数据流可用性：在自主系统中实现时间保证

李敖和张宁，***圣路易斯华盛顿大学***

https://www.usenix.org/conference/osdi24/presentation/li

本文收录于第 **18** 届 **USENIX** 操作系统设计与实现研讨会论文集。

2024 年 **7** 月 **10-12** 日 **-** 美国加利福尼亚州圣克拉拉

978-1-939133-40-3

第 **18** 届 **USENIX** 操作系统设计与实现研讨会论文集》的开放获取由以下机构赞助

数据流可用性：实现自主系统的时间保证

李敖　　张宁

圣路易斯华盛顿大学

摘要

由于与物理世界的持续交互，自主网络物理系统（CPS）需要同时具备功能正确性和时间正确性。尽管实时计算的理论基础取得了最新进展，但在现代 CPS 平台中有效利用这些成果往往需要领域专业知识，这给许多开发人员带来了非同小可的挑战。

为了了解构建实时软件的实际挑战，我们对 7 个具有代表性的 CPS 开源项目中的 189 个软件问题进行了调查。通过这一实践，我们发现大多数错误都是由于网络状态和物理状态之间的时间错位造成的。这启发我们抽象出三个关键的时间属性：新鲜度、一致性和稳定性。数据流可用性（Data-flow Avail- ability，DFA）旨在捕捉数据流的时间/可用性期望，我们利用这个新开发的概念，展示了如何将这些基本属性表示为数据流的时序约束。为了实现 DFA 的时序保证，我们设计并实现了 Kairos，它能自动检测和缓解违反时序约束的情况。为了检测违规行为，Kairos 将基于 API 注释的策略定义转化为运行时程序仪表。为了减少违规行为，它提供了一种基础设施，用于不同抽象层调度程序之间的语义差距，从而降低协调工作的难度。在三个

真实世界的 CPS 平台表明，Kairos 提高了时序可预测性和安全性，而运行时开销仅为 2.8%。

1　导言

人工智能和机器人技术的最新进展推动了各种自主网络物理系统融入社会，包括自动驾驶汽车[94]、无人机[31]和家庭服务机器人[32]。与普通系统不同，CPS 必须感知物理世界，计算适当的控制行动，并及时对物理世界采取行动。因此，保证

自主 CPS 的时间属性是系统正确性的基础。

实时网络物理系统的系统挑战。实时系统领域认识到其重要性，因此在确保计算的及时性方面投入了大量精力。然而，尽管有关实时计算理论基础的文献非常丰富，例如可调度性分析[70]、混合关键性调度[43]和组合调度[51, 83]，但对于非专业人员来说，在开发 CPS 软件时利用这些成果仍具有相当大的挑战性。此外，最近在多核执行和多模式传感方面取得的进展也使这一问题即使对专家来说也具有挑战性还有大量开放的研究问题正在积极研究中[66, 70]。一项重新进行的行业调查[29]（问题 23）也表明，只有一小部分（9.38%）系统在设计时使用了商业可调度性分析工具。

了解真实世界 CPS 中的定时问题。为了更好地了解 CPS 中的系统挑战，我们从最近的并发错误调查[64,67]中汲取灵感，对 7 个主流开源 CPS 软件项目中的 189 个定时错误进行了系统研究。我们的目标是了解 CPS 应用程序中的定时错误类别、每种错误类别的根本原因以及开发人员在预防这些错误时面临的挑战。我们发现，大多数定时错误都是由网络状态和物理状态之间的时间错位造成的。因此，在网络-物理控制回路抽象的基础上，我们提取了三个最基本的时间属性：新鲜度、一致性和稳定性。此外，我们还发现许多现有的缓解措施都是对数据时间戳进行人工检查，这促使我们从数据流的角度对问题进行建模。

我们的解决方案--数据流可用性。受定时错误研究结果的启发，我们提出了数据流可用性这一可在自主系统中实现定时保证的新概念。根据对现代 CPS 中数据流驱动网络物理控制回路的观察，我们认为

