SDaaS:一种传感流数据的服务化封装方法

张仲妹[1),2),3)] 刘 晨[2),3)] 苏 申[2),3)] 张守利[1),2),3)] 韩燕波[1),2),3)]

[1)](天津大学计算机科学与技术学院 天津 300072)

[2)](北方工业大学大规模流数据集成与分析技术北京市重点实验室 北京 100144)

[3)](北方工业大学云计算研究中心 北京 100144)

摘 要 来自不同传感器网络的流数据共享和集成对于带动相关业务和行业的创新具有重要意义.现有的传感网络往往是任务导向或领域专用的,仅适用于特定的应用场景,难以有效地在不同应用间共享和重用其数据资源.传感流数据的服务化是一种有效解决物理传感网络数据资源共享和重用的方法.针对已有服务化方法在应对大规模传感流数据共享和用户并发访问方面存在的局限性,该文提出了一种面向传感流数据的服务化封装方法——SDaaS(Stream Data as a Service),该方法使用事件的方式驱动传感流数据的处理和传输,通过对传感数据的融合操作实现服务对传感流数据的深层次加工,并基于 Pub/Sub 机制实现传感流数据的按需分发.文中基于 Spark Streaming 实现对大规模流数据加工操作的封装,并通过对传统的基于匹配树的事件匹配算法进行改进实现了高效的流数据内容分发,以保障将传感数据实时的分发给相应需求.该文通过实验验证了流数据服务的性能,印证了流数据服务能够响应不同的数据需求,在毫秒级别将数据流分发给不同应用.

关键词 传感流数据;流数据服务;Pub/Sub 机制;事件驱动;事件匹配;云计算

中图法分类号 TP393 **DOI 号** 10.11897/SP.J.1016.2017.00445

SDaaS: A Method for Encapsulating Sensor Stream Data as Services

ZHANG Zhong-Mei[1),2),3)] LIU Chen[2),3)] SU Shen[2),3)] ZHANG Shou-Li[1),2),3)] HAN Yan-Bo[1),2),3)]

[1)](*School of Computer Science and Technology, Tianjin University, Tianjin* 300072)

[2)](*Beijing Key Laboratory on Integration and Analysis of Large-scale Stream Data, North China University of Technology, Beijing* 100144)

[3)](*Cloud Computing Research Center, North China University of Technology, Beijing* 100144)

Abstract The sharing and integration of multiple stream data derived from different Sensor Networks significant for the innovation of business and industry. The current Sensor Networks are generally domain-specific and task-oriented, tailored for particular applications with little possibility of sharing and reusing sensor data for different applications. The servitization of stream sensor data is an effective solution for sharing and reusing sensor data resources. Considering the limitations of existing methods on processing large-scale stream data and concurrent requests, this paper proposes to encapsulate Stream Data as a Service(SDaaS), which process sensor stream data with service modeling operations and distribute data on-demand based on Pub/Sub mechanism. Then this paper fulfills the encapsulation by utilizing event-driven mechanism for stream data service processing, using Spark Streaming framework to process sensor events, and improving traditional matching-tree algorithm to distribute stream data efficiently. Finally, we evaluate our

收稿日期:2016-01-08;在线出版日期:2016-09-29.本课题得到国家自然科学基金(61672042)、北方工业大学"人才强校计划"青年拔尖人才培育计划和北京市市委组织部资助青年骨干个人项目(2015000020124G024)资助.**张仲妹**,女,1990 年生,博士研究生,主要研究方向为服务计算、大规模流数据集成与分析等.E-mail: gloria_z@126.com. **刘 晨**,男,1980 年生,博士,副研究员,中国计算机学会(CCF)会员,主要研究方向为服务计算、大规模流数据集成与分析等. **苏 申**,男,1985 年生,博士,讲师,主要研究方向为服务计算、域间路由建模与分析及舆情分析. **张守利**,女,1988 年生,博士研究生,主要研究方向为服务计算、大规模流数据集成与分析等. **韩燕波**,男,1962 年生,博士,教授,博士生导师,中国计算机学会(CCF)会员,主要研究领域为分布式系统、互联网服务、业务流程管理和协同、大规模流数据集成与分析等.

approach through experiments, and the stream data service can handle multiple requests and deliver data to corresponding applications in milliseconds level.

Keywords stream sensor data; stream data service; Pub/Sub mechanism; event-driven; event matching; cloud computing

1 引 言

近年来,随着传感网络技术的不断发展,越来越多的传感器被部署到物理世界中并连接入网.据欧洲委员会预计,到 2020 年将有 500～1000 亿的传感设备连接到因特网上[1].传感设备能够持续产生大量的传感数据,进而帮助用户(人类或者机器)与其所处的环境互动,并对现实生活中的事件及时反应.这些传感数据具有实时到达、持续不间断、变化快等特征,是典型的流数据.

来自不同传感器网络的流数据的共享和集成对于带动相关业务和行业的创新具有重要意义.然而现有的传感器网络一般是领域专用或者任务导向的,仅服务于特定的应用.因此在不同企业或组织之间共享传感流数据存在极大障碍.首先,数据提供者并不希望对外提供与隐私相关的数据,因此传感流数据不能直接交付给数据消费者.例如,对于交管局的车牌自动识别流数据(Automatic Number Plate Recognition Data,ANPR),实时导航可以利用车牌流数据做路况分析,但交管局并不希望暴露具体的车牌.其次,传感流数据规模一般比较大,数据消费者只关心其中很少的一部分,而不希望接受并处理一个大规模的数据流.因此更好的方案是按需的推送传感流数据.

为了克服上述障碍,一些现有工作本着"数据即服务"[2]的思想,以服务化的方式来对外提供传感流数据.比如,一些传感器网络(如 USTL[3],HomePort[4]等)以 RESTful 服务接口的形式对外提供传感数据,以实现方便的数据访问.为了使应用能够获取不同来源的传感流数据,一些研究团队建立了面向社区的流数据服务化平台(如微软的 SenseWeb[5],Global Sensor Networks[6]及 LiveWeb[7]),使得用户能够将不同来源的传感流数据组合起来以创建更具有价值的应用.然而,前者仅仅对外提供了获取数据的接口,并不考虑第三方对获取数据的需求;后者虽然允许编码流数据的处理逻辑,但往往采用集中处理的方式,且需要为不同种类的用户需求提供不同的服务接口.因此无法有效的处理大规模流数据,也就在保障用户并发访问方面具有一定的局限性.

针对已有方法在流数据服务化方面存在的局限性,本文提出面向传感流数据的服务化封装方法——SDaaS(Stream Data as a Service),以实现大规模传感流数据在服务化过程中的深层次数据加工和按需定制.具体的,本文主要贡献如下:

(1)提出一个针对传感流数据的服务化封装方法,通过对流数据的融合操作实现传感流数据服务的深层次加工,通过 Pub/Sub 系统实现传感流数据的按需分发,并将该服务封装方法进行模型抽象,然后面向服务提供者和数据消费者抽象封装接口;

(2)根据本文提出的服务封装方法,提出了一个可行的实现方案,该方案基于 Spark Streaming 实现了对流数据加工操作的封装,基于对传统匹配树算法的改进实现了高效的流数据内容分发,并进一步通过性能检验验证了本文提出的服务封装方法的可行性.

2 相关工作

传感流数据的服务化是一种有效解决传感数据资源共享和重用的方法,能够依托互联网提供跨组织共享数据能力,保障数据获取时的安全性和私密性.服务需要依托流数据的实时处理来对原始的传感流数据进行加工转换,在流数据实时处理领域,文献[8-10]采用事件驱动的方式对数据进行抽象.本文采用类似的思路,将传感设备产生的数据记录视为传感事件.这样一个传感数据流可以视为一个传感事件流,多个传感事件流经过过滤、聚合和关联等加工之后可以生成具有更丰富语义的传感事件流.然后本文利用事件订阅系统来实现服务内容的分发过程,将经过加工处理的传感事件流按需分发给不同的订阅请求.

2.1 流数据服务化

SenseWeb[5]允许应用在其提供的 Web 平台上共享和获取传感数据.Global Sensor Networks[6]提

供一个灵活的、能够动态集成和管理不同传感网络数据的中间件. 文献[5-6]采用了轮询的方式在互联网上向用户提供数据. 早期工作均采用集中式的方式实现对流数据的转换、查询能力. 文献[11-13]采用了分布式的方式实现对流数据的处理. 上述工作均采用传统的服务模型，需要创建不同服务来满足流数据共享的不同需求，对这些服务的管理势必会对整个系统造成巨大的负担.

文献[14-15]提出了“Web of Thing”的概念，将来自不同物联网中的数据在 Web 之上共享，并提供与其他 Web 服务进行组合(Mashup)的能力. 文献[15]讨论了基于 Web 向用户主动推送消息的几种方式，由于其认为传感设备的采样频率很少超过 1 Hz，因此采用了 WebHooks 的推送方式. LiveWeb[7] 提供了一个 SensorWeb 服务门户网站，允许用户随时对物理世界进行实时查询、监控，并提供离线通知功能. 文献[15]和文献[7,16]分别采用了基于主题和基于内容的 Pub/Sub 机制实现流数据在不同的用户之间的共享，但不支持对流数据的转换融合以及用户的灵活按需获取，因此更适用于流数据产生速率较慢的情况.

目前已有部分工作在其传感网络[3-4]或者基于传感流数据的具体应用[17-18]中采用了服务化的方式共享和获取传感数据. 其中，文献[4,17]采用了服务器发送事件(Server Send Event，SSE)技术实现服务端主动推送数据. 这些工作对外共享其传感数据或者处理传感数据的能力，但第三方应用在获取其传感流数据或者处理能力时需要根据需求对传感数据进一步处理或者对处理能力进行改进.

2.2　事件匹配方法

文献[19]指出 Pub/Sub 机制由于其以数据为中心、多对多的通信模式、动态性及低耦合的特性，非常适用于基于传感网络的应用. 然而目前在传感网络领域很少有工作涉及事件匹配算法. 目前基于内容的 Pub/Sub 系统中事件匹配算法主要分为 3 类：基于树状结构的算法，基于谓词索引的算法，及基于分布式哈希表的算法.

Gough 等人[20]提出一种基于搜索树的匹配算法，但是当订阅请求增加或者取消时，该算法难以对搜索树进行修改. Aguilera 等人在文献[21]中也通过对匹配树进行遍历来得到所有匹配成功的订阅请求，但是该算法中只能考虑部分重复的谓词. Silvia 等人在文献[22]中提出了基于 R-tree 的匹配方法，将每个订阅请求视为一个范围，并根据订阅请求范围之间的关系，构建一个 R-tree. 文献[23-24]中同样使用了基于 R-tree 的匹配算法，主要基于订阅请求的位置约束来构建树. 本文中的订阅请求结构较为复杂，订阅请求整体之间的关系难以表达，因此不适合采用 R-tree 的方法.

文献[25]提出了基于谓词索引的内容匹配算法，为所有订阅请求包含的谓词建立一个保存谓词和订阅请求映射关系的表，在匹配时遍历这个表找出满足订阅请求的谓词数目，如果与其包含的谓词数目相同，则说明该订阅请求与事件相匹配. 在订阅请求包含的谓词数目较少的情况下，文献[25]有较高的吞吐量和较低的时间延迟. 文献[26-29]在谓词索引的基础上进行了改进，其中[28]通过判断事件属性间的重复关系、订阅请求之间的重复和包含关系减少不必要的匹配，能够较快的找到与事件相匹配的订阅请求. 谓词索引方法避免了相同的谓词被多次验证，但是需要对所有的谓词进行验证.

