****

本科毕业设计(论文)

论文翻译

GRADUATION DESIGN(THESIS)

Thesis Translation

|  |  |
| --- | --- |
| 题 目 | 基于Spring MVC框架的物业管理系统设计与实现 |
| 学生姓名 | 杨成 |
| 指导教师 | 李玺 |
| 学 院 | 信息科学与工程学院 |
| 专业班级 | 计算机1404 |

本科生院制

2018年3月

JSON数据流查询

Yang Bo

摘要

数据流管理系统（DSMS）类似于数据库管理系统（DBMS），但可以直接在联机流中搜索数据。使用mediator-wrapper方法，可扩展数据库系统Amos II允许查询不同种类的分布式数据资源进行查询。 它已被扩展为流数据类型以查询可能的无限流，这提供了DSMS功能。现在，越来越多的Web应用程序开始以JSON提供他们的服务这是一个基于文本的格式，用于表示简单数据结构和关联数组的人类可理解的格式。 例如，全球最受欢迎的网站之一Twitter已经开发了一个流接口，可以注册接收大量JSON格式的Twitter消息。 另一个流行网站Facebook和一些天气服务也通过JSON提供流接口。该项目的目标是开发一个通用的JSON流阅读器作为Amos AIwrapper，允许DSMS轻松查询和管理这些流。 为了实现这样的系统，定义了三个原始外部函数，以允许使用Amos II的查询语言AmosQL查询JSON流。 在流行的Twitter社交网络数据流中体现了开发的包装的有用性。

目录

[1. 简介 3](#_Toc27357)

[2. 背景 5](#_Toc10192)

[2.1 数据库管理系统 5](#_Toc26292)

[2.1.1关系数据模型 5](#_Toc4542)

[2.1.2结构化查询语言 5](#_Toc15931)

[2.2 数据流管理系统 6](#_Toc15165)

[2.2.1 DSMS架构 6](#_Toc25865)

[2.2.2 流式查询语言 7](#_Toc2397)

[2.3 JSON 8](#_Toc27310)

[2.3.1 JSON 数据模型 9](#_Toc384)

[2.4 Twitter流媒体API 11](#_Toc23461)

[2.4.1受保护的和公共账户 11](#_Toc17557)

[2.4.2认证和连接 11](#_Toc27162)

[2.4.3流式访问方法 11](#_Toc17727)

[2.4.4状态/过滤器 12](#_Toc19103)

[2.4.5解析响应 13](#_Toc18499)

[2.5 Amos II 13](#_Toc28541)

[2.5.1数据模型 13](#_Toc32014)

[2.5.2外部接口 14](#_Toc24458)

[3. JSON流查询系统（JSQ） 15](#_Toc6076)

[3.1示例查询和结果 15](#_Toc3241)

[3.2系统架构 17](#_Toc24584)

[3.3实现 18](#_Toc28518)

[3.4评估 20](#_Toc1292)

[4.相关工作 21](#_Toc6215)

[5.结论和未来的工作 21](#_Toc17344)

[参考文献： 22](#_Toc11463)

[附录： 24](#_Toc7821)

1. 简介

JSON（JavaScript Object Notation） [12] 是一种轻量级的计算机数据交换格式，它是一种基于文本，人类可理解，并且独立于语言的表示简单的数据结构和关联数组。 近年来，越来越多的网络应用程序，如雅虎和谷歌开始用JSON提供他们的一些Web服务。

网络监控，电信数据管理，制造和传感器网络等越来越多的应用正在处理一种新型的数据，其形式为连续数据流，而不是传统数据库中的存储表格。 他们的客户需要长时间运行的连续查询，而不是一次性查询。 数据流管理系统（DSMS）[10]处理联机流查询以及对常规存储数据的查询。

在可扩展的主存DBMS [17] Amos II [21]中，数据库查询以AmosQL [4]表示，这是一种基于函数的查询语言。 AmosQL已经扩展了一个特殊的流数据类型来支持可能的无限流的查询。 这个扩展以及一些流原语为Amos II提供了DSMS功能。

Amos II还提供了一个通用的外部函数机制，并在Amos II和编程语言C，Lisp和Java之间具有本地API。 通过使用外部函数接口，可以通过AmosQL查询外部数据源。

Twitter是当今全球最受欢迎的网站之一。 这是一个免费的社交网络，使用户能够分享和发现现在正在发生的事情。 Twitter开发了一个通用API来访问其内部数据库以及一个可以注册的流接口，以接收大量的Twitter消息。 这个项目特别感兴趣的是Twitter流媒体API [5]，它以JSON对象流的形式返回数据。

该项目的目的是使用外部函数和库为JSON流数据开发通用的AMOS II包装器。因此，定义了一个JSON流查询系统（JSQ）。现有的AmosQL扩展为支持查询JSON流，而不是开发新的查询语言。 JSQ系统主要可以分为两个功能组件：认证和查询。 第一个将URL，用户名和密码作为参数发送给Web应用程序进行身份验证，并在传递后返回连接标识。 第二个函数使用连接标识查询流数据并将它们处理为Amos II可识别的数据类型。

作为概念证明，系统针对Twitter流接口发出的消息进行测试。 Twitter流接口是基于HTTP的接口，以URL，用户凭证和关键字作为参数，以JSON格式返回连续的Twitter消息流。

为了实现封装，我们研究了Twitter流的公共接口以及其他一些概念，如HttpClient [8]，它用于生成HTTP请求并递增地返回响应数据。

本报告的其余部分安排如下。 第2节介绍与该项目相关的技术。 第3节介绍包装器的实现细节。 第4节介绍相关工作。 该报告已完成，并有结论和未来的工作。

1. 背景

2.1 数据库管理系统

数据库管理系统（DBMS）是使用户能够创建和维护数据库的一系列程序[17]。 它允许用户和其他软件通过使用某种数据模型以结构化的方式存储和检索数据，例如， 数据用表格表示的关系数据模型。 它还为需要操作数据库的用户提供适当的语言和界面。

DBMS用于存储持久数据并支持复杂的应用程序查询。 尽管此模型充分代表了商业目录或个人信息存储库，但许多当前和新兴的应用程序都需要支持对快速变化的数据流进行在线分析。 因此，将在2.2章中介绍一种管理流数据的新型系统，即所谓的数据流管理系统（DSMS）。