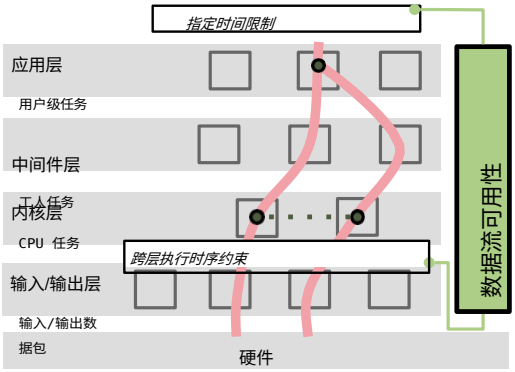


图 1：系统堆栈中的数据流可用性。

通过新的时间维度来增强数据流，这就是*定时数据流图*（TDFG）。从概念上讲，每个变量（捕捉网络或物理状态）都有一个时间属性（标签），它们之间的信息流尊重软件的期望。因此，时间策略被编码为图边上的时间约束。

为了在系统中实现数据流可用性的概念，我们设计并开发了一个编程模型 Kairos，用于自动检测和缓解违反时序约束的情况。Kairos 由一个用于在运行时检测违反时序约束的 DFA 嵌入工具和一个用于缓解的跨层调度系统组成（如图 1 所示）。使用 DFA 嵌入工具，开发人员使用 Kairos 提供的应用程序接口手动注释源代码，或使用提供的动态剖析器指定所检测到的时序属性。然后，编译器扩展会将时序预期以数据流约束的形式表示出来，并自动对软件进行检测，以便在运行时检测违反时序约束的情况。然而，仅仅检测并不能提供时序保证。一旦出现违反时序约束的情况，就必须采取措施恢复系统。为此，Kairos 基于可调度实体路径的概念，为网络物理数据流构建了操作系统不同抽象层中可调度实体的关联。这就弥合了抽象层之间的语义鸿沟，使系统中的调度器之间能够更有效地协调，以减少违规行为。

原型和评估。为了了解 DFA 在缓解时序错误方面的有效性，我们分析研究了如何利用 Kairos 实现现有的错误修复发现在 189 个错误中，有 111 个可以通过 Kairos 缓解。为了了解 Kairos 的性能特点，我们构建了 Kairos 的原型，并在三个真实世界的机器人平台上对其进行了评估：Autoware [36]、Jackal UGV [59] 和 Turtlebot3 [89]，每个平台都有不同的工作负载和计算能力要求。在这三个平台上，我们展示了如何在 Kairos 中构建和使用 TDFG，以便在运行时，Kairos 的平均开销.8%，在可扩展性分析中显示出了可控的性能。运行时，Kairos 的平均开销为 2.8%，在可扩展性分析中表现出了可控的性能。在系统高度过载的情况下，与其他先进系统（ROS [80]、ERDOS [55] 和 ghOSt [57]）相比，Kairos 在对时序违规做出反应方面的响应时间更快、更稳定。此外，端到端评估表明，Kairos 可以在系统超负荷的情况下提高安全性。

贡献。我们做出了以下贡献[1]：

- 数据流可用性是一个从数据流角度实现时序保证的新概念。
- 设计并实现 DFA 的概念验证--Kairos。Kairos 通过在应用程序中嵌入时序属性监控器来检测违反时序的情况，并通过跨层调度基础设施来缓解这些违反时序情况。
- 在三个真实世界的机器人平台上对 DFA 和 Kairos 进行评估，每个平台都有不同的工作负载和操作领域。

2 背景介绍

实时网络物理系统。自主网络物理系统的一个独特特征是与物理世界的过程紧密相连。网络物理系统软件通常建立在网络物理控制回路的抽象之上，该回路不断感知物理世界，计算适当的控制行动，然后对系统进行操作，以达到所需的状态。这一控制环的实现通常使用多个任务（过程），其中每个任务在实时模型中被建模为周期性任务或零星任务。

网络物理系统中的时间性抽象。由于其网络物理特性，自主系统的正确性取决于功能正确性和时间正确性。为此，需要根据实时任务模型对每个系统进行实时可调度性分析 [77]。满足最后期限通常被认为是实时系统中最重要的要求。利用可调度性分析得出的任务参数，系统的调度器会强制执行任务间的时间隔离，确保没有任务错过最后期限。根据容忍错过截止日期的能力，系统可以是硬实时、稳固实时或软实时。由于各种实际挑战，如确定实时任务模型难度、实现系统保证的处理器效率以及最坏情况执行时间的准确估计，许多已部署的实时系统都是软实时系统。

