

文章编号:1001-9081(2009)07-1771-04

分布式数据流查询方案及优化

徐署华, 胡 君

(湖南科技职业学院 软件学院, 长沙 410118)

(gtsgxsh@tom.com)

摘 要:流式数据库系统是一种新型数据库系统, 方便于执行连续数据流查询。许多基于流的应用都是分布式的, 由于输入流速率及其他系统参数如可用的计算资源是易变的, 所以一个流查询方案必须能适应这些变化。提出一种分布式流查询方案并进行了优化, 使用元组响应时间及系统吞吐量来评价方案的性能。同时, 通过实验和其他方案进行比较, 证明了方案是最佳的。

关键词:分布式数据流; 查询方案; 查询优化; 路由策略

中图分类号: TP319 **文献标志码:** A

Query scheme and its optimization for distributed data stream

XU Shu-hua, HU Jun

(Software Institute, Hunan Vocational College of Science and Technology, Changsha Hunan 410118, China)

Abstract: Stream database system is a new type of database system designed to facilitate the execution of queries to continuous streams of data. Many stream-based applications are inherently distributed. Since input stream rates and other system parameters such as the amount of available computing resources can fluctuate significantly, a stream query plan must be able to adapt to these changes. This paper proposed a distributed stream query plan and optimized, and defined performance metrics for response time and system throughput. The experimental result proves that this plan is optimal compared with other plans.

Key words: distributed data stream; query plan; query optimization; routing policy

0 引言

流式数据库系统是一种新型数据库系统, 许多基于流的应用如网络监控应用系统、在线信息跟踪等都是分布式的, 所以许多标准数据库的基本属性不再适用于数据流管理系统^[1]。典型的流查询很花费时间, 它要监听几个连续的流, 并产生连续的结果流。传统数据库优化指标——运行时间, 不再适应于流数据管理系统, 必须使用其他指标来评价。因输入流速率及可用计算资源随时间而变化, 导致此时工作状态良好的执行方案可能马上变得效率低下, 因此一个流执行方案必须能适应这些改变。当前有许多这方面的研究, 文献[2]介绍在操作之间路由元组的概念是作为查询优化来考虑的, 本文扩展这个概念到分布式环境; 文献[2-3]中描述的路由策略是几个和本文相反的路由策略(后面对这些策略进行了比较); Brandeis、Brown、MIT的合作项目 Borealis^[4-5]在数据流处理的近似查询算法及负载均衡等方面做了大量的工作。这里提出使用队列网络来建立一种分布式流查询方案, 并对方案进行了优化。提出使用元组响应时间及系统吞吐量来评价方案的性能。同时, 通过实验和其他方案进行比较, 并证明了本方案是最佳的。

1 流查询执行机制

一个漩涡^[2]是一个流查询执行机制, 能在查询方案中不断重新排列操作顺序。漩涡的每个输入元组携带自己的执行历史, 执行历史由两个位图起作用。一个位图记录元组已访

问过的操作, 另一位图记录元组下一个能访问的操作。有两种类型的漩涡, 集中式漩涡和分布式漩涡。

集中式漩涡根据元组的执行历史路由由元组到下一操作, 并由漩涡来维护这些统计数据。如果元组满足某操作的操作条件, 操作将对两个位图做适当修改, 并返回元组给漩涡。漩涡继续这种工作直到元组访问了所有操作。图1显示一个包含三个操作的漩涡, 漩涡的主要优势为, 执行方案能很好地适应对每个单独元组决定操作顺序的路由结果^[1-2]。

然而, 集中式漩涡不能直接用于分布式数据流管理系统, 必须使用分布式漩涡。对于分布式漩涡, 在操作处理元组后, 并不返回元组给漩涡, 而是基于元组执行历史及保存在操作上的统计数据建立一个路由方案。图2显示一个带三个操作的分布式方案。带虚线的箭头表示操作间可能的路线。四条带实线的箭头表示一个实际执行顺序, 即一个元组的实际路线。路由策略决定每个元组在所有操作上的执行顺序。

