基于 `PTOLEMY II 11.0.1` 的实时系统实例设计与仿真

1. 系统概述

本系统基于 Ptolemy 实现了对小球运动的仿真系统，通过对小球的上、下、左、右四种运动的仿真，使用先进先出（First In First Out，FIFO）、固定优先级（Fixed Priority，FP）、Round-Robin、最早截止时间优先（Earliest Deadline First，EDF）四种调度策略对小球的四种运动进行调度，从而探究这四种调度策略的特征。本系统主要包含三个模块：

1. 任务模块：对小球的四个方向的运动进行建模，抽象为实时系统中的任务（Task）；
2. 调度器模块：对小球的四个方向的运动所对应的Task进行调度，调度策略包括
    - 先进先出算法（First In First Out，FIFO）
    - 固定优先级算法（Fixed Priority，FP）
    - 循环调度算法 （Round-Robin）
    - 最早截止时间优先算法（Earliest Deadline First，EDF）
3. 可视化模块：对小球的运动以及系统的调度决策进行可视化输出。

本文档将针对实时系统的建模方式、基于 `Ptolemy II 11.0.1` [1] 的建模实现、四种调度算法的特性分析、以及可视化结果进行叙述。最后一章中包括我们的源码结构以及系统的运行方法。

2. 整体设计

本节将针对系统概述中描述的顶层建模设计进行叙述，描述系统的基本运作原理。`Ptolemy II 11.0.1` 来自于 UC Berkeley 项目组成员所制作的信号仿真系统，经过多年的发展完善，它具有对实时系统进行仿真的功能。我们使用它对我们设计的小球运动系统进行建模，仿真其实时环境下的运作模式，探究实时系统调度算法的特征与优缺点。顶层视图如图1所示，其中参数TaskNum表示系统中的任务数量，Frequency表示时钟周期：

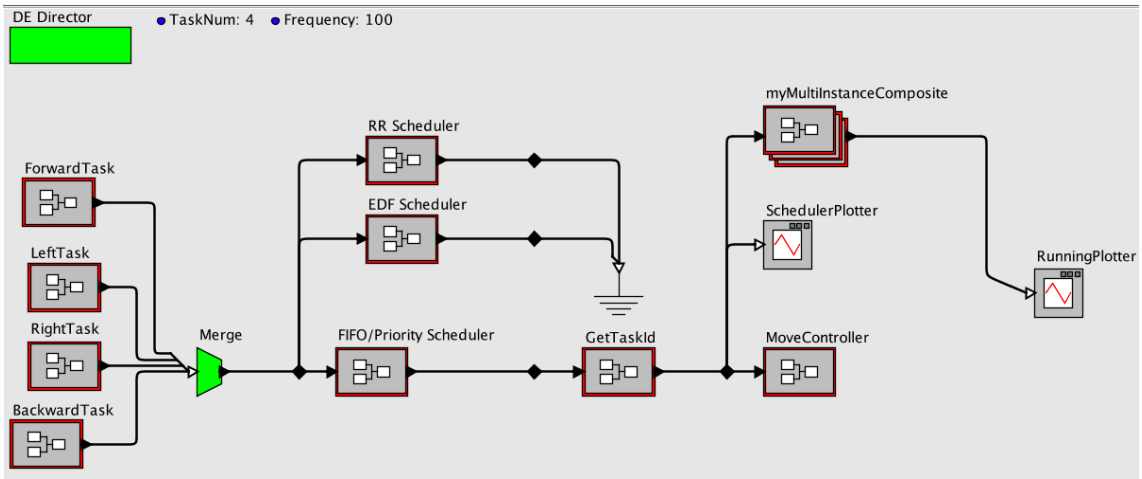

图 1 整体设计

2.1 任务模块

本模块负责对小球的四个方向的移动进行建模。在我们的模型中，需要进行的简化假设如下：

1. 小球只能在一个平面上运动，且只有四个移动方向，分别为：
    - 向前 (Forward)
    - 向后 (Backward)
    - 向左 (Left)
    - 向右 (Right)
2. 小球在每个给定的时刻只能向一个方向移动，如向左移动。不存在类似于向左上方移动的情况
3. 小球移动方向的切换不需要时间

根据以上三条假设，可以将小球的四个方向的移动分别抽象为实时系统中的四个任务：ForwardTask, BackwardTask, LeftTask, RightTask。这四个Task遵循实时系统建模的所有条件，即：只有一个处理任务的中央处理器CPU，每个给定时刻只有一个任务可以在运行；两个任务之间的上下文切换开销不计。这四个任务在 `Ptolemy II 11.0.1` 仿真软件中的建模结构图如图2所示：

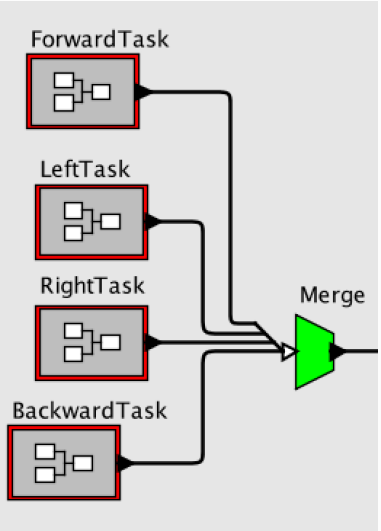

图 2 任务模块设计

同时，我们为每个任务添加了实时系统中任务调度的相关参数，包括

- computationTime，计算时间。即需要向某一方向行进的时间长度。
- priority，优先级。优先级高的任务可以抢占优先级低的任务。
- startTime，即任务到达的时间。
- id，任务的编号。每个任务都有唯一的编号。
- relativeDeadline，相对deadline。