[1] 源代码以及包含更多分析和实验的扩展版可在 https://dataflow-availability.github.io/ 获取。

行业调查 [29]。此外，时间限制可以
除最后期限错过外，还表现在其他方面，包括
但不限于任务响应时间、执行时间、释放
抖动和响应抖动。

网络物理系统中的及时性实现。现代自主系统通常涉及多个抽象层，如图 1 所示。除了典型的用户空间层和内核空间层，现代 CPS 软件还利用中间件（如机器人操作系统 (ROS) [68, 80]）来简化编程。有些 CPS 软件甚至在应用程序中实现了自己的用户空间调度程序，从而实现了跨多层抽象的时间管理。这为开发人员实现网络事件与物理世界事件的一致性带来了独特的挑战。此外，还经常出现时间驱动[33]或事件驱动[80]任务的组合。

3 真实世界定时错误研究

动机实时理论建议将单个计算建模为单个任务。然而，为现代复杂数据驱动的 CPS 开发实时任务模型对于非专业人员来说可能相当具有挑战性。此外，高效任务模型的公式化通常需要实时调度方面的深厚专业知识。最近的一项行业调查[29]（问题 23）也表明，只有一小部分（9.38%）系统在设计时使用了商业可调度性分析工具。现有的并发错误研究[64, 67]为业界提供了重要启示，受此启发，我们对 7 个开源机器人软件中的时序错误进行了系统研究。目的是更好地了解开发人员面临基本实际挑战。因此，研究的重点是*定时*错误，即由网络物理系统内数据流的非确定性定时引起的错误。

方法。所选的七个开源 GitHub 机器人软件项目是 Autoware [53]、MoveIt [2]、Google Cartographer [56]、百度 Apollo [37]、ORB- SLAM2/3 [3,4]、ROS Navigation [5] 和 ROS2 rcl [7]。之所以选择这些项目，是因为它们代表了现代网络物理控制回路中的重要子系统，包括感知、定位、规划和控制。此外，它们也已被广泛采用[1, 35, 52, 75]。为了收集漏洞，我们使用了一组关键字（如 "timing"、"sched"、"times- tamp"、temporal "等）来筛选问题，最终得到了一份包含 189 个漏洞的清单。

系统化总结。如表 1 所示，我们发现在收集到的时序错误中，两类根本原因占了大多数（189 个中的 169 个）：时序约束的规范和执行不足。其余为设计缺陷和硬件问题。

表 1：真实世界应用中的定时错误

项目	错误数量	定时约束规范		定时约束执行			其他
		失踪	错误规格	缺失	假的		
		制约因素	可表达性参数	制约因素	执法		
制图员 [47]	34	14	12	1	1	4	2
阿波罗[37]	49	11	23	2	3	0	10
移动[2]	23	4	7	2	2	5	3
ORB-SLAM [4]	6	1	1	0	1	2	1
Autoware [53]	16	6	5	0	1	0	4
导航 [5]	15	3	3	1	2	4	2
ROS rcl [7]	46	3	3	1	3	35	1
总计	189	42	54	7	13	50	23
工作范围		✓	✓	✓	✓		

3.1 定时规范错误

时序保证失败的最常见原因是时序约束规范不完整（103/189 个错误）。如前文所述，虽然实时理论为时序保证行为提供了坚实的基础，但对于不具备实时计算专业知识的开发人员来说，将理论转化为实践仍存在差距。由于没有实时理论工具（如可调度性分析）提供的正式保证，开发人员目前为缓解这一问题而采用的做法是，开发人员在创建或传输数据时为数据标记时间戳，然后在使用数据使用这些时间戳来检查数据的有效性（如新鲜度）。这种以数据为中心的时间戳检查方法在我们调查的代码库中随处可见。，Autoware 和 Google Cartog- rapher 分别在 340 多处和 110 多处使用时间戳检查来确定执行逻辑。此外，最先进的中间件，如 ROS [80]、ROS2 [68] 和 ERDOS [55]，也在任务间传输的数据中内置了时间戳。

