计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(8):72−79 [doi: 10.15888/j.cnki.csa.007588]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于扩展 ASP 的 RDF 流处理系统①

林维淦[1,2], 刘　峰[2], 于碧辉[2]

[1](中国科学院大学, 北京 100049)
[2](中国科学院 沈阳计算技术研究所, 沈阳 110168)
通讯作者: 林维淦, E-mail: l1620119821@163.com

摘　要: 对传感器产生的语义数据流执行复杂推理的能力, 最近已成为语义网社区中的重要研究领域, 而目前大多数 RDF 流处理系统是以 SPARQL (W3C 标准 RDF 查询语言) 为基础实现的, 但这些引擎在捕获复杂的用户需求和处理复杂的推理任务方面存在局限性. 针对此问题, 本文结合并扩展了回答集编程 (Answer Set Programing, ASP) 技术用于对 RDF 流进行连续的处理. 为了验证本方法的有效性, 首先以智能家居本体为实验对象, 并分析传感器设备间的共有特性及复杂事件以构建本体库; 然后基于本体库产生实例对象, 并通过中间件产生 RDF 数据流; 接下来通过扩展 ASP, 充分利用其表达和推理能力以减少推理时间, 并设计了 RDF 流的窗口划分策略等, 然后根据用户的请求, 选择性地进行静态知识库加载等; 最后通过实验与 Sparkwave 和 Laser 进行对比, 证明了该方法在延迟和内存上的性能优势.

关键词: RDF 流; 复杂事件处理; 查询推理; ASP; SPARQL

引用格式: 林维淦,刘峰,于碧辉.基于扩展 ASP 的 RDF 流处理系统.计算机系统应用,2020,29(8):72–79. http://www.c-s-a.org.cn/1003-3254/7588.html

RDF Stream Processing System Based on Extended ASP

LIN Wei-Gan[1,2], LIU Feng[2], YU Bi-Hui[2]

[1](University of Chinese Academy of Sciences, Beijing 100049, China)
[2](Shenyang Institute of Computing Technology, Chinese Academy of Sciences, Shenyang 110168, China)

Abstract: The ability to perform complex reasoning on semantic data streams generated by sensors has recently become an important research area in the Semantic Web community. Currently, most RDF stream processing systems are implemented based on SPARQL (W3C Standard Protocol and RDF Query Language), but these engines have limitations in capturing complex user requirements and processing complex reasoning tasks. In response to this problem, this study combines and extends Answer Set Programming (ASP) technology for continuous processing of RDF streams. In order to verify the effectiveness of this method, we firstly take the smart home ontology as the experimental object, and analyze the common characteristics and complex events between the sensor devices to build the ontology library; then generate instance objects based on the ontology library and generate RDF data stream through middleware. Next, through extending ASP, making full use of its expression, and reasoning capabilities and reducing the reasoning time, a window partitioning strategy for the RDF stream in this method is designed. The static knowledge base is selectively loaded according to the user's request. Finally, the comparison with Sparkwave and Laser through experiments proves the performance advantage of this method in terms of latency and memory.

Key words: RDF stream; complex event processing; query and reasoning; ASP; SPARQL

① 收稿时间: 2020-01-09; 修改时间: 2020-02-08; 采用时间: 2020-03-24; csa 在线出版时间: 2020-07-29

近年来, 越来越多的传感器加入物联网络, 由此持续产生了大量异构的语义数据流, 为了能够更好地表达、更快地处理这些数据流, 如需要根据用户请求实时查询这些数据流中的隐藏知识等, 语义流处理技术被提出[1]. 传感器语义网络 (SSN) 的研究为推进语义流处理方面的发展做出了贡献, 语义网络具有异构、动态变化的特点, 不同设备间的数据异构以及表达方式多样化, 从而导致了信息所有者对这些数据存在不够理解或者理解不一致的诸多问题. 语义物联网 (SWoT)在此基础上引入了语义网技术, 特别是通过扩展 RDF以表示和操作语义数据, 为物联网中的信息及其关系添加了语义描述, 使得用户能够更好理解并且利用这些信息, 有效解决了物联网中的资源异构及流处理的问题. 由此 RDF 流处理 (RSP) 被提出, 并产生了众多以 SPARQL 为基础引擎, 如 C-SPARQL 和 CQELS、Sparkwave 等. 这些引擎主要基于模式匹配技术来处理实时性的流数据, 并在一定限度下可以做推理, 但推理能力仍然有限, 并不能支持复杂的推理模式, 如默认值、常识、偏好、递归和不确定性等. 另一方面, 回答集推理 (ASR) 和流规则 (StreamRule) 分别通过使用 Clingo 和 DL Vhex 解算器来支持较复杂的推理功能, 这些系统通过回答集对子过程进行硬编码, 并重复调用解算器从数据流和给定的规则集中推理出新知识, 但是它们没有提供集成流处理和推理功能等更为灵活的方式. 因此本文在此基础上扩展 ASP来进行持续的 RDF 流处理, 提出了 ASPR (ASP Reasoning) 方法, 并充分利用其良好的表达和推理能力.

