculateBolt Template */
public class _CalculateBoltTemplate_
implements IRichBolt {
  ...
  public void execute(Tuple input) {
    ...
    /* Instantiate algorithm */
    BigDecimal index =
algo.calculate(valueList);
    ...
  }
  public void declareOutputFields(
OutputFieldsDeclarer declarer) {
    /* Declare stream-id: algorithm name
*/
    declarer.declareStream("default",new
Fields("srcIdInRC", "dataKey", "time",
"index"));
  }
  ...
}
```

(a)CalculateBolt 的模板代码

```
/* Auto-generated bolt name */
public class MaximumBolt implements
IRichBolt {
  ...
  public void execute(Tuple input) {
    ...
    /* Auto-generated algorithm instance
*/
    Algorithm algo = new Maximum();
    BigDecimal index =
algo.calculate(valueList);
    ...
  }
  public void declareOutputFields(
OutputFieldsDeclarer declarer) {
    /* Auto-generated stream-id declare*/
    declarer.declareStream("Maximum",
new Fields("srcIdInRC", "dataKey", "time",
"index"));
  }
  ...
}
```

(b)实例化的计算单元 SumBolt

图 8 CalculateBolt 模板的代码替换

Fig. 8 Code replacement of CalculateBolt template

## 5.4 计算需求实例化(生成 Topology 代码)

基于上述生成的数据解析和数据计算模块，以及 Topology 的基础结构，本节将 CR 的半结构化描述实例化为具体的 Topology 代码。Topology 代码的生成效果如图 9 所示，其结构可以用图 10 来描述。

```
/* Topology Template */
public class _TopologyTemplate_ {
  public static void main(String[] args) {
    ...
    TopologyBuilder builder = new TopologyBuilder();
    builder.setSpout("kafka-spout", new
KafkaSpout(spoutConfig), parall1);
    /* Insert bolt here */
    ...
    config.put("CR", xml2Json(CR));
    config.put("DF", xml2Json(DF));
    config.put("DC", xml2Json(DC));
    StormSubmitter.submitTopology("topoName",
config, builder.createTopology());
  }
}
```

(a)Topology 模板代码

```
public class TopologyMain {
  public static void main(String[] args) {
    ...
    TopologyBuilder builder = new TopologyBuilder();
    builder.setSpout("kafka-spout", new KafkaSpout(spoutConfig), parall1);
    /* Auto-generated bolt */
    builder.setBolt("parse-bolt", new ParseBolt(), parall2)
      .shuffleGrouping("kafka-spout");
    builder.setBolt("maximum-bolt",new MaximumBolt(), parallelism3)
      .fieldsGrouping("parse-bolt", "Maximum", new Fields("srcIdInCR", "dataKey"));
    builder.setBolt("mean-bolt",new MeanBolt(), parall3)
      .fieldsGrouping("parse-bolt", "Mean", new Fields("srcIdInCR", "dataKey"));
    builder.setBolt("maximum-merge-bolt",new MaximumMergeBolt(), parall4)
      .fieldsGrouping("maximum-bolt", "Maximum", new Fields("srcIdInCR", "dataKey"));
    builder.setBolt("metric-bolt",new MetricBolt())
      .globalGrouping("maximum-merge-bolt")
      .globalGrouping("mean-bolt");
    ...
  }
}
```

(b)实例化后的 Topology 代码

图 9 Topology 模板的代码替换

Fig. 9 Code replacement of Topology template

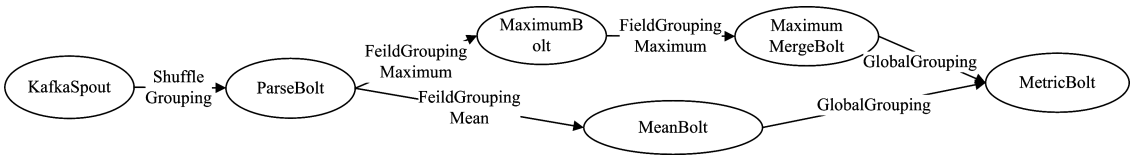


图 10　自动生成的 Topology 结构示意图

Fig. 10　Auto-generated topology instance

数据计算模块使用〈sourceId,dataKey〉或〈srcIdInRC,dataKey〉域组合对输出数据流进行分组,相同域组合的 Tuple 被分发到同一个计算 Bolt 实例的任务中进行处理,以保证数据项窗口的数据完整性。

