S4:分布式流计算平台

概要

S4 是一个通用的，分布式的，可扩展的，分区容错的，可插拔的平台。开发者可以很容易的在其上开发面向无界不间断流数据处理的应用。编键的数据事件被分类路由到处理单元（Processing Elements，PEs），处理单元消费这些事件，做如下事情之一或全部：（1）发出一个或多个可能被其他 PE 处理的事件。（2）发布结果。这种架构类似提供了封装和地址透明语义的 Actor 模式，因此允许应用在大规模并发的同时暴露简单的编程接口给应用开发者。在这篇论文里，我们将勾画 S4 的架构细节，描述各种各样的应用，包括实际中的部署。我们的设计主要由大规模应用在生产环境中的数据采集和机器学习所驱动。我们展示了 S4 设计令人惊奇的灵活性，使其运行在构筑于普通硬件之上的大规模集群中。

关键词：编程模型(programming model); 复杂事件处理(complex event processing);并发编程(concurrent programming); 数据处理(data processing); 分布式编程（distributed programming); map-reduce; 中间件(middleware); 并行编程(parallel programming); 实时搜索(real-time search); 软件设计(software design); 流计算(stream computing)

一、介绍

S4（简单可扩展流系统的首字母简称:Simple Scalable Streaming System）是一个受 Map-Reduce 模式启发的分布式流处理引擎。我们设计这个引擎是为了解决使用数据采集和机器学习算法的搜索应用环境中的现实问题。当前的商用搜索引擎，像 Google、Bing 和 Yahoo!，典型的做法是在用户查询响应中提供结构化的 Web 结果的同时插入基于流量的点击付费模式的文本广告。为了在页面上的最佳位置展现最相关的广告，科学家开发了算法来动态估算在给定上下文中一个广告被点击的可能性。上下文可能包括用户偏好，地理位置，之前的查询，之前的点击等等。一个主搜索引擎可能每秒钟处理成千上万次查询，每个页面都可能会包含多个广告。为了处理用户反馈，我们开发了 S4，一个低延迟，可扩展的流处理引擎。

为了便于在线实验算法，我们设想一种既适合研究又适合生产环境的架构。研究的主要需求是要具有将算法快速发布到特定领域的高度灵活性。这使得以最小的开销和支持在实际流量中测试在线算法成为可能。生产环境的主要需求是可扩展性（以最小的代价通过增加更多的机器来提高吞吐量的能力）和高可用性（在

存在系统故障的情况下不需要人工介入仍然能持续提供服务的能力）。我们考虑过扩展开源的 Hadoop 平台来支持无界流计算但是我们很快认识到 Hadoop 平台是为批处理做了高度优化的。MapReduce 系统典型的是通过调度批量任务操作静态数据。而在流计算中的典型范式是有一个在我们无法控制的数据比率之上的事件流流入系统中。处理系统必须赶得上事件流量，或者通过消减事件优雅的降级，这通常被称为负载分流（load shedding）。流处理的这一模式决定了要和批处理使用非常不同的架构。试图建造一个既适合流计算又适合批处理的通用平台结果可能会是一个高度复杂的系统，并且最终可能都不是两者最理想的实现。一个作为 Hadoop 扩展构建的 MapReduce 在线架构的例子可以在[3]中找到。

MapReduce 编程模型可以很容易的将多个通用批数据处理任务和操作在大规模集群上并行化，而不必担心像 failover 管理之类的系统问题。MapReduce 编程模型在 Hadoop 之类的开源软件浪潮推动下加速被采用，并且从实验室走向了 Web 搜索、欺诈检测、在线约会等各种各样的实际应用中。但是通用的分布式流计算软件却没有类似的发展趋势。虽然已经有各种各样的工程和商业引擎（[6] ,[7] ,[8] ,[9] ,[10]），但是它们的使用仍然局限于高度专业化的应用。Amini et. al.[7]给出了各种系统的评论。

实时搜索、高频交易、社交网络等新应用的出现将传统数据处理系统所能做的推向了极限[11]。对能够在高数据流量下操作，处理巨量数据的高可扩展流计算解决方案有了一个清晰的需求。例如，为了个性化搜索广告，我们需要实时处理来自几百万唯一用户每秒成千上万次的查询，典型的包括分析用户最近活动如查询、点击等。我们发现用户的会话特征可以提高广告相关性预测模型的精确度。这个性能改善用来提高显示给每个特定用户的广告的相关性[12]。S4 致力于一个通用的分布式流计算平台的需求。

值得注意的是，某些现实世界的系统实现了这样一种流处理策略：将输入数据分隔成固定大小的片段，再由 MapReduce 平台处理。这种方式的缺点在于其延迟与数据片段的长度加上分隔片段、初始化处理任务的附加开销成正比。小的分段会降低延迟，增加附加开销，并且使分段间的依赖管理更加复杂（例如一个分段可能会需要前一个分段的信息）。反之，大的分段会增加延迟。最优化的分段大小取决于具体应用。与其尝试将方形的木钉嵌入圆形的孔，我们决定探索一种简单的可以操作实时数据流的编程模型。我们的设计目标是：