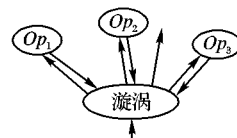


图1 集中式漩涡

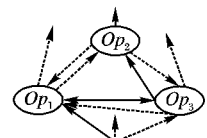


图2 分布式查询方案

2 分布式流查询方案及优化

2.1 查询方案模型概述

把分布式流查询方案设计为一系列由网络连接的操作 $Op_i, i=1, \dots, n$ 。操作的输入元组被加入到先来先服务的队

收稿日期: 2009-02-14; 修回日期: 2009-03-27。 基金项目: 教育部高职高专计算机类专业教指委科研资助项目(jzw59010817)。

作者简介: 徐署华(1970-), 男, 湖南长沙人, 副教授, 主要研究方向: 数据库技术、分布式应用; 胡君(1980-), 男, 湖北黄石人, 硕士, 主要研究方向: 计算机网络、软件工程。

列中。假设 Op_i 资源(CPU,内存,网络带宽)对每个操作是可用的,并假设 Op_i 的每个输入元组平均开销为 $Op_i \cdot r$,可知 Op_i 在每时间单元内最多能处理 $Op_i \cdot R / Op_i \cdot r$ 个输入元组,且对每个单独元组的平均服务时间为 $Ts = Op_i \cdot r / Op_i \cdot R$ 。事实上,许多队列网络的正常资源在 $Op_i \cdot R$ 到 1 之间。元组在 Op_i 中的居留时间 Tr 是队列延时和处理元组时间之和。

方案的输入元组 是由特殊的数据源操作产生的,元组 t 在生成时被加上系统时间 ts 时间戳^[2],每个元组携带自己的执行历史 h 。执行历史记录了元组已访问的操作,元组需要访问的下一操作通常视元组已访问的操作而定。带时间戳 ts 和操作历史 h 的元组标识为 t_h^s 。数据源操作不会从其他操作接收输入元组,假设元组局留在数据源操作中的时间为 0。

操作的选择率 对操作 Op_i 的每个输入元组 t_h^s , Op_i 将平均产生 ρ 个输出元组, ρ 叫着 Op_i 的选择率。比如选择操作的 ρ 总是小于 1;连接操作的 ρ 可能大于 1。

元组的执行路径 对每个输出元组, Op_i 根据路由加权函数 $Op_i \cdot W(h) = (w_{i1}(h), w_{i2}(h), \dots, w_{in}(h))$ 选择下一操作并向它输送该输出元组, n 是分布式方案中操作的数目,每个输出元组被加上和相应输入元组相同的时间戳 ts ,且在被送到下一操作前它的执行历史被相应设置。持续这种操作,直到元组到达特殊的称为数据池(Sink)操作。池操作得到分布式方案的最后结果。元组被送到池操作相当于元组已离开系统,可假设元组在池操作中无资源开销且居留时间为 0。当池操作收到一输入元组,操作就能计算出元组的响应时间(此刻的系统时间减去元组的 ts),元组依次访问操作的线路叫元组的执行路径。

以上模型中的操作只有一个输入队列,下面介绍连接操作的执行。这里逻辑上有两个输入流^[3],假设所有连接操作使用滑动窗口表示,以便一个元组将不会被连接到任意多的数据。且假设所有连接操作使用对称 hash 连接算法^[6]来执行。

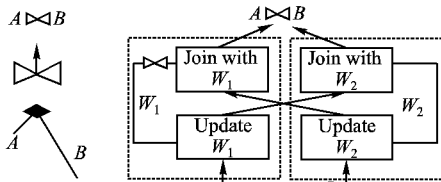


图3 $A \triangleleft B$ 的分布式对称算法

图3显示了 $A \triangleleft B$ 的分布式对称算法,流 A 中的输入元组先由为流 A 维护滑动窗口的操作来处理,这个元组然后被送到一个和流 B (可能在不同的物理节点上)窗口连接元组的操作上,流 B 中的元组同样以对称的方式被处理。图中虚线方框表示单一节点上的连接算法的组成部分。

2.2 方案性能度量指标

提出如下两个指标:

ART 平均响应时间。元组到达池操作的平均响应时间,注意,在查询处理过程中被过滤出去的元组不进行这种评估。

MDR 最大数据速率。系统超载前数据源操作的最大数据速率。当任何操作的输入队列的长度超过了事先定义的最大长度时,认为系统超载了。当有多个数据源操作时,对每个数据源操作的数据速率进行一些功能优化是必要的。这里将对数据源操作的所有数据速率之和进行优化。