2.1.1关系数据模型

关系模型将数据库表示为关系集合。 非正式地，每个关系类似于一张价值表[17]。 表中的每一行都是相关数据值的集合，表示通常对应于真实世界实体或关系的事实。 表名和列名用于表示每行中值的含义。 在正式的关系模型术语中，一行称为元组，一个列标题称为属性，而该表称为关系。

图2.1显示了一个表格的例子：

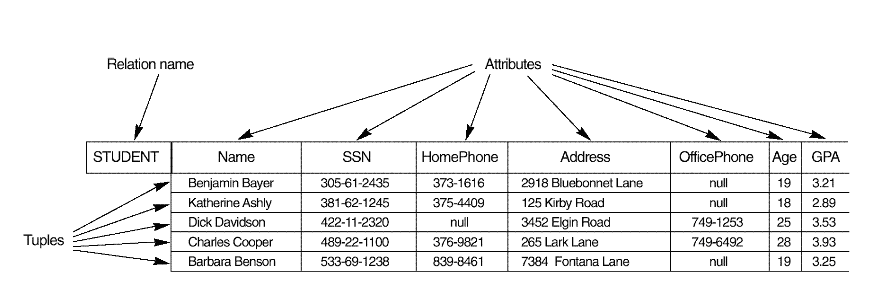


图2.1关系格式的数据

2.1.2结构化查询语言

SQL是用于定义，查询和更新商业关系数据库管理系统的标准语言。 它分别使用表，行和列来表示关系，元组和属性[17]。 创建新表的一般语法是：

create table <table name>

从数据库中搜索信息的一个基本语句是SELECT语句。 SELECT语句的基本形式，有时称为select-from-where语句块，由三个子句SELECT，FROM和WHERE组成，具有以下形式[17]：

SELECT <attribute list>

FROM <table list>

WHERE <condition>

属性列表是逗号分隔的属性名称列表，其值将由查询检索; 表格列表是处理查询所需的关系名称的逗号分隔列表; condition是一个条件（布尔）表达式，用于标识要由查询检索的元组。

2.2 数据流管理系统

传统的数据库管理系统最适合在有限存储的数据集上运行一次查询。 在诸如网络监控，电信数据管理，点击流监控等现在的许多应用中，数据采用连续数据流的形式而不是有限的存储数据集，并且客户端需要长时间运行的连续查询而不是一次性的查询[7]。

传统数据库管理系统在支持流媒体应用方面的局限性已经得到认可，促使研究扩大现有技术并构建一种称为数据流管理系统（DSMS）的新型系统来管理流媒体数据。

2.2.1 DSMS架构

DSMS的抽象体系结构如图2.2所示[16]

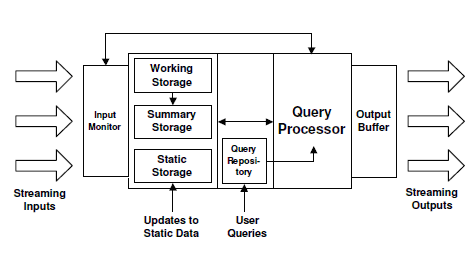


图2.2数据流管理系统的抽象体系结构

在图2.2中，输入监视器可能会通过丢弃数据包来控制输入速率。 输入数据通常存储在三个分区中：临时工作存储（例如，用于窗口查询），流概要的概要存储，以及元数据的静态存储（例如，每个源的物理位置）。 长时间运行的查询在查询库中注册，并被放入组中进行共享处理。 对流的当前状态进行快照查询也可能出现。 在DSMS系统中注册连续查询时，将从中编译查询计划。 查询计划由运算符组成，这些运算符执行实际的处理，队列，在运算符之间移动时缓存元组（或元组的引用），以及存储运算符状态的提要。 查询处理器与输入监视器通信，并可响应于输入速率的改变而重新优化查询计划。 结果流式传输给用户或临时缓冲[16]。

2.2.2 流式查询语言

对于流中的简单连续查询，可以使用关系查询语言（如SQL），将关系引用替换为流的引用，并在结果中创建新的元组。

然而，随着连续查询变得越来越复杂，例如，通过添加聚合，子查询，窗口构造以及流和关系的连接，应用于这些查询的传统关系语言的语义很快变得不清楚[16]。 为了解决这个问题，设计了一些用于流数据的查询语言。 例如，斯坦福大学开发了一种名为连续查询语言（CQL）的新查询语言[1]。

CQL的抽象语义基于两种数据类型 - 流和关系 - 以及这三种类型的运算符：运算符,它们从流中产生一个关系（流到关系），产生与其他关系的运算符关系（relation-to-relation）以及从关系（流到关系）产生流的运算符。

一些示例CQL查询如下[1]：

1.Select Istream(\*) From S [Rows Unbounded] Where S.A > 10

流S通过应用[Rows Unbounded]被转换为关系。 关系关系过滤器“S.A> 10”作用于该关系，并且Istream（）函数将过滤关系作为结果进行流式传输。

2.Select \* From S1 [Rows 1000], S2 [Range 2 Minutes] Where S1.A = S2.A And S1.A > 1

上面的查询是流S1的最后1000个元组和在前2分钟到达的流S2的元组的窗口连接。

3.Select Rstream(S.A, R.B) From S [Now], R Where S.A = R.A

这个连续的查询基于流S中的每个元组来探测存储的表R并且流式传输结果。

可扩展的主内存DBMS Amos II通过特殊的流数据类型扩展了其查询语言AmosQL，以允许查询可能的无限流，从而启用DSMS功能。 这些扩展在本工作中用于提供由Twitter Streaming API [5]提供的JSON流的查询。

2.3 JSON

JSON（JavaScript Object Notation）是一种轻量级的数据交换格式，它基于JavaScript编程语言的一个子集，标准ECMA-262第3版 - 1999年12月[12]。 像XML1一样，它是一种基于文本的人类可读格式，用于表示简单的数据结构和关联数组。 XML不如JSON那样适合于数据交换，因为它带有大量的额外信息，并且与大多数编程语言的数据模型不匹配。 相反，JSON完全是独立于语言的，它使用C语言家族的程序员熟悉的惯例，如C，C ++，C＃，Java，JavaScript，Perl，Python等等。