- 提供一种简单的编程接口来处理数据流
- 设计一个可以在普通硬件之上可扩展的高可用集群。
- 通过在每个处理节点使用本地内存，避免磁盘 I/O 瓶颈达到最小化延迟
- 使用一个去中心的，对等架构；所有节点提供相同的功能和职责。没有担负特殊责任的中心节点。这大大简化了部署和维护。
- 使用可插拔的架构，使设计尽可能的即通用又可定制化。
- 友好的设计理念，易于编程，具有灵活的弹性

为了简化 S4 初始的设计，我们作了如下假设：

- 不完全的 failover 是可以接受的。在一个服务器故障时，处理自动的转移到稳定的服务器。存储在本地内存中的处理状态在交接中会丢失。（新的处理）状态会根据输入数据流重新生成。下游系统必须能够优雅降级。
- 不会有节点从正在运行的集群中增加或移除。

我们发觉这些要求对于我们大部分的应用都可以接受。将来我们计划为无法接受这些限制的应用找出解决方案

允许在常用硬件之上进行分布式操作，和避免集群内使用共享内存这两个目标引导我们为 S4 采用 Actor 模式[1]。这种模式有一个简单的原语集并且在工业级规模下的各种框架使用中被证明是有效的[13]。在 S4 中，通过处理单元（Processing Elements (PEs)）进行计算，消息在处理单元间以数据事件的形式传送。每个 PE 的状态对其他 PE 不可访问。PE 之间唯一的交互模式就是发出事件和消费事件。框架提供了路由事件到恰当的 PE 和创建新 PE 实例的能力。这方面的设计提供了封装和地址透明的特性。

S4 的设计和 IBM 的流处理核心（SPC）中间件有很多相同的特性。两个系统都是为了大数据量设计的。都具有使用用户定义的操作在持续数据流上采集信息的能力。两者主要的区别在架构的设计上：SPA 的设计源于一种订阅模式，而 S4 的设计是源于 MapReduce 和 Actor 模式的结合。我们相信因为其对等的结构，S4 的设计达到了非常高程度的简单性。集群中的所有节点都是等同的，没有中心控制。就像我们将要描述的，这得益于 ZooKeeper[14]，一个简单优雅的集群管理服务，可以给数据中心的多个系统共用。

二、设计

我们定义一个流为一个由(K,A)形式的元素组成的序列，这里 K 和 A 分别是键（key）和属性（attribute）的元组。我们的目标是设计一个弹性的流计算平台，在分布式计算环境中消费这样的流，计算中间值，视情况发出其他流。这节包含了一个样例应用，接着是 S4 各种组件的细节描述。

A．例子

图 1 的例子中，输入事件包含了一个英文报价单(Quote)文档。我们的任务是以最小的延迟持续产生一个所有文档范围中出现频率最高的前 K 个单词的排序列表(TOP K)。Quote 事件没有 key，直接发送给 S4。QuoteSplitterPE 对象(PE1)监听 Quote 事件。QuoteSplitterPE 是一个无 key 的 PE 对象，处理所有 Quote 事件。对文档中每一个唯一的 word，QuoteSplitterPE 对象对其计数并发出一个新的 WordEvent 事件，以 word 为 key。WordCountPE 对象监听以 word 为 key 发出的

WordEvent 事件。例如，key 为 word=”said”的 WordCountPE 对象（PE2）接受所有 word=”said”的 WordEvent 类型事件。当一个 key 为 word=”said”的 WordEvent 事件到达，S4 以 word=”said”为 key 查找 WordCountPE 对象。如果 WordCountPE 对象存在，则该 PE 对象被调用，计数增加，否则一个新的 WordCountPE 对像被初始化。每当一个 WordCountPE 对像增加其计数，它就将更新后的计数发给一个 SortPE 对象。SortPE 对象的 key 是一个[1,n]之间的随机整数，n 是想要的 SortPE 对象的总数。当一个 WordCountPE 对象选择了一个 sortID 之后，在其后续生存期就一直使用这个 sortID。使用一个以上 SortPE 对象的目的是为了更好的在多个节点/处理器之间分布负载。例如，key 为 word=”said”的 WordCountPE 对像发送一个 UpdatedCountEvent 事件给一个 key 为 sortID=2（PE5）的 SortPE 对象。每个 SortPE 对象在收到 UpdatedCountEvent 事件时就更新其 TOP K 列表。每个 SortPE 对象定时发送其作为部分的 TOP K 列表给一个单个的 MergePE 对象（PE8），使用任意一个约定的属性键，在这个列子中是 topK=1234。MergePE 对象合并接收到的这些部分列表，然后产出最后的权威 TOP K 列表。