2.3 方案的元组路由策略

2.3.1 ART 和 MDR 值计算

只考虑资源分配固定的情况。假设方案中 Op_i 的计算资

源 R 是固定的,并假设 Op_i 的路由加权函数 W 为: $Op_i \cdot W(h) = (w_{i1}(h), w_{i2}(h), \dots, w_{in}(h))$ 。如果带有操作历史 h 的元组不能被路由到操作 Op_k ,那么 $w_{ik}(h) = 0$ 。相反的,能被路由到 Op_k ,那么 $w_{ik}(h) = cik$, cik 表示从操作 Op_i 路由到 Op_k 的加权概率。

计算速率算法,是计算 Op_i 总的输入速率 $Op_i \cdot \lambda$ (关于 cik 的一个函数,数据源操作产生数据的速率),如果把操作 Op_i 当作一个具有输入速率 $Op_i \cdot \lambda^2$ 的 M/M/1 服务器,可以计算一个元组在操作 Op_i 中的平均居留时间 $Op_i \cdot Tr$,以及平均队列长度 $Op_i \cdot qlen$ 。将有:

$$Op_i \cdot Tr = Op_i \cdot Ts / (1 - Op_i \cdot \lambda \times Op_i \cdot Ts)$$

$$Op_i \cdot qlen = (Op_i \cdot \lambda \times Op_i \cdot Ts) / (1 - Op_i \cdot \lambda \times Op_i \cdot Ts)$$

$Op_i \cdot Ts = Op_i \cdot r / Op_i \cdot R$ 是操作 Op_i 对元组的平均服务时间。

该算法也计算在池操作中每个不同执行路径的到达速度,表示为 $Sink \cdot \lambda h$ 。响应时间 Th 表示带有执行历史 h 的元组沿着它的执行路径在所有操作中居留时间的总和, $Th = \sum_{op \in h} Op_i \cdot Tr$,可以计算系统的平均响应时间为:

$$ART = \sum_h (Sink \cdot \lambda h * Th) / \sum_h Sink \cdot \lambda$$

计算速率算法如下:

```

1  for 每个操作 op:
2      if op is not 数据源操作
3           $Op \cdot \lambda = 0$ 
4      end if
5      if op is 数据源操作
6          new_branch.op = op
7          new_branch.rate =  $Op \cdot \lambda$ 
8          new_branch.history = {op}
9          active_branches.add(new_branch)
10     end if
11 end for
12 while active_branches is not null
13     curr = active_branches.first()
14     active_branches.remove(curr)
15      $(w_1, w_2, \dots, w_n) = curr.op.W(curr.history)$ 
16     total_w =  $\sum_{i=1}^n w_i$ 
17     for i in 1, 2, ..., n:
18         if  $w_i \neq 0$ :
19              $p = w_i / total\_w$ 
20              $r = curr.rate * curr.op.p * p$ 
21             if  $op_i$  is not 池操作 v
22                  $Op_i \cdot \lambda += r$ 
23                 new_branch.op =  $Op_i$ 
24                 new_branch.flow = r
25                 new_branch.history = curr.history.append( $Op_i$ )
26                 active_branches.add(new_branch)
27             else
28                  $Sink \cdot \lambda, curr.history += r$ 
29             end if
30         end if
31     end for
32 end while

```

说明 算法中的路径看成是一个数据结构,有字段 {op, rate, history}, active_branches 是一个路径列表。对 Op_i 总的输入速率表示为 $Op_i \cdot \lambda$,带有操作历史 h 的元组到达池 Sink 的到达速率表示为 $Sink \cdot \lambda h$ 。算法中 1~11 行为初始化数据源操作的数据速率。12~33 行递归计算一个执行方案中各路径都到达数据池操作的速率。13~14 行在路径列表中取第一条路径并从列表中删除,15~16 计算元组路径的加权概

率,17~31行操作总的输入速率及元组到达数据池操作的到达速率。

2.3.2 方案优化

对 ART 的优化:给定网络的特性(包括每个操作的选择率)和数据源操作中数据产生的速率,为每个操作 Op_i 选择一个加权路由函数 W ,使 ART 达到最小化。对 MDR 的优化:给定网络特性,为每个操作 Op_i 选择一个加权路由函数 W ,使 MDR 达到最大化。

优化 ART: $\arg\min_{\{cik\}} (\sum_h (Sink_i \cdot \lambda h * Th) / \sum_h Sink_i \cdot \lambda h)$

具有约束 $Op_i \cdot \lambda < Op_i \cdot R / Op_i \cdot r$, 注意,这里的数据速度 λ 在数据源节点已给定。

优化 MDR: $\arg\max_{\{cik, Op_i \mid \text{where } Op_i \text{ is data source}\}} (\sum_{Op_i \text{ is data source}} Op_i \cdot \lambda)$