XML和JSON的例子见图2.3和2.4 [13]：

XML（可扩展标记语言）是一套电子编码文档的规则。

{"widget": {

"debug": "on",

"window": {

"title": "Sample Konfabulator Widget",

"name": "main\_window",

"width": 500,

"height": 500

},

"image": {

"src": "Images/Sun.png",

"name": "sun1",

"hOffset": 250,

"vOffset": 250,

"alignment": "center"

},

"text": {

"data": "Click Here",

"size": 36,

"style": "bold",

"name": "text1",

"hOffset": 250,

"vOffset": 100,

"alignment": "center",

"onMouseUp": "sun1.opacity = (sun1.opacity / 100) \* 90;"

}

}}

图2.3 JSON示例

<widget>

<debug>on</debug>

<window title="Sample Konfabulator Widget">

<name>main\_window</name>

<width>500</width>

<height>500</height>

</window>

<image src="Images/Sun.png" name="sun1">

<hOffset>250</hOffset>

<vOffset>250</vOffset>

<alignment>center</alignment>

</image>

<text data="Click Here" size="36" style="bold">

<name>text1</name>

<hOffset>250</hOffset>

<vOffset>100</vOffset>

<alignment>center</alignment>

<onMouseUp>

sun1.opacity = (sun1.opacity / 100) \* 90; </onMouseUp>

</text>

</widget>

图2.4 XML示例

上述示例分别以JSON和XML格式表示相同的数据。 如果没有XML中使用的复杂层标签，JSON对于人类和计算机来说都更容易读写。

2.3.1 JSON 数据模型

JSON有两个基本的（可能嵌套的）结构：JSONObject和JSONArray。

JSONObject是名称/值对的集合。 它以{（左大括号）开始并以}（右大括号）结束。 每个名称后跟:(冒号），每个名称/值对用逗号（，）分隔[12]。 JSONObject的结构如图2.5所示。

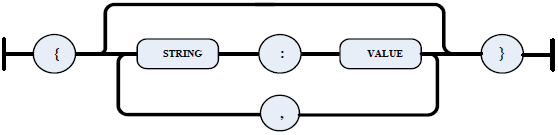


图2.5 JSONObject的结构

在各种语言中，这被实现为对象，记录，结构，字典，散列表，键控列表或关联数组。 在AmosQL中，它对应于一个名为Record的数据类型。

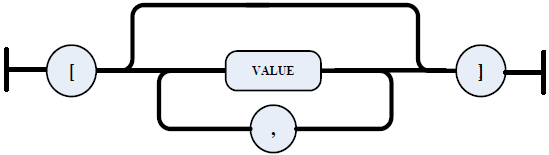


图2.6 JSONArray的结构

JSONArray是值的有序集合。 它以左括号（[）开始并以右括号（]）结束。 值由逗号（，）分隔[12]。 JSONArray的结构如图2.6所示。

在大多数语言中，这是作为数组，矢量，列表或序列实现的。 在AmosQL中，它被映射到一个名为Vector的数据类型。值可以是双引号中的字符串，数字，true，false，null，JSONObject或JSONArray。

这些结构可以嵌套。 图2.7给出了价值的描述。

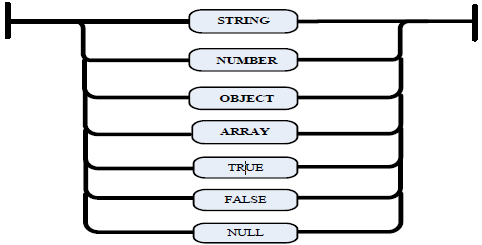


图2.7可能的数据类型的值

以上结构是通用数据结构。 几乎所有的现代编程语言都以不同的形式支持它们。 有意义的是，与编程语言互换的数据格式也可以基于这些结构[12]。

在目前的工作中，使用相应的Amos II数据类型表示的JSON对象用于表示通过互联网传送的JSON对象流中的内容。

2.4 Twitter流媒体API

Twitter是由Jack Dorsey于2006年首次创建的，现在已经在全球范围内获得了知名度和知名度。这是一个免费的社交网络和微博客服务，它的用户可以发送和阅读被称为推文的消息[7]。推文长度不得超过140个字符，并且可以通过Twitter网站，短消息服务（SMS）或外部应用程序发送。发件人可以将发送限制在朋友圈中的发件人，或默认情况下允许开放访问[7]。

Twitter Streaming API允许近实时访问Twitter公共状态的各种子集1。但是，受限资源的访问受此API的限制，并且只有在接受附加的服务条款文档[5]后，才会根据具体情况授予访问权限。

2.4.1受保护的和公共账户

有两种Twitter帐户：受保护和公共。受保护的帐户是属于明确检查其个人资料中的“保护我的推文”选项的用户的帐户。只有由非受保护帐户创建的状态才是流API中的候选人。

2.4.2认证和连接

当连接到Streaming API时，需要HTTP基本认证[8]，这意味着客户端必须提供凭据 - 以用户名和密码的形式 - 有效的Twitter帐户。

由于Streaming API是基于HTTP的，因此需要使用HTTP客户端来形成HTTP请求。所选的HTTP客户端应该能够递增地返回响应数据[5]。大多数健壮的HTTP客户端满足这个要求在这个项目中，Apache HttpClient [8]被选择来处理这个用例。

要访问Streaming API，每个帐户只能创建一个流式连接。后续来自同一个账户的连接可能导致先前建立的连接被断开[5]。无论成功与否，过多的连接尝试都会导致自动暂时禁止客户端的IP地址[5]。

2.4.3流式访问方法

目前，Twitter的Streaming API中提供了四种方法：

状态/流水

返回所有公共状态。

状态/样品

返回所有公共状态的随机样本。默认访问级别提供了从statues / firehose方法返回的一小部分结果。

状态/过滤器

返回匹配一个或多个筛选谓词的公共状态2。

状态/转推

返回所有retweets3。

在默认访问级别下，所有帐户都可以访问方法状态/样本和状态/过滤器。访问其他方法需要与Twitter进行特殊安排，很少有应用程序需要这种访问级别。由于Twitter不建议应用程序访问状态/ firehose和statues / retweet方法，因此我们只会考虑此项目中的前两种方法。此外，由于状态/过滤器方法可以使用谓词来过滤流数据，使其结果更有意义，因此JSQ系统主要基于状态/过滤器方法进行开发和测试。状态/过滤器的详细描述将在2.4.4章节中。