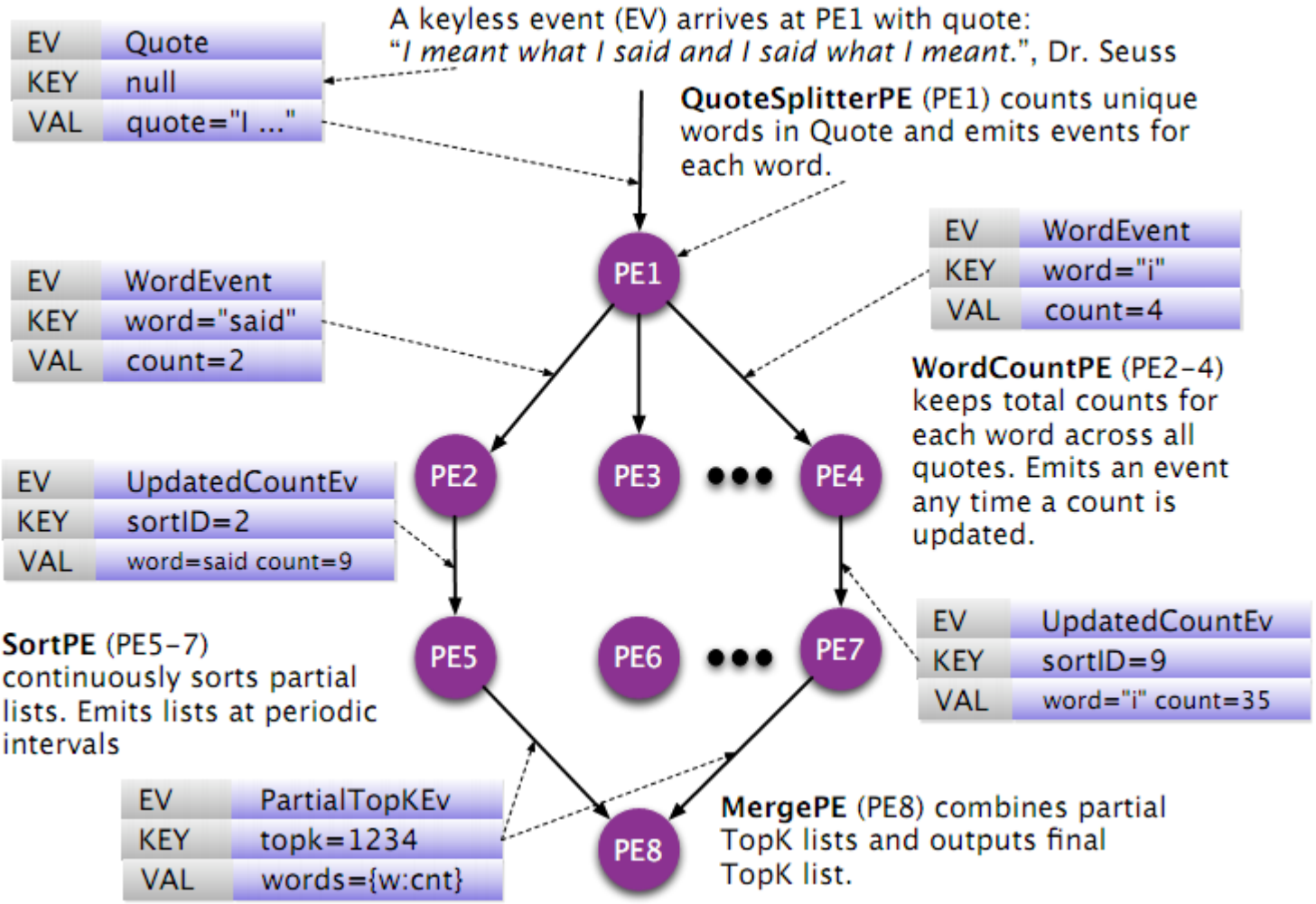

图 1 单词计数例子

PE ID	PE Name	Key Tuple
PE1	QuoteSplitterPE	null
PE2	WordCountPE	word="said"
PE4	WordCountPE	word="i"
PE5	SortPE	sortID=2
PE7	SortPE	sortID=9
PE8	MergePE	topK=1234

图 1 单词计数例子

B. 处理单元(Processing Elements)

处理单元(PEs)是 S4 中最基本的计算单元。每个 PE 的实例被四个要素唯一标识：

(1) 由一个 PE 类和相关配置定义的功能
(2) 它所消费的事件的类型
(3) 这些事件的带 key 的属性（keyed attribute）
(4) 这些带 key 的属性的属性值（value）

每个 PE 只消费事件类型、属性 key、属性 value 都匹配的事件。它可能会产生输出事件。注意会为每个属性值初始化一个 PE。这个实例化由平台进行。例如，在这个单词计数的例子中，会为输入中的每个单词实例化一个 WordCountPE。每当事件中出现一个新的单词，S4 就为其创建一个新的 PE 实例。

有一种特别的无 key 的 PE，没有属性 key 和属性 value。这种 PE 消费相关类型的所有事件。无 key 的 PE 一般在一个 S4 集群的输入层使用，在这里事件会被赋予一个 key。

有一些内置的 PE 用来处理像 count、aggregate、 join 等等标准任务。许多任务可以使用这些标准 PE 来完成，不需要额外的编码。任务使用一个配置文件来定义。可以使用 S4 的 sdk 很容易的编写定制的 PE。

对于有大量唯一 key 的应用，可能有必要随着时间的推移清除 PE 对象。最简单的方法可能是给每个 PE 对象加一个时间戳。如果在一个特定的时间段内没有这个 PE 相关的事件到达，那就可以清除了。当系统内存回收时，PE 对象被清除，其之前的状态也丢失了（在我们的例子中会丢失单词的计数）。这种内存管理策略比较简单，但是并不高效。为了使服务质量(QoS)最大化，我们应该在系统可用内存和对象对系统整体性能影响的基础上最恰当的清除 PE 对象。我们设想一种 PE 对象可以提供其优先级或重要性的方案，这个值是由应用决定的，因此其逻辑应该由应用开发者实现。