具有约束 $Op_i \cdot \lambda < Op_i \cdot R / Op_i \cdot r$ 及 $Op_i \cdot qlen < Op_i \cdot MaxQ$, 是对 Op_i 给定的最大队列长度(在操作 Op_i 超载之前的最大长度)。

2.3.3 实验

只对选择操作进行了实验,实验是由三个选择操作组成的方案, Op_1, Op_2 和 Op_3, Op_0 是数据源操作。这三个选择操作能以任何顺序进行评价,有关的参数(如每个操作可用的计算资源)已在表1中给定。

表1 相应参数

操作	R	r	$Ts = r/R$	ρ
Op_1	10	0.1	0.01	0.5
Op_2	20	0.1	0.005	0.2
Op_3	20	0.5	0.025	0.4

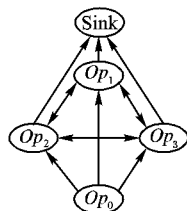


图4 实例元组可能路径示意图

表2 实例中可能的执行路径

操作	路径	操作	路径
P1	$Op_0, Op_1, Op_2, Op_3, Sink$	P4	$Op_0, Op_2, Op_3, Op_1, Sink$
P2	$Op_0, Op_1, Op_3, Op_2, Sink$	P5	$Op_0, Op_3, Op_1, Op_2, Sink$
P3	$Op_0, Op_2, Op_1, Op_3, Sink$	P6	$Op_0, Op_3, Op_2, Op_1, Sink$

如图4所示,这里元组从 Op_0 到 Sink 有6条可能的执行路径(P1~P6,如表2所示),在 Op_0 处数据产生的速度为 $Op_0 \cdot \lambda$,大体上没有什么损耗,所以可以假设 $\sum_j cij = 1$,可以计算总的到达速率及在操作 Op_1 的居留时间如下:

$$\begin{aligned}
 Op_1 \cdot \lambda &= Op_0 \cdot \lambda \times c01 + && \text{经 P1, P2} \\
 &+ Op_0 \cdot \lambda \times c02 \times Op_2 \cdot \rho \times c21 + && \text{经 P3} \\
 &+ Op_0 \cdot \lambda \times c02 \times Op_2 \cdot \rho \times c23 \times Op_3 \cdot \rho + && \text{经 P4} \\
 &+ Op_0 \cdot \lambda \times c03 \times Op_3 \cdot \rho \times c31 + && \text{经 P5} \\
 &+ Op_0 \cdot \lambda \times c03 \times Op_3 \cdot \rho \times c32 \times Op_2 \cdot \rho && \text{经 P6} \\
 Op_1 \cdot Tr &= Op_1 \cdot Ts / (1 - Op_1 \cdot \lambda \times Op_1 \cdot Ts)
 \end{aligned}$$

类似地,以上计算过程也适用于 Op_2 和 Op_3 ,既然一元组从 Op_0 到 Sink 总是要访问 Op_1, Op_2, Op_3 ,严格地说,不管经哪条执行路径,在 Sink 的平均响应时间都为:

$$ART = Op_1 \cdot Tr + Op_2 \cdot Tr + Op_3 \cdot Tr$$

现在可以解答 ART 和 MDR 的数值,如果给定 $Op_0 \cdot \lambda =$

200,解决方案中的 ART 值就可算出,如表3所示,假设 $Op_i \cdot MaxQ = 10000; i = 1, 2, 3$,MDR 值如表4所示。

表3 当 $Op_0 \cdot \lambda = 200$ 时 ART 值

ATR	C01	C02	C03	C12	C13	C21	C23	C31	C32
0.14	0.21	0.78	0.01	0.98	0.02	0.99	0.01	0.15	0.85

表4 当 $Op_i \cdot MaxQ = 10000$ 时 MDR 值

MDR	C01	C02	C03	C12	C13	C21	C23	C31	C32
243	0.31	0.68	0.01	0.77	0.23	0.50	0.50	0.16	0.84

从数据结果可以看到更多的元组将被路由到具有较低服务时间及选择率较大的操作上,在解决方法中观察几个执行方案都有这一现象,这是一个在优化分布式路由策略和优化集中漩涡路由策略之间重要的差异。