2.4.4状态/过滤器

状态/过滤器方法返回匹配一个或多个过滤器谓词的公共状态。

必须指定至少一个谓词参数，track或follow，[6]。 通过follow参数，该方法返回给定用户集合的公共状态。 使用track参数，该方法返回在其文本字段中包含指定关键字的数据。 跟踪是不区分大小写的。 术语是精确匹配的，并且精确匹配忽略了标点符号。 短语，带空格的关键词不被支持[6]。 包含标点符号的关键字只能完全匹配令牌[6]。

以下是Twitter Streaming API文档提供的关于如何在curl1中指定跟踪谓词的示例：

示例：创建一个名为'tracking'的文件，其中包含，完全排除

引号：“track =篮球，足球，棒球，足球，足球”。 执行：curl -d @tracking http://stream.twitter.com/1/statuses/filter.json -uAnyTwitterUser：密码。 您将收到有关各种重要运动会主题和事件的JSON更新。

示例：创建一个名为'follow'的文件，其中包含，正好和不包含

引号：“跟随=12,13,15,16,20,87”。执行：卷曲-d

@遵循http://stream.twitter.com/1/statuses/filter.json -

uAnyTwitterUser：密码。 您将收到来自Jack Biz，Crystal，Ev，Krissy的JSON更新，但不是Jeremy，因为他是私人用户。

两个可选的查询参数（计数和分隔符）也是有效的。 计数表示要考虑与实时流一起交付的先前状态的数量。 它的范围是-150,000到150,000。 积极的价值观可以无缝地转换到现场直播。 当历史数据流2完成时负值终止，对调试有用[6]。 分隔符表示应该在流中限定状态。 状态由一个长度（以字节为单位），一个换行符以及恰好为长度字节的状态文本表示[6]。

总之，状态/过滤方法可以这样描述[6]：

网址：http：//stream.twitter.com/1/statuses/filter.format

格式：xml，json

方法：POST

参数：count，分隔，跟随，跟踪

返回：状态元素流

2.4.5解析响应

Streaming API可以以XML或JSON格式返回数据。 Twitter鼓励客户使用更紧凑的JSON表示。

从Streaming API中解析JSON响应很简单：每个对象都在自己的行上返回，并以回车结束[6]。 换行符（\ n）可能出现在对象元素（例如状态对象的文本元素）中，但回车符（\ r）不应该[5]。 解析XML更具挑战性，因为对象包含换行符和回车符。

在这个项目中，收到的流数据是JSON格式。

2.5 Amos II

Amos II是一个对象关系DBMS。 它允许使用其mediator-wrapper方法来查询不同种类的分布式数据资源。 Amos II在AmosQL中表达了它的数据库查询，这是一种基于函数的查询语言[21]。

2.5.1数据模型

数据库中的所有实体均表示为对象并由系统管理。 它们主要分为两类：文字和代理[14]。 文字是系统维护的对象，例如数字和字符串。 代理对象由系统用户定义，并被定义为具有明确的对象标识符（OID）[14]。 代理人的例子可以是代表真实世界实体的对象，如汽车或人。

Amos II中的对象被创建为数据类型的实例。 创建新类型的一般语法是[19]：

create type <typename>

鉴于类型，一个新的对象被定义为：

create <typename> instances <variable>

Amos II中的函数模拟对象的语义，例如对象的属性，对象的计算以及对象之间的关系[20]。 函数定义由签名和实现组成。 该签名定义参数和结果参数的类型和可选名称。 该实现使用给定的参数值计算结果。 创建函数签名的一般语法是[19]：

create function <functionname(type)>

-> <return type>

这个功能可以被用户或其他功能调用。

2.5.2外部接口

Amos II包含了通过为每种外部源定义包装器来访问不同外部数据源的基元数目[19]。使用包装器可以使用AmosQL查询外部数据源。

从Amos II访问外部系统的基础是定义外部功能[19]。外部函数在外部编程语言中定义，Amos II目前提供了Java，C和Lisp的接口。在这个项目中，JSON流封装器是用Java实现的。

在Java中实现一个外部函数通常需要三个步骤[4]：

1. 在Java代码中实现函数。在Java中实现的外部函数与任何其他具有签名和主体的Java函数类似，但它们总是有两个参数：上下文和元组。上下文是由系统管理的数据结构，用于在外部函数和系统其余部分之间传递信息[4]。参数元组是一个Tuple对象，它持有函数[4]的参数和结果。语法是：

public void <functionname>(CallContext cxt, Tuple tpl) throws AmosException;

1. 使用AmosQL与Amos II挂钩。 通过执行AmosQL语句[4]必须为外部函数分配一个函数resolvent：

create function <fn>(<argument declaration>) -><result declaration>

as foreign 'JAVA:<class file>/<method>';

其中fn是AmosQL中外部函数的名称，参数声明是参数的签名，结果声明是结果的签名，类文件是在Java中实现的类（文件）的名称， 方法是实现它的方法的名称。 这个定义是通过调用方法execute或通过顶层循环中的AmosQL命令从Java完成的[4]。

1. 定义一个可选的成本提示来估计执行该功能的成本。 Amos II函数可以是多方向的，这意味着当一些结果已知并且一些相应的参数值不是[19]时，它们的逆可以被执行。 因此，一个多方向的外部功能可能会有不同的执行成本。 成本提示用于告诉查询优化器应该在复杂查询中选择哪个实现作为最有效的实现[19]。

3.JSON流查询系统（JSQ）

JSQ是对Amos II的扩展，以支持包装JSON流数据。 从用户的角度来看，它是一个可以通过调用Amos II外部函数接收JSON流数据并使用AmosQL查询流数据的系统。 JSON流由Web应用程序（例如Twitter Streaming API）生成。 该系统的结构如图3.1所示。

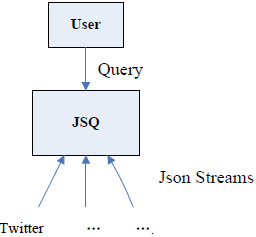


图3.1 JSQ的用户级体系结构

3.1示例查询和结果

1. 在Twitter上查找最近发布的关于“奥巴马”的5条推文：

JavaAMOS 1>set :s=jsonstream(0,{"track":"Obama"}); 0.026 s

JavaAMOS 2>select r["text"] from Record r where r in section(:s,0,5);

查询结果是：

"RT @washingtonpost: President Obama will promote more education spending in his State of the Union speech Wednesday. - http://wpo.st/PLR ..."