C. 处理节点

处理节点（PNs）是 PE 的逻辑主机。它们负责监听事件，在到达事件上执行操作，通过通讯层的协助分发事件，还有发出输出事件。S4 通过一个哈希函数将每个事件路由到 PN，这个哈希函数作用于事件的所有已知属性值上。单个事件可能被路由到多个 PN 上。所有可能的属性 key 的集合通过 S4 集群的配置获知。PN 中的事件监听器将到来的事件传递给 PE 容器（PEC），PE 容器以适当的顺序调用适当的 PE

有一种特殊的 PE 对象类型：PE 原类型(PE prototype)。它有其身份标识的前 3 个元素（功能、事件类型、属性 key）；而属性值是未赋值的。这个对象在初始化时配置，对于任意值 V，它能够克隆自己来创建相同类型，相同配置，以 V 为属性值的全限定的 PE。PN 对其遇到的每一个唯一属性值触发一次这个操作。

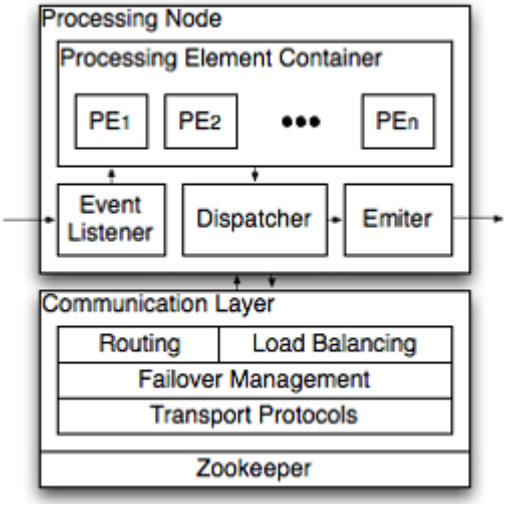

图 2 处理节点

以上设计的一个结果是，所有包含特定属性值的事件保证会到达特定的相应的 PN，并被路由到 PN 内相应的 PE 上。每个编键的(keyed) PE 能够被映射到一个确定的 PN，映射的规则也是通过一个哈希函数，作用于 PE 的属性值上。无属性键的 PE 会在每个 PN 上被初始化。

D. 通讯层

通讯层提供集群管理，故障恢复(failover)到备用节点，逻辑节点到物理节点的映射。它自动检测硬件故障并相应的更新这个映射。

发送消息时只指定逻辑节点，发送者不会感知物理节点的存在或故障导致的逻辑节点重映射。

通讯层的 API 提供几种语言的绑定（如 java、C++）。遗留系统可以使用通讯层的 API 以 round-robin 的模式发送输入事件到 S4 集群中的节点。

通讯层使用一个插件式的架构来选择网络协议。事件可能以可靠或不可靠的方式发送。控制消息可能需要可靠发送，而数据消息可能不需要可靠发送以最大化吞吐量。

通讯层使用 ZooKeeper[16]在 S4 集群节点之间进行一致性协作。ZooKeeper 是 Hadoop 下的开源子项目。它为分布式应用提供分布式一致性协作服务。

E. 配置管理系统

我们预想一个管理系统，在上面操作者可以为 S4 任务创建和销毁集群，并且进行其他的管理操作。分配物理节点到这些 S4 任务集群的操作使用 ZooKeeper[16]进行协作。一个活动节点的集合被分配给特定的任务，剩余的空闲节点仍然留在池中以备需要时使用（例如故障恢复或动态负载均衡）。特别的，一个空闲节点可能同时作为多个不用任务的多组活动节点的冷备。

三、编程模型

最上层的编程范例是编写通用的，可重用的，可配置的处理单元，以能够在各种各样的应用中使用。开发者使用 Java 语言编写 PE。使用 spring 框架将 PE 装配到应用中。

处理单元 API 相当简单和富有弹性。开发者本质上只需实现两个主要的 handler：一个输入事件 handler processEvent()和一个输出机制 handler output()。另外开发者可以为 PE 定义一些状态变量。processEvent()在 PE 订阅的每个事件到达时被调用。这个方法实现输入事件处理逻辑，通常为一个内部 PE 状态的更新。output()方法是可选的，可被配置为以各种方式调用。既可以在特定时间间隔 t 调用，也可以在接收到 n 个输入事件时调用。这也意味着它可以每个消息调用一次，即 n==1 的情况。output()方法实现 PE 的输出机制，通常是将 PE 的内部状态发布到一些外部系统。

图 3. QueryCounterPE.java 摘录

```
private queryCount = 0;
public void processEvent(Event event)
{
      queryCount ++;
}
public void output()
{
      String query = (String) this.getKeyValue().get(0);
      persister.set(query, queryCount);
}
```