3 与其他方案的比较

3.1 其他方案

其他数据流查询方案有许多,这里例举几个和本方案进行比较。

路由策略 Q(Q length)^[2] 元组被路由到负荷较轻的操作上,当操作输出一元组,该元组被路由到可用操作中具有最短队列长度 Q 的操作上。Q 策略不是路由等量元组到各操作,有较小 Ts 的操作将以较快速率处理元组,所以更多元组将被路由到此操作上。

路由策略 T(Ticket)^[2-3] 元组被路由到有最大选择率的操作,因更多元组将在有最大选择率的操作上被排除,所以可以节约被排除的元组在有较小选择率操作上不必要的处理。但结果显示,最大选择率的操作将成为系统的瓶颈。

路由策略 SC(Selectivity-Cost) 使用表达式 $Benefin_SC_i = (1 - Op_i \cdot S) / Op_i \cdot C$ 来决策路由,如果 $Op_i \cdot S$ 大于 1, $Benefin_SC_i$ 设为 0。每个操作的输出元组将被路由到下一个带有最大 benefit 值的可用操作上。

路由策略 WSC(Weighted Selectivity-Cost) 在 2.3 方案中, Op_i 根据 cij 路由元组到下一操作 Op_j , WSC 模仿 2.3,根据概率 W_SC_j 路由元组到 Op_j , $W_SC_j = (Benefit_SC_j)^2$ 。

路由策略 SCQ(Selectivity-Cost-Qlength) SC 考虑操作上的元组平均处理开销,开销不含排队时间(元组在队列中等待时间),元组被路由到有最大 benefit 值的操作,而不管这个操作负荷有多大。SCQ 中,定义: $Benefin_SCQ_i = (1 - Op_i \cdot S) / (1 + Op_i \cdot Q) \times Op_i \cdot C$,把队列长度纳入考虑,元组的开销为处理元组时间和排队等待时间之和。

路由策略 WSCQ(Weighted Swlctivity-Cost-Qlength) 类似 WSC, WSCQ 根据某一概率路由元组到下一可用操作,路由元组到 Op_i 的加权概率为: $W_SCQ_i = (Benefit_SCQ_i)^2$ 。

表5显示了用来进行路由选择的统计信息。输入队列的长度表示每个操作的负荷。Ticket 和选择率是研究操作对输入流的选择率的不同度量方法。

3.2 模拟实验比较

模拟系统由通过网络连接一定数量的物理节点组成。分布式方案中每个操作在一个物理节点上执行,一个物理节点可同时执行多个操作。元组就是操作间交换的信息,如果一个物理节点有多个操作,元组将多路输入到各操作队列中。元组被组成传输页在网络上迁移。元组通常可缓存一个时间单

元,当页达到半满时且时间单元过了就会被迁移。数据源操作以某一输出速率产生元组。每个操作收集统计信息(表 5),且每过 5 个时间单元或每处理 40 个输入元组后传送这些统计信息给其他操作。包含统计数据的信息比纯数据有优越性,它们一旦到达就被处理。模型中的重要参数为:各物理节点计算资源值为 500 ~ 1 000;每个操作处理一个元组所需计算资源为 1 ~ 20;两个物理节点间的网络连接带宽,随机地选

择每时间单元 1 MB ~ 100 MB;网络传输页大小为 4 KB;元组平均大小为 100 B,所以,每页有 40 个元组。

先来看 2.3.3 的实验(图 4),表 6 显示了 2.3.3 实验中使用不同策略的 ART 值(表中“本方案”指 2.3 中的方案)。这里规定操作超载条件为它的队列长度超过 10 000,模型能处理的最大数据速率为 $\lambda = 243$,表中下划线部分表示 ART 有大的衰退。

表 5 路由策略用到的统计信息

统计信息	符号	含意和度量方式
Qlength	Q	平均输入队列的长度,可以通过由个操作的资源监测器来获得
Ticket	T	当一个输入元组到达时增 1,一个元组被加入到输出流中时减 1, T 成为负数或服务器超载时, T 设置为 0
Selectivity	S	特定时间段内选择率 $S_{interval}$ 可以通过一监测器来度量,操作的选择率在每个时间段内可通过公式 $S_{new} = 0.8 \times S_{old} + 0.2 \times S_{interval}$ 进行改变
Cost	C	处理元组的平均服务时间,可以通过监测每个操作得到

