基于时标的分布式事件处理系统框架

2019 年 7 月 19 日

1. 系统描述

本系统的最大特点是严格按照事件具备的唯一时标进行全程处理，以保证等式$y_{j(t)=f(x_i(t))}$是完整正确的。

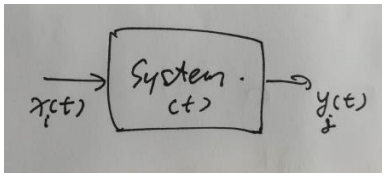

图 1 基于时标的处理系统

系统主要由节点、通道和事件等三个基本元素。

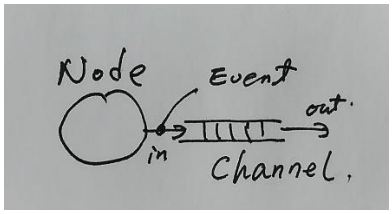

图 2 三个基本元素

事件是本系统中唯一被处理的对象，是一个带有唯一、不变的时标标签的抽象对象实体。事件可以定义成任何一个物理实体，例如一个分布在某个服务器上的数据集、文件或一个共享内存、一个数据库的表等。

通道是用做传递事件的单元，其主要任务是按时标顺序组织输入事件，并按时标顺序输出。

节点是一个处理单元，其内含一个可动态外挂一个事件处理程序的插件机制，由事件处理程序对关联到本节点的所有输入通道接收具有相同时标的事件进行处理，并按要求把处理结果事件向与本节点关联的通道输出。

分布式是系统基本特色，每个系统内的节点、通道等单元实体可以分别部署到不同的计算资源上，并通过网络实现各个计算资源的连通。

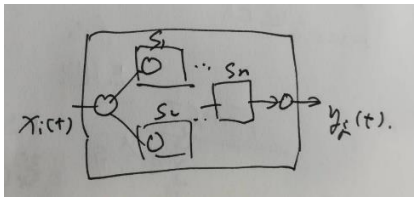

图 3 分布式设计

在实用环境中，可引用已有系统，如消息队列、共享存储系统等，作为工程的通道及事件存储管理。

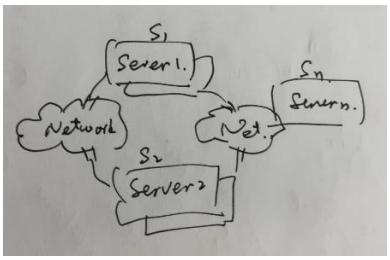

图 4 分布式部署

2. 基本单元

这里重点介绍系统中基本的组成单元，它们包括有节点、通道、事件、注册器和执行器。

2.1. 节点

节点在系统中是事件的处理实体，核心是它内部有一个外挂事件处理程序的插件机制。每个节点都可对应多个输入、输出通道，用于向节点的事件处理程序输入事件，并将处理事件输出。端特性的节点是一个特殊情况，它或许无需定义输入通道，例如对于事件采集节点；或许无需定义输出通道，例如对于事件处理呈现（如报告、展现等）节点。

也可以根据要求，定义具有其它特殊功效的节点，如多通道汇聚节点或事件多通道分配节点等。

2.2. 通道

事件在系统中的传递是由通道实现。它有两个功能层面，其一是事件管理层面，主要实现按时标顺序组织输入和输出的事件，实现严格按时序传递事件的机制；其二是事件传输层面，用于实现事件实体有效载荷的传输，例如采用消息队列、共享存储或其它网络传输系统等实现事件在不同计算系统间的传递。

2.3. 事件

事件是系统运行的带有特定时标的载荷，各种可计算的物理实体都可通过事件进行抽象，例如对一个存放在一个共享内存系统中的数据集，或存放在文件系统中的共享文件，或数据库系统中的表等。