我们通过一个例子来描述。考虑一个 PE 订阅了用户搜索查询的事件流，从开始计数每个查询实例，并立即将计数值写到一个外部存储。事件流由 QueryEvent 类型的事件组成。类 QueryCounterPE 实现图 3 中描述的 processEvent() 和 output()。在这个例子中，queryCount 是 PE 的内部状态变量，持有这个 PE 对查询的计数。最后 PE 的配置在图 4 中描述。在这里，属性 keys 告诉我们 QueryCounterPE 订阅了一个 QueryEvent 类型的事件，并且关注事件的 queryString 属性。配置将 PE 和数据处理组件 externalPersister（这个可以是一个抽象的数据服务系统）绑定起来，并且构造 output()方法为每 10 分钟调用一次。

图 4. QueryCounterPE.xml 摘录

```
<bean id="queryCounterPE"
class="com.company.s4.processor.QueryCounterPE">
   <property name="keys">
      <list>
         <value>QueryEvent queryString</value>
      </list>
   </property>
   <property name="persister" ref="externalPersister">
   <property name="outputFrequencyByTimeBoundary" value="600"/>
</bean>
```

四、性能

我们引入了一个现实世界中的基准应用，来描述如何解决 S4 面临的问题，并有一些性能结果。

A. 点击通过率 (Click-Through Rate) 流计算

用户点击是 Web 上最有价值的用户行为之一。它们提供了用户喜好和当前从事事情的即时反馈，这些信息可以用来在突出的位置显示更受用户欢迎的项目以提高用户体验。在点击付费模式的搜索广告中，发布者，代理商，广告客户基于点击计数决定付费多少。点击通过率(CTR)是点击数除以广告显示数得到的比率。当有了足够的历史数据后，CTR 是用户点击一个项目的可能性的一个很好的估算。精确的来源点击对于个性化和(搜索)排名价值无限，但也受点击欺骗的影响。点击欺骗可以用来操纵一个搜索引擎的排名。点击欺骗一般通过运行在远程电脑(bot)或成群电脑(botnet)上的恶意软件来实现。另一种潜在的威胁是 impression spam,就是使请求源自 bot。这些请求可能是无恶意的，但是会影响 CTR 估算。

在这个例子中(图 5)，我们演示如何使用 S4 来实时测量 CTR。在搜索广告上下文中，用户查询通过广告引擎处理，返回一个广告排名列表。在例子中，我们为每一个广告查询组合测量 CTR。为了消除点击和显示干扰，我们使用一个启发

式规则集合来消除可疑的服务和点击。（在这个例子中，一个服务对应一个用户查询，并被分配一个唯一 ID）。对每个服务，返回一个搜索结果页面给用户，用户可能点击也可能不点击超链接。这个页面相关的点击以唯一 ID 分类）

服务事件包含服务数据，如服务 ID、查询、用户、广告等等，而点击事件只包含点击信息和所点击的服务 ID。在 S4 中计算查询广告级别的 CTR，我们需要使用一个由查询和广告组成的 key 来路由点击事件和服务事件。如果点击负载不包含查询和广告信息，我们需要将上次事件路由到的服务 ID 和查询广告关联作为 key。关联之后，事件必须通过 bot 过滤器。最终，服务和点击聚集到一起来计算 CTR。图 5 显示了一个事件流的快照。