基于树状结构和谓词索引的算法大都基于这样的思想：对于所有订阅请求中的一些重复的部分，只需要判断一次，并充分利用索引结构来缩短匹配时间. 基于分布式哈希表的算法将订阅请求和事件分布到不同的哈希区域中，事件仅和同一区域的订阅请求进行匹配. 文献[30]利用哈希树将订阅请求分布到不同的 bucket 中，每个事件只需要对相应的 bucket 的订阅请求进行匹配. 文献[31]将事件和订阅请求按照事件的属性值和订阅请求定义的范围分配到不同的分区中，对不同分区的订阅请求分别进行验证. 基于分布式哈希表的匹配算法易于实现对大规模的订阅请求进行并行化匹配，可以与基于树状结构或基于谓词索引的算法相结合，更进一步的提高事件匹配的效率.

2.3　流数据实时处理

早期的流数据管理系统(Data Stream Management System，DSMS)采用集中式架构，包括 Aurora[32]，STREAM[33]，TelegraphCQ[34] 等. 显然，一旦负责查询处理的节点资源出现饱和状态，查询处理的速度就会减慢，从而导致查询结果的输出时间延迟变长. 近年来，分布式的流处理系统(Distributed Stream Processing System)，如 S4[35]，Storm①，Spark Streaming[36] 成为主流. 这些平台都采用分布式架构，处理能力能够随着节点数目的增长而得到扩展. S4 是 Yahoo 开发的分布式、可扩展的、对等的流数

① Storm，http://storm.apache.org/2012

据处理平台，处理节点间保持相对的独立性、对等性和高并发性，极大的提高系统性能. S4 仅支持部分容错，当数据流速率超过一定界限时，数据处理的错误率会随着速率增加. Storm 是 Twitter 支持开发的一款分布式、开源、实时、主从式的流数据计算系统，采用简单的编程模型，能够保证消息的快速计算，目前不支持数据负载的动态变化. Spark Streaming 是 Berkeley 开源的通用分布式流数据处理平台，引用了微批次的概念，具有高容错性. Spark Streaming 可以与 Spark 框架中的其他项目（如 Spark、SparkR、MLlib 等）无缝对接，能够减少单独编写流处理程序和历史数据处理程序.

3 SDaaS 服务化封装方法

为了方便进一步的细节描述，本节首先参考文献[37-39]给出 SDaaS 模型的基本定义，然后介绍流数据服务模型和服务化封装方法.

3.1 传感流数据相关概念和定义

本文将传感流数据视为由部署在物理世界的传感设备或应用程序产生的、有时间属性的传感数据记录序列.

定义 1. 传感流数据. 传感流数据可以被表示为 $S=\{d_1,d_2,\cdots,d_i,\cdots\}$，其中每一条数据记录都具有相同的属性集合 $\{a_1,a_2,\cdots,a_m\}$. $d_i=\langle v_{i1},v_{i2},\cdots,v_{im}\rangle$ 表示传感流数据的一条数据记录，其中，v_{ik} 表示属性 a_k 在记录 d_i 中所对应的取值.

图 1 给出了一个来自某出租车公司的真实 GPS 数据集片段. GPS 数据流 S 的属性集合为{车 id，检测时间，经度，维度，速率，方向，卫星个数}，根据定义 1，流数据中的数据记录可以表示为

车ID	检测时间	经度	维度	速率	方向	卫星个数
18834,	2015-04-01 08:00:21,	121.441895,	31.206928,	38.5,	228.0,	7
27320,	2015-04-01 08:01:21,	121.562235,	31.249510,	0.0,	249.0,	10
16926,	2015-04-01 08:02:02,	121.666902,	31.282538,	28.4,	15.0,	11
26337,	2015-04-01 08:03:32,	121.424717,	31.235583,	0.0,	135.0,	10
23807,	2015-04-01 08:04:43,	121.478988,	31.341545,	0.0,	294.0,	6
23390,	2015-04-01 08:05:00,	121.388335,	31.263890,	42.2,	1.0,	3
10827,	2015-04-01 08:06:57,	121.333718,	31.173365,	0.0,	307.0,	9
20043,	2015-04-01 08:07:53,	121.357767,	31.385620,	0.0,	171.0,	8
00049,	2015-04-01 08:08:01,	121.303755,	31.247965,	0.0,	10.0,	8
21941,	2015-04-01 08:09:53,	121.720453,	31.179758,	86.4,	67.0,	9
15042,	2015-04-01 08:10:50,	121.458723,	31.272005,	0.0,	68.0,	8
27615,	2015-04-01 08:11:46,	121.389823,	31.098167,	66.9,	336.0,	9
14891,	2015-04-01 08:12:37,	121.406283,	31.304457,	0.0,	273.0,	11

图 1 GPS 数据记录示例

$d_1=\{18834,2015\text{-}04\text{-}0108{:}00{:}21,121.441895,31.206928,38.5,228.0,7\}$,

$d_2=\{27320,2015\text{-}04\text{-}01\ 08{:}01{:}21,121.562235,31.249510,0.0,249.0,10\}$,

…

传感流数据每条最新的数据记录都可以视为一个事件 e，数据消费者可以通过提交订阅请求的方式获取符合其需求的事件.

定义 2. 事件. 事件由一组包含属性及其属性值的键值对组成，可以表示为

$$e=\{\langle a_1,v_1\rangle,\langle a_2,v_2\rangle,\cdots,\langle a_n,v_n\rangle\},$$

其中：a_i 表示属性名；v_i 表示具体的属性值.

传感流数据 S 的每条数据记录 $d_i=\langle v_{i1},v_{i2},\cdots,v_{im}\rangle$ 可以对应一个更新事件 $e_i=\{\langle a_{i1},v_{i1}\rangle,\langle a_{i2},v_{i2}\rangle,\cdots,\langle a_{im},v_{im}\rangle\}$. 比如来自出租车公司的 GPS 传感流数据所产生的事件可以由 $\{\langle id,18834\rangle,\langle longitude,121.441895\rangle,\langle latitude,31.206928\rangle,\langle direction,228.0\rangle,\langle speed,38.50\rangle,\langle siteNum,7\rangle,\langle time,2015\text{-}04\text{-}01\ 08{:}00{:}21\rangle\}$ 表示.

定义 3. 事件约束. 事件约束 $constraints$ 表示事件 e 的值域，由其包含的属性可能存在的取值范围组成，可以表示为

$$constraints=\bigcap c_{a_i},$$

其中：a_i 是事件所包含的属性；属性约束 c_{a_i} 表示属性 a_i 所能取值的范围.

如 GPS 传感流数据所产生的事件中的车速属性取值一定大于 0，该事件中存在一个属性约束 $c_{speed}=(speed>0)$.

显然一个事件流中所有事件的格式相同，因此一个事件流的事件约束即这个事件流中每个事件的约束.

定义 4. 操作算子. 对传感流数据的操作算子 op 可以表示为一个四元组：$op=\langle name,func,in\_events,out\_event\rangle$，其中：

(1) $name$ 表示操作算子的名称；

(2) $func$ 描述具体的操作逻辑；

(3) in_events 表示操作算子作用的事件流对象集合；

(4) out_event 表示经过操作转换之后所生成的事件流.

数据消费者可以通过提交订阅请求的方式来定义对传感流数据的内容需求.

定义 5. 订阅请求. 订阅请求 r 可以由 $k(k\geq1)$ 个谓词 p_i 通过逻辑运算符“∧”或“∨”连接而成的合取或析取范式来表示，r 可以等值演算为由有限个合取式所组成的析取式，记作：

$$r=\bigvee P_i,\ P_i=\bigwedge p_{ij},$$

其中，谓词 p_{ij} 可以定义为一个三元组：

$$p_{ij}=\langle a_{ij},ro_{ij},v_{ij}\rangle,\ p_{ij}\subseteq c_{a_i},$$

其中：a_{ij} 是属性名；关系运算符 $ro_{ij}\in\{<,\leq,>,\geq,=,\neq\}$；$v_{ij}$ 是对应属性 a_{ij} 的属性值，谓词的定义一定在事件约束的范围之内.

如应用可以通过提交订阅请求：

$r_1=\langle vid,=,$京 B34C91$\rangle$；

$r_2=\langle longitude,<,126.89\rangle\wedge\langle longitude,>,116.29\rangle\wedge(\langle speed,>,10\rangle\vee\langle speed,<,60\rangle)$

来获取相应的传感数据. 其中，订阅请求 r_2 可以转换为一个析取范式：

$r_2'=(\langle longitude,<,126.89\rangle\wedge\langle longitude,>,116.29\rangle\wedge\langle speed,>,10\rangle)\vee(\langle longitude,<,126.89\rangle\wedge\langle longitude,>,116.29\rangle\wedge\langle speed,>,10\rangle)$.

定义 6.　事件和订阅请求的匹配. 给定事件 e 和订阅请求 r，事件需要 r 中的所有谓词 $p_i=\langle a_i,ro_i,v_i\rangle\in r$ 进行验证匹配，其中对于任意的谓词 p_i，当 e 包含键值对 $\langle a',v'\rangle$，并且 $(a'=a_i)\wedge(v'\ ro_i v_i)$ 时，e 与谓词 p_i 相匹配，记作：

$$e\propto p_i.$$

将 r 所有谓词的匹配结果按照定义 5 进行逻辑计算，当计算结果为真时，事件与订阅请求相匹配，记作：

$$e\propto r.$$

例如，事件中的属性键值对 $\langle speed,62\rangle$ 与谓词 $\langle speed,>,60\rangle$ 相匹配，因为 62>60；事件{$\langle id,18834\rangle$，$\langle longitude,121.441895\rangle$，$\langle latitude,31.206928\rangle$，$\langle direction,228.0\rangle$，$\langle speed,38.5\rangle$，$\langle siteNum,7\rangle$，$\langle time,2015\text{-}04\text{-}01\ 08{:}00{:}21\rangle$}与订阅请求 r_2' 相匹配.

3.2　流数据服务模型

原始传感流数据除了安全性和私密性的问题，还存在着价值密度较低的问题. 因此我们对原始的传感流数据进行转换、以服务的方式共享更具有价值的流数据. 为了更好的为应用所重用，流数据服务能够响应不同订阅请求，仅向应用分发其所需的事件. 本文借鉴前期数据服务模型[40]和基于内容的 Pub/Sub 机制，对流数据服务进行定义. 流数据服务相当于部署在流数据源和业务应用系统之间的具有处理能力、可配置的“软传感器”，能够灵活响应不同的应用需求.

定义 7.　流数据服务. 一个流数据服务是一个传感流数据资源的统一抽象表示，从构成角度，一个流数据服务可以表示为一个多元组：$sds=\{uri,in\_events,ops,out\_event,constraints,rs,disLogic\}$，其中：

(1) uri 是服务创建者用来描述其流数据服务的唯一标示，服务消费者可以使用 HTTP 标准方法访问该 uri 来获取数据；

(2) in_events 表示服务所接入事件流，事件可以来自于原始传感流数据，也可以来自于其它的流数据服务；

(3) ops 表示服务创建者定义在 in_events 上的一组由操作算子组成的操作逻辑；

(4) out_event 表示 in_events 在经过 ops 操作逻辑转换之后所形成的事件流；

(5) $constraints$ 表示 out_event 的事件约束，服务消费者定义在该服务上的订阅请求必须符合该约束条件；

(6) rs 表示定义在流数据服务之上的订阅请求；

(7) $disLogic$ 分发逻辑决定流数据服务输出的事件向哪些数据消费者进行分发.

在一个流数据服务中，in_events，ops，out_event 元素面向服务创建者，$constraints$，rs 元素面向服务消费者.

流数据服务接入、输出，及通过操作算子所产生的中间事件都按照定义 2 所示的格式. 流数据服务对数据转换融合的实时性要求较高，需要及时的输出新的流数据，在创建过程中选择以事件驱动的方式接入、转换、输出流数据. 当接收到来自流数据源的数据后，主动将数据以事件的方式发送到相应的操作算子，最终得到输出结果.

3.3　服务化封装

流数据的共享和重用问题可以映射为用户订阅请求和流数据服务所输出数据的适配问题. 图 2 展示了传感流数据共享和重用的服务化封装的总体框架图.