2.4. 注册器

注册器的主要功能是实现节点、通道和事件等实体的命名管理，按名称注册和获取指定类型的对象实体是其基本功能。

注册器包含三个链表，分别与节点、通道和事件对应，允许按某种策略把单个对象实体插入链表中，或转移链表中的对象实体位置。

注册器提供一个外部 API 用于获取它提供的服务。

2.5. 执行器

执行器是系统运行的主程序，每个系统允许有一个执行器。

执行器在运行时，会按链表顺序调度执行每个节点的事件处理程序。

3. 命名及注册管理

系统中每个节点、通道和事件实体都具有其唯一的 ID 和名称。

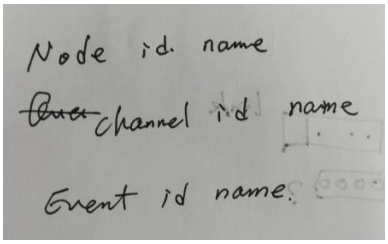

图 5 命名

在每次创建节点、通道和事件实体时，系统会自动将其 ID 和名称注册到系统内一个注册器中，由注册器按类别维护每个实体 ID、名称和实体对象的映射表。系统及其内每个节点上的处理程序都可从注册器按名称获取相应的实体对象。

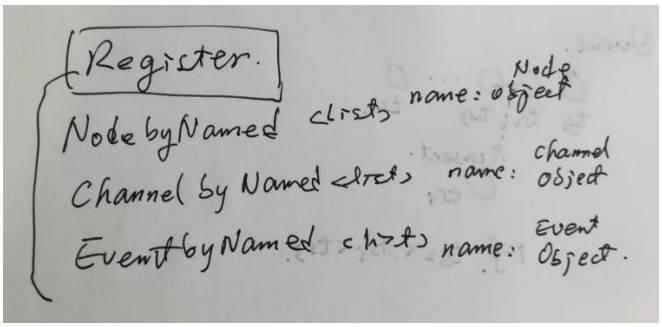

图 6 注册器

4. 通道对事件的排序

由通道来维护输入事件的时标的序列，确保事件按序排列，并根据时标按照“先进先出，即 FIFO”原则输出事件。

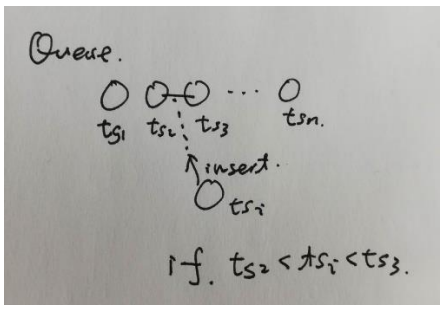

图 7 通道按时序对事件排序

在向通道输入一个事件时，通道会根据该事件的时标按时序将其插入到正确位置上。

5. 同步器

同步器是系统核心功能单元，是系统实现等时标处理的关键组件，用来保证来自多个通道的具有相同时标事件的完整输出。当其中一个通道没有指定时标事件时，有两种策略选择，其一是等待，直到所有通道都能提供相同时标的事件，其二是处理下一个满足所有通道都能提供相同时标的事件。

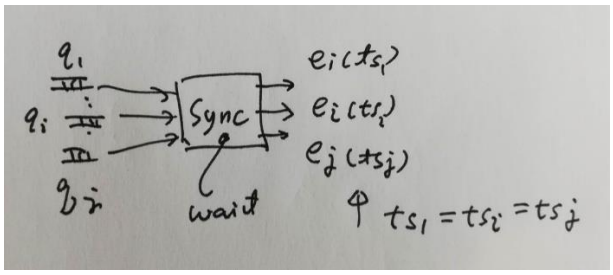

图 8 同步器

6. 端节点

端节点是系统不可缺的资源类型。有两种端节点分别对应前后两端，一个是用于事件生成，一个用于事件呈现。

例如定义一个前端节点，来定期采集某个数据源并生成事件，把它输出指定的输出通道上。

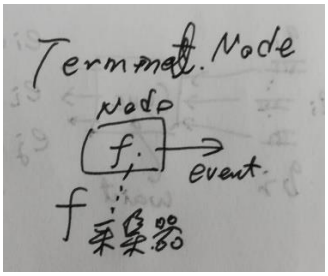

图 9 端节点

7. 定义一个处理框架

处理框架是构建系统的定义文件，用 json 语言对系统内部构建进行描述。

基本的描述语句为：

```
Node:{
    Name: 节点名称,
    Input:{
        Policy: 输入策略执行程序入口,
        Name: 通道名称,
        …
    },
    Output:{
        Policy: 输出策略执行程序入口,
        Name: 通道名称
        …
    }
```