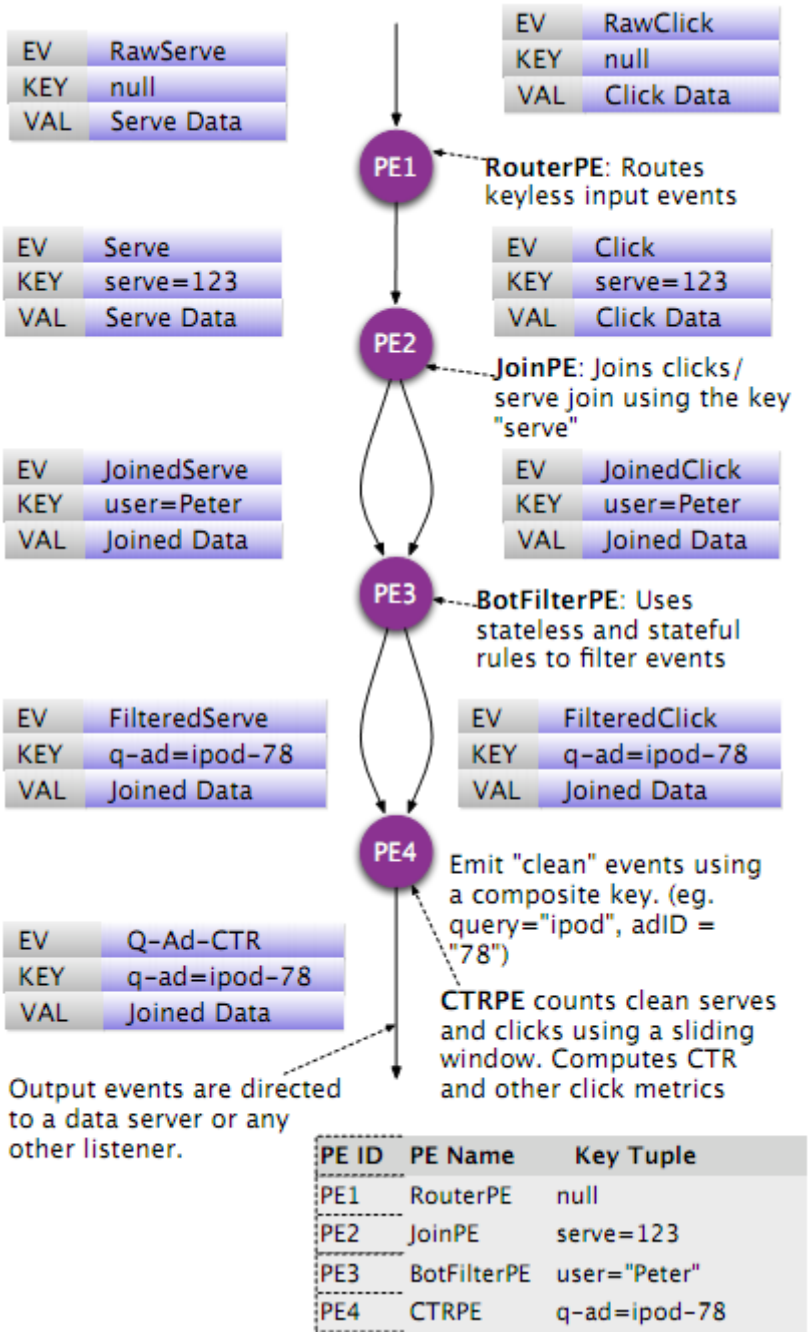

图 5. CTR 计算

B. 实验搭建

1) 在线实验：我们在实际搜索流量的一个随机抽样上运行 CTR 流计算。为了保证体验的一致性，随机分配到这个实验的搜索引擎用户会再根据他们的浏览器 cookie 的一个哈希做一个修正。平均下来，每天大约有一百万的搜索由 250000 用户发起。实验运行了两个星期，观察到的峰值是每秒 1600 个事件。实验集群有 16 台服务器，每台 4 个 32 位处理器，2GB 内存。

要完成的任务是以非常低的延迟计算查询和广告组合的点击通过率(CTR)。CRT 是一个 24 小时滑动窗口内的累计值。这通过将窗口划分为多个 1 小时的 slot，对每个 slot 的点击和显示做累计来实现。窗口内所有 1 小时 slot 的累计值加起来推送到一个服务系统上。这种方法非常节约内存，但是以更新延迟为代价。使用更多的内存，我们可以维护更小粒度的 slot，例如 5 分钟，从而降低更新延迟。如此实现的系统，提供了短时间段的 CTR 估算，合并后就是长时间段的 CTR 估算。在 PN 故障的时候，我们丢失了那个节点的数据，并且正好路由到那个节点的查询/广告无法做短时间段 CTR 估算。在这种情况下我们的故障策略是放弃(该节点的估算数据)回到长时间段估算

2) 离线实验：我们也运行了一个离线压力测试。我们搭建了 8 个服务器的测试集群，每台服务器 4 个 64 位处理器，16GB 内存。在这些机器上跑了 16 个 PN，每台 2 个。我们使用来自搜索流量 log 的真实点击和服务数据，重建了点击和服务事件来计算搜索查询的真实 CTR，这个会作为精确测试的标准(gold standard)。事件数据由 300 万服务和点击组成。

C. 结果

在线流量上的实验显示我们可以在不影响收入的前提下将 CRT 提高 3%，主要通过快速检测低质量的广告并把它们过滤出去达成。

离线压力测试的目标是评估系统在远远高于期望的事件流量下的性能。在我们上文描述的测试集群上，我们将离线生成的事件流导向 S2 网格，以一个递增的事件流速率多次运行。在每次运行之后，将系统估算的 CTR 的值和来自搜索 log 的实际 CTR 做比较。表 6 显示了这次测试的结果：

Events per second	Relative Error in CTR	Data Rate
2000	0.0%	2.6 Mbps
3644	0.0%	4.9 Mbps
7268	0.2%	9.7 Mbps
10480	0.4%	14.0 Mbps
12432	0.7%	16.6 Mbps
14900	1.5%	19.9 Mbps
16000	1.7%	21.4 Mbps
20000	4.2%	26.7 Mbps

表 6

系统显示了在 10Mbps 左右(性能的)下降。下降的原因归结为 S4 无法足够快地处理这个速率下的事件流，因此导致了事件丢失。

五、 应用：在线参数调优

这节中，我们引入一个现实中的 S4 应用实例。一个在线参数调优(OPO)系统[17]，使用搜索广告系统的在线流量自动调整一个或多个参数。系统避免了手工调优和持续的人工介入，同时相比于人工操作能在更短的时间内尝试更大范围的参数。系统接受目标系统发出的事件（在我们的列子中是搜索广告系统），测量性能，应用一个适配算法来决定新的参数，然后将新的参数注入回到目标系统。这种闭合循环的方式原理上与传统的控制系统类似。

A．功能设计

我们假设目标系统(TS)产生的输出可以用一个流来表示，并且可以通过一个可配置的目标功能(OF)测量其性能。

我们将目标系统的输出流随机地分割为 slice1、slice2 两个 slice(分片)。我们要求目标系统能够对每个 slice 应用不用的参数值，并且每个 slice 能够以此标识出自己。

在线参数调优系统(OPO)有 3 个上层功能组件：测量器、比较器和优化器

1）测量器(Measurement)：测量器组件吸收 slice1 和 slice2 流，在每个 slice 中测量 OF 的值。对一个连续的 slot（duration of a slot）测量 OF。slot 可以按时间段来定义，也可以按输出流的事件数定义（连续 n 个事件作为一个 slot）。