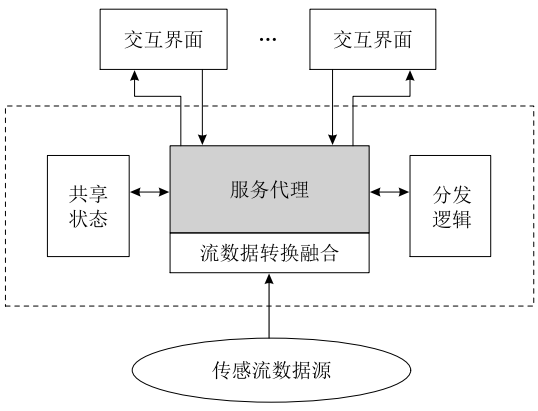


图 2　传感流数据服务化封装框架

从体系结构上看，传感流数据的服务化封装主要由 3 个部分组成：服务代理、事件的分发逻辑以及

所有在服务代理上注册的流数据服务及用户的状态信息.服务代理负责管理流数据服务的创建和事件分发的整个过程,是传感流数据服务化封装中的核心.事件分发逻辑是把流数据服务产生的事件分发到事件目的地(用户)时所使用的策略.服务状态提供所有流数据服务的实时运行状态和服务元数据,以及与服务对应的用户订阅状态和元数据.

基于图 2 所示的结构,图 3 展示了传感流数据服务化封装的实例.为了实现对传感流数据实时、按需的共享和重用,本文的服务化封装方法需要对流数据服务和订阅请求进行管理.其中,流数据服务管理允许数据提供者(服务创建者)对传感流数据源进行转换、融合,将新的、可公开的流数据以服务的方式公布给外界.订阅请求管理在流数据服务的基础上进一步支持数据消费者(服务消费者)按需的共享和重用流数据服务所提供的数据,流数据服务能够同时响应来自不同数据消费者的订阅请求.

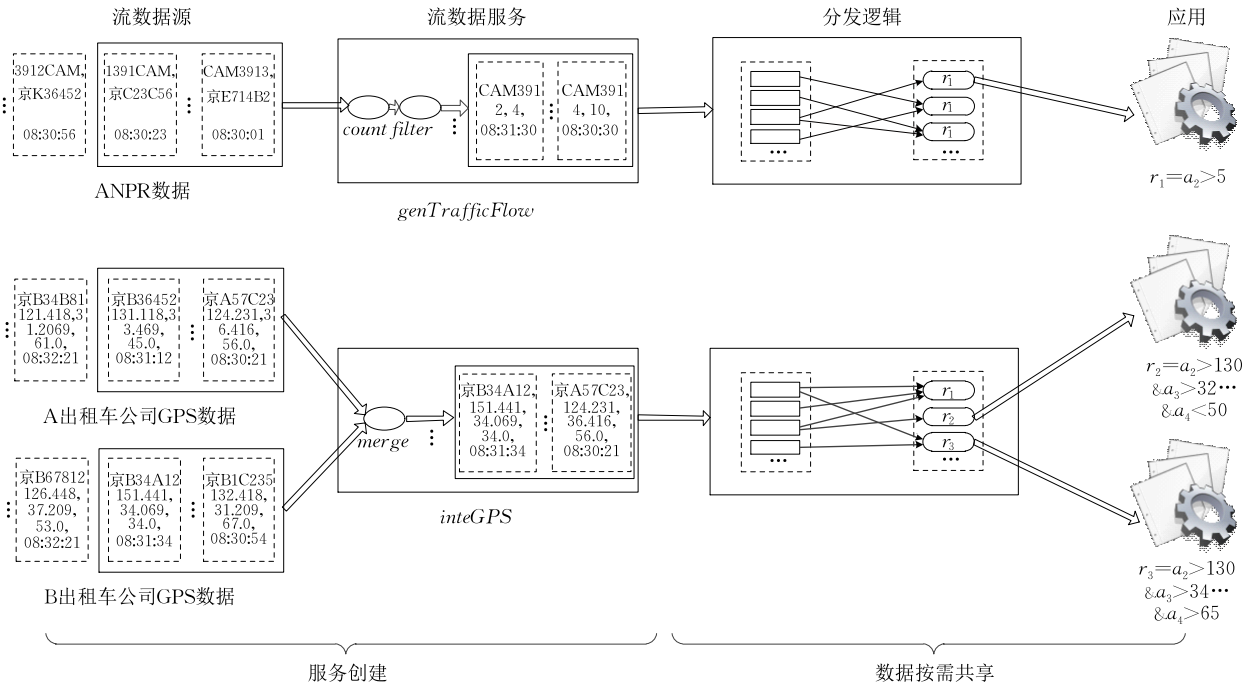


图 3 传感流数据服务化封装实例

3.3.1 流数据服务管理过程

服务创建者通过在服务代理上创建并注册流数据服务的方式来共享和重用传感流数据.

原始的传感流数据具有数据规模大,价值密度低的问题,服务创建者可以将其拥有的传感流数据根据需求配置相应的处理逻辑,将其转换为一个新的数据流,并根据输出事件流形成事件约束.与此同时,服务创建者还可以选择来自其它服务的流数据来创建流数据服务.如 ANPR 数据的拥有者可以将原始的 ANPR 数据进行聚集操作(*count*)将其转换为车流量数据 TrafficFlow,创建流数据服务 ***genTrafficFlow***.其中,ANPR 数据中的一条数据记录可以表示为 $d_1=\{$CAM3913,京 K36452,2015-04-01 08:30:56$\}$,其产生的事件具有属性集$\{cid,vid,time\}$和事件约束 $constraints=\{\langle cid=*\rangle,\langle vid=*\rangle,\langle time>2010/01/01\ 00:00:00\rangle\}$,表示 ANPR 数据的时间范围为 2010 年以后.TrafficFlow 数据的事件具有属性集合$\{cid,flow,time\}$和事件约束 $constraints=\{\langle cid=*\rangle,\langle flow>0\rangle,\langle time>2010/01/01\ 00:00:00\rangle\}$,表示车流量必须大于 0,时间范围是 2010 年以后.相比较于 ANPR 数据,车流量数据更具有价值意义.又如服务创建者可以选择不同来源的 GPS 数据作为原始传感流数据,对其进行融合(merge)处理,转换为一个更完整的 GPS 数据流,创建流数据服务 ***inteGPS***.原始的 GPS 数据(如图 1 所示)和转换后的 GPS 数据具有相同的属性集合和事件约束 $constraints=\{\langle vid=*\rangle,\langle 115.25<longitude<117.30\rangle,\langle 39.26<latitude<41.03\rangle,\langle speed>0\rangle,\langle time>2010/01/01\ 00:00:00\rangle\}$,表示 GPS 事件的属性值中,地理位置在北京市,速率必须大于 0,时间范围在 2010 年以后.

一般情况下,服务创建者还需要自定义操作或者算法,以对流数据源进行更复杂的数据处理或融合.如本文所在团队的其他工作[41]中实现对 ANPR

数据的实时分析，将 ANPR 数据转换为一个包含伴随车组信息的流数据. 对于车辆追踪、实时拼车等应用都更具有价值意义.

在流数据服务创建过程中，对原始的传感流数据的处理逻辑是基于事件驱动的，不需事先将数据存储起来，因此流数据服务所接入和输出的数据是持续不断的. 基于分布式流数据处理平台，流数据服务对原始流数据的处理具有高可用性和可伸缩性，能够满足流数据实时处理的需求[42].

3.3.2　订阅请求管理过程

数据消费者可以通过向指定流数据服务提交订阅请求的方式获取传感数据. 只要数据消费者的订阅请求符合服务输出的事件约束，数据消费者便可以获得其所需的数据，而无需创建专门的流数据服务或者自行处理原始的传感流数据.

面对流数据服务所产生的持续不断的事件以及应用不同的订阅请求，流数据服务需要一个分发逻辑——即事件匹配方法，负责找到与当前事件相匹配的所有订阅请求，将事件分发给相应订阅请求. 在图 3 的例子中，数据消费者想要获取指定需求的 GPS 车辆数据，首先找到流数据服务 ***inteGPS*** 并得到订阅请求需要遵循的服务约束，即服务输出事件的事件约束 $constraints=\{\langle vid=*\rangle, \langle 115.25<longitude<117.30\rangle, \langle 39.26<latitude<41.03\rangle, \langle speed>0\rangle, \langle time>2010/01/01\ 00{:}00{:}00\rangle\}$. 遵循服务约束，数据消费者向流数据服务提交订阅请求；服务将按照分发逻辑将其产生的每个事件分发给相应的订阅请求. 例如：假设数据消费者提交了订阅请求 $r_2=\{\langle longitude,>,130\rangle, \langle latitude,>,32\rangle, \langle speed,<,50\rangle\}$ 和 $r_3=\{\langle longitude,>,130\rangle, \langle latitude,>,34\rangle, \langle speed,>,55\rangle\}$ 来说，对于服务 ***inteGPS*** 产生的事件 $e_1=\{\langle vid,$ 京 A57C23$\rangle, \langle longitude,134.2\rangle, \langle latitude,36.41\rangle, \langle speed,56.0\rangle\}$ 和 $e_2=\{\langle vid,$ 京 B34A12$\rangle, \langle longitude,151.4\rangle, \langle latitude,34.06\rangle, \langle speed,35.0\rangle\}$ 来说，事件 e_1 仅发送到 r_3，事件 e_2 仅发送到 r_2.

高效的事件匹配算法是保证数据消费者能够实时获取所需数据的关键之一. 一种简单的事件匹配方法是将事件与所有的订阅请求分别进行验证. 这种方法的执行时间会随着订阅请求的增加而线性增长，并不适合于多流数据来源及多应用请求的场景. 当订阅请求的数量较多时，组成订阅请求的各谓词之间可能存在着大量相等或者包含关系，本文中的分发逻辑通过考虑不同订阅请求之间的关系，来减少谓词匹配的数目，以提高事件匹配的效率. 如订阅请求 r_2 和 r_3 中相同的谓词（$longitude>130$）仅需要匹配一次，具有包含关系的谓词（$latitude>32$ 与 $latitude>34$），存在 $(\uparrow e\propto(latitude>32)\rightarrow\uparrow e\propto(latitude>34))\vee(e\propto(latitude>34)\rightarrow e\propto(latitude>32))$.

在事件向数据消费者发送时，可以选择三种传输模式：轮询、长连接和主动连接方式. 其中轮询模式属于被动拉取的模式，相当于传统数据服务的轮询方式实现；长连接和主动连接的方式属于主动推送的模式. 长连接模式主要基于由客户端向服务器端建立的长连接：数据消费者通过建立一个长连接向流数据服务提交订阅请求，流数据服务在该长连接之上将符合订阅请求的事件持续的发送给数据消费者. 主动连接传输模式更为灵活：数据消费者通过短连接向流数据服务提交或者取消其订阅请求；每当事件符合订阅请求时，流数据服务主动向相应数据消费者建起连接并发送数据.

相比较而言，主动推送方式获取数据的效率更高. 其中，长连接方式对于每个订阅请求都需要维护一个长连接，流数据服务支持的订阅请求数目受到系统本身的限制[43]. 主动连接方式不会出现连接资源浪费以及连接限制的情况，然而由于连接的重复建立，并不适用于数据发送频率较高的场景. 由于本文目前侧重于支持大规模流数据的实时共享，因此暂时采用了基于长连接的传输方式，在后续研究中，我们将界定不同方式的适用范围，灵活选用传感数据的传输方式.

4　SDaaS 实现

4.1　流数据融合操作

本文当前实现的操作算子主要包含 3 类：针对单个流数据源的转换算子、基于滑动窗口的聚集算子以及针对多源流数据的融合算子. 在当前实现中，针对接入的传感流数据源可能出现数据规模较大的情况，操作算子被映射为一个或多个 Spark Streaming 方法，利用 Spark Streaming 并行框架实现. 表 1 展示了流数据服务的建模操作算子及其对应的部分 Spark Streaming 方法.