此模块输出的任务及其包含的调度相关信息（computationTime, priority, id）将被Merge模块所合并，统一输入到调度器模块。本模块设定了默认的startTime，即无需用户进行操作，即可触发任务。具体设计见第3章。

2.2 调度器模块

调度器模块接收来自于任务模块的任务信号，对来自于三个Task的Job按照三种调度策略进行调度。三种调度策略分别有：

- 先进先出算法（First In First Out，FIFO）：

  先来的任务先被调度运行，后来的任务需要等待先来的任务执行完毕才可以执行。这种调度算法有较为简洁的实现（在建模软件中实现也较为简洁），但没有考虑到任务的优先级；

- 固定优先级算法（Fixed Priority，FP）：

  优先级较高的任务先被执行。此算法虽然考虑到了任务的优先级，使得更为重要的任务优先执行，但存在饿死低优先级任务的风险；

- 循环调度算法 （Round-Robin）：

  设置一个时间片，每个任务在每次循环中至多执行一个时间片的时间，超过该时间片后即刻放弃CPU，而在每个循环中所有任务按照FIFO算法进行调度。该算法的优点是达到了所有任务之间执行时间的公平，最小化了等待时间，但是由此带来了较为频繁的任务切换，即产生更多的上下文切换开销。本文没有对上下文切换的开销进行仿真，如2.1节中的简化假设所述。

- 最早截止时间优先算法（Earliest Deadline First，EDF）：

  截止时间最早的算法优先执行。本文中实现的EDF是抢占式的EDF算法，它最优化了deadline miss的发生，即发生的deadline miss最少。

在 `Ptolemy II 11.0.1` 中，我们使用了三个模块来对这四种调度算法进行建模，如图3所示，其中FIFO和FP算法被合并在同一个模块实现，这是由于FIFO本质上是所有任务优先级相同的FP调度算法。调度模块接收来自于Merge的任务信号，向可视化模块输出Task Id。注意，可视化模块只接收来自于一个调度器的输出，其他调度器的输出将被抛弃。如图3，我们默认选择接收来自于FIFO/FP调度器的输出信号。

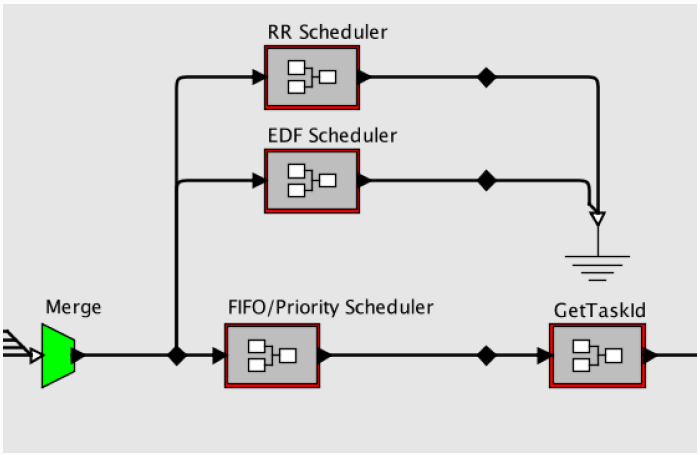

图 3 调度器模块设计

2.3 可视化模块

可视化模块接收来自于调度器模块的id，即当前运行的任务的id，来进行两部分可视化，分别是调度器调度决策可视化模块（折线图）以及小球运动的动画显示模块。如图4所示，MoveController实现了对小球运动的动画显示，使用了 `Ptolemy II 11.0.1` 软件中的Translate 3D组件以及ViewScreen组件。RunningPlotter对每个任务的调度情况绘制折线图，SchedulerPlotter对系统中正在运行的任务的ID绘制折线图。MyMultiInstance对ID信号进行多路复用，输入RunningPlotter产生4条折线图，对应每