2）比较器（Comparator）：比较器组件将测量器的产出作为输入，确定 slice1 和 slice2 的性能是否有统计意义上的不同。当检测到不同时，发送一个消息给优化器。如果在指定个数的 slot 之内没有检测到不同，则声明两个 slice 是相同的。

3）优化器：优化器实现了适配器模式。它将参数影响历史包括最近的比较器输出作为输入，为 slice1 和 slice2 产生新的参数值，这也是新的实验循环开始的信号

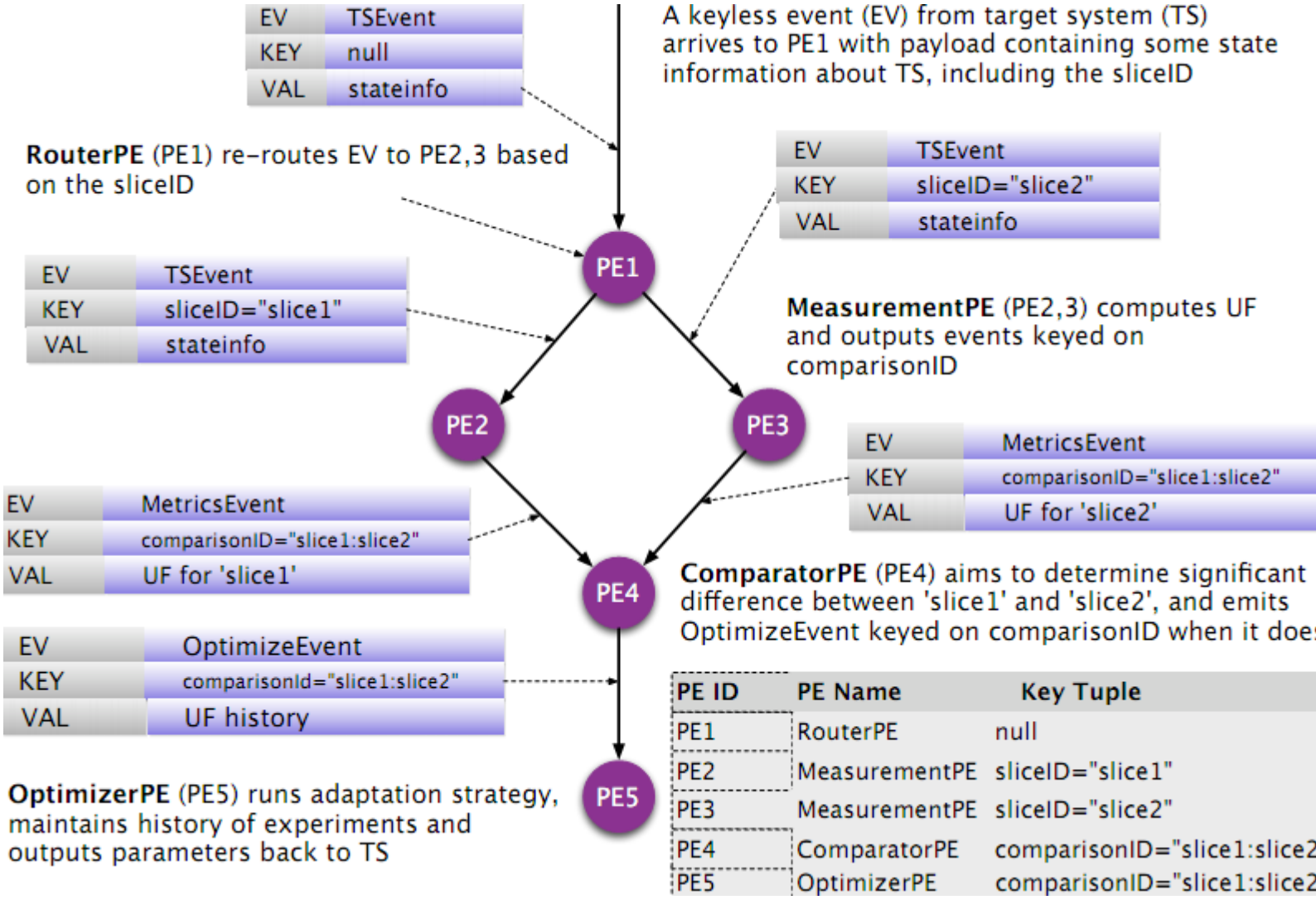

图 7 opo 在 s4 中的实现

B. S4 实现

图 7 描述了 OPO 系统在 S4 中的实现。3 个功能组件以 3 个 PE 的形式实现。MeasurementPE 以 sliceID 为 key，slice1 和 slice2 各有一个实例（我们能够很容易的将其扩展为更多 slice 以支持更高级的策略）。目标功能的度量在固定长度的时间片上进行。ComparatorPE 以 ComparatorID 为 key，对应到一对 slice，在我们的例子中是 slice1，slice2。基于一个对度量结果对的独立的 t-test 来确定两个 slice 之间是否有统计意义上的不同。它被配置为取合格的度量结果中的最小数值。OptimizerPE 以 slice1 和 slice2 为 key。为了实现适配器模式，我们使用一个 Nelder-Mead(aka Amoeba)算法[18]的修改版：一个任意斜率的最小化算法。OTO 系统输出的参数反馈回搜索广告服务系统。这些参数值控制服务系统的外观(aspects)，因此导致不同的用户行为。