"An article about President Obama support for a new agency to protect consumers against lending abuses misstated... http://bit.ly/81VFXC"

"US troop buildup is big; Afghan buildup is key (AP): AP - President Barack Obama has his troop surge. Afghan 7FEyBA !"

"Shepard Fairey Faces Criminal Investigation In AP/Obama Poster Case (source: Arts Journal): ''In October, the L.A.... http://bit.ly/dgsjr8" "Loaded: Jobs and Obama on the same day: Two important dudes are giving

big keynote addresses today: Steve Jobs and... http://bit.ly/aU0Nqu"

执行时间约为12秒。

1. 目前在Twitter上查找五个正在讨论'奥巴马'的人的位置：

JavaAMOS 1>set :s=jsonstream(0,{"track":"Obama"}); 0.016 s

JavaAMOS 2>select r["user"]["location"] from Record r where r in section(:s,0,5);

查询结果：

"Miami"

"null"

"iPhone: 40.769150,-73.865921"

"California"

"Canada"

执行时间是6.668秒

1. 从下面的人和他们的名字中实时获取推文：

JavaAMOS 1>set :s=jsonstream(0,{"follow":"92778994,70900570,26480012"} );

JavaAmos 2>select r["text"],r["user"]["name"] from Record r where r in :s;

查询结果：

<"This is a testing tweet. ","Bo Yang">

<"This is the second testing tweet.","Bo Yang">

…

3.2系统架构

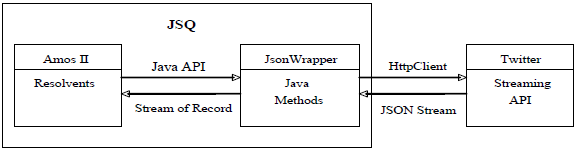


图3.2 JSQ的系统架构

图3.2显示了基于Twitter Streaming API的JSON流查询系统的体系结构。 左边是Amos II外部函数在Java中定义并由Amos II通过Java API调用的系统核心。 Java类被称为JsonWrapper。 它被实现为通过Jakarta Commons HttpClient访问Web应用程序，并将接收到的JSON流数据处理为Amos II可识别的类型的流数据（如记录）。 因此Amos II的用户可以通过使用这个系统直接查询JSON流。 值得注意的是，JsonWrapper右侧的组件可以是其他Web应用程序，它们将数据作为JSON流返回。

3.3实现

由于Twitter Streaming API将数据作为JSON流返回，因此这是演示JSQ系统的好实例。 JSQ只需要Twitter URL，帐户信息和查询关键字作为HttpClient的参数并将它们传递给服务器端。 JSQ的实现是通用的，并且不包含Twitter相关的库，因此它可以轻松地应用于其他JSON流Web服务。

JSQ的实现可以分为两个阶段：

阶段1：使用给定的URL，用户名和密码初始化HttpClient连接。 此步骤验证URL，用户名和密码是否有效。 如果是，连接将在JSQ系统中存储一个标识符，稍后可以被其他HttpClient方法重用。 如果不是，则会抛出异常。

阶段2：使用上面定义的连接将查询参数发送到服务器端，并将响应数据作为JSON流获取。 然后将JSON流处理为Amos II可识别数据类型的流，例如， 记录或矢量。 当流查询结束时（例如，通过停止条件或显式中断），finally {...}语句中止HTTP方法的执行并释放HTTPClient方法所使用的连接。 这是保持事物流动的关键步骤，因为我们必须告诉HttpClient我们已经完成了连接，现在可以重用资源[9]。

上述阶段在Java中实现，因为Java API由所有HttpClient，JSON和Amos II提供。 每个阶段都作为一个独立的外部功能来实现。 根据Amos II的包装机制，在使用这些外部功能之前，需要将它们连接到Amos II。 这是通过创建函数签名并将其分配给相应的外部函数完成的。 因此，该系统由三个外部函数定义组成，如图3.3所示：

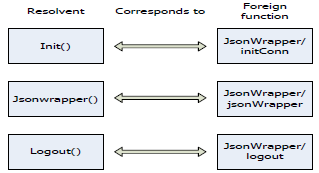


图3.3外部函数及其相应的resolvent

在Amos II中，这些功能被定义为：

create function init(charstring url,charstring name, charstring pass)-> Integer

as foreign "JAVA:JsonWrapper/initConn";

create function jsonwrapper(Integer id,Record r)

->Bag of Record

as foreign "JAVA:JsonWrapper/jsonWrapper";

其中id是从init（）函数返回的连接标识符，r是查询参数，例如， {“track”：“a”}

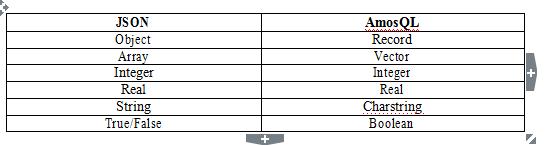
Amos II中的Stream数据类型支持对可能无限流的查询，函数jsonstream（）被定义为流jsonwrapper（）生成的不确定结果。 系统函数streamof指示一个流而不是一个包被返回：

create function jsonstream(Integer id,Record r)

->Stream of Record

as streamof(jsonwrapper(id,r));

Twitter发布的流数据是一系列JSONObjects。 那些JSONObjects将被处理成类型为Record的Amos II对象，其中Amos II中的记录表示名称/值对，与JSONObject的结构相同。 如果流数据是JSONArray格式，则在返回到Amos II之前，它将被处理成Vector类型的对象。 换句话说，AmosQL被扩展为在JSQ系统中支持JSON流查询。 下表列出了从JSON数据类型到Amos II数据类型的类型映射：



根据此表，除了JSONObject和JSONArray之外，JSON中的其他基本数据类型将映射到Amos II中的相应文字对象。 这些扩展允许用户在JSON流中搜索仅使用高级声明式AmosQL语法表示的复杂条件。 虽然通用过滤器方法只允许在JSONObjects的文本字段中搜索断言的分离，但此扩展允许在JSONObjects的任何字段上搜索任何谓词的分离和连接。 例如，它可以查询包含关键字“a”和“is”的推文，如下所示：