个任务的调度情况。

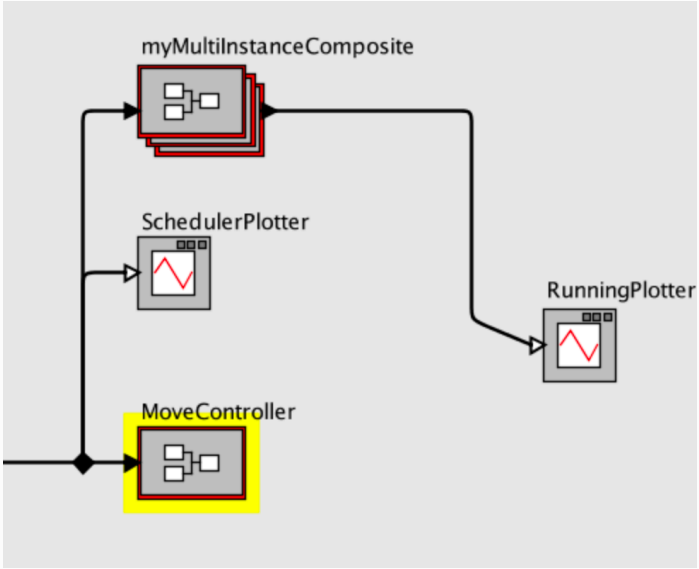

图 4 可视化模块设计

3. 详细设计

本章对系统的三个模块的设计进行叙述，描述其运行原理，以及如何对不同的调度算法进行仿真。我们的设计使用了 `Ptolemy II 11.0.1` 软件中的各类组件，每个组件发挥着不同的功能，详细参考文献2。我们用 `特殊字体` 表示 `Ptolemy II 11.0.1` 软件中的组件，用普通字体表示其他参数。

3.1 任务模块

如图5所示为ForwardTask模块的内部结构，我们使用 `Const` 组件表示任务的参数，即computationTime，priority，id，relativeDeadline。而 `SingleEvent` 组件用于在给定时刻startTime（任务的到达时间）产生一个信号，输入到各个 `Const` 的trigger端口。这样，每个 `Const` 组件将会在startTime时刻向 `RecordAssembler` 发送其value（即computationTime，priority，id，relativeDeadline），而其他时刻 `Const` 组件没有输出（输出为空）。 `RecordAssembler` 将这四个value（可能为空）合并成一条信息，在startTime时刻输出到out端口。

我们基于此结构设计了四个任务，通过对每个任务的startTime、relativeDeadline、computationTime等参数进行手动设定，从而探究不同调度算法的特性。如2.1节所示。这四个模块产生的信号被 `Merge` 组件聚合后，发向调度器模块。

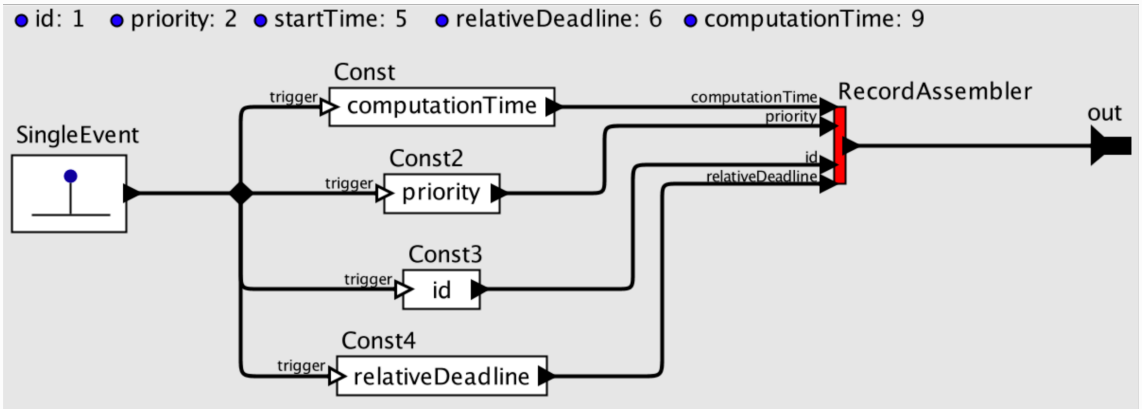

图 5 任务模块详细设计

3.2 调度器模块

调度器模块接收 `Merge` 组件发来的Task信号。我们实现了四种不同的调度算法，本节将一一讲解。

3.2.1 先进先出、固定优先级算法（FIFO、FP）

由于FIFO算法是FP算法priority = 1的特殊情况，故我们用同一模块实现FIFO算法和FP算法。如图6，in端口输入的是来自四个Task的信号，in信号发送到 `RecordDisassembler` 进行解码，得到两个信号，分别是computationTime和priority，这两个信号是Task的执行参数。我们添加了一个参数isFIFO，让我们不加修改即可实现两种调度算法：

- 当isFIFO设置为1时，该模块是是一个FIFO调度器，所有Task的优先级相同，都为1；
- 当isFIFO设为0时，该模块是FP调度器，priority信号被 `expression` 组件保留。

我们将Task信号以及computationTime、priority信号作为 `AdvancedServer` 组件的输入，同时使用 `DiscreteClock` 组件作为 `AdvancedServer` 的trigger，使其在每个时钟周期都对到达的任务进行处理，并且输出正在执行的Task，即

`AdvancedServer` 组件改造自 `Server` 组件，其本质是一个固定优先级的服务器，接收任务信号以及任务对应的computationTime、priority，当任务执行完成后输出对应Task。这样，可视化模块不能在每个时钟周期获知正在执行的Task，也无法绘制相关图像。为此，我们定制了 `AdvancedServer` ，添加trigger输入，使得 `AdvancedServer` 可以按照时钟周期输出正在执行的Task，我们称之为currentExec输出端口。为此，我们修改了 `Server.java` 的fire函数（见第5章），在每次接收到trigger信号之后，向currentExec端口输出当前执行的Task，如果当前Server中没有正在执行的Task，currentExec端口将输出null。