1 相关工作

目前该领域研究的热点工作是利用基于逻辑规则的非单调推理技术的表达能力来构建流处理系统 (图 1), 一方面依靠语义流处理技术的方法来表示和处理数据流, 另一方面是基于规则的复杂推理方法. 例如, EP-SPARQL[2] 扩展了 SPARQL 查询语言进行事件处理, 其中 EP-SPARQL 查询被转换成逻辑表达式, 并由基于 SWI-Prolog7 实现的 ETALIS 进行连续推理评估; C-SPARQL[3] 引入了新功能, 如 RDF 流数据类型, 聚合, 窗口管理和时间戳. 另一个解决方案是 Sparkwave[4,5], 也是目前比较流行的一个 RDF 流处理引擎, 它通过 RETE 网络对 RDF 流数据进行持续模式匹配和推理, 在执行的过程中, 将包含在语义查询中的 RDF 图转换成 RETE 网络, 使用了 RDF/RDFs 推理规则对 RDF 流数据进行推理. 另一方面, 一些基于 ASP 的推理引擎最近被提出[6–11], 如 Ticker 和 Laser. Laser 是一种基于逻辑规则以用于分析流推理的框架, 通过不同的方式来抽象化窗口, 并且提供了单调性语义, 但是 ASP 的表达能力并没有得到充分利用, 包括划分数据、优化、聚合等方面.

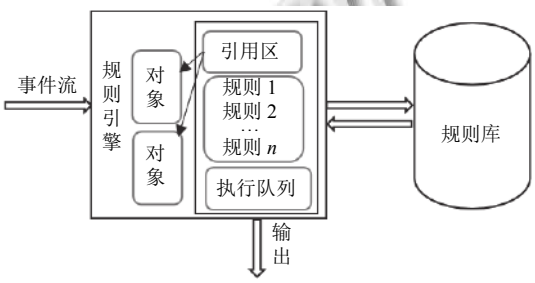

图 1 基于规则的流处理方法

本文在此基础上提出了基于扩展 ASP 的 RDF 流推理 (ASPR) 方法, 该方法对 ASP 进行了扩展, 旨在利用 ASP 的全部功能在 RDF 流上执行推理, 以及将语义流处理和非单调推理无缝结合. 该方法允许用户指定查询请求, 这些请求在 ASPR 引擎中完成注册, 并通过窗口策略连续执行查询推理任务. 本文在最后使用 Sparkwave 和 Laser 及 ASPR 进行对比实验, 来验证用 ASPR 进行 RDF 流推理的有效性和性能优势.

2 关键理论与技术

2.1 RDF 与 RDF 流

资源描述框架 (RDF) 是 W3C 的一个标准数据模型, 描述了资源及相互间的关系.

$$D_i = (S_i, P_i, O_i) \tag{1}$$

$$G = U(D_i) \tag{2}$$

其中, S 指的是实体, P 是指与实体关联的属性, O 指的则是客体或者是属性 P 的值, 每一个实体由唯一的统一资源定位符 (URI) 表示. 大量 RDF 数据通过内在联系构成了带语义信息的图网络结构 G. 传感器动态、实时、有序地产生实例数据, 由此构成 RDF 流, 有序序列 S:

$$S = \cdots, (d_i, t_i), (d_{i+1}, t_{i+1}), \cdots \tag{3}$$

每一组数据均由 RDF 三元组 d 及其时间戳 t 组成, t 表示该条数据产生的时间并且是单调非递减的 ($t_i \le t_{i+1}$). 流数据随着时间变化, 因此其属性值或内部关系也可能不断变化, 所以需要构建高效的 RDF 流处理系统来保证这些数据的时效性, 易失性和顺序性.

2.2 复杂事件处理

复杂事件处理 (Complex Event Processing, CEP) 将每一条传感器数据看成是正在发生的某个事件, 它以事件为驱动[12]. 事件之间存在着时序、因果以及构成关系等各种形式的联系, 当事件流经 CEP 系统时, 事件被聚集到一起并相互关联, 试图去理解它们的关系时, 将会复合生成具有高价值的高层次事件. CEP 技术使得用户能够准确、实时地从流数据中获得感兴趣的信息, 但目前大多数 CEP 系统缺乏了灵活性, 因此在 CEP 的基础上引入了资源描述框架 (RDF) 和网络本体语言 (OWL), 使得在处理数据异构的问题上, 可以对复杂事件进行有效的描述与推理.

2.3 本体建模

本体是用来描述某个语义网络的术语集合, 是对特定领域中的概念及其相关关系的形式化表达,复杂事件处理通过对某领域的事件进行本体建模, 就可以根据其属性等自动进行推理以发现隐藏的知识. 本体建模是进行复杂事件处理的准备阶段, 为了对事件本体进行建模, 一般在建模过程需要详细考虑该事件模式的特点, 在传感语义网络领域通过结合使用现有的 SSN 本体[13,14], 使得构建的本体具有良好的可表达性, 后续进行更有效的查询推理. 本文以智能家居为对象, 结合已有 SSN 本体进行建模, 并生成实例数据进行后面的实验.