3.1.1 缺失时间限制（检查内容）

由于来自不同任务的数据之间存在复杂的依赖关系，要想知道在哪里以及如何手动添加时间戳检查相当具有挑战性[8,12,14,17,19-21,23,50,54]。天真地认为，我们可以简单地在所有指令上添加定时检查。然而，这会给系统带来过高的开销，导致不良的物理结果。

意义 - 不仅要了解哪些程序语句需要检查，还要了解时间属性的哪些方面验证，这样最大限度地减少保护对性能的影响。

为了进一步深入探讨问题的根本原因，我们回到网络物理控制回路的基本抽象概念，提出这些时序错误违反了哪些属性的问题。通过物理世界影响的视角，我们在分析时序违规时发现了三个关键属性。

新鲜度--*描述的是物理现象发生与网络代表消费之间的延迟时间。*

重新显示。虽然数据应尽可能新鲜，但由于传感和计算的原因，总会有一些延迟。

关键是要确保特定数据的新鲜度控制执行所能接受的。图 2 显示了 Cartographer-Pull-153 中的一个示例。制图师 [47] 使用队列以协调、有时间顺序的方式管理和处理来自多个来源的传感器数据流。然后，它利用这些数据构建机器人的轨迹，进行定位。代码片段会检查输入的数据是否早于当前轨迹的开始时间，如果是，则会丢弃过时的数据。

```
1  void OrderedMultiQueue::Dispatch() {
2    // 我们查看下一个数据之后的时间。如果是
3    // 没有超出 "common_start_time"，我们将其删除
3    std::unique_ptr<Data> next_data= next_q->queue.Pop();
5      last_dispatched_time_= next_data->time;
6
7      next->callback(std::move(next_data));
7    }
8    // 否则： 删除数据
```

图 2：过时数据被删除的新鲜度检查。来自 Cartographer-Pull-153 的简化代码片段。

一致性--*描述数据流中的物理世界观测数据在时间上的一致性，这些观测数据汇聚在程序的特定语句中。理想情况下，这些网络状态捕捉到的物理事件应尽可能同步。*

图 3 显示了 ROS 导航[5]（Navigation-Pull-1121）中的一个示例，其中控制任务使用 `tf_` 缓冲区检索机器人姿态，该缓冲区保存了历史姿态。在原始代码中（第 1 行用红色标出），它直接使用最新姿势。但是，由于 `tf_` 缓冲区是由其他任务动态更新的，控制任务使用的当前地图的时间戳（时间）可能比 `tf_` 中的最新姿势要早。这可能导致使用的姿势在时间上早于当前映射，从而使运动规划产生不正确的路径。为了解决这个问题，绿色突出显示的代码（第 3 行）采用了基于时间戳检查方法，将 `tf_` 的时间戳与控制任务的时间戳进行比较。如果时间不比 `tf_` 中的最新时间新，就会调用 `lookupTransform()` 函数来插入与当前地图在时间上一致的姿势。

```
1 tf_.transform (robot_pose、global_pose、global_frame) ;
2 // 检查 curr_time 是否小于 tf_ 的最新更新时间。
3 如果 (tf_.canTransform(global_frame, robot_base_frame, curr_time)) {
4   // 如果是，则在当前时间的时间点进行转换
5   transform= tf_.lookupTransform (global_frame、
6                                   机器人基本框架、
7                                   current_time) ;
8   tf2::doTransform(robot_pose, global_pose, transform);
9 否则 {
10  // 使用最新版本，否则
11  tf_.transform(robot_pose, global_pose, global_frame_);
12 }
```