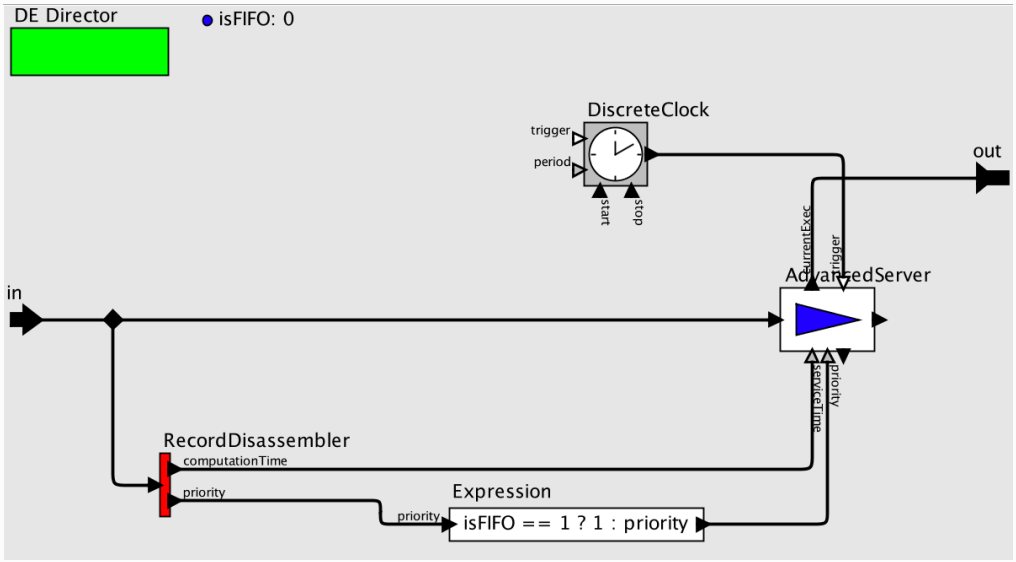

图 6 FIFO/FP调度器设计

3.2.2 最早截止时间优先算法（EDF）

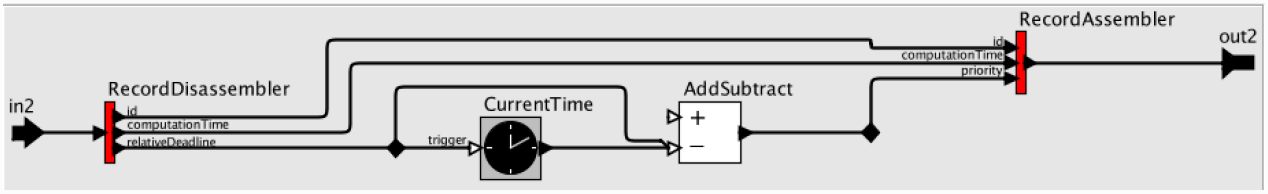

图 7 GetPriorityByDeadline模块设计

有了之前 `AdvancedServer` 的实现，我们可以不费力地将FIFO调度器转化为一个EDF调度器。如图8所示，我们增加了一个 `GetPriorityByDeadline` 模块，用来将距离截止时间近的任务设为高优先级任务，其实现如图7所示。 `CurrentTime` 模块用来获取当前时间，将当前时间currentTime加上relativeDeadline，再取负值，作为任务的优先级priority。这样，绝对截止时间越早的进程将会有更高的优先级，即为抢占式的EDF调度算法。