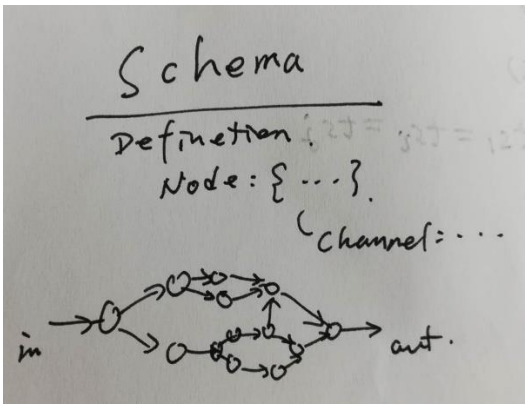

图 10 用描述文件定义一个处理框架

8. 系统运行原理

在构建系统时首先要完成系统构建过程。通过配置文件定义系统结构，在系统初始化时根据这个定义文件创建每个节点、通道及其与节点的关系。

在系统运行时，由执行器统一调度本系统内的每个节点的处理程序来推动整个框架的运转。在系统框架中，汇集节点和分路节点是节点的两个特殊用途，分别用于通道的多对一和一对多或多对多处理需要。

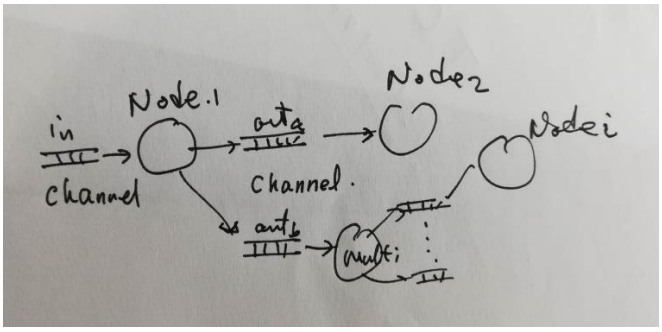

图 11 事件处理机制

在事件生成时，会自动带上一个时标，此时标会始终依附在事件及与其派生的事件上。

9. 应用样本

这里以一个数据采集系统为例进行应用样本说明，系统结构如图所示。

它有三个前端数据采集节点 N1、N2 和 N3，分别针对三个不同的数据源进行定时数据采集，这三个节点与一个时钟服务器保持时标同步（秒级），假设 N1 定时从数据传感器上采集数据，并生成事件；N2 定时从一个业务系统上采集数据，并生成事件；N3 定时从一个服务接口采集数据，并生成事件。

这三个节点把各自的采集到的事件按定义的规则通过通道 C1、C2、C3 和 C4 输出到 N4 和 N5 上。

节点 N4 对来自 N1 和 N2 的事件进行对时标处理，并将处理后的事件通过 C5 通道输出给 N6，同样，节点 N5 对来自 N2 和 N3 的事件进行对时标处理，并将