(1) 转换算子

过滤. 过滤操作可以仅输出符合相关条件的事件. 过滤操作可以表示为

$$filter(a,ro,v),$$

表 1 流数据服务的操作算子及对应 Spark Streaming 方法

类型	算子	Spark Streaming 方法
转换算子	*filter*	*filter*(*fun*(*a*,*rO*,*v*))
	project	*map*(*fun*(*A*,*V*,*M*))
	sort	*window*(*wL*,*sI*).*sortByKey*(*o*,[*numTasks*])
聚集算子	*aggregator*	*reduceByWindow*(*fun*(),*wL*,*sI*)
	incrAggregator	*reduceByWindow*(*fun*(),*wL*,*sI*).*reduceByKey*(*sumfun*())
融合算子	*merge*	*union*(*s*′)
	join	*join*(*s*′,[*numTasks*])

其中:a 为事件所包含的属性;$ro\in\{<,>,=,\neq,\leq,\geq\}$是关系运算符;$v$ 为具体的属性值.经过过滤操作转换所得的流数据与原始的流数据具有相同的属性集合.

投影.投影操作可以选择输入流数据的若干属性及值映射新的数据流,可以表示为

$$project(A,A',M,V)$$

其中:A 为从原始流数据中选择的属性集合;A'为新指定的属性集合;$M=\{\langle a_1,a_1'\rangle,\langle a_2,a_2'\rangle,\cdots,\langle a_k,a_k'\rangle\}$是新旧属性集合的映射关系,$V=\{\langle a_i',v_i\rangle\}$在 A'中包含原始流数据属性集合以外的属性时指定相应的默认属性值.经过投影操作转换所得到的流数据具有新的属性集合 A'.

排序.排序操作可以对指定窗口内的事件按照指定属性值进行排序,可以表示为

$$sort(a,o,wL,sI)$$

其中:a 为输入事件所包含的属性;$o\in\{asce,desc\}$用来指定排序的方式是降序还是升序;wL 表示窗口大小;sI 表示窗口的滑动距离.经过排序算子转换所得到的流数据与原始的流数据具有相同的属性集合.

与过滤操作和投影操作不同的是排序操作属于阻塞式操作[44],对于单条数据记录做排序操作是毫无意义的.排序操作需要额外的指定 wL 和 sI 参数来指定操作的流数据输入范围和操作频率.本文中所使用的滑动窗口是基于元组的滑动窗口[45].

(2)聚集算子

聚集操作包含求和、计数、最小值、最大值以及平均数操作.与排序操作相同,聚集操作也属于阻塞式操作,需要指定 wL 及 sI 参数,针对指定窗口 wL 内的流数据记录进行操作.本文实现两类聚集操作 $aggregator$ 和 $incrAggregator$,其中:

$aggregator=\{sum,count,min,max,avg\}$仅对窗口内的数据进行操作,每次窗口数据改变时以新计算的结果代替上一次的计算结果.以最大值操作为例,最大值操作仅输出窗口内指定属性值最大的事件,可以表示为

$$max(a,wL,sI),$$

其中:a 为输入事件所包含的属性;wL 表示窗口大小;sI 表示窗口的滑动距离.

$incrAggregator=\{incrSum,incrCount,incrMin,incrMax,incrAvg\}$为递增式的聚集操作,可以对多次的计算结果进行累加.同样以递增式最大值操作为例,当前窗口内指定属性值最大的事件会与上一次输出的事件相比较输出属性值较大的事件,可以表示为

$$incrMax(a,wL,sI),$$

其中:a 为输入事件所包含的属性;wL 表示窗口大小;sI 表示窗口的滑动距离.

经过聚集算子转换所得到的流数据与原始的流数据具有相同的属性集合.

(3)融合操作

流数据服务可以利用融合操作对接入的多个传感流数据进行融合,输出更具有价值的流数据.

合并.合并操作能够将两个同构的流数据合并为一个新的流数据,可以表示为

$$merge(S'),$$

其中,S'所产生的事件和原始流数据产生的事件具有相同的属性集合,经过合并算子转换所得到的流数据与原始的流数据也具有相同的属性集合.

连接.连接操作可以将两个描述同一对象或具有相同属性的传感数据连接为一个流数据,可以表示为

$$join(S',a,wL,sI),$$

其中:S'与原始流数据描述同一个对象或者具有相同的属性 a;wL 表示窗口大小;sI 表示窗口的滑动距离.连接操作能够对窗口内的两个流数据所产生的事件进行笛卡尔积连接,或者将具有相同属性值的事件进行连接.使用连接操作所转换得到的流数据包含的属性集合为$\{a_1,a_2,\cdots,a_m\}\cup\{a_1,a_2,\cdots,a_n\}$,其中,$\{a_1,a_2,\cdots,a_n\}$和$\{a_1,a_2,\cdots,a_m\}$为原始流数据的属性集合.

目前的融合操作均为二元操作,流数据服务可以通过多个连续的融合操作,实现对多源流数据的融合.

4.2 事件订阅系统

现有的事件匹配算法大都是将订阅请求组织为特定的数据结构,对其进行遍历,从而得到与事件相匹配的订阅请求.文献[20-21,46]中使用树状结构来对订阅请求集合进行预处理,增加事件的匹配效

率，这种结构被称为匹配树. 本文使用了改进的匹配树来获取与事件相匹配的订阅请求，并实现了匹配树的并行化验证.

4.2.1　匹配树的数据结构

匹配树中包含 3 种类型的节点：根节点、叶子节点和谓词节点，其中，根节点用虚拟节点表示；叶子节点代表一个订阅请求；谓词节点包含订阅请求中的一个谓词.

定义 8.　匹配树. 一个匹配树 T 可以由其根节点和多个索引结构来表示

$$T=\langle t\_r,\{PL\}\rangle,$$

其中：t_r 是一个虚拟节点，代表所有订阅请求的起点；索引结构 PL 用来表述树中不同谓词节点之间的相等或者包含关系，PL 的数目与匹配树所对应的事件所包含的属性数目有关.

匹配树具有以下性质：

(1) 给定事件 e，对于匹配树中的任意谓词节点 n，仅当 $e\propto n.p$ 时，访问 n 的所有子节点 N，并对于谓词节点 $n'\in N$，验证 e 与 $n'.p$ 的匹配关系；

(2) 给定事件 e，设匹配树中的所有叶子节点为 N，$N'\subseteq N$ 是事件 e 对匹配树遍历验证时所访问到的所有叶子节点，则 $R=n_1.r\cup n_2.r\cup\cdots\cup n_k.r$，$ni\in N$，$N'\wedge 1\leq i\leq k$ 为所有与事件相匹配的订阅请求；

(3) 对于任意的订阅请求 $r_i=p_{i1}\wedge p_{i2}\wedge\cdots\wedge p_{in}$ 和 $r_j=p_{j1}\wedge p_{j2}\wedge\cdots\wedge p_{jn}$，令 $p_{i1}\wedge p_{i2}\wedge\cdots\wedge p_{ik}$ 和 $p_{j1}\wedge p_{j2}\wedge\cdots\wedge p_{jk}$ 为 r_i 和 r_j 的前缀谓词，其中 $1\leq k\leq \min(n,m)$，如果 $(p_{i1}=p_{j1})\wedge(p_{i2}=p_{j2})\wedge\cdots\wedge(p_{ik}=p_{jk})$，那么 r_i 和 r_j 的前 k 个谓词可以在匹配树上共用 k 个谓词节点.

(4) 对于任意的谓词节点，$n\in PL$ 可以根据 $n.p$ 的验证结果推理获取部分谓词节点 $n\in PL$ 的谓词匹配结果.

下面通过介一个简单的匹配树实例来说明匹配树的性质.

例如，假如存在 3 个订阅请求 r_1，r_2 和 r_3：

$$r_1=p_1\wedge p_2;$$
$$r_2=p_1\wedge p_3;$$
$$r_3=p_2\wedge p_4$$

其中：r_1 和 r_2 包含相同的谓词 p_1；r_2 和 r_3 包含相同的谓词 p_2；谓词 p_3 和 p_4 存在一个包含关系 $p_3\prec p_4$，这 3 个订阅请求组成的匹配树如图 4 所示.

在图 4 中，对于事件 e，仅在 $e\propto p_1$ 时才需要对节点 p_2 和 p_3 进行访问和验证(性质 1). 对于订阅请求 r_1 来说，路径：虚拟节点→p_1→p_2→r_1 代表了订阅请求 $r_1=p_1\wedge p_2$. 当事件 e 对匹配树进行遍历验证时，访问到的所有叶子节点即为与事件 e 相匹配的订阅请求(性质 2). 订阅请求 r_1 和 r_2 共同拥有谓词 p_1，它们在匹配树中共用同一个谓词节点(性质 3).

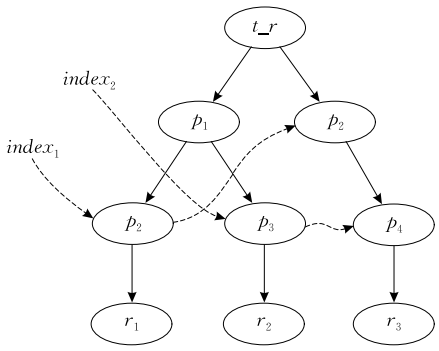


图 4　一个匹配树实例

传统匹配树只在不同订阅请求具有相同的前缀谓词时才能共享谓词节点，无法涵盖所有的谓词关系. 如图 4 中订阅请求 r_2 和 r_3 包含同样的谓词 p_2，但它们无法共用同一个谓词节点.

为了进一步减少谓词的验证次数，我们在匹配树中使用了多个索引结构来表述谓词节点的谓词之间关系，对匹配树进行优化. 如图 4 中包含两个索引结构，其中，对于索引结构 $index_1$ 来说，两个谓词节点的匹配结果相同，对于索引结构 $index_2$ 来说，由于 p_3 p_4，因此 $(e\propto p_4\rightarrow e\propto p_3)\wedge(\neg e\propto p_3\rightarrow e\propto p_4)$ (性质 4).

4.2.2　匹配树创建

根据定义 5，本文将组成订阅请求 r_i 的每一个简单合取式称为条件表达式 c_{ij}，对于订阅请求 r_i 来说，$\exists(e\propto c_{ij})\rightarrow(e\propto r_i)$. 定义在同一个流数据服务之上的订阅请求所包含的所有条件表达式可以转换为一颗匹配树. 匹配树构建的步骤可以分为：(1) 初始化匹配树及 (2) 依次向匹配树中添加新的条件表达式.

在向匹配树添加新的条件表达式时，给定已有的匹配树 T 和条件表达式 c_i，对于任意的谓词 $p_{ij}\in c_i$，需要遍历 T 中所有谓词节点才能获取所有可能的谓词关系，这种方式复杂且不易于实现. 因此，我们将谓词集合按照其所定义的属性名称和关系运算符来逐级建立多级索引，每一个最终索引项为事件所包含的属性，按照不同的关系运算符形成谓词列表 PL.

如图 5 所示，对于包含判定大小关系的运算符的谓词，在 PL 中按照谓词的包含关系进行排序，即对 $\forall p_i,p_j\in PL$，$j<j\rightarrow p_i\prec p_j$. 对于这一类 PL，能

够得到$(e\propto p_i\rightarrow e\propto\{p_j\})\vee(\uparrow e\propto p_i\rightarrow\uparrow e\propto\{p_k\})$，其中$p_j$为排在$p_i$后面的谓词，$p_k$为排在$p_i$前面的谓词.对于相等和不相等关系运算符，该$PL$实质上是一个Hash表，对于相等运算符的$PL$中的谓词来说，存在$e\propto p_i\rightarrow\uparrow e\propto\{p_j\}$，其中，$p_j$为除$p_i$之外的谓词.对于不相等运算符的$PL$来说，存在$\uparrow e\propto p_i\rightarrow e\propto\{p_j\}$，其中，$p_j$为除$p_i$之外的谓词.