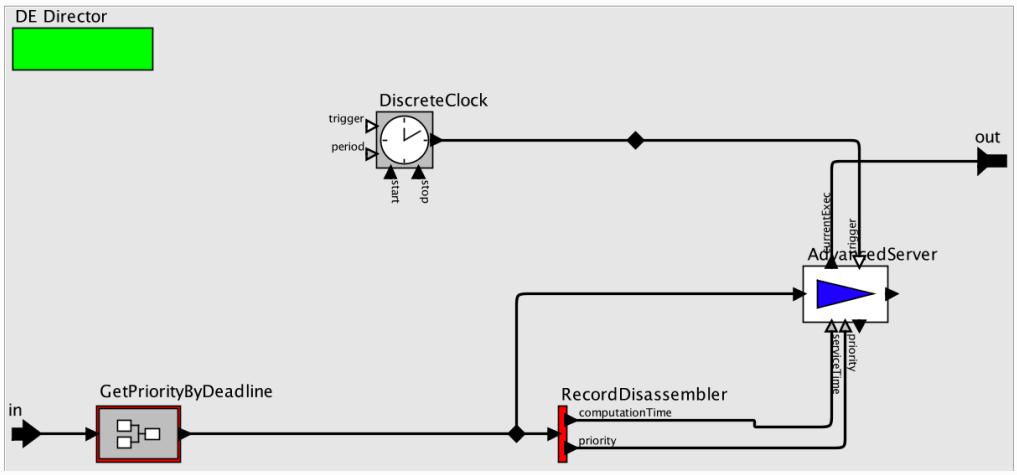

图 8 EDF调度器设计

3.2.3 循环调度算法 （Round-Robin）

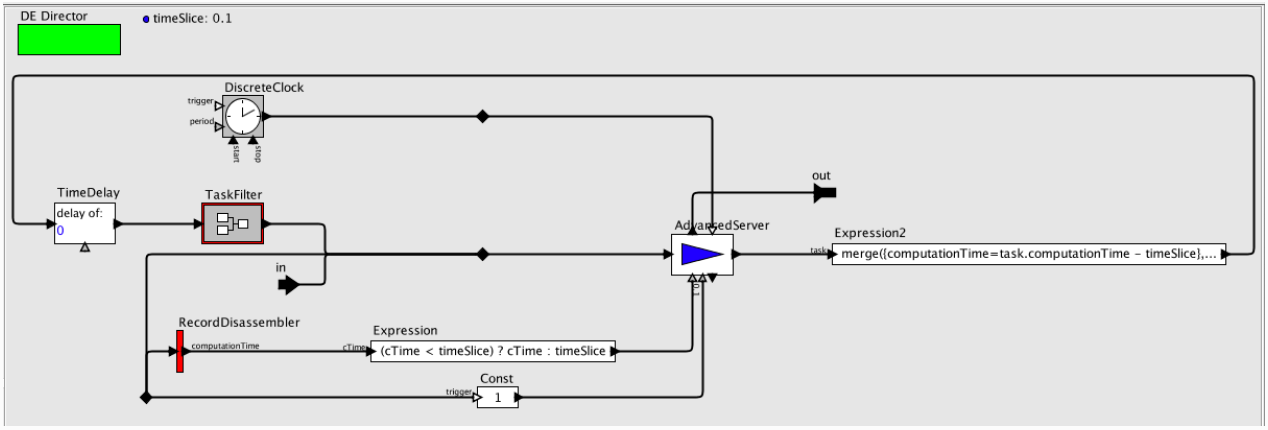

图 9 RR调度器设计

循环调度算法不能复用来自FIFO调度器的设计，需要重新进行设计。基于已有的`AdvancedServer`，我们修改serviceTime（computationTime）输入来实现RR调度算法。如图9所示，我们将in端口的输入使用`RecordDisassembler`进行解码，获取当前到达任务的执行时间，作为ctime输入到`Expression`组件中。`Expression`对ctime进行判断，如果ctime大于RR调度器的时间片参数timeSlice，则将timeSlice作为serviceTime输入到`AdvancedServer`组件；否则将剩余的ctime输入到`AdvancedServer`组件。`Const`组件将固定的priority（1）输入到`AdvancedServer`组件。`AdvancedServer`组件输出的Task再经过`Expression2`，其computationTime被减去了timeSlice，该Task经过`TaskFilter`组件的判断，如果computationTime为0，即已经完成执行，那即可过滤掉此Task。`TaskFilter`的实现如图10所示。

经过如此的设计，每个任务不会占用Server超过timeSlice的时间就会将Server释放给下一个Task，而`AdvancedServer`组件保证了在每个调度循环中，所有Task遵循FIFO调度算法。于是，我们即可实现RR调度器。