JavaAMOS 6> set :m=jsonstream(0,{"track":"a"}); 0.016 s

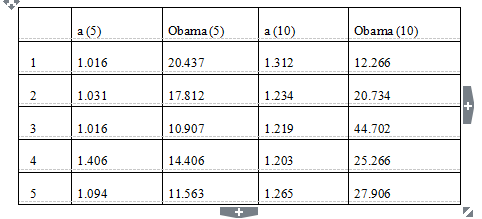
JavaAMOS 7> set :n=jsonstream(0,{"track":"is"}); 0.015 s

JavaAMOS 8> select r1["text"] from Record r1,Record r2 where r1 in section(:m,0,2) and r2 in section(:n,0,2) and r1["id"]=r2["id"]; 4.89 s

3.4评估

查询的执行时间可能受到HTTP请求中查询参数内容，查询时间等因素的影响。 例如，使用“奥巴马”和“一个”每秒钟的结果数量会非常不同。

以下是执行时间表，用Twitter Stream API分别查询带有关键字“a”和“Obama”的推文，分别为5个和10个结果。 时间单位是秒。



上面的表格不应被视为对JQS系统的评估，而是表明不同查询的响应强度。 执行时间高度依赖于返回Twitter结果的频率。 根据上表，显然询问“奥巴马”比“a”要花费更长的时间，因为“a”比“奥巴马”更频繁地出现在推文中。 即使您在一天或一段时间的不同时间请求相同的关键字，执行时间也会有很大差异。 换句话说，在长时间运行的流应用中，流的数据和到达特性可能会随着时间而显着变化[1]。

4.相关工作

目前大多数实施的数据流管理系统，如Aurora [3]和STREAM [1]，都用于监测传感器，网络和库存等应用。 另一个名为Xstream的DSMS被开发用于支持高速率信号处理。 JSQ系统专门用于查询和管理JSON格式的流数据。

在上面提到的所有系统中，Aurora [3]，STREAM [1]和Xstream [15]都定义了新的语义和查询语言，用于连续查询流和关系。 与其在JSQ系统中开发新的查询语言，现有的AmosQL查询语言已扩展为允许用户通过JSON流进行查询。

1. 结论和未来的工作

JSON流查询系统（JSQ）是Amos II的扩展。 它采用了Amos II数据库的DSMS功能，并为用户提供了一种查询和操作JSON格式的流数据的简单方法。

JSQ使用用Java编写的外部函数实现，并基于Twitter Streaming API进行测试。 并没有为流数据开发新的查询语言，而是扩展了AmosQL以允许用户查询JSON流。

目前，Java外部函数仅在将JSONObject处理成类型为Record的Amos II对象时进行了测试，将JSONArray处理为Vector类型的Amos II对象的接口原本是开发的，需要在将来进行扩展。

参考文献：

1. Arvind Arasu, Brian Babcock, John ieslewicz, Keith Ito, Mayur Datar, Rajeev Motwani, Shivnath Babu, Utkarsh Srivastava, and Jennifer Widom: STREAM: The Stanford Data Stream Management System, Technical Report. Stanford InfoLab, USA, March, 2004
2. A. Arasu, S. Babu, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and

Query Execution. Technical report, Stanford University, Oct.

2003. <http://dbpubs.stanford.edu/pub/2003-67>

1. D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun,J. Hwang, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, Y. Xing, R. Yan, S. Zdonik Aurora: A Data Stream Management System, Brandeis University, Brown University, M.I.T.
2. Daniel Elin and Tore Risch: Amos II Java Interface, UDBL, Uppsala University, Sweden, Aug. 2000
3. http://apiwiki.twitter.com/Streaming-API-Documentation#ParsingResponses
4. <http://apiwiki.twitter.com/Streaming-API-Documentation#statuses/filter>
5. <http://en.wikipedia.org/wiki/Twitter>
6. <http://hc.apache.org/httpclient-3.x/>
7. http://hc.apache.org/httpclient-3.x/tutorial.html
8. <http://infolab.stanford.edu/stream/>
9. <http://www.everydaytweet.com/2009/04/what-is-retweet-and-how-do-i-retweet.html>
10. <http://www.json.org/>
11. <http://www.json.org/example.html>
12. Kjell Orsborn: Management of Product Data Using an ExtensibleObject-Oriented Query Language, the Sixth International Conference on Data and Knowledge Systems for Manufacturing and Engineering, DKSME ’96, <http://user.it.uu.se/~udbl/publ/dksme96.pdf>
13. Lewis Girod, Yuan Mei, Ryan Newton, Stanislav Rost, Arvind Thiagarajan, Hari Balakrishnan, Samuel Madden: XStream: a Signal-Oriented Data Stream Management System, Computer Science and Artificial Intelligence Laboratory, MIT, 32 Vassar St, Cambridge, MA, 02139, USA
14. Lukasz Golab and M. Tamer Ozsu: Issues in Data Stream Management, University of Waterloo, Canada, 2003
15. Ramez Elmasri and Shamkant B. Navathe: Fundamentals of Database Systems, Third Edition, Addison-Wesley, 2003.
16. Risch, T, Josifovski, V: ”Distributed Mediation using a Light-Weight OODBMS”, in 1st ECOOP Workshop on Object-Oriented Databases, Lisbon Portugal, June 1999
17. Staffan Flodin, Martin Hansson, Vanja Josifovski, Timour Katchaounov, Tore Risch, Martin Sköld, and Erik Zeitler: Amos II Release 12 User's Manual, UDBL, Uppsala University, Sweden, March 29, 2005
18. T Risch, V Josifovski: Distributed Mediation by Object-Oriented Mediator Servers, To be published in “Concurrency – Practice and Experience”, J Wiley & Sons, 2001
19. Tore Risch, Vanja Josifovski, Timour Katchaounov: “AMOS II Concepts”,UDBL,UppsalaUniversity,Sweden, http://user.it.uu.se/~udbl/amos/doc/amos\_concepts.html, 2000