2.4 基于回答集编程 (ASP) 的 RDF 流处理

RDF 流处理 (RSP) 在各个领域已变得越来越流行, 如实时城市环境、交通监控, 社交网络信息流, 商业 RFID 数据流等. 在流处理过程中需要用到的 RDF 处理通常需要借助本体推理规则中的传递性、对称性、和等价性等 (表 1) 性质进行. 推理本身是需要经过大量复杂的计算, 并且不断地产生中间结果, 再进行重复的过滤、去重等工作来得到推理结果. 尤其是在实时推理的任务中, 为了保障查询推理的性能, 因此需要构建更高效的流处理系统.

ASP 是基于逻辑规则和对应回答集的声明式编程范式, 主要是针对复杂的搜索问题, 用于逻辑规则推理. 本文提出的 ASPR 处理模型结合 ASP 能够对 RDF 流进行推理, 通过窗口策略和请求来定义 RDF 流, 这些请求通过输入的 RDF 流和背景知识库及规则库来进行动态推理, 并产生输出结果, 图 2 所示.

表 1 推理规则集

Rule name	If	Then add
rdf1	<x p y>	<p rfs:type:Property>
rdfs2	<p rdfs:domain c><x p y>	<x rdf:type c>
rdfs3	<p rdfs:range c><x p y>	<x rdf:type c>
–	–	–
inv1	<p owl:inverseOf q>	<q owl:inverseOf q>
tra	<p rdf:tyze owl:TransitiveProperty><x p y><y p z>	<x p z>

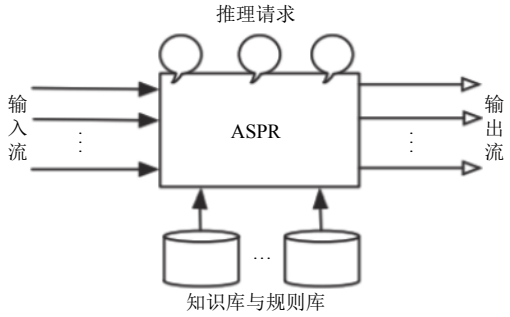

图 2 ASPR 流处理模型

3 ASPR 流处理系统

本章以智能家居传感器语义网络为示例, 来描述本文提出的 ASPR 模型的各个组成模块. 本节分为以下 3 个部分.

3.1 本体构建及 RDF 流实例生成

首先在本体构建阶段, 通过分析智能家居环境的事件模式, 并结合已有的 SSN 本体, 从本体中提取概念和属性联系. 定义 Device、Action、Location、Phenomenon 相关概念及事件, Protege 工具可以快速地根据需求建立好本体库并生成 OWL 文件 (图 3); 用户可以根据具体情况扩展, 自定义传感器设备实体及其他属性等信息.

传感器产生数据后, 需要使用 Jena 等工具对传感器产生的数据进行语义标注. 本文通过上一步构建好的本体文件产生 RDF 实例数据, 同时可以对本体库进行扩展, 形成带有时间戳的 RDF 三元组实例 (清单 1),

该数据表示编号为室内温度传感器设备 In01 在时间戳为 1576814400 时数值为 25.2 摄氏度.

将产生的实例数据保存至 SmartHome.owl 文件中, 以保证每次实验都用到相同的数据, 排除数据不相同带来的结果差异. 最后使用消息中间件 Kafka 按照时间顺序读取并构建待消费的 RDF 流.