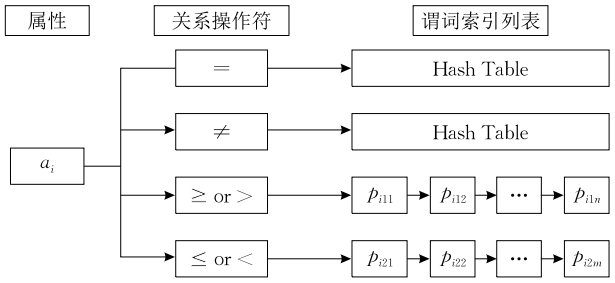


图5 包含属性a_i的谓词多级索引结构

因此，对于事件$e=\{\langle a_1,v_1\rangle,\langle a_2,v_2\rangle,\cdots,\langle a_m,v_m\rangle\}$来说，每一个属性$a_i$都可以定义一个多级索引结构来表述定义在该属性上不同谓词的关系.定义在e上的订阅请求所形成的匹配树可以由一个根节点以及m个多级索引结构组成.

在匹配树中，如果能够事先知道每个谓词的出现概率，并根据概率来对谓词排序，可以增加不同的条件表达式共享谓词节点的可能性[47].然而，统计谓词出现的概率并进行排序需要复杂的处理过程，且谓词出现的概率会随着订阅请求的改变而改变.简化起见，本文中的谓词顺序使用字母排序，即对于任意条件表达式：

$$c=\langle a_1,ro,v_1\rangle\wedge\langle a_2,ro,v_2\rangle\wedge\cdots\wedge\langle a_k,ro,v_k\rangle.$$

其包含的属性$\{a_1,a_2,\cdots,a_k\}$存在$a_1\leq a_2\leq\cdots\leq a_k$.

图6展示了一个优化后的匹配树，由于匹配树对应的事件包含4个属性，匹配树中包含一个根节点以及4个多级索引结构.简单起见，图中并没有将所有的索引结构在图中画出.在匹配树中从根节点到任一叶子节点的路径上，对于任意的$a_i\leq a_j$，a_i一定出现在a_j之前.

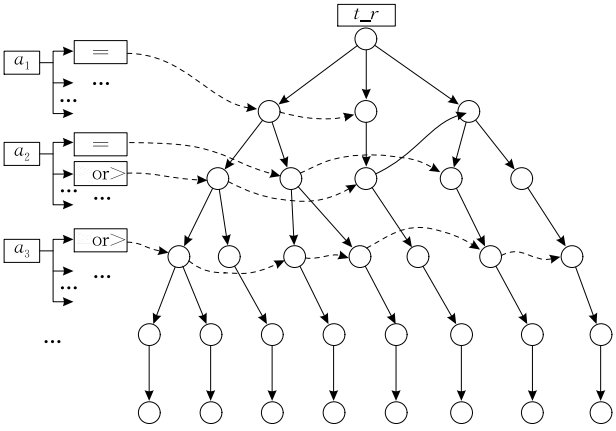


图6 优化后的匹配树示例

将所有的条件表达式转换为匹配树的伪代码如算法1所示.

算法1. 匹配树创建算法.

输入：条件表达式集合C

输出：新的匹配树T

```
1.  过程 create_Tree(C)
2.  INIT T;
3.  FOR EACH c in C
4.     addConInTree(T, c);
5.  RETURN T;
6.  过程 addConToTree(T, c)
7.  q is the number of predicates in c;
8.  p_i is the i-th predicate in c;
9.  n←T.root; i←1;
10. IF n has no child THEN flag←FALSE;
11. ELSE IF
12.   flag←TRUE;
13. WHILE flag and i≤q DO
14.    FOR EACH child node m of n
15.      IF m.value==p_i
16.      THEN n←m; i←i+1; BREAK;
17.      IF there are no child of n equal with p_i
18.     THEN flag←FALSE;
19. IF flag==FALSE
20.    WHILE i≤q DO
21.      new node n'←p_i;
22.      find PL for n' and insert n';
23.      n.addChild(n');
24.      n←n';
25. new node n'←c; n.addChild(n');
```

创建匹配树时，首先需要初始化匹配树(行2)，然后向匹配树中依次添加新的条件表达式(行3,4).添加新的表达式时，我们首先判断原始的匹配树的根节点是否不含任何子节点(行10)，如果不含任何子节点，则将条件表达式的每个谓词形成的谓词节点依次添加到树中(行19～24).如果根节点具有子节点，判断并找到匹配树中包含的条件表达式的前缀谓词，共享谓词节点(行12～18).然后向树中依次添加表达式中剩余的谓词(行19～24)，在添加谓词节点时，需要根据谓词所包含的属性及关系运算符找到谓词所在的索引列表PL，并将其插入到PL中(行22).最后在树中添加一个代表该条件表达式

的叶子节点(行 25).

在创建匹配树时，每往匹配树中插入一个新的节点都需要查找并改变该节点所属的索引结构.假设事件包含的属性数目为 m，需要对 n 个条件表达式创建匹配树.这里假设每个表达式所包含的谓词为 $2m$，每个属性之上的谓词数目为 $2n$.最好的情况下，每次插入请求时，插入谓词均包含其他谓词或者与其他谓词等价，匹配树的创建时间复杂度为 $2mn$.最坏的情况下，每次插入一个谓词节点时，都需要与同层次的其他节点进行比较，此时的时间复杂度为 $2n\times(n+(m-1)\times n)$.创建匹配树的平均时间复杂度为 $O(mn^2)$.

空间复杂度方面，每个条件表达式的插入需要占用至少 $2m+1$ 个节点，由于每个节点需要有链表的前后驱信息和树的父子信息，所以占用空间复杂度为 $O(mn)$.

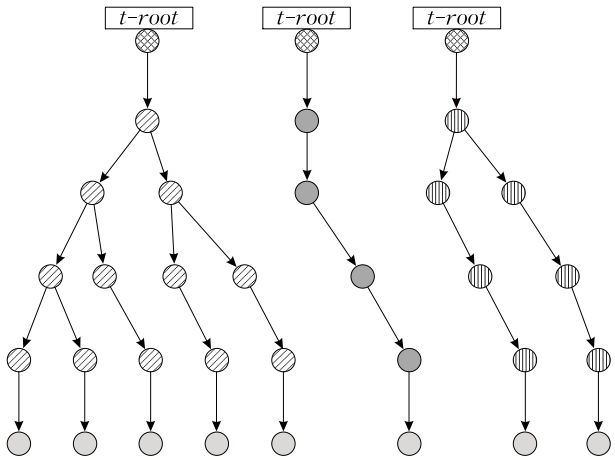
匹配树的生成算法允许订阅请求的动态增加.当新的订阅请求到来时，仅需要对其进行转换，然后对相应的条件表达式调用算法 1 中的过程 addConToTree 即可.当删除订阅条件时，可以通过对匹配树的局部节点进行删除，同时对相应的索引结构进行调整.

4.2.3　匹配树并行化验证

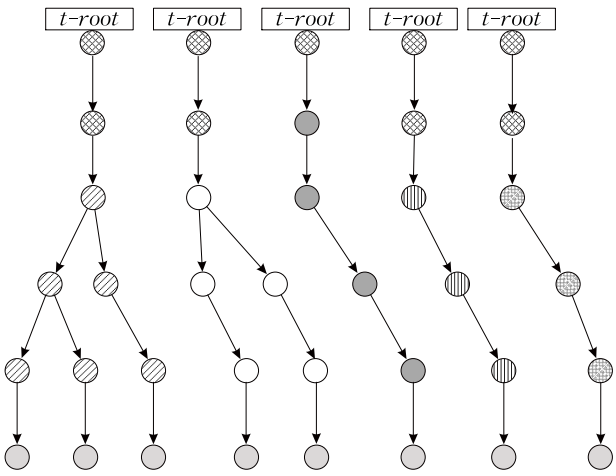
对于可能出现的大规模的应用订阅请求，为了更好的保障应用获取传感流数据的时效性，在利用匹配树对事件进行匹配时，我们可以将匹配树划分为多个子树并对其行进并行化遍历.如图 6 中的匹配树可以划分为图 7 所示的子树.

由图 7 可以看到，匹配树可以根据不同层次的子节点划分为不同的子树.由于构成匹配树的订阅请求数目及其所包含的谓词数目都不相同，匹配树划分的子树数目会受到每层节点数目的影响，无法动态的调节匹配树验证的并行度.而且每个子树包含的谓词数目也会出现倾斜的状况.因此我们可以先将匹配树划分成多个子树，然后将子树分配到不同的分块中重新组成一个子树，并行的对每个分块中的子树进行遍历.

为了避免出现分块负载过重或者负载过小的情况，在对子树进行划分时需要尽量保证分块的均匀性.因此本文按照子树包含的谓词数目对其进行划分，尽量保证同一分块中子树包含谓词的总数目相近.本文将匹配树的划分转换为典型的整数划分问题[48]以进行求解，定义 9 给出了对匹配树进行划分的形式化定义.



(a) 根据第1层子节点进行划分



(b) 根据第2层子节点进行划分

图 7　匹配树的子树划分示例

定义 9.　匹配树的划分策略.给定匹配树 T，可以将其划分为子树集合 $\boldsymbol{T}$，设分块数目为 k，基于谓词数目的匹配树划分策略 sp 是一个分块函数 $sp(\boldsymbol{T},k)=\{\boldsymbol{T}_1,\boldsymbol{T}_2,\cdots,\boldsymbol{T}_k\}$，它满足：

(1) $\boldsymbol{T}=\bigcup_{i=1\cdots k}\boldsymbol{T}_i$；

(2) $\boldsymbol{T}_i\cap\boldsymbol{T}_j=\varnothing$，$i,j=1\cdots k\wedge i\neq j$；

(3) $\min\left(\sum_{i,j=1\cdots k}(|\boldsymbol{T}_i|-|\boldsymbol{T}_j|)\right)$，$|\boldsymbol{T}_i|$ 是 $\boldsymbol{T}_i$ 中所有子树包含谓词的总数目.

整数划分问题就是如何把一个数组 L 分成 k 个子数组，使得每个子树组的元素之和大小尽量相等.如一个数组{3,1,1,2,2,1}可以分成两个子数组 $L_1=\{3,1,1\}$ 和 $L_2=\{2,2,1\}$，显然这两个子数组满足定义 9 的所有要求.

在本文中，我们先按照设置的并行度及匹配树中每层谓词节点的数目将匹配树划分为多个子树，所有子树包含的谓词数目便形成一个数组，其中每个子树中包含的谓词数目相当于数组中的一个元

素.如图 7 中,假设匹配树的并行度设置为 2,按照第一层子节点的数目将匹配树划分为 3 个子树,这些子树的包含的谓词数目可以组成数组{12,4,7}.该数组可以分成两个子数组 $L_1=\{12\}$ 和 $L_2=\{4,7\}$.

整数划分问题是一个 NP 难问题.本文采用了文献[48]给出的时间复杂度为 $O(kn^2)$ 的算法,该算法并不打乱原有数组元素的顺序,通过将数组顺序的进行划分得到一个次优解.它计算各种划分的可能情况的最小开销 $M[i][j]$ 以及相应的数组划分的位置 $D[i][j]$.其中,$M[i][j]$ 表示将数组的前 i 个元素分成 j 个分块的最小开销,$D[i][j]$ 用来记录元素划分的位置信息.算法的详细信息可以见参考文献[48].利用匹配树的划分策略,我们对匹配树并行的进行遍历验证.匹配树并行化验证的伪代码如算法 2 所示:

算法 2. 匹配树并行化验证算法.