附录：

A：从Twitter Streaming API收到的公共状态的一个示例

{"text":"@deano\_BondiLG ur pretty handy in the kitchen tho arent you?? I

remember seeing u and Whippet cooking up a storm at Karrinyup that day...",

"contributors":"null",

"geo":"null",

"in\_reply\_to\_screen\_name":"deano\_BondiLG",

"truncated":"false",

"id":"11238489534",

"source":"web",

"favorited":"false",

"in\_reply\_to\_status\_id":"11237710943",

"in\_reply\_to\_user\_id":"118339349",

"created\_at":"Mon Mar 29 07:47:34 +0000 2010",

"place":"null",

"user":{"location":"Perth, WA",

"statuses\_count":"260",

"profile\_background\_tile":"true",

"lang":"en",

"profile\_link\_color":"FF0000",

"id":"28741593",

"following":"null",

"favourites\_count":"60",

"protected":"false",

"profile\_text\_color":"3D1957",

"description":"",

"verified":"false",

"contributors\_enabled":"false",

"profile\_sidebar\_border\_color":"65B0DA",

"name":"Fiona Mackenzie",

"profile\_background\_color":"642D8B",

"created\_at":"Sat Apr 04 04:42:22 +0000 2009",

"followers\_count":"30",

"geo\_enabled":"false",

"profile\_background\_image\_url":"http://s.twimg.com/a/1269387398/images/themes/theme10/bg.gif", "url":"null",

"utc\_offset":"28800",

"time\_zone":"Perth",

"notifications":"null",

"friends\_count":"166",

"profile\_sidebar\_fill\_color":"7AC3EE",

"screen\_name":"finona\_99",

"profile\_image\_url":"http://a3.twimg.com/profile\_images/542783733/Dean\_and\_I\_normal.jpg"}, "coordinates":"null"}

1. 实现JSON流封装的AmosQL代码

create function init(charstring url,charstring name,charstring pass)-> Integer as foreign "JAVA:JsonWrapper/initConn"; create function jsonwrapper(Integer id,Record r)->Bag of Object as foreign "JAVA:JsonWrapper/jsonWrapper"; create function jsonstream(Integer id,Record r)->stream of Object as streamof(jsonwrapper(id,r)); /\*recieve JSON stream from twitter\*/

create function trackstream(Integer i,charstring k,Integer stop)->Stream of Record as section(jsonstream(i,Record({"track",k})),0,stop);

1. Json流封装器的Java类实现

/\*\*

* JsonWrapper uses Httpclient to request resource from certain url.
* Then processes the requested resource which is in JSON format to
* Amos Vector data type
* @author Bo Yang

\*/

import callin.AmosException;

import callin.Connection;

import callin.Oid;

import callin.Scan;

import callin.Tuple;

import callout.CallContext;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.Collection;

import java.util.logging.Level;

import java.util.logging.Logger;

import org.apache.commons.httpclient.Credentials;

import org.apache.commons.httpclient.HttpClient;

import org.apache.commons.httpclient.HttpStatus;

import org.apache.commons.httpclient.HttpURL;

import org.apache.commons.httpclient.NameValuePair;

import org.apache.commons.httpclient.URIException;

import org.apache.commons.httpclient.UsernamePasswordCredentials; import org.apache.commons.httpclient.auth.AuthScope; import org.apache.commons.httpclient.methods.\*;

import org.json.JSONArray;

import org.json.JSONException;

import org.json.JSONObject;

import org.json.JSONTokener;

public class JsonWrapper {

private AuthScope authScope;

private HttpURL url = null;

private String baseUrl = null;

private String username = null;

private String password = null;

private static ArrayList<HttpClientUrl> arraylist = new ArrayList<HttpClientUrl>(); private static PostMethod postMethod = null;

/\*\*

* Extracts the host and post from the baseurl and constructs an
* appropriate AuthScope for them for use with HttpClient

\*/

private AuthScope createAuthScope(String baseUrl) throws AmosException { AuthScope authscope = null;

try {

url = new HttpURL(baseUrl);

authscope = new AuthScope(url.getHost(), url.getPort(), "realm"); } catch (URIException ex) {

Logger.getLogger(JsonWrapper.class.getName()).log(Level.SEVERE, null, ex); throw new AmosException(ex.getMessage());

}

return authscope;

}

/\*\*

* Process JSONObject to Amos Record \*/

public Tuple processJson(CallContext cxt, JSONObject jsonobject, Tuple t) { JSONArray keys = jsonobject.names();

for (int i = 0; i < keys.length(); i++) { try {

//put the key of JSONObject into Record

t.setElem(i \* 2, keys.getString(i));

//put the corresponding value of JSONObject into Record

if (jsonobject.get(

keys.getString(i)).getClass().getName().equals("org.json.JSONObject"))

{

Scan s;

Tuple temp1 = new Tuple(1);

JSONObject temp = jsonobject.getJSONObject(keys.getString(i)); temp1.setElem(0, processJson(

cxt, temp, new Tuple(temp.names().length() \* 2))); s = cxt.connection().callFunction(

"vector.make\_record->Record", temp1); t.setElem(i \* 2 + 1, s.getRow().getOidElem(0));

} else if (jsonobject.get(

keys.getString(i)).getClass().getName().equals("org.json.JSONArray"))

{

JSONArray temp = jsonobject.getJSONArray(keys.getString(i)); t.setElem(i \* 2 + 1, processArray(cxt, temp, new Tuple(temp.length())));

} else {

//System.out.println(jsonobject.get(keys.getString(i)).getClass().getName());

t.setElem(i \* 2 + 1, jsonobject.getString(keys.getString(i)));

}

} catch (AmosException ex) { Logger.getLogger(

JsonWrapper.class.getName()).log(Level.SEVERE, null, ex); } catch (JSONException ex) {

Logger.getLogger(

JsonWrapper.class.getName()).log(Level.SEVERE, null, ex);

}

}

return t;

}

/\*\*

* Process JSONArray to Amos Vector \*/

public Tuple processArray(CallContext cxt, JSONArray jsonarray, Tuple t) { for (int i = 0; i < jsonarray.length(); i++) {

try {

if (jsonarray.get(i).getClass().getName().equals("org.json.JSONObject")) { JSONObject temp = jsonarray.getJSONObject(i);

t.setElem(i, processJson(

cxt, temp, new Tuple(temp.names().length() \* 2)));

} else

if (jsonarray.get(i).getClass().getName().equals("org.json.JSONArray"))