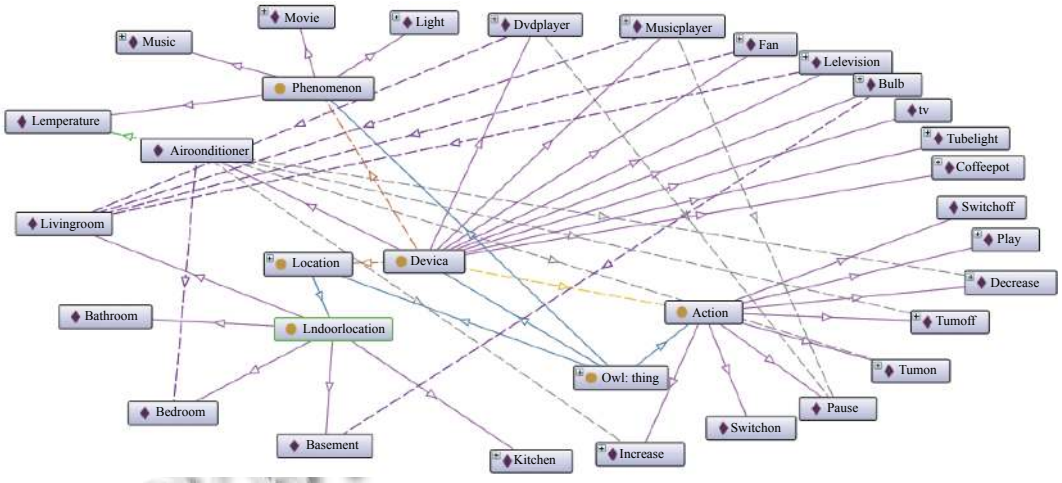

图 3 智能家居 OWL 本体

3.2 RDF 流推理

该模块为本节的核心模块, 模块实现了 RDF 流推理功能. 通过预先设定好查询请求, 使用 ASP 语言来形式化地表达要求和偏好, 请求被 ASP 引擎注册并在输入流、背景知识库及规则库中连续地进行推理. 另外, 通过改进动态加载背景知识库和规则库以减少内存的消耗, 因此本文以用户查询驱动来维护本地静态数据, 即在本地视图中标识最新和过时的数据项, 并分别进行更新和回收. 和大多数查询引擎一样, ASPR 处理模型也需要使用基于窗口的数据流, 并结合基于规则的 CEP 模型, 从输入数据到输出结果, 该过程设计为 3 个步骤:

清单 1. 产生的 RDF 流实例

```
<owl:NamedIndividual rdf:about="http://www.w3c.com/
    ssn/ont/sensor/Temperature_In01_1576814400">
<rdf:type rdf:resource="http://www.w3c.com/ssn/ont/
    sensor/Observation"/>
<sn-owl:floatValue rdf:datatype="&xsd;float">25.2
    </sn-owl: floatValue>
</owl:namedIndividual>
```

(1) 窗口化及预处理. 该过程使用了流操作符来捕获与请求最相关的数据, 并让这些 RDF 数据对知识库进行状态更新, 如选择输入流中时间上最新的子集等.

(2) 查询推理. 该过程对时间窗口内有限的 RDF 数据, 由 ASPR 引擎解析并执行推理计算. 同时, 通过维护当前流的查询中涉及的局部视图, 在产生中间结果后, 再将本地视图与当前流中相关的内容进行连接, 从而动态更新本地视图.

(3) 输出结果. 查询处理完成后将结果呈现给用户.

3.2.1 窗口化及预处理

关于窗口化处理的一些设计如下.

给定 RDF 流 S, 在时间 t 上的窗口为:

$$W(S,t)=\{(d,t')\in S:t'\le t\} \tag{4}$$

为了从流 S 中选择元素, ASPR 支持两种类型的窗口, 分别为基于时间和基于计数的窗口, 每种类型的窗口采用参数$\omega\in N$来定义从流中选择元素的方式. 给定 RDF 流 S, 则在时间 t 上的基于时间的窗口为:

$$W_T^{\omega}(S,t)=\{(d,t')\in S:t'\in[\max\{0,t-\omega\},t]\} \tag{5}$$

给定 RDF 流 S, 在时间 t 上的基于计数的窗口定义为: $W_C^{\omega}(S,t)=W(S,t)$, 并满足以下两个条件:

$$\begin{cases}|W_C^{\omega}(S,t)|=\omega, \text{并且}\\ \forall(d',t')\in W_C^{\omega}(S,t), \nexists(d'',t'')\in S\setminus W_C^{\omega}(S,t)\\ \text{使得} t'\le t''\le t\end{cases} \tag{6}$$

为了处理连续 RDF 流, 使用参数β来表示两个连续窗口间的时间间隔, 在 RDF 流$S(W_T^{\omega,\beta}(S,t))$上, 基于时间的滑动窗口在每$\beta$个时间单位产生一个窗口$W_T^{\omega}(S,t)$.

同理, 基于计数的滑动窗口通过删除旧窗口, 为每个新窗口添加新窗口, 在每β个时间单位产生一个窗口$W_C^{\omega}(S,t)$, 并且该窗口正好包含ω个三元组. 如果删除(添加) 相同时间戳的三元组数使得窗口小于 (大于)ω, 则随机选择要删除 (添加) 的三元组.

3.2.2 查询推理

在上一步选择输入流的窗口化操作后, 接下来是 ASPR 处理模型对用户的请求进行推理. 给定推理请求 *RR* (Reasoning Request):

$$RR = (P, I, R, O) \tag{7}$$

其中, P 表示一组命名空间的常量 (即 URI),也即 RDF; $I = I_{\text{stream}} \cup I_{kb}$, $I_{\text{stream}} = \{(S_i, W(S_i,t)), i = 1..n\}$, 表示一组 RDF 输入流窗口以指定如何从流中提取元素, $I_{kb} = \{kb_j, j = 1..m\}$表示一组静态 RDF 数据, 即背景知识库; R 表示一个规则库; O 表示输出. ASPR 处理模型在时间 t 会对推理请求 RR 进行计算并生成该窗口对应结果. 给定一个推理请求 RR, 在时间 t 产生的结果集为:

$$ans(RR,t) = \Omega(ans(\cup_n^{i=1} W(S_i,t), ans(\cup_m^{j=1} kb_j, R))) \tag{8}$$