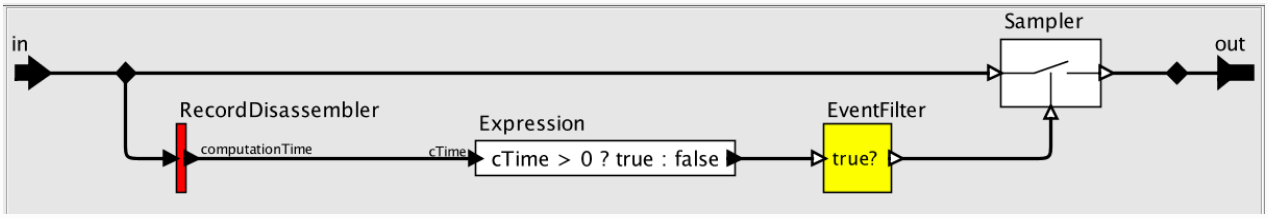

图 10 TaskFilter模块设计

3.3 可视化模块

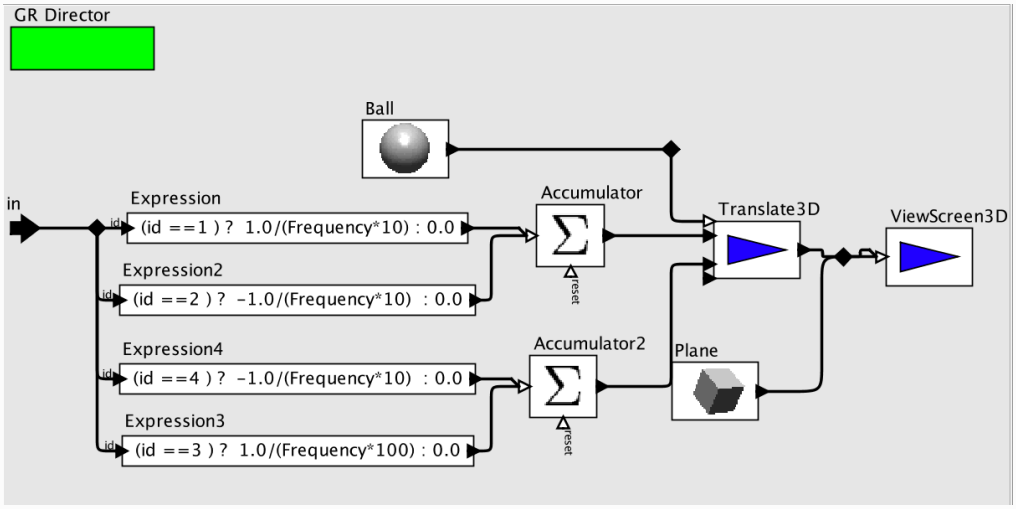

图 11 小球运动可视化模块设计

如图4，我们通过 `Plotter` 组件接收当前正在执行的Task对应的id，绘制两个调度折线图，其中 `myMultiInstanceComposite` 可以使得 `RunningPlotter` 组件绘制每个Task的运行情况。 `MoveController` 负责对小球运动进行动画显示，其实现如图11所示。当ForwardTask被调度运行时，小球的纵坐标加1；当BackwardTask被调度运行时，小球的纵坐标减1；当LeftTask被调度运行时，小球的横坐标加1；当RightTask被调度运行时，小球的横坐标减1；将平面、小球以及计算好的小球坐标输入 `Translate3D` 、 `ViewScreen3D` 组件进行显示即可完成动画显示，图12展示了小球运动的显示效果。

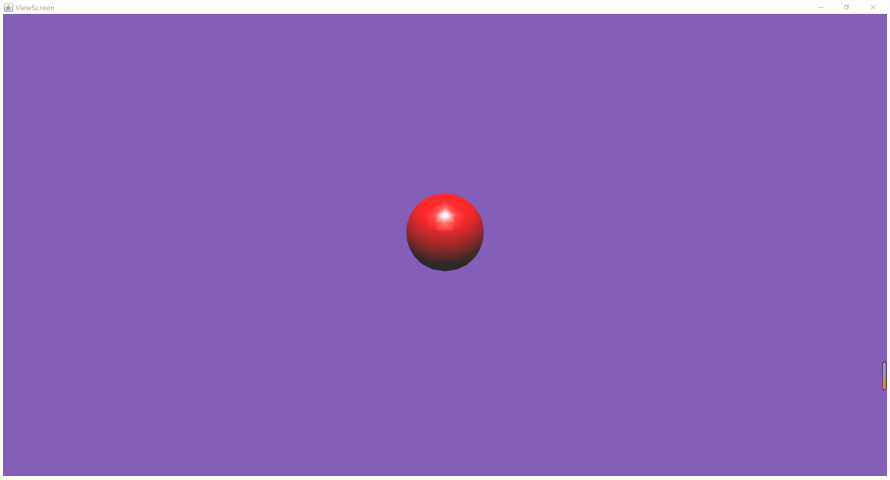

图 12 小球运动效果图

4 调度算法实验分析

本章通过设定每个任务的computationTime，priority，startTime，relativeDeadline，运行上述章节所设计的仿真系统，对四种调度算法进行仿真，获取调度折线图，从而观察其优缺点。

4.1 实验设定

如下表所示，在四个调度器的实验中，我们设定了相同的实验参数，从而比较不同调度算法的特性与优缺点。我们对RunningPlotter所做的图像进行了采集，而SchedulerPlotter包含了与RunningPlotter相同的调度信息，本文限于篇幅，没有对SchedulerPlotter做出的图像进行采集。对于Round-Robin算法，我们取timeSlice = 0.1

Task ID	computationTime	priority	startTime	relativeDeadline	absoluteDeadline
1	9	2	5	6	11
2	2	4	8	3	11
3	9	1	1	3	4
4	5	3	10	6	16

4.2 先进先出算法（FIRST IN FIRST OUT，FIFO）

如图13所示，红色折线代表Task 1，蓝色折线代表Task 2，绿色折线代表Task 3，黑色折线代表Task 4。当任务被调度运行时，折线处于高位。当任务在等待，或任务已经完成执行时，折线处于低位。图13展示了先进先出算法的调度结果。观察可知，task 1具有较长的computationTime，但由于其到达时间最晚，它被最后执行，导致其完成时间（finish time）远远超过了截止时间（deadline）。若小球要向前运动从而避免撞到障碍物，那么小球将会大概率地撞到障碍物。

这是因为先进先出算法没有将任务的优先级、deadline考虑在调度算法中，仅仅根据任务到达时间的先后对任务进行调度。但是该调度算法有实现简单的优点，如3.2.1节所示，我们几乎无需修改原有的`Server`模块就实现了先进先出的调度算法。由于其实现简单，对实时系统中任务的性能影响十分小。