{

JSONArray temp = jsonarray.getJSONArray(i);

t.setElem(i, processArray(cxt, temp, new Tuple(temp.length()))); } else {

t.setElem(i, jsonarray.getString(i));

}

} catch (AmosException ex) { Logger.getLogger(

JsonWrapper.class.getName()).log(Level.SEVERE, null, ex); } catch (JSONException ex) {

Logger.getLogger(

JsonWrapper.class.getName()).log(Level.SEVERE, null, ex);

}

}

return t;

}

/\*\*

* Initialize the httpclient by setting credentials and creating Authorization Scope.
* Then add the httpclient and url into an arraylist

public void initConn(final CallContext cxt, final Tuple tpl) throws AmosException {

Boolean find = false;

this.baseUrl = tpl.getStringElem(0);

this.username = tpl.getStringElem(1);

this.password = tpl.getStringElem(2);

int status;

authScope = createAuthScope(baseUrl);

HttpClient httpClient = new HttpClient();

Credentials creds = new UsernamePasswordCredentials(username, password); httpClient.getHttpConnectionManager().getParams().setSoTimeout(60000); httpClient.getState().setCredentials(authScope, creds); httpClient.getParams().setAuthenticationPreemptive(true);

GetMethod get = new GetMethod(baseUrl);

get.setDoAuthentication(true);

try {

status = httpClient.executeMethod(get);

//System.out.println(status);

if ((status != 200) && (status != 406)) {

throw new AmosException(

"Got status " + get.getStatusText());

}

} catch (IOException ex) {

Logger.getLogger(JsonWrapper.class.getName()).log(Level.SEVERE, null, ex);

throw new AmosException(ex.getMessage());

} finally {

get.releaseConnection();

}

if ((status == 200) || (status == 406)) {

for (int i = 0; i < arraylist.size(); i++) {

if ((arraylist.get(i) == null) && (!find)) {

arraylist.remove(i);

arraylist.add(i, new HttpClientUrl(httpClient, baseUrl));

find = true;

tpl.setElem(3, i);

cxt.emit(tpl);

}

}

if (!find) {

arraylist.add(new HttpClientUrl(httpClient, baseUrl));

tpl.setElem(3, arraylist.size() - 1);

cxt.emit(tpl);

}

}

}

/\*\*

* Send a name value pair to url and process the recieved data \*/

public void jsonWrapper(final CallContext cxt, final Tuple tpl) throws AmosException { Scan s;

Tuple temptpl = new Tuple(1); Tuple temptpl2 = new Tuple(1);

Collection<NameValuePair> namevaluepair = new ArrayList<NameValuePair>(); int snumber = tpl.getIntElem(0);

if (arraylist.isEmpty()) {

throw new AmosException("No connection initialized!");

} else if ((snumber >= arraylist.size()) || (arraylist.get(snumber) == null)) { throw new AmosException("Connection is not defined or has been released");

} else {

HttpClient httpclient = arraylist.get(tpl.getIntElem(0)).httpClient;

Oid oid = tpl.getOidElem(1);

temptpl2.setElem(0, oid);

1. = cxt.connection().callFunction("Record.all\_value->Object", temptpl2); while (!s.eos()) {

Tuple row; String name; String value; row = s.getRow();

name = row.getStringElem(0); s.nextRow();

row = s.getRow();

value = row.getStringElem(0); namevaluepair.add(new NameValuePair(name, value)); s.nextRow();

}

try {

postMethod = new PostMethod(arraylist.get(tpl.getIntElem(0)).baseurl); postMethod.setRequestBody(namevaluepair.toArray(

new NameValuePair[namevaluepair.size()])); httpclient.executeMethod(postMethod);

if (postMethod.getStatusCode() != HttpStatus.SC\_OK) {

throw new AmosException(

"Got status " + postMethod.getStatusText());

}

InputStream is = postMethod.getResponseBodyAsStream();

if (is != null) {

JSONTokener jsonTokener = new JSONTokener(

new InputStreamReader(is, "UTF-8"));

String jsontype = jsonTokener.nextValue().getClass().getName(); if (jsontype.equals("org.json.JSONObject")) {

while (true) {

JSONObject jsonObject = new JSONObject(jsonTokener);

temptpl.setElem(0, processJson(cxt, jsonObject, new Tuple(

jsonObject.names().length() \* 2)));

s = cxt.connection().callFunction(

"vector.make\_record->Record", temptpl);

tpl.setElem(2, s.getRow().getOidElem(0));

cxt.emit(tpl);

}

} else if (jsontype.equals("org.json.JSONArray")) { while (true) {

JSONArray jsonarray = new JSONArray(jsonTokener);

tpl.setElem(2, processArray(

cxt, jsonarray, new Tuple(jsonarray.length())));

cxt.emit(tpl);

}

} else {

throw new AmosException("Data is not in JSON format ");

}

}

} catch (JSONException ex) { Logger.getLogger(

JsonWrapper.class.getName()).log(Level.SEVERE, null, ex); throw new AmosException(ex.getMessage());

} catch (IOException ex) {

Logger.getLogger(

JsonWrapper.class.getName()).log(Level.SEVERE, null, ex);

throw new AmosException(ex.getMessage());

} finally {

* + Abort the method, otherwise releaseConnection() will
  + attempt to finish reading the never-ending response.
  + These methods do not throw exceptions.

postMethod.abort();

postMethod.releaseConnection();

}

}

}

/\*\*

\*Abort the method and finish reading the never-ending response \*/

/\*\*

* Get the number of available connections \*/

public void available(final CallContext cxt, final Tuple tpl) throws AmosException { if (arraylist.isEmpty()) {

throw new AmosException("No connection available!");

} else {

Boolean find = false;

for (int i = 0; i < arraylist.size(); i++) {

if (arraylist.get(i) != null) {

find = true;

tpl.setElem(0, i);

cxt.emit(tpl);

}

}

if (!find) {

throw new AmosException("No connection available!");

}

}

}

public static void main(String argv[]) throws AmosException {

Connection.initializeAmos(argv);

Connection theConnection = new Connection(""); theConnection.amosTopLoop("JavaAmos II"); // Enters the AmosQL top-loop

}

}

class HttpClientUrl {

public HttpClient httpClient = null;

String baseurl;

public HttpClientUrl(HttpClient httpclient, String Url) {

this.httpClient = httpclient;

this.baseurl = Url;

}

}