其中, $ans(RR,t)$ 的计算步骤共分为两步, 首先在规则集 R 中计算出回答集, 并在静态加载的知识库$\left(\cup_m^{j=1} kb_j\right)$和在时间 t 从输入流产生的窗口$(\cup_n^{i=1} W(S_i,t))$进行查询推理, 然后根据 O 中定义的谓词从回答集中选择输出项, 如下:

$$\begin{cases} \Omega(ans(\cup_n^{i=1} W(S_i,t), ans(\cup_m^{j=1} kb_j, R))) \\ \{p(.) \in ans(\cup_m^{j=1} kb_j, R) : p \in O\} \end{cases} \tag{9}$$

由于产生的回答集可能包含多个, 因此 ASPR 为请求 RR 提供了多个可能的结果集, 表示为$Ans(RR,t) = \{ans_i(RR,t)\}$.

选对知识库的选择性加载策略能减少内存消耗, 因为其中可能会有大量知识与用户需要的事件模式无关, 从而会降低推理效率. 此外, 为了加快数据查找与传输速率, 使用 Redis 进行快速数据映射来加速计算过程. 在对本地视图进行维护 (图 4 所示) 的过程中, 识别出一组与请求相关的映射, 当 RDF 流到来后经过窗口化及预处理后形成子图 (局部视图), 提取组件 (proposer) 从子图中选择用于更新的候选映射集, 然后排序器 (ranker) 根据时间戳对相关的 RDF 进行排序, 直接刷新至时间最新的映射集, 最后维护器 (maintainer) 执行连接 (join) 操作, 以更新本地视图的状态.

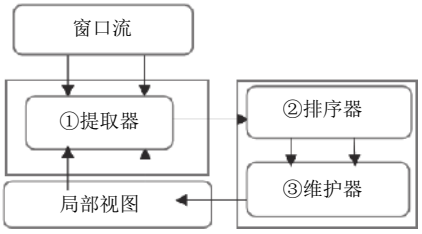

图 4 本地视图更新流程

3.2.3 输出结果

对于结果集, 需要将它进行序列化构建输出流, 再呈现给用户. 在该过程中, 本文给出了一个流操作符表达结果集中的数据输出项, 并为其分配产生输出的时间戳.

RStream 操作符如下:

$$RStream(ans(RR,t)) = \{(d,t) : d \in ans(RR,t)\} \tag{10}$$

对于多个输出, 有:

$$RStream(Ans(RR,t)) = \{RStream(ans(RR,t) : ans(RR,t) \in Ans(RR,t)\} \tag{11}$$

为了生成 RDF 输出流, 输出 O 中的谓词符号 p 必须以 $p(s,o)$ 形式的谓词出现. 此时 RStream 运算符调整如下:

$$RStream(ans(RR,t) = \{(< s,p,o >,t) : p(s,o) \in ans(RR,t)\} \tag{12}$$

3.3 扩展 ASP

对于 ASPR 处理模型, 本文扩展了具有 RDF 流推理功能的 ASP 语言来表达推理请求. 对于一个请求 RR, 由扩展的 ASP 语法进行表达 (清单 2), 首先, 为了方便处理 RDF 格式的数据流, 使用一个前缀词条 (Prefix Clause, IRI 中的一个语法) 来捕获 P 中的元素.

I 中的输入流和静态知识库分别通过 FromStream Clause 和 FromClause 表示. 在 FromStreamClause 中, 每个输入流都与一个 Window 对应, 以使得 ASPR 引擎知道如何从流中提取数据. 在 FromClause 中, 加载的静态知识库由路径指定, 并且 ASPR 处理模型在进行推理之前, 会将它们与输入流集成在一起. R 中的规则遵循 ASP 语言标准, 而扩展的 ASP 依赖于 Clingo 求解器. 用于处理 RDF 流的规则集中引入了谓词符号, 这些符号是通过 prefixName_p 的形式转换成 RDF 三元组<*S*, *P*, *O*>而获得的, 而 P 是在 ASP 内部规则中定义好的谓词. 因此, 带有时间戳的 RDF 三元组 (< *S*, *P*, *O* >, *t*) 将自动转换为 prefixName_p(*s*, *o*, *t*) 的形式.

清单 2. 扩展的 ASP

PrefixClause → # prefix prefixName:<IRI>;
FromStreamClause → from stream streamIRI window;
Window → Time-basedWindow | Tuple-baseWindow
Time-baseWindow → [time number TimeUnit step number TimeUnit]
TimeUnit → d | h | m | s | ms
Tuple-basedWindow → [count number step number]
FromClause → #from <knowledge base>;
RuleClause → ASP rules;
OutputClause → #show predicateSymbol/number;

此外, 在推理之后需要提供输出谓词, *O* 中的输出语句在 OutputClause 中定义, 其变量数量由 predicate Symbol 的参数 number 决定, 当 number=2 时, ASPR 引擎通过将输出 (例如, predicateSymbol(*S*, *O*)) 转换为三元组 (例如, <*S*, predicateSymbol, *O*>),并设定时间戳代表该结果的生成时间, 使用 RStream 标识 (例如, <*S*, predicateSybol, *O*>, *t*).