表 6 实验中不同方案的 ART 值

方案	数据速率				
	200	210	220	230	240
本方案	0.14	0.18	0.25	0.43	1.57
T	0.35	<u>22</u>	<u>26</u>	<u>35</u>	<u>45</u>
Q	1.5	2.1	2.6	3.0	3.8
SC	0.11	2.3	<u>23</u>	<u>34</u>	<u>44</u>
WSC	0.12	0.49	9.0	<u>20</u>	<u>29</u>
SCQ	0.54	0.95	1.2	1.4	2.2
WSCQ	0.22	0.44	0.61	0.8	1.6

从表 6 中可知,本方案的平均响应时间结果是最理想的。策略 Q、SCQ 和 WSCQ 都可在不引起 ART 重大衰退情况下达到 $\lambda = 243$ 。结果表明,把队列长度作为路由参数对元组速率起关键作用。策略 T 显示最坏的 MDR 结果,一旦具有最大选择率的操作(Op_2)变得饱和,将设置它的 ticket 为 0,但这将马上引起下一个具有最大选择率的操作阻塞。Q 策略均衡操作间的负载,但太多的元组输送到低选择率的操作,将引起太多的不必要的操作。把 Q 策略的 ART 和其他策略 SCQ、WSCQ 的 ART 进行比较,可以发现,为了降低 ART,考虑选择率和开销是很重要的。SCQ 和 WSCQ 在低访问次数和高数据速率方面有优越性。

在一个真实的数据流管理系统中,通常会同时有许多费时的查询在运行。在模拟模型中模拟 20 个查询(每个查询有 10 ~ 40 个操作,平均每个方案有 25 个操作)同时实施,假设网络有 40 个节点。表 7 显示了 ART 和 MDR 的结果。再次发现本方案有最小的 ART 值和最大的 MDR 值。策略 T、SC 和 WSC 的 ART 是很差的。策略 Q、SCQ 和 WSCQ 的 MDR 没什么差别,它们的 MDR 要好于不考虑队列长度的策略。可以得到结论,在多方案系统中,队列长度是路由决策的重要因素。

表 7 ART 和 MDR 的几何平均值

方案	ART	MDR
本方案	0.17	98
T	331	22
Q	3.3	94
SC	21	61
WSC	26	63
SCQ	1.52	96
WSCQ	1.3	97

4 结语

研究了路由策略对执行一个分布式数据流管理系统的影响。对分布式查询方案使用一个队列网络来建立一个分析模型并进行了优化,并确定两种性能度量方式,即平均响应时间和最大数据速度。将本优化方案和其他路由策略进行了比较,并使用分离的模拟实验比较了它们相对的性能。发现在每个操作分配固定的计算资源的分布式方案中,考虑操作的选择率、执行开销、操作负荷的路由策略要好于只考虑一到二个因素的简单策略。集中式漩涡路由策略^[2]不适用于分布式环境,路由元组使用加权概率是一种使 ART 达到最小的有效技术。通过实验得知,本优化方案具有较大优势。

参考文献:

- [1] CARNEY D, CETINTEME L U, CHERNIACK M, *et al.* Monitoring streams: A new class of data management application[C]// Proceedings of the 28th International Conference on Very Large Data Bases. Hong Kong: VLDB Endowment, 2002: 215 - 226.
- [2] AVNUR R, HELLERSTEIN J. Eddies: Continuously adaptive query processing[C]// Proceedings of the ACM SIGMOD International Conference on Management of Data. New York: ACM, 2000: 261 - 272.
- [3] MADDEN S, SHAH M, HELLERSTEIN J, *et al.* Continuously adaptive queries over streams[C]// Proceedings of the ACM SIGMOD International Conference on Management of Data. New York: ACM, 2002: 49 - 60.
- [4] TATBUL N, ZDONIK S. Dealing with overload in distributed stream processing systems[C]// Proceedings of IEEE International Workshop on Networking Meets Databases: NetDB' 06. Washington, DC: IEEE Computer Society, 2006: 24.
- [5] ABADI D J, AHMAD Y, BALAZINSKA M, *et al.* The design of the Borealis stream processing engine[EB/OL]. [2008 - 12 - 11]. <http://cs-www.cs.yale.edu/homes/dna/papers/cidr05.pdf>
- [6] ARASU A, BABCOCK B, BABU S, *et al.* Characterizing memory requirements for queries over continuous data streams[C]// Proceedings of the ACM Symposium on Principles of Database Systems (PODS). New York: ACM, 2002: 221 - 232.
- [7] BABCOCK B, BABU S, DATAR M, *et al.* Models and issues in data stream systems[C]// Proceedings of the ACM Symposium on Principles of Database Systems. New York: ACM, 2002: 1 - 16.