输入:匹配树 T,事件 e,并行度 k

输出:匹配成功的条件表达式集合 C

```
1.  过程 match(T,e)
2.  T←T.childTree;
3.  PT←sp(T,k);
4.    FOR EACH T_i in PT
5.      exec sub_match (T_i,e) in parallel;
6.  过程 sub_match(T,e)
7.  INIT C;
8.  FOR EACH T_i in T
9.    n is the root of T_i;
10.   visit(T,n,e,R);
11. RETURN C;
12. 过程 visit(T,n,e,C)
13. IF n is a leaf node THEN C.add(n.value);
14. ELSE
15.   IF n.result is NULL
16.     verify n.value on e;
17.     find PL for n and set result for relative nodes;
18.   IF n.result is TRUE
19.     FOR EACH child n' of n
20.     visit(T,n',e,R);
```

我们首先获取匹配树的所有子树(行 2),并将子树集合按照定义 6 进行划分(行 3).对于每一个分块我们并行的执行子匹配过程 sub_match(行 4,5).在每个子匹配过程中,对每个子树递归的进行遍历(行 8~10),所有访问到的叶子节点为与事件相匹配的条件表达式(行 7).当访问到谓词节点时,某些谓词节点在没有遍历到的情况下便已经通过索引结构得到了匹配结果,可以先查看该谓词节点是否已经存在匹配结果(行 15).如果没有结果再对其进验证(行 16),然后按照其匹配结果为所在索引上的相关节点进行赋值(行 17).

通过对匹配树进行遍历,我们可以获得更新事件相匹配的所有的条件表达式.通过条件表达式与订阅请求的关系,便可以得到与更新事件相匹配的所有订阅请求,并将更新事件发送到相应的订阅请求.

在匹配树验证阶段,匹配树验证的时间复杂度与每次访问验证的谓词节点的数目相关.假设将匹配树平均划分为若干子树,根据整数划分原理每棵子树包含的节点数量尽可能相等.子树上的每个谓词节点的验证消耗包含访问和谓词验证两部分的时间消耗.对于任意子树中的第 j 层的第 i 个谓词节点 n_{ij},遍历的复杂度可以表示为

$$cost(n_{ij})=\sum_{k=1}^{k<j} p_k\times cost_{vi}+p'\times cost_{ve} \tag{1}$$

其中:p_k 为从根节点到 n_{ij} 的路径上第 k 个谓词验证结果为真的概率;p' 为已知 n_{ij} 验证结果的概率;$cost_{vi}$ 为访问的时间消耗;$cost_{ve}$ 为谓词验证的时间消耗.

简单起见,我们假设条件表达式中仅包含等值谓词,设 K 是事件所包含的属性个数,V 是每个属性可以取值范围,S 是子树涉及的条件表达式集合,$C(S)$ 是事件与该子树进行验证的期望时间.

利用上文介绍的整数划分原理将匹配树尽可能均匀的划分为若干子树.假设所有的事件是等可能发生的,且每个订阅请求与事件相匹配的概率是相同的.当并行地将大量事件与匹配树进行匹配时,订阅请求集合将均衡分布于各个子树,根据概率论可以得到每个事件将以相似的概率与子树中的订阅请求相匹配,从而保障并行化过程的负载均衡.

基于此并行化方式,匹配任意一个事件所消耗的期望时间 $C(S)$ 满足

$$C(S)\leq 2(k+1)|S|^{1-\lambda}(\ln V+\ln(K+1))\times cost_{vi}+(k+1)|S+1|^{1-\lambda}(\ln V+\ln(K+1))\times cost_{ve} \tag{2}$$

其中,$\lambda:=\dfrac{\ln V}{\ln V+\ln K}>0$.

首先考虑访问的时间消耗.

引理 1. 对于等概论出现的 V^K 个事件,存在

$$C_{vi}(S)=\frac{\sum cost_{vi}(n)}{V^K} \tag{3}$$

引理 2. 最多有 V^η 个节点满足

$$C_{vi}(S)=V^{K-\eta} \tag{4}$$

证明. 满足 $C_{vi}(S)=V^{K-\eta}$ 的节点只存在于第 η 层.根据匹配树的构建方式,最坏情况下第 η 层所

有节点都需要匹配. 这种情况下，最多有 V^{η} 个节点满足 $C_{vi}(S)=V^{K-\eta}$. 证毕.

显然可得：

$$V^{\eta}\leq V^{\eta}\times(K+1)^{\eta+1}.$$

易知，一个树最多含有 $(K+1)\times|S|$ 个节点. 将一棵树的所有节点按照 $cost_{vi}(n)$ 排序，设 $f(n)$ 表示第 n 个节点的代价 $cost$. 根据 $C_{vi}(S)=\dfrac{\sum cost_{vi}(n)}{V^K}$，有 $C_{vi}(S)=\dfrac{\sum_{n=1}^{(K+1)\times|S|}cost(n)}{V^K}$.

设 $g(x)=(Ax+B)^{-\lambda}$，其中：

$$A=V^{-(K+1)/\lambda}\times[V-1/(K+1)],$$

$$B=V^{-(K+1)/\lambda},$$

$$\lambda=\frac{\ln V}{\ln V+\ln K}<1.$$

$f(j)$ 表示第 j 个节点的访问次数，则有如下引理.

引理 3.　$f(x)\leq g(x)$.

证明.　对于任意 $0\leq i\leq K$，正整数 j 满足

$$\sum_{p=0,1,\cdots,i-1}(K+1)\times[(K+1)\times V]^{p}<j\leq\sum_{p=0,1,\cdots,i}(K+1)\times[(K+1)\times V]^{p}.$$

由于 $f(j)$ 表示第 j 个节点的访问次数，且 $f(j)$ 为 $V^{(K+1)-\eta}$ 的节点个数最多为 $(K+1)\times[(K+1)\times V]^{\eta}$，所以 $f(j)\leq V^{(K+1)-i}$.

根据等比数列求和公式：

$$g\Big(\sum_{p=0,1,\cdots,i}(K+1)\times[(K+1)\times V]^{p}\Big)=g\Big[(K+1)\times\frac{[(K+1)\times V]^{i+1}-1}{KV-1}\Big].$$

根据 $g(x)$ 定义

$$g\Big(\sum_{p=0,1,\cdots,i}(K+1)\times[(K+1)\times V]^{p}\Big)=V^{(K+1)-i}.$$

由于 $g(x)$ 是非增函数：

$$f(j)\leq V^{(K+1)-i}=g\Big(\sum_{p=0,1,\cdots,i}(K+1)\times[(K+1)\times V]^{p}\Big)\leq g(j).$$

证毕.

推论 1.

$$C_{vi}(S)\leq 2(k+1)|S|^{1-\lambda}(\ln V+\ln(K+1))\times cost_{vi}.$$

证明.

$$C_{vi}(S)=V^{-K}\sum_{x=1}^{(k+1)|S|}f(x)cost_{vi}\leq V^{-K}\sum_{x=1}^{(k+1)|S|}g(x)cost_{vi}\leq V^{-K}\int_{0}^{(k+1)|S|}g(x)\mathrm{d}x\,cost_{vi}$$

$$=V^{-K}\frac{(A(K+1)|S|)^{1-\lambda}-B^{1-\lambda}}{A(1-\lambda)}cost_{vi}=\frac{V(K+1)((V(K+1)|S|-|S|+1)^{1-\lambda}-1)cost_{vi}}{(V(K+1)-1)(1-\lambda)}.$$

由于 $(V(K+1)|S|-|S|+1)\leq V(K+1)|S|$，$V(K+1)^{1-\lambda}=(K+1)$，$V\geq 2$，$K\geq 1$，于是 $\dfrac{V\times(K+1)}{V\times(K+1)-1}\leq\dfrac{4}{3}$，$\dfrac{1}{\ln(K+1)}<3/2$.

因此：

$$C_{vi}(S)\leq\frac{V(K+1)[(K+1)|S|^{1-\lambda}-1][\ln V+\ln(K+1)]\times cost_{vi}}{(VK-1)\ln K}\leq 2(k+1)|S|^{1-\lambda}(\ln V+\ln(K+1))\times cost_{vi}.$$

证毕.

同理可得

$$C_{ve}(S)\leq(k+1)|S+1|^{1-\lambda}(\ln V+\ln(K+1))\times cost_{ve}.$$

5　实验与评价

本节首先对 SDaaS 方法与现有流数据共享和重用的方法进行分析对比，然后从性能角度比较这几种方法在服务响应时间和系统负载方面的差别.

5.1　实验数据及环境

本文实验使用真实的交通信息数据集及模拟数据集. 近年来，本文所在团队搜集了来自市交管局的 ANPR 数据(S_1)以及来自某出租车公司的 GPS 数据(S_2). 根据 GPS 数据集，我们模拟了 GPS 数据集，并将其划分为某公司的 GPS 数据集(S_3)和多个私人的 GPS 数据(S_4等). 表 2 展示相应的交通数据集.

表 2　数据集信息

名称	属性集合	来源	规模
ANPR(S_1)	*cid*, *vid*, *time*	市交管局	64 GB
GPS(S_2)	*vid*, *time*, *longitude*, *latitude*, *speed*, *direction*, *satelliteNum*, *siteNum* …	出租车公司 A	46 GB
GPS(S_2)	*vid*, *time*, *longitude*, *latitude*, *speed*, *direction*, *satelliteNum*, *siteNum* …	出租车公司 B(模拟)	340 MB
GPS(S_4)	*vid*, *time*, *longitude*, *latitude*, *speed*, *direction*, *satelliteNum*, *siteNum* …	私家车 A(模拟)	1.4 MB

实验在拥有 5 个节点的集群上进行，所有节点均为装有 CentOS-6.4 的虚拟机，jdk 版本为 1.70. 集群所使用的 Spark 版本为 1.1.0，Hadoop 版本为 2.3.0. 集群的详细信息见表 3. 其中主节点、工作节点 1 及工作节点 2 作为流数据服务实现的服务端；工作节点 3 用来模拟传感流数据源；工作节点 4 用来模拟应用的不同请求.

表 3 集群配置信息

节点角色	CPU	核数	内存/GB
主节点	Intel Xeon E312xx	6	6
工作节点 1	Intel Xeon E312xx	6	6
工作节点 2	Intel Xeon E312xx	3	3
工作节点 3	Intel Xeon E312xx	3	3
工作节点 4	Intel Xeon E312xx	3	3

5.2 服务评价指标

本文参考文献[31]定义了以下几个指标,对流数据服务进行评价.

(1)服务延迟.流数据服务 sds 对用户 i 的平均延迟(Service Latency,SL)可被定义为

$$SL_i=\frac{\sum(t_{ij\_rec}-t_{ij\_in})}{n},$$

其中:t_{ij_in} 表示服务接入数据记录 r_{ij} 的时间;t_{ij_rec} 表示用户 i 接收到数据记录 r_{ij} 的时间;n 为用户 i 接收到的数据记录数目.

服务延迟由两部分的时间延迟组成,分别为原始的流数据服务接入流数据服务到其输出新的流数据的时间延迟以及流数据服务输出流数据到应用获取到数据的时间延迟.

(2)系统负载.系统负载指用户在调用流数据服务 sds 时的系统负载情况,主要分为 CPU,内存以及网络的负载状况.

(3)事件匹配率.给定流数据服务 sds,假定 m 个用户同时对服务 sds 进行请求,其中,事件匹配率(Event Matching Rate,EMR)表示流数据服务单位时间内(s)能够处理的事件数目.

5.3 服务效果分析

为了模拟传感流数据的共享和重用,我们选取了一天的 ANPR 数据和 GPS 数据并遵循其事件约束各自随机生成 1000 个不同的订阅请求.我们将具有相同属性的订阅请求视为一类数据需求,表 4 展示了订阅请求所包含的数据需求种类数目.

表 4 数据需求种类

数据集	需求种类数目
ANPR	6
GPS	126

我们按照数据记录的产生时间将其转换为流数据,然后使用不同的方式实现传感流数据的服务化.现有的传感流数据服务化方法主要有:采用传统服务模型的服务化方式(TS)[5]、基于主题的 Pub/Sub 机制(TB_P/S)[15]和基于内容的 Pub/Sub 机制(CB_P/S)[16]的服务化实现方式.表 5 展示了使用不同服务化方式封装服务的效果分析.