C．结果

我们在一个搜索广告系统的实际流量分片上运行了 OPO 系统。目标是在当前的参数值（使用传统方法调整）上尽可能地提高系统度量。目标功能是一个搜索引擎上代表总收益和用户体验关系的公式（用户体验和收益的关联性）。流量分片基于搜索引擎用户空间的分布(partitions)：每个 slice 接受大约每天 200000 用户的流量。系统运行了 2 周，尝试调优一个已经知道对搜索引擎性能有重大影响的参数。OPO 系统生成的优化参数值被证明在系统性能的主要度量上有显著的提高：收益提高了 0.25%，点击跳转提高了 1.4%

D．总结

我们展示了使用 S4 的一个在线参数调优系统的设计和实现。我们应用这个系统来调整一个搜索广告系统，得到了可喜的结果。这个系统可以用来调整任何有可调参数的动态系统（只要动态系统满足之前描述的几个要求）。

六、未来的工作

当前系统使用静态路由，通过 ZooKeeper 自动 failover，但是缺乏动态负载均衡和健壮的在线 PE 迁移。我们计划增加这些特性。

七、致谢

作者们感谢所有支持和帮助这个项目的雅虎同事，尤其是 Khaled Elmeleegy, Kishore Gopalakrishna, George Hu, Jon Malkin, Benjamin Reed, Stefan Schroedl 和 Pratyush Seth。我们也感谢雅虎使 S4 在 Apache2.0 许可下开源[19]。

参考资料

[1] G. Agha, Actors: A Model of Concurrent Computation in Distributed Systems. Cambridge, MA, SA: MIT Press,1986.

[2] B. Edelman, M. Ostrovsky, and M. Schwarz, “Internet ad-vertising and the generalized second-price auction: Selling billions of dollars worth of keywords,” American Economic Review, vol. 97, no. 1, pp. 242–259, 2007.

[3] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein,K. Elmeleegy, and R. Sears, “MapReduce online,” EECS Department, University of California, Berkeley, Tech.Rep. UCB/EECS-2009-136, Oct 2009. [Online].
Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-136.html

[4] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” commun. ACM, vol. 51, no. 1,pp. 107–113, 2008.

[5] Apache Hadoop. http://hadoop.apache.org/.

[6] D. C. Luckham, The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[7] L. Amini, H. Andrade, R. Bhagwan, F. Eskesen, R. King,P. Selo, Y. Park, and C. Venkatramani, “SPC: a distributed,scalable platform for data mining,” in DMSSP ’06: Pro-ceedings of the 4th international workshop on Data mining standards, services and platforms. New York, NY, USA:ACM, 2006, pp. 27–37.

[8] D. J. Abadi, Y. Ahmad, M. Balazinska, U. C¸ etintemel,M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey,A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. B. Zdonik, “The Design of the Borealis Stream Processing Engine,” in CIDR, 2005, pp. 277–289.

[9] D. J. Abadi, D. Carney, U. C¸ etintemel, M. Cherniack, C. Con-vey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik,“Aurora: A new model and architecture for data stream management,” The VLDB Journal, vol. 12, no. 2, pp. 120–139, 2003.

[10] Streambase. http://streambase.com/.

[11] M. Stonebraker, U. C¸ etintemel, and S. Zdonik, “The 8 re-quirements of real-time stream processing,” SIGMOD Rec.,vol. 34, no. 4, pp. 42–47, 2005.

[12] S. Schroedl, A. Kesari, and L. Neumeyer, “Personalized ad placement in web search,” in ADKDD ’10: Proceedings of the 4th Annual International Workshop on Data Mining and Audience Intelligence for Online Advertising, 2010.

[13] R. K. Karmani, A. Shali, and G. Agha, “Actor frameworks for the JVM platform: a comparative analysis,” in PPPJ ’09: Proceedings of the 7th International Conference on Principles and Practice of Programming in Java. New York, NY, USA: ACM, 2009, pp. 11–20.

[14] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, “ZooKeeper: wait-free coordination for internet-scale sys-tems,” in USENIXATC’10: Proceedings of the 2010 USENIX conference on USENIX annual technical conference. Berke-ley, CA, USA: USENIX Association, 2010, pp. 11–11.

[15] K. Gopalakrishna, G. Hu, and P. Seth, “Communication layer using ZooKeeper,” Yahoo! Inc., Tech. Rep., 2009.

[16] Apache ZooKeeper. http://hadoop.apache.org/zookeeper/.
[17] J. Malkin, S. Schroedl, A. Nair, and L. Neumeyer, “Tuning Hyperparameters on Live Traffic with S4,” in TechPulse 2010: Internal Yahoo! Conference, 2010.
[18] J. A. Nelder and R. Mead, “A simplex method for function minimization,” Computer Journal, vol. 7, pp. 308–313, 1965.
[19] The S4 Open Source Project. http://s4.io/.

翻译：林轩 linxuan@taobao.com
原文地址：http://labs.yahoo.com/files/KDCloud 2010 S4.pdf