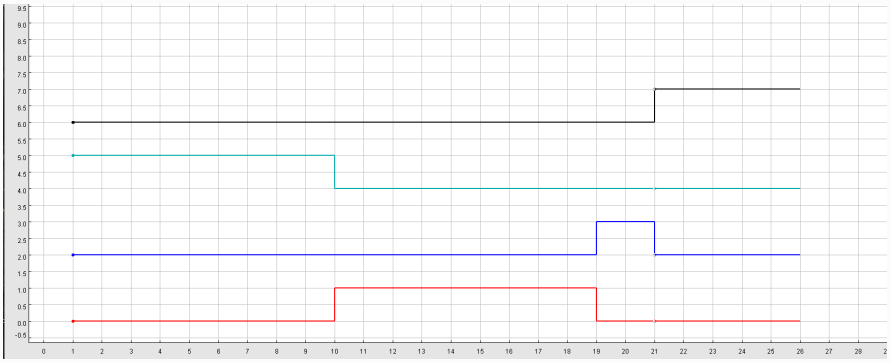

图 13 先进先出算法调度折线图

4.3 固定优先级算法（FIXED PRIORITY，FP）

折线颜色、高低含义同4.2节所述，图14展示了固定优先级算法的调度结果。在我们的系统中，Task 2具有最低的优先级，故会经常被其他任务所抢占，导致其完成时间远远超出了截止时间。然而优先级最高的Task 3在到来时就被立刻执行。假设Task 3对应的向左运用对于小球优先级最高，如在一个向左追赶小球的游戏场景，固定优先级算法能最大化的保证游戏的胜利。然而，向前的活动被经常抢占，出现了“饿死”（Starvation）现象。

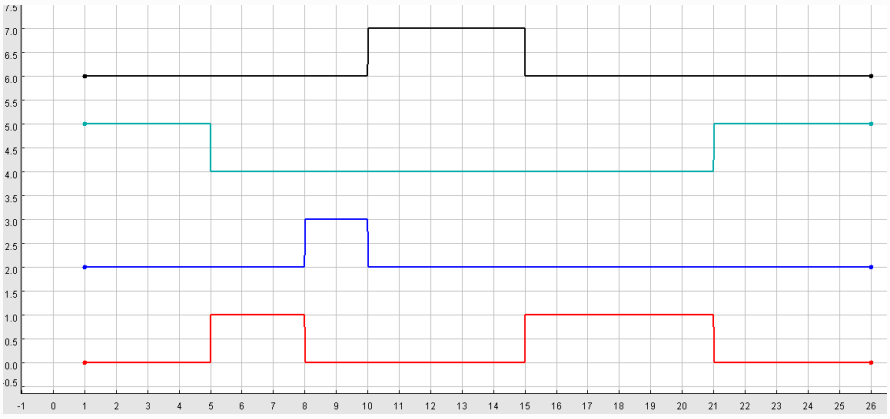

图 14 固定优先级算法调度折线图

4.4 循环调度算法 （ROUND-ROBIN）

折线颜色、高低含义同4.2节所述，图15展示了循环调度算法的调度结果。在循环调度算法中，系统出现了频繁的上下文切换。虽然所有任务均得到了公平的执行时间，但是会给优先级很高的任务，如Task 3，造成巨大的影响。考虑4.3节提到的追赶游戏场景，小球将无法逃脱追赶；而且由于频繁的上下文切换，小球变换方向所需的能量很大，这对小球是一种负担。

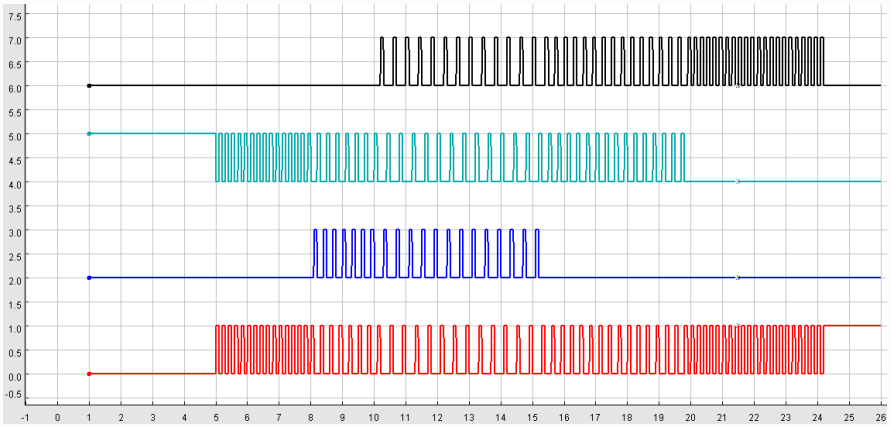

图 15 循环调度算法调度折线图

4.5 最早截止时间优先算法（EARLIEST DEADLINE FIRST，EDF）

折线颜色、高低含义同4.2节所述，图16展示了最早截止时间优先算法的调度结果。由于EDF对deadline miss有最优性，该算法保证了数量最少的deadline miss（见实验配置表格所示）。假设有这样一个游戏场景，小球必须在指定时间到达某一平面区域，到达该平面区域所需的时间是computationTime，那么到达时间的要求是absoluteDeadline。EDF调度算法保证了这类游戏的最大胜利。