表 5 服务效果分析

服务化方式	服务数目	是否需要进一步处理数据
DSaaS	2	否
TS	132	否
TB_P/S	4	需要
CB_P/S	4	部分需要

采用传统服务模型的服务化方式需要为每一类数据需求创建一个流数据服务,基于主题或者内容的 Pub/Sub 机制需要创建 4 个主题分别对应不同的传感流数据.相比较而言,SDaaS 方法仅需要创建 2 个数据服务 ***getANPR*** 及 ***inteGPS*** 便可以满足不同用户的订阅请求,其中,***inteGPS*** 服务对原始的 3 个 GPS 数据进行了融合操作,输出一个完整的 GPS 数据流.

对于基于主题的 Pub/Sub 机制,它会向任一个主题提交订阅请求的用户都会获取该主题上的所有数据,用户需要对其获取的数据进行处理才能够获取其需求的数据.虽然基于内容的 Pub/Sub 机制能够向用户发送其所需的数据,但对于对所有 GPS 数据都有需求的用户来说,仍需要对其获取的数据进行融合才能得到完整的数据.相比较而言,使用 SDaaS 方法用户无需对其获取的数据进行进一步的处理.

为了验证 SDaaS 方法的效果,我们将其与采用传统服务模型的服务化方式进行比较,以检验 SDaaS 是否能够按需的将数据分发给不同订阅请求.我们首先将传感流数据的数据频率设为 1000 Hz,将用户的订阅请求数目设置为 250,分别测试在订阅请求包含不同谓词数目情况下,服务端和用户端接收数据量的情况.实验表明在相同的订阅请求数目下,本文的 SDaaS 方法和采用传统服务模型的服务化方式能够接收到相同的数据.

此外,由于用户定义在同一个服务上的订阅请求并不是完全独立的,通过对服务输出事件与订阅请求的匹配过程进行优化,我们可以进一步提高用户获取数据的时效性.在本文的实验环境下,当传感流数据的频率设置为 10000 Hz 时,传统的服务方式在单个服务节点上仅能同时支持 950 个用户的订阅请求,而 SDaaS 方式可以支持 2697 个订阅请求,并且服务延迟保持在 30 ms 以内.

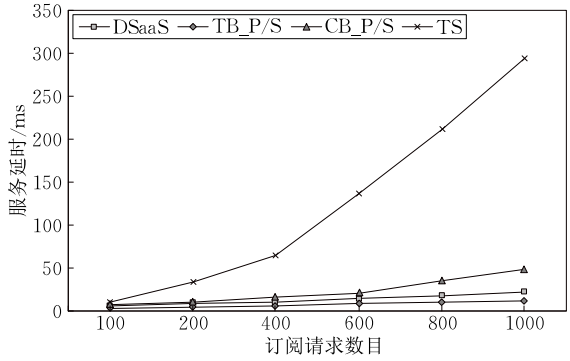
5.4 服务性能评价

5.4.1 请求数目对服务性能的影响

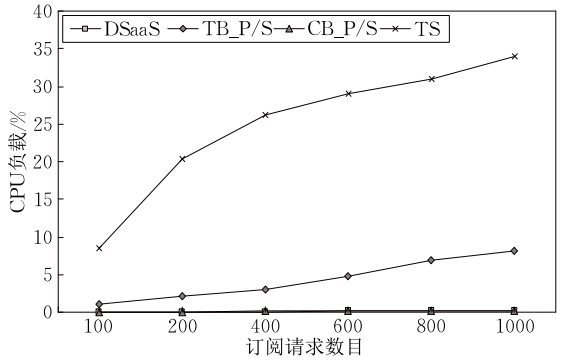
本节采用传统的服务化方式,基于主题及基于内

容的 Pub/Sub 方式作为基准，与本文方法（SDaaS）进行了对比. 我们随机选择 1000 条订阅请求，分别利用 4 种服务化方式获取数据. 对于 Pub/Sub 机制，定义在 GPS 数据上的订阅请求需要同时向 3 个 GPS 源提交请求. 我们将流数据的频率设置为 10 000 Hz，订阅请求的数目设置为 100、200、400、600、800 和 1000，分别计算 4 种方式下的平均服务延迟和系统负载.

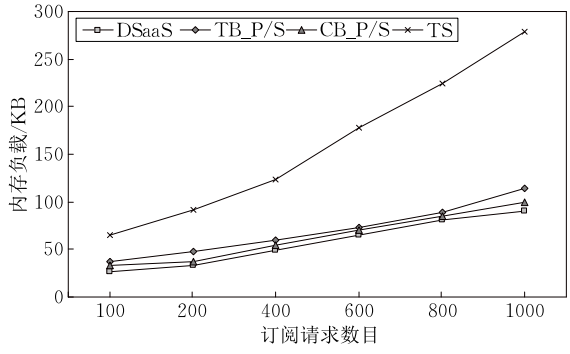
图 8(a)展示了服务响应时间的比较，可以看出基于主题的 Pub/Sub 机制具有最低的服务响应时间，这是因为基于主题的 Pub/Sub 机制并不对用户的订阅请求进行任何处理，直接将所有的数据转发给用户，因此具有最低的响应时间. 图 8(b)、(c)、(d)分别展示了订阅请求数目对系统负载的影响，其中由于 TS，TB_P/S 与 DSaaS 方式的网络负载相同，因此图 8(d)中仅展示了 DSaaS 的变化曲线. 由图 8(b)、(c)可以看出基于传统服务模型的服务化方法具有最高的内存和 CPU 负载，这是因为这种方式对于每一个订阅请求都要建立一个服务实例来进行相应的处理，每个服务处理之间是彼此独立的. 由图 8(d)可以看出，基于主题的 Pub/Sub 机制具有最高的网络负载. 这是因为基于主题的 Pub/Sub 机制需要将所有的数据转发给用户，而其他方法可以仅向用户传输其所需要的数据，因此基于主题的 Pub/Sub 机制具有更高的网络负载，而且用户还需要对其获取的数据进行进一步的处理. 基于内容的 Pub/Sub 机制和 DSaaS 方法具有较低的系统负载和较少的服务响应时间，其中由于 DSaaS 方法采用了更高效的事件匹配机制和分布式的数据处理框架，因此具有更高的效率和较低的系统负载. 在 1000 个并发订阅请求的情况下，服务响应时间一直维持在 20 ms 以内. 实验说明 SDaaS 方法能够支持在多用户并发请求的情况下在毫秒级别内将数据发送给相应用户.



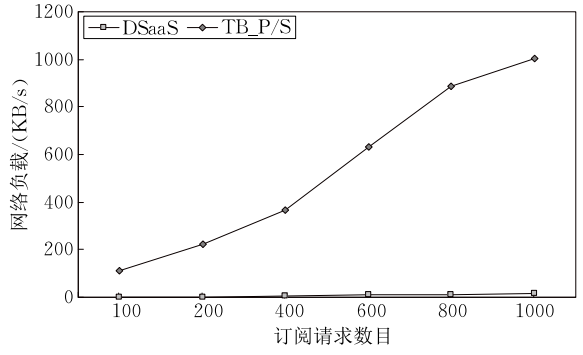
(a) 请求数目对服务响应时间的影响



(b) 请求数目对CPU负载的影响



(c) 请求数目对内存负载的影响



(d) 请求数目对网络负载的影响

图 8　订阅请求数目对流数据服务性能的影响

5.4.2　事件匹配算法对服务性能的影响

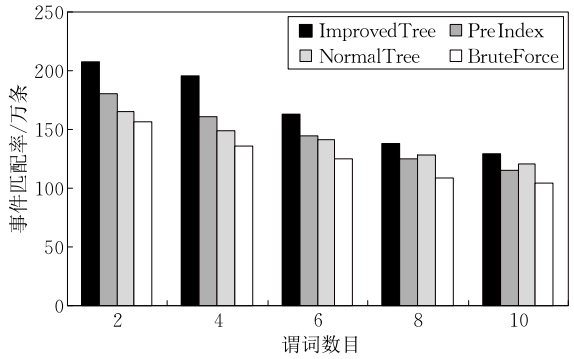
本文中的事件匹配方法是保障流数据服务能够高效的将数据分发给不同应用的关键，本节对本文采用的事件匹配方法（ImprovedTree）与普通的匹配树（NormalTree）[21]，谓词索引算法（PreIndex）[28]以及简单的事件匹配方法（BruteForce）进行比较. 为了能够应对大规模的应用并发的向流数据服务发起订阅请求，本文还实现了事件匹配方法的并行化，并与基于子匹配树数目均匀划分的策略进行比较.

我们首先将流数据服务输出的流数据频率设置为 10 000 Hz，订阅请求数目设为 100，比较在订阅请求包含的谓词数目对不同事件匹配方法效率的影

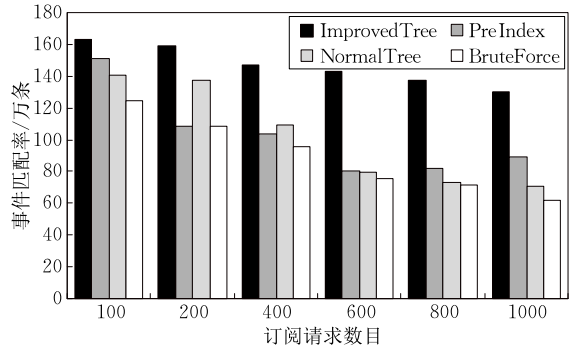
响.然后将订阅请求包含的谓词数目设为6,比较不同数目的订阅请求对事件匹配算法效率的影响.最后我们将订阅请求的数目设置为1000,比较两种不同的并行化方式的事件匹配率.

如图9(a)、(b)所示,当订阅请求数目或者订阅请求包含谓词数目较大时,BruteForce算法的效率最低,普通的匹配树和谓词索引分别在订阅请求数目较大和谓词数目较大时效率次低.本文采用的改进的匹配树的事件匹配算法由于采用了多级索引结构,利用谓词之间的关系减少了不必要的匹配使得该算法的效率在不同的谓词数目和不同的订阅请求数目下均具有较高的事件匹配率.

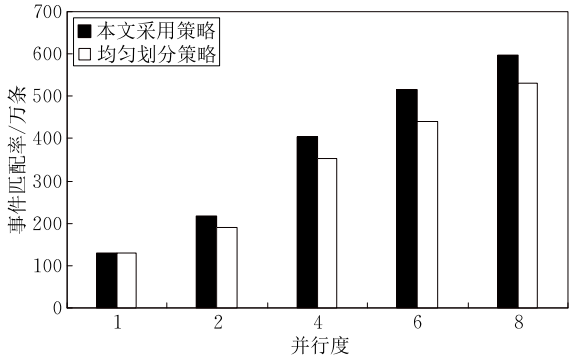
如图9(c)所示,随着并行度的增加,两种并行化方式下的事件匹配率均得到提高,而基于谓词的划分策略具有更高的事件匹配率.这是因为不同的子匹配树所包含的谓词数目并不相同,简单的将所有的子匹配树均匀的划分不能保证每个分块中的事件匹配过程的均衡.然而,虽然相比较对子匹配树均匀的划分来说,本文的划分策略能够保证每个分块中包含近似数目的谓词,却仍不能保证分块中进行访问验证的谓词节点数目的均衡.在后期工作中我们会针对该问题对划分策略进行进一步的优化.



(a) 谓词数目对事件匹配算法的影响



(b) 请求数目对事件匹配算法的影响



(c) 并行度对事件匹配算法的影响

图9 事件匹配算法的性能比较

6 总 结

为了实现传感流数据在不同应用之间的共享和重用,本文提出了一种面向传感流数据的服务化建模方法SDaaS,能够在流数据源和应用系统之间部署可配置的流数据服务,并借鉴Pub/Sub机制实现传感流数据的按需分发.本文基于Spark Streaming实现了对流数据加工操作的封装,能够对原始流数据进行转换融合,以服务的方式公布给外界;由于借鉴基于内容的Pub/Sub系统,相比较于传统的服务模型,能够更好的实现流数据服务的重用,减少系统需要维护的服务数量;对传统匹配数算法进行了改进并实现了高效的流数据内容分发.实验验证了流数据服务能够实现对不同应用请求实时、按需的发送数据,相比于现有的服务化方法具有更高的时效性和更低的系统负载.