4 实验与评估

4.1 实验数据

本文以智能家居环境中的传感器为实验对象, 由本体库生成 RDF 实例, 其中包含温度、光、娱乐设备、厨房设备等 21 个类别, 以及位置信息 (Location) 的 5 个类, 15 个动作行为等, 用户可以根据实际需求任意扩展. 以 Device 的具体类为基础生成实例数据, 共包含 200 个不同设备对象的 RDF 数据. 如温度传感器, 以 temperature_In、temperature_Out 分别表示室内外传感器对象, 数字表示设备编号. 生成的实例数据保存在 SmartHome.owl 文件中, 共包含 1000 万条 RDF 三元组, 然后通过 Kafka 按顺序构建 RDF 流, 以保证每一次实验都用到相同且时间上有序的数据.

4.2 评估指标

在进行基于规则的 RDF 流数据的查询推理过程中, 其准确性基本上都已确定, 因此主要的测量指标有两个, 分别是延迟 (ms) 和占用内存 (MB). 其中延迟指的是输入流到达到生成输出之间消耗的时间, 内存消耗则反映了系统执行期间内存的占用情况, 具体是实验过程中在这两个指标上每分钟的平均情况.

4.3 实验方案

为了体现 ASPR 模型的有效性以及与传统流推理模型的性能表现, 本实验通过 ASPR 与目前较为流行的 Sparkwave 流处理系统、以及基于 ASP 流处理系统 Laser 作性能对比, 后者作为参考组来说明 ASP 方法相比于 Sparkwave 的有效性, 并确保其余条件一致. 在 RDF 流推理阶段, 选取窗口的大小为 3 s 和 10 s, 窗口更新频率分别为 1 s 和 5 s, 然后根据用户设定的请求语句进行查询推理. 在此根据构建好的智能家居本体库, 设定已有的基准查询语句 Q1C、Q2C, 基于 ASP 的模型则使用本文扩展的 ASP 语言, 分别设定为 R1C、R2C, 清单 3、4 所示, Q1C 表示需要查询室内外不同传感器的温度, 并根据规则作出推理, 以此产生之后的行为. Q2C 则是更为复杂的推理语句, 在查询模式上比 Q1C 拥有更多的数量和连接运算符. 此外, 为了方便比较, 确保返回相同的数据格式, 因此本文将 SPARQL 中的查询语句关键字 SELECT 改成关键字 CONSTRUCT.

清单 3. Q1C

```
CONSTRUCT {? Id1 sao:hasValue ? v1. ? Id2 sao:hasValue ? v2.}
FROMSTREAM <… /SmartHome/temperature_In>
[ range 3000ms step 1s]
FROMSTREAM <… /SmartHome/temperature_out>
[ range 3000ms step 1s]
FROM<http://localhost:8 080/SmartHome.owl>
WHERE{      ? p1 act:TemperatureLevel.
            ? p2 act:TemperatureLevel.
            ? Id1 ssn:observedProperty ? p1.
            ? Id1 sao:hasValue ? v1 .
            ? Id1 ssn:observedBy temperature_In.
            ? Id2 ssn:observedProperty ? p2.
            ? Id2 sao:hasValue ? v2 .
            ? Id2 ssn:observedBy temperature_Out.
      }
```

清单 4. R1C

```
#from stream <…/SmartHome/temperature_In> [time 3 s
      step 1 s ];
#from stream <…/SmartHome/temperature_Out> [time 3 s
      step 1 s ];
#from <dataset/SmartHome/SmartHome.owl>;
observeryBy(Id):− ssn_observedBy(Id,"_temperature_In");
observeryBy(Id):− ssn_observedBy(Id,"_temperature_Out");
result(Id, V):− observedBy(Id) , sao_hasValue(Id, V)
      ssn_observedBy(Id, P),
```

rdf_type(P, ”ct_TmperatureLevel”);
#show result / 2

4.4 实验结果与分析

本实验以 Sparkwave 与 ASPR 作对比实验结果, Laser 作为两者的参照组进行对比. 当窗口设置为 3 s 时, 对于查询 Q1C(R1C), Q2C(R2C), Laser 模型相比 Sparkwave 可以看出基于 ASP 的流推理方法上的有效性, 尽管性能上并没有太大优势. 而 ASPR 对比 Laser, 在内存上的消耗平均减少接近一半, 在延迟上有平均 2 倍的提升, 说明了 ASPR 改进了 Laser 上没有充分利用 ASP 的表达和推理能力; 而对于 ASPR 与 Sparkwave, 在平均执行速度上前者速度分别约为后者的 2.2 倍和 2.5 倍, 在内存消耗上前者少于一半 (如图 5、图 6).