处理后的事件通过 C6 通道输出给 N6。

节点 N6 对时标处理来自 C5 和 C6 上的事件，并将最终处理结果存入目标数据库中。

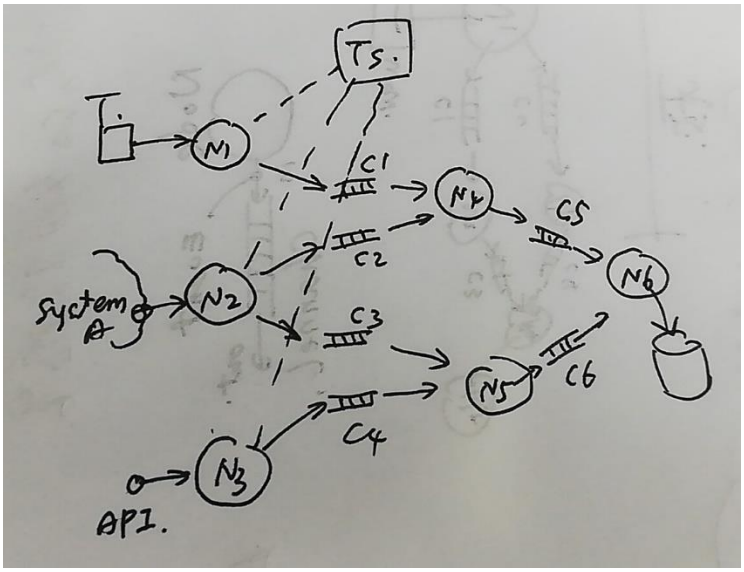

图 12 一个数据采集系统

10. 实例

```
# 定义系统
# 三信号源的信号经过一个“乘法器”处理后输出到展示节点
#
system = {
    "node": {
        "sin": {
            "output": [
                "C1.1",
                "C2.3"
            ],
            "function": sin_func
        },
        "cos": {
            "output": [
                "C1.2",
                "C2.2"
            ],
            "function": cos_func
```

```
        },
        "noise": {
            "output": [
                "C1.3",
                "C2.4"
            ],
            "function": noise
        },
        "nN1": {
            "input": [
                "C1.1",
                "C1.2",
                "C1.3",
            ],
            "output": [
                "C2.1",
            ],
            # 定义一个同步器
            "synchronizer": True,
            "function": multi_func
        },
        "N2": {
            "input": [
                "C2.1",
                "C2.2",
                "C2.3",
                "C2.4"
            ],
            # 定义一个同步器
            "synchronizer": True,
            "function": func2
        }
    },
    "channel": [
        "C1",
        "C1.1",
        "C1.2",
        "C1.3",
        "C2.1",
        "C2.2",
        "C2.3",
        "C2.4"
    ]
```

```
}
```

系统输出为：

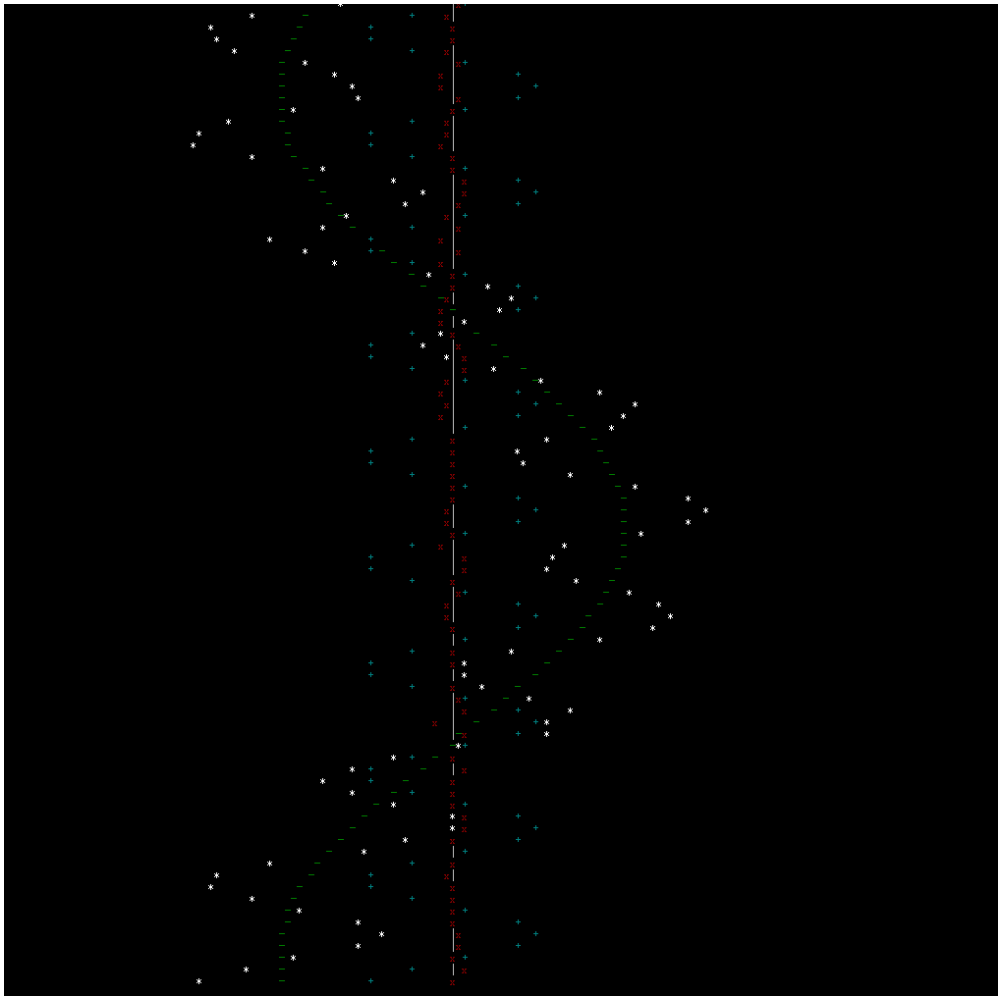

图 13 三信号源处理后的输出