参 考 文 献

[1] Reed D A, Gannon D B, Larus J R. Imagining the future: Thoughtsoncomputing. Computer, 2012, 45(1): 25-30

[2] Carey M J, Onose N, Petropoulos M. Data services. Communications of the ACM, 2012, 55(6): 86-97

[3] Pinto J, Martins R, Sousa J B. Towards a REST-style architecture for networked vehicles and sensors//Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops. Mannheim, Germany, 2010: 745-750

[4] Guilly T L, Olsen P, Ravn A P, et al. HomePort: Middleware for heterogeneous home automation networks//Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops. San Diego, USA, 2013: 627-633

[5] Grosky W I, Kansal A, Nath S, et al. SenseWeb: An infrastructure for shared sensing. IEEE Multimedia, 2007, 14(4): 8-13

[6] Aberer K, Hauswirth M, Salehi A. Infrastructure for data processing in large-scale Interconnected sensor networks//Proceedings of the 8th International Conference on Mobile Data Management. Mannheim, Germany, 2007: 198-205

[7] Yang X, Song W, Debraj D. LiveWeb: A sensorweb portal for sensing the world in real-time. Tsinghua Science & Technology, 2011, 16(5): 491-504

[8] Gyllstrom D, Wu E, Chae H J, et al. SASE: Complex event processing over streams//Proceedings of the CIDR 2007, the 3rd Biennial Conference on Innovative Data Systems Research. Asilomar, USA, 2007: 407-411

[9] Silberstein A, Filpus G, Munagala K, et al. Data-driven processing insensor networks//Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research. Asilomar, USA, 2007: 325-333

[10] Wang F, Zhou C, Nie Y. Event processing in sensor streams. Managing and Mining Sensor Data, 2013(12): 77-102

[11] Zhang J, Iannucci B, Hennessy M, et al. Sensor data as a service—A federated platform for mobile data-centric service development and sharing//Proceedings of the IEEE International Conference on Services Computing. California, USA, 2013: 446-453

[12] Mafrur R, Nugraha I G D, Choi D. Concept, design and implementation of sensing as a service framework//Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication. Bali, Indonesia, 2015: 1-4

[13] Kotsev A, Pantisano F, Schade S, et al. Architecture of a service-enabled sensing platform for the environment. Sensors, 2015, 15(2): 4470-4495

[14] Guinard D, Trifa V, Guinard D. Towards the web of things: Web mashups for embedded devices//Proceedings of the International World Wide Web Conferences. Madrid, Spain, 2009: 15-23

[15] Trifa V, Guinard D, Davidovski V, et al. Web messaging for open and scalable distributed sensing applications//Proceedings of the 10th International Conference on Web Engineering (ICWE 2010). Vienna, Austria, 2010: 129-143

[16] Zhang J, Radia N, Li Z, et al. An infrastructure supporting considerate sensor service provisioning//Proceedings of the 2013 IEEE 6th International Conference on Service-Oriented Computing and Applications (SOCA). Koloa, USA, 2013: 69-76

[17] Han Y, Wang G, Yu J, et al. A service-based approach to traffic sensor data integration and analysis to support community-wide green commute in china. IEEE Transactions on Intelligent Transportation Systems, 2016, 17(9): 2648-2657

[18] Sashima A, Yoda I, Kawamoto M, et al. A sensor data streaming service for visualizing urban public spaces//Proceedings of the ACM Conference on Embedded Networked Sensor Systems. Rome, Italy, 2013: 1-2

[19] Sheltami T R, Al-Roubaiey A A, Mahmoud A S H. A survey on developing publish/subscribe middleware over wireless sensor/actuator networks. Wireless Networks, 2015, 22(6): 1-22

[20] Gough J, Smith G. Efficient recognition of events in a distributed system//Proceedings of the Australasian Computer Science Conference. Canberra, Australia, 1995: 173-179

[21] Aguilera M K, Strom R E, Sturman D C, et al. Matching events in a content-based subscription system//Proceedings of the 18th ACM Symposium on Principles of Distributed Computing. Georgia, USA, 1999: 53-61

[22] Bianchi S, Felber P, Gradinariu M. Content-based publish/subscribe using distributed R-trees//Proceedings of the Euro-Par 2007 Parallel Processing. Berlin, Germany, 2007: 537-548

[23] Hu H, Liu Y, Li G, et al. A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions//Proceedings of the IEEE International Conference on Data Engineering. Brisbane, Australia, 2015: 711-722

[24] Guo L, Chen L, Zhang D, et al. Elaps: An efficient location-aware pub/sub system//Proceedings of the IEEE International Conference on Data Engineering. Santa Clara, USA, 2015: 1504-1507

[25] Fabret F, Jacobsen H A, Llirbat F, et al. Filtering algorithms and implementation for very fast publish/subscribe systems. ACM SIGMOD Record, 2001, 30(2): 115-126

[26] Carzaniga A, Wolf A L. Forwarding in a content-based network. SIGCOMM, 2003, 33(4): 163-174

[27] Xue Tao, Feng Bo-Qin, Li Bo, et al. Efficient matching for content-based publish-subscribe system. Mini-Micro Systems, 2006, 27(3): 529-533(in Chinese)
(薛涛，冯博琴，李波等. 基于内容的发布订阅系统中快速匹配算法的研究. 小型微型计算机系统，2006，27(3)：529-533)

[28] Liu Guo, Zhou Zhong, Wu Wei. Event matching algorithm based on the judgment of redundant attributes in publish/subscribe systems. Journal of Computer Research and Development, 2010, 47(10): 1690-1699(in Chinese)
(刘国，周忠，吴威. 发布/订购系统中基于重复属性判定的事件匹配算法研究. 计算机研究与发展，2010，47(10)：1690-1699)

[29] Pan Z, Liang X, Zhou Y C, et al. Intelligent push notification for converged mobile computing and internet of thing//Proceedings of the IEEE International Conference on Web Services. New York, USA, 2015: 655-662

[30] Qian S, Cao J, Zhu Y, et al. H-Tree: An efficient index structure for event matching in content-based publish/subscribe systems//Proceedings of the IFIP Networking Conference. New York, USA, 2013: 1-9

[31] Wang Y, Ma X. A general scalable and elastic content-based publish/subscribe service. Concurrency & Computation Practice & Experience, 2015, 27(1): 2100-2113

[32] Balakrishnan H, Balazinska M, Carney D, et al. Retrospective on aurora. The International Journal on Very Large Data Bases, 2004, 13(4): 370-383

[33] Arasu A, Babcock B, Babu S, et al. Stream: The stanford data stream management system. Data Stream Management, 2016(7): 317-336

[34] Chandrasekaran S, Cooper O, Deshpande A, et al. TelegraphCQ: Continuous dataflow processing//Proceedings of the ACM International Conference on Management of Data. San Diego, USA, 2003: 668-668

[35] Neumeyer L, Robbins B, Nair A, et al. S4: Distributed stream computing platform//Proceedings of the IEEE International Conference on Data Mining Workshops. Sydney, Australia, 2010: 170-177

[36] Zaharia M, Das T, Li H, et al. Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters//Proceedings of the Usenix Conference on Hot Topics in Cloud Ccomputing. Berkeley, USA, 2012:10-10

[37] Henzinger M R, Raghavan P, Rajagopalan S. Computing on data streams//Proceedings of the External Memory Algorithms. New Jersey, USA, 1998: 107-118

[38] Tirthapura S, Xu B, Busch C. Sketching asynchronous streams over a sliding window//Proceedings of the ACM Symposium on Principles of Distributed Computing. New York, USA, 2006: 82-91

[39] Campailla A, Chaki S, Clarke E, et al. Efficient filtering in publish-subscribe systems using binary decision diagrams//Proceedings of the International Conference on Software Engineering. Hyderabad, India, 2001: 443-452

[40] Wang G, Yang S, Han Y. Mashroom: End-user mashup programming using nested tables//Proceedings of the International Conference on World Wide Web 2009. Madrid, Spain, 2010: 861-870

[41] Zhu Meiling, Liu Chen, Wang Jiangwu, et al. A service-friendly approach to discover traveling companions based on ANPR data stream//Proceedings of the IEEE International Conference on Services Computing. San Francisco, USA, 2016: 171-178

[42] Stonebraker M, Çetintemel U, Zdonik S. The 8 requirements of real-time stream processing. ACM Sigmod Record, 2005, 34(4): 42-47

[43] Dickerson R, Lu J, Lu J, et al. Stream feeds: An abstraction for the world wide sensor web//Proceedings of the International Conference on the Internet of Things. Zurich, Switzerland, 2008: 360-375

[44] Babcock B, Babu S, Datar M, et al. Models and issues in data stream systems//Proceedings of the ACM Sigact-Sigmod-Sigart Symposium on Principles of Database Systems. Madison, USA, 2002: 1-16

[45] Arasu A, Babu S, Widom J. CQL: A language for continuous queries over streams and relations//Proceedings of the Database Programming Languages, International Workshop. Potsdam, Germany, 2003: 1-19

[46] Jerzak Z, Fetzer C. Prefix forwarding for publish/subscribe//Proceedings of the Inaugural International Conference on Distributed Event-Based Systems ACM. Toronto, Canada, 2007: 238-249

[47] Chen Ming-Wen, Xiao Zheng, Hu Chong-Lin, et al. Optimized covering detection algorithm for big event data processing. Journal of Computer Research and Development, 2013, 50(S2): 196-207(in Chinese)
(陈明文，肖政，虎嵩林等. 大规模事件数据处理的覆盖检测优化算法. 计算机研究与发展，2013，50(S2)：196-207)

[48] Skiena S. The Algorithm Design Manual. 2nd Edition. London: Springer London, 2008

ZHANG Zhong-Mei, born in 1990, Ph. D. candidate. Her research interests include service computing, large-scale stream data integration and analysis, etc.

LIU Chen, born in 1980, Ph. D., associate professor. His research interests include service computing, large-scale stream data integration and analysis, etc.

SU Shen, born in 1985, Ph. D., lecturer. His research interests include service computing, inter-domain routing modeling and analysis, sentiment analysis, etc.

ZHANG Shou-Li, born in 1988, Ph. D. candidate. Her research interests include service computing, large-scale stream data integration and analysis, etc.

HAN Yan-Bo, born in 1962, Ph. D., professor, Ph. D. supervisor. His research interests include distrusted system, web service, business process collaboration and management, large-scale stream data integration and analysis, etc.

Background

With the rapid development of sensor network technology, an increasing number of sensors are deployed, and generate volume, high-speeded sensor data continuously, which is typical stream data. It is very valuable to share and fuse such stream data form multiple sources for further analysis and innovation. However, most of the current sensor networks

are domain specific or task-oriented, which merely available for the third-party organizations.

There are mainly two challenges to share and reuse the stream data in real-time continuously. Firstly, to transfer a volume stream data over Internet is quite expensive. And secondly, it is impractical to provide to provide the raw stream data directly to users. To solve the above problem, a lot of efforts propose to encapsulate the stream data as a service or provide a community-oriented platform for sharing and managing stream data. However, these works either consider no user's demand, or need to provide different interfaces for multiple users.

Towards a better solution, we propose to encapsulate the stream data into service, which focuses on in-depth data processing and on-demand data distributions. Our contributions are mainly two-folded:

(1) we propose a stream data service model to encapsulate stream data, which provides in-depth data processing by defining transform and fusion operations on stream data, and distributes stream data based on Pub/Sub system.

(2) we realize the transform and fusion operations based on Spark Streaming, and realize event matching for stream data distribution by an improved matching tree algorithm.

This work is supported by the National Natural Science Foundation of China (No. 61672042), the Program for Youth Backbone Individual supported by Beijing Municipal Party Committee Organization Department (No. 2015000020124G024) and the Training Plan of Top Young Talent in North China University of Technology.