### 5.5　实现效果和评价

#### 5.5.1　研究问题

研究问题 1(RQ1)　有效性:为研究上述 Storm 应用辅助开发框架能否支持不了解 Storm 编程的业务人员快速开发和配置 Storm 应用,需要选择没有 Storm 编程经验但理解基本流式数据处理概念的实验对象(人员),在采用本框架辅助开发的情况下开发指定的 Storm 应用。由于学习 Storm 开发本身是一个系统性的学习任务,没有可重复性,因此实验中并没有通过对比实验的方式来说明采用框架和不采用框架之间的学习效率的差异,而是仅通过验证性实验的方式研究采用 Storm 应用辅助开发框架能否支持非 Storm 技术人员的快速开发。

研究问题 2(RQ2)　适应性:即研究对于复杂的 Storm 应用,开发框架能否提供足够的适应性支持。由于本文所研发的 Storm 辅助开发框架在已实现的算法和模块中仅覆盖了数学统计等简单算法,一般认为可适用于物联网、互联网数据的监控。而在金融风控等领域需要用到实时数据关联分析等复杂的算法,由于无法预先给出具体实现,因此也无法预先集成在开发框架中。因此,如何通过框架的可扩展算法接口实现复杂的自定义算法及判断其实现效果,则需要通过实验来进行研究。

#### 5.5.2　针对 RQ1(有效性)的实验

实验选择 2 名有 Storm 开发经验和 4 名无 Storm 开发经验但有 Java 开发基础的计算机专业的硕士研究生,针对环境监测数据实时统计的应用需求开发相应的 Storm 应用。考虑到在现实中一些业务人员本身也具有计算机技术背景,但仍然面临着开发技术问题而无法开展相应的 Storm 应用开发,因此,将计算机专业的硕士研究生作为实验对象,在一定程度上能够说明辅助开发框架对于无 Storm 开发基础的业务人员的辅助效果。

选定了实验对象后,首先向实验对象介绍具体的业务需求和数据格式,并要求实验对象采用我们的数据需求描述工具和 6 台云服务器搭建的 Storm 标准测试环境,在 2h 内开发出具有粉尘、噪音、温度、风力 24h 监控的实时数据处理应用。为了确保实验对象了解业务需求,我们没有对业务介绍环节做任何时间限制,并且在实验对象进行开发时,若有任何业务问题也可向我们提出。但是,为了验证辅助开发框架本身对于辅助开发的效果,不对实验对象进行配置工具的细节培训。实验对象可以自行尝试配置、运行、重配置、再运行,通过"试错"的方式完成所要求的开发功能。

实验内容:给定数据样例和处理需求,2 名专业 Storm 开发人员编写 Storm 应用代码,4 名非 Storm 开发人员通过框架生成应用。数据样例形如[2017-10-01 12:05:00],s182-0,{pudong:{noise:15.6,dust:33.8,temperature:28.7}}。实验处理需求为:对于每个名称包含 s182 的设备,分别计算在过去 1h 内浦东地区噪声(pudong.noise)的均值(数据量达到 90%开始计算),在过去 4h 内浦东地区温度(pudong.temperature)的最小值;对于所有名称包含 s184 的设备,计算在过去 8h 内杨浦地区粉尘(yangpu.dust)数据的最大值。

实验结果:2 名熟悉 Storm 编程的开发人员和 3 名非 Storm 开发人员分别在 2h 内完成了符合需求的 Storm 应用开发,并且使其在模拟数据下成功运行。其中 1 名开发人员根据处理需求生成了两个 Storm 应用,实际上可以只生成一个。还有 1 名非 Storm 开发人员由于参数配置错误,导致计算结果不正确,没有在规定时间内完成。

在上述实验中,3 名(75%)非 Storm 开发人员能够在没有任何编程技术帮助的情况下,在 2h 内配置出可运行的流式计算应用,因此可以认为 Storm 应用辅助开发框架的相关工具能提供有效的开发支持。

#### 5.5.3　针对 RQ2(适应性)的实验

针对复杂的计算需求,Storm 应用辅助开发框架提供了开放的算法接口,需要开发人员自行编写符合接口的 Java 程序。

实验内容:给定新的数据样例和处理需求,2 名有工具使用经验的实验人员需要编写符合接口的算法,并加入到辅助开发框架中,从而生成相应的 Storm 应用。数据样例形如:10.131.223.65 - - [01/10/2017 18:01:22] "GET/news.htm 200 http://www.mypage.com/ v0772"。实验处理需求为:统计网页 www.mypage.com 每小时的访问流量,即计算过去 1h 内的 *PV* 值和 *UV* 值。

实验结果:通过自定义算法(*PV* 值表示直接统计的网页的访问次数,*UV* 值表示对网页的访问者 id 去重后再统计的访问次数),2 名实验人员分别配置出了满足需要的 Storm 应用。

由于复杂数据需求的不确定性,目前尚未能一一枚举实验。同时,由于具体算法的资源需求不同,对整个辅助开发框架的配置要求也更多。因此,目前我们仅设计了符合 Algorithm 接口的常见算法,且这些算法本身也是通过实现可扩展

接口并根据实际计算需求逐步增加到辅助开发框架中的,可见该设计本身就具有可扩展性和适应不同需求的能力。

5.5.4 讨论

由于辅助开发框架可以适应多种数据格式和处理需求,在需求比较简单的情况下,专业 Storm 开发人员编写的代码比框架生成的代码更加简洁。但是在需求不断变化的情况下,使用框架生成应用具有更高的开发效率,也缩短了程序调试过程耗费的时间。

第一次使用框架开发 Storm 应用的实验人员,可能会因为参数配置错误导致生成的代码出现异常,因此提高生成代码的健壮性和优化界面的配置操作,是未来改进的方向之一。为了减少人为的误操作,配置界面应该尽可能提供选项和默认值,而不是让用户输入。配置过程也可以进一步分为基本配置和高级配置,以便扩展。对于类型更加复杂的计算需求,希望在应用实践中对辅助开发框架进行持续维护和增强,从而使其适应更多不同类型的数据需求。

**结束语** 本文面向业务需求人员,设计并实现了数据需求驱动的 Storm 应用辅助开发框架。通过对数据规范需求和数据计算需求的建模和描述,以及对 Storm 编程单元的模板化设计,实现了一个 Storm 应用的辅助开发框架,从而缩短了流式数据实时处理应用的开发周期,降低了 Storm 应用开发的技术门槛。开发实验表明了该辅助开发框架和相关工具对于简化应用开发部署过程的有效性,以及应对多种数据需求的部分适应性。在后续工作中,将进一步根据数据规模动态地调整 Storm 应用代码中 Topology 的并行度(Worker 数量)以及每个处理组件的并行度(Executor 和 Task 的数量),从而优化不同负载下的处理性能问题。

## 参 考 文 献


[1] SUN D W, ZHANG G Y, ZHENG W M. Stream Computing in Big Data Environment: Key Technologies and System Examples [J]. Journal of Software, 2014, 25(4): 839-862. (in Chinese)