图 3：检测两个任务数据时间一致性的一致性检查（导航-拉动-1121）。

稳定性 - *描述新鲜度的变化。这与实时和控制领域的抖动概念类似，理想情况下应尽量减少抖动。*

许多控制算法和系统在设计时，不仅隐含着对新鲜度边界的假设，而且也隐含着对不同循环之间新鲜度变化（通常相对较小）的假设。从本质上讲，这涉及到数据流在时间维度上与空间维度上的一致性（如上所述的一致性）。图 4 显示了 AutowareAuto-Pull-980 的代码片段，其中添加了一个计时器，以确保控制输出的稳定性。定时器在轮询循环中检查经过的时间，以便在预期时间间隔内触发控制输出功能。

```
1  NERaptorInterface::NERaptorInterface(. . ){
2    /* 使用 ROS 定时器确保稳定性 */
3
4    m_timer= node.create_wall_timer(m_pub_period、                this));
                                       std::bind(&cmdCallback、
5   }
6
7  /* 在执行 ROS 定时器时 */
8  while (rclcpp::ok()) {
9  // 通过轮询循环，使用消除时间来检查计时器是否准备就绪
10   rcl_timer_get_time_until_next_call(m_timer, &time_until_next_call);
11   if (time_until_next_call<= 0)      m_timer->call();
12 }
```

图 4：使用 ROS 定时器进行稳定性检查，以尽量减少控制抖动（AutowareAuto-Pull-980）。

小结 - 这三个关键属性提供了一个独特的机会，只需少量的时间属性检查就能解决大量错误。

3.1.2 时间限制不足（如何检查）

即使解决了 "检查什么 "的难题，开发人员还必须解决 "如何检查 "的难题。有 61 个错误是由于时序约束说明不充分造成的；在这些错误中，我们发现了两个常见的原因。第一类是一些硬编码的时间限制可能不适合部署。这通常是由于测试不足或系统的软硬件[11]或运行环境[55]发生了变化。第二类情况就不那么简单了。在实时网络物理系统中，除了延迟（与前面讨论的新鲜度相对应）之外，还有其他重要的时序维度，例如对齐（与讨论的一致性相对应）和抖动（与前面讨论的稳定性相对应）。例如，在到达抖动[10]、数据丢失检测[24]、处理数据时间比[13, 27]和开发定时工具的要求[9, 16]等方面都可能出现问题。图 5 显示了谷歌制图师[47]（Cartographer-Issue-242）中的一个简化错误示例，该错误在最终修复之前跨越了多个补丁。代码片段通过将相邻两帧之间的位置差除以时间间隔来估算机器人的速度。在这种情况下，数据的新鲜度是一个问题，因为如果传入的激光雷达帧比最新的帧要早（即失序），就会导致时间差（`delta_t`）为负。补丁[79]中增加了新鲜度检查（图中第 3 行用黄色标出）。然而，即使 `delta_t` 为正值，数据新鲜度之外的其他问题依然存在。不规则的时间可能会导致两个激光雷达帧在时间上过于接近，从而导致 `delta_t` 过大。

短。在这种情况下，位置差会被一个很小的 `delta_t` 值除以，这会显著放大任何估计误差，可能导致速度大。通过插入检查（第 8 行），最终解决了这一问题，即丢弃间隔小于 1 毫秒的帧。

```
1 // 估算速度估计值。
2 如果 (time> common::Time::min()
3     && time> last_scan_match_time) {
4
5   // 防止数据失序
6 double delta_t=  common::ToSeconds(time - last_scan_match_time); 7
8   if (delta_t< 1e-6) return;
9  // 防止间隔时间过短
10  velocity_estimate_+=  (pose_estimate_.translation() -) 。
11                        model_prediction.translation()) /
12                        delta_t;
13 }
```

图 5：时间戳检查在语义上不完整。

意义--考虑到不同平台和物理环境中时序预期的动态范围，开发一种机制来简化开发人员对这些范围的配置非常重要。理想情况下，这种机制还能自动发现必要的范围，以维护系统安全。