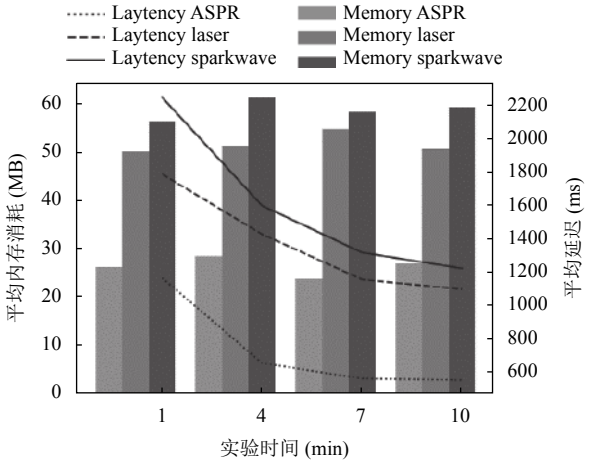

图 5 Window=3 s, Q1C 与 R1C

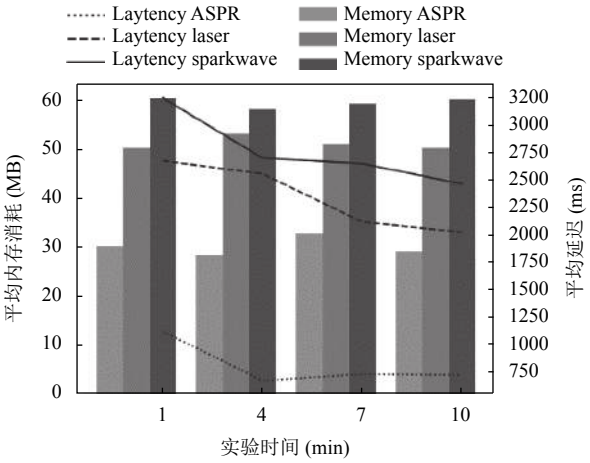

图 6 Window=3 s, Q2C 与 R2C

当窗口大小设置为 10 s 时, 等同在每一个窗口内处理更多的 RDF 流, 而这时对于查询 Q1C(R1C), Q2C(R2C) 的情况是, ASPR 在 Laser 上的优势基本保持不变, 原因可能是都使用了表达能力强的 ASP. 而 ASPR 的平均执行速度分别比 Sparkwave 约快 3.4 倍和 3.3 倍, 内存上则保持相同的优势 (如图 7、图 8). 实验结果表明 ASPR 在延迟和内存消耗方面均优于 Laser 和 Sparkwave 模型, 对于越复杂的查询推理问题, 对 Laser 和 ASPR 推理性能优势没有太大影响, 随着窗口大小逐渐增加, 窗口内处理的数据越多, 并且推理任务越复杂, 可以看到 ASPR 也能够保持更低的延迟和内存消耗.

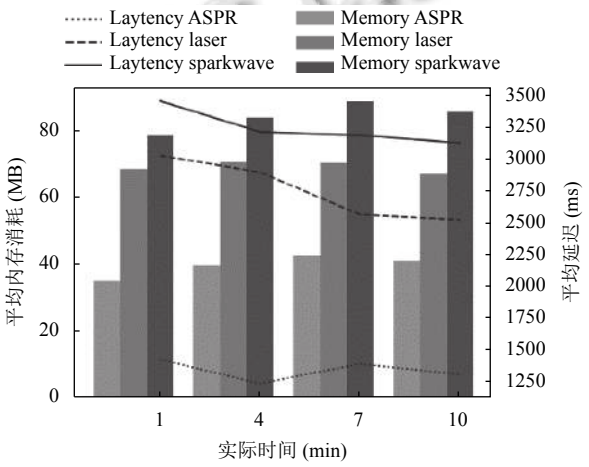

图 7 Window=10 s, Q1C 与 R1C

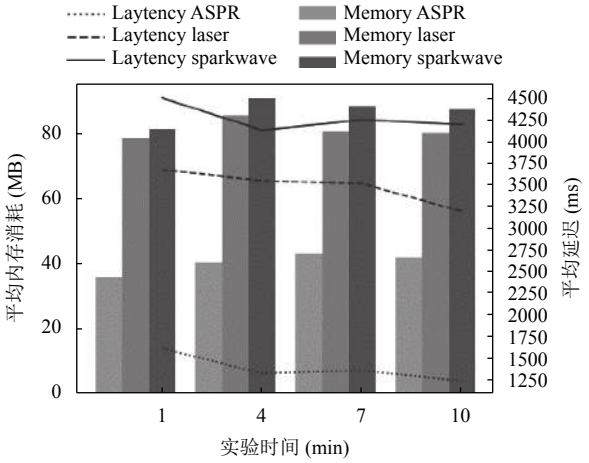

图 8 Window=10 s, Q2C 与 R2C

5 总结展望

在现有技术中有很多用于建模和处理 RDF 流的方法, 但大部分都是以扩展了 SPARQL 为基础. CEP 启发了另一个研究方向[11], 将每个数据元素视为一个原子事件, 并以复杂事件的形式从输入流中提取出隐式知识,