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.

[2] TOSHNIWAL A, TANEJA S, SHUKLA A, et al. Storm@twitter[C]// Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2014: 147-156.

[3] NEUMEYER L, ROBBINS B, NAIR A, et al. S4: Distributed stream computing platform[C]// The 10th IEEE International Conference on Data Mining Workshops. Washington: IEEE Computer Society, 2010: 170-177.

[4] KULKARNI S, BHAGAT N, FU M, et al. Twitter Heron: Stream Processing at Scale[C]// Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2015: 239-250.

[5] AKIDAU T, BALIKOV A, BEKIROGLU K, et al. MillWheel: Fault-Tolerant Stream Processing at Internet Scale[J]. Proceedings of the Vldb Endowment, 2013, 6(11): 1033-1044.

[6] QIAN Z, HE Y, SU C, et al. TimeStream: reliable stream computation in the cloud[C]// Proceedings of the 8th ACM European Conference on Computer Systems. New York: ACM, 2013: 1-14.

[7] ZAHARIA M, DAS T, LI H, et al. Discretized streams: fault-tolerant streaming computation at scale[C]// ACM SIGOPS 24th Symposium on Operating Systems Principles. New York: ACM, 2013: 423-438.

[8] PAPAGEORGIOU A, POORMOHAMMADY E, CHENG B. Edge-Computing-Aware Deployment of Stream Processing Tasks Based on Topology-External Information: Model, Algorithms, and a Storm-Based Prototype[C]// 2016 IEEE International Congress on Big Data. Washington: IEEE, 2016: 259-266.

[9] ANIELLO L, BALDONI R, QUERZONI L. Adaptive online scheduling in storm[C]// The 7th ACM International Conference on Distributed Event-Based Systems. New York: ACM, 2013: 207-218.

[10] XIN Q, YAO X. Distributed QoS-Aware Scheduling in Cognitive Radio Cellular Networks[C]// Proceedings of the 2015 International Conference on Network and Information Systems for Computers, Wuhan, China. 2015: 106-110.

[11] XIONG A P, WANG X W, ZOU Y. Scheduling Algorithm Based on Storm Topology Hot-edge[J]. Computer Engineering, 2017, 43(1): 37-42.

[12] LI T, TANG J, XU J. Performance Modeling and Predictive Scheduling for Distributed Stream Data Processing[J]. IEEE Transactions on Big Data, 2016: 2(4): 353-364.

[13] SANTURKAR S, ARORA A, CHANDRASEKARAN K. Stormgen-A Domain specific Language to create ad-hoc Storm Topologies[C]// Proceedings of the 2014 Federated Conference on Computer Science and Information Systems. Washington: IEEE, 2014: 1621-1628.

[14] SUN C H. The Design and Implementation of Data Analysis System Based on Storm[D]. Beijing: Beijing University of Posts and Telecommunications, 2014. (in Chinese)
孙朝华. 基于 Storm 的数据分析系统设计与实现[D]. 北京:北京邮电大学,2014.

[15] LONG S H. Research and Implementation of Real-time Big Data Analysis System Based on Storm[D]. Shanghai: Shanghai Jiao-Tong University, 2015. (in Chinese)
龙少杭. 基于 Storm 的实时大数据分析系统的研究与实现[D]. 上海:上海交通大学,2015.