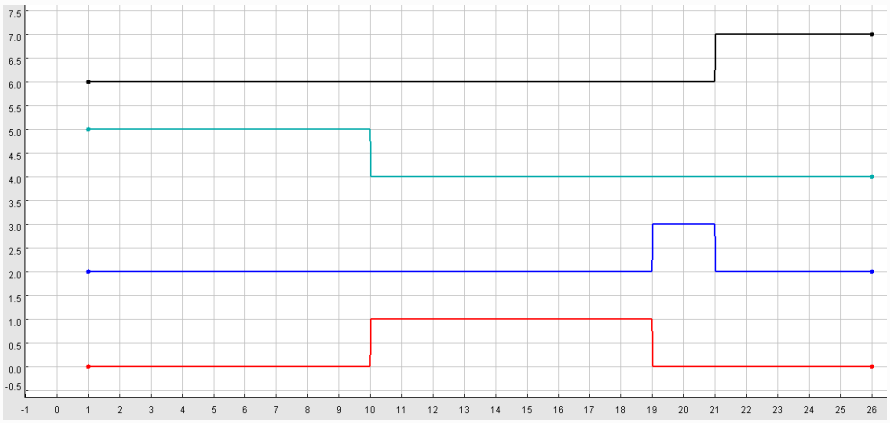

图 16 最早截止时间优先算法调度折线图

5 代码实现与运行步骤

本系统的实验代码以及实验环境已开源，Github地址为 https://github.com/snake0/ptolemy-hw1。本章对运行过程进行简要的叙述，并描述代码的实现。

5.1 运行步骤

如图17所示，只需点按绿色的运行按钮，系统就开始运行。可以看到 `RunningPlotter` 以及 `SchedulerPlotter` 中展示的折线随着时间慢慢被画出，也可观察到小球的运动。当所有任务执行完成后，点按红色的停止按钮，即可结束仿真。我们可以通过将 `GetTaskId` 的输入切换到不同的Scheduler模块，从而实现对不同调度策略的观察。

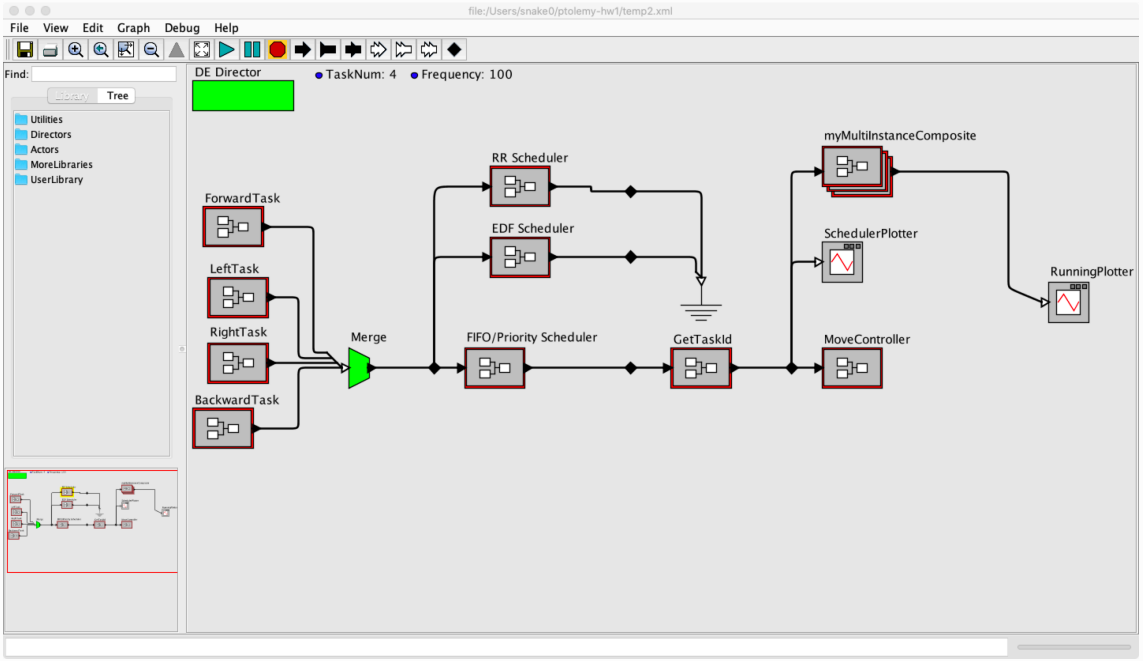

图 17 仿真系统的运行

5.2 代码结构

如下是我们在AdvancedServer.java中添加的代码。其含义是，在Server模块每次调用 `fire()` 函数时，我们检查currentJob。如果currentJob不为空，那么输出这个Job，否则输出null。这样我们设计的 `AdvancedServer` 模块即可在每个时钟周期输出其正在运行的Task。我们还给Server模块添加了trigger输入，从而接收时钟信号。

```
for (int i = 0; i < trigger.getWidth(); i++) {
    if (trigger.hasToken(i)) {
        trigger.get(i);
        currentJob = this.peekQueue();
        if (currentJob != null) {
            Token outputToken = currentJob.payload;
            currentExec.send(0, outputToken);
        } else {
            currentExec.send(0, null);
        }
    }
}
```

添加上述代码后，重新编译Ptolemy，用Ptolemy打开Github repo中的ball.xml文件，即可运行上述仿真系统。

参考文献

- [1] 2019. The Ptolemy Project. https://ptolemy.berkeley.edu/ptolemyII/index.htm
- [2] 信息物理融合系统（CPS）设计、建模与仿真 基于Ptolemy II平台 （美）爱德华·阿什福德·李编著_北京：机械工业出版社，2017.02_P374