如 EP-SPARQL 以及 Sparkwave, 但其并不能进行复杂的推理请求, 并且推理效率低下. 为了解决并优化这一问题, 本文提出了一种基于扩展 ASP 的方法 ASPR 用于连续执行 RDF 流的复杂推理, 该方法充分利用 ASP 的表达能力来进行 RDF 流推理, 并无缝地将语义流处理与非单调推理相结合, 支持通过将 RDF 流添加到自定义数据类型, 并使用窗口化操作符来捕获与查询推理请求中最相关的数据流部分, 允许以推理请求的形式来表达用户的要求和偏好, 请求在引擎中只注册一次, 然后在 RDF 数据流到来时通过流运算符连续执行推理产生结果, 另外, 为了进一步减少推理时间, 通过有选择性地加载静态知识库, 并使用 Redis 作为映射工具. 此外, 实验表明 ASPR 在内存和延迟性能上更优于最新的 Sparkwave 与 Laser, 但还有进一步优化的地方, 因此在未来的工作中, 有以下两个方向可以继续探讨,一是可以尝试其他流处理框架中的窗口类型和流操作符来与 ASP 技术结合, 如 Lars[15]; 二是采用其他技术来扩大引擎的规模, 如采用并行推理方法等[16–18].

参考文献

1 杜方, 陈跃国, 杜小勇. RDF 数据查询处理技术综述. 软件学报, 2013, 24(6): 1222–1242. [doi: 10.3724/SP.J.1001.2013.04387]

2 Anicic D, Fodor P, Rudolph S, *et al*. EP-SPARQL: A unified language for event processing and stream reasoning. Proceedings of the 20th International Conference on World Wide Web. Hyderabad, India. 2011. 635–644.

3 Barbieri DF, Braga D, Ceri S, *et al*. C-SPARQL: SPARQL for continuous querying. Proceedings of the 18th International Conference on World Wide Web. Madrid, Spain. 2009. 1061–1062.

4 Komazec S, Cerri D, Fensel D. Sparkwave: Continuous schema-enhanced pattern matching over RDF data streams. Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems. Berlin, Germany. 2012. 58–68.

5 王玮昉. 语义物联网中基于 RDF 流的复杂事件处理方法研究 [硕士学位论文]. 大连: 大连海事大学, 2016.

6 Pham TL, Ali MI, Mileo A. C-ASP: Continuous ASP-based reasoning over RDF streams. Proceedings of the 15th International Conference on Logic Programming and Nonmonotonic Reasoning. Philadelphia, PA, USA. 2018. 45–50.

7 Kolchin M, Wetz P, Kiesling E, *et al*. YABench: A comprehensive framework for RDF stream processor correctness and performance assessment. Proceedings of the 16th International Conference on Web Engineering. Lugano, Switzerland. 2016. 280–298.

8 Dehghanzadeh S, Dell'Aglio D, Gao S, *et al*. Approximate continuous query answering over streams and dynamic linked data sets. Proceedings of the 15th International Conference on Web Engineering. Rotterdam, the Netherlands. 2015. 307–325.

9 Beck H, Eiter T, Folie C. Ticker: A system for incremental ASP-based stream reasoning. Theory and Practice of Logic Programming, 2017, 17(5–6): 744–763.

10 Bazoobandi HR, Beck H, Urbani J. Expressive stream reasoning with laser. Proceedings of the 16th International Semantic Web Conference. Vienna, Austria. 2017. 87–103.

11 Do TM, Loke SW, Liu F. Answer set programming for stream reasoning. Proceedings of the 24th Canadian Conference on Artificial Intelligence. St. John's, Canada. 2011. 104–109.

12 何恒靖, 赵伟, 黄松岭. 复杂事件处理技术的应用现状及展望. 计算机工程, 2017, 43(1): 20–26, 31. [doi: 10.3969/j.issn.1000-3428.2017.01.004]

13 张晓双. 基于智能家居的本体查询扩展研究 [硕士学位论文]. 青岛: 中国海洋大学, 2014.

14 王海渊, 张雅涵, 黄佳进, 等. 语义传感器 Web 中传感器本体的构建及应用. 计算机系统应用, 2018, 27(10): 80–84. [doi: 10.15888/j.cnki.csa.006574]

15 Beck H, Dao-Tran M, Eiter T. LARS: A logic-based framework for analytic reasoning over streams. Artificial Intelligence, 2018, 261: 16–70. [doi: 10.1016/j.artint.2018.04.003]

16 邱慧. RDF 数据分布式查询处理与优化方法研究 [硕士学位论文]. 哈尔滨: 哈尔滨工业大学, 2018.

17 叶怡新. 分布式 RDF 数据并行推理方法研究与实现 [硕士学位论文]. 福州: 福州大学, 2017.

18 Pham TL, Ali MI, Mileo A. Enhancing the scalability of expressive stream reasoning via input-driven parallelization. Semantic Web, 2019, 10(3): 457–474. [doi: 10.3233/SW-180330]