3.2 定时执行错误

有 62 个错误源于时序约束执行不力。其中大部分（50/62）是由于软件本身的缺陷造成的，如调度器和定时器等执行基础设施的内存损坏。另外 13 个错误是由于时序约束没有被传送到执行机制造成的。造成这些问题的主要原因是，指定的定时约束仅限于用户空间应用程序，而不会传播到其他调度层。图 1 以示意图的形式展示了在设计和部署通用系统时所涉及的调度层。由于对跨调度器传递调度上下文的支持不足，一个调度器指定的时序约束（或调度决策）无法传播到其他调度器。这类问题可表现为优先级跨层颠倒[22]，导致关键任务无法可靠触发[26]、在不同时间段执行[25]或执行顺序混乱[15]。因此，仅在用户层或中间件上保持相对优先级的缓解方法，即使不是不可能，也往往相当具有挑战性。

```
1  void SchedulerChoreography::CreateProcessor() {
2    proc-> BindContext(ctx);
3    /* 为任务预留一组 CPU 内核 */
3    SetSchedAffinity(proc->Thread(), pool_cpuset_, pool_affinity_, i);
4    SetSchedPolicy(proc->Thread(), pool_processor_policy_、
5                   pool_processor_prio_, proc-> Tid()) ;
6 }
```

图 6：缓解跨调度层的断开连接。图 6 是处理 Apollo-Issue- 的 9433.它引入了一种新的调度策略，为中间件任务保留了一组 CPU 内核，使它们可以直接调度到这些内核上，避免了层与层之间的断开。然而，要实施这种调度策略，就必须全面了解各项任务，包括它们之间的依赖关系和执行时间。

机制。

含义--当调度上下文在所有抽象层中都可见时，对时序预期的保证会更有效。

3.3 摘要

表 1 显示了我们研究的定时错误，以及拟议机制 DFA 的范围。根据研究结果，我们总结了 DFA 设计的机遇和启示：

- 程序员通常会通过检查数据的年龄来增加对时间的预期，这暗示了使用信息（数据）流作为捕捉程序员意图的机制的可能性。
- 基于网络物理控制回路抽象，我们系统化了三类关键时间属性，即新鲜度、一致性和稳定性。
- 操作系统中不同抽象层之间的可视性往往有利于定时执行。

4 数据流可用性

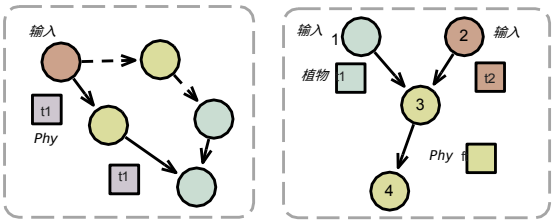
受第 3 节所述挑战的启发，本文介绍了***数据流可用性***（DFA），它从数据流的角度来定义时态属性的策略。

4.1 定时数据流图表

定时数据流图（TDFG）是由 DFA 程序的数据流图扩展而成的程序表示法。

图定义。TDFG 是一个有向图 **G**= (**V**, **E**, **T**, **C**)
根据程序的中间表示构建：

- *顶点*集合 **V** 中的每个顶点v 都对应中间表述中的一个语句。
- *边：* 边 **E** **V**⊆ × **V** 表示顶点之间的数据依赖关系。如果相应的状态元素之间存在数据依赖关系，就会添加一条边。
- *时间标签：* 时间戳t_{py}⊆**T** 由图中的内存 SSA（静态单赋值）生成，并在运行时沿边传播。它包括两种类型的定时，一种是物理世界的传感器读数，另一种是定时传感器值的导出值范围。



(a) 顺序传播。 (b) 合并不同的输入。

图 7：数据流中时序传播的两种情况。

```
1 void EvaluatorManager::DumpCurrentFrameEnv() {
2   FrameEnv curr_frame;
3   auto obstacles= ContainerManager()->Get(PERCEPTION_OBSTACLES);
4    curr_frame.set_timestamp(obstacles-> timestamp());
5 }
```

(a) 时间戳在单个数据流中传播（Apollo-Pull-8503）。

```
1 bool Fusion::GenerateMsg(Obstacles* obstacles) {
2   common::Header * header;
3   header->set_lidar_stamp(lidar_timestamp * 1e9);
4   header->set_camera_stamp(camera_timestamp * 1e9);
5   header->set_radar_stamp(radar_timestamp * 1e9);
```

(b) 多个数据流的时间戳合并（Apollo-Pull-5459）。

图 8：数据流中时序传播的代码示例。

- *时序约束：* 可为边分配定时约束 $C \in \mathbf{C}$。这些约束定义了边在指定容差阈值内应满足的时间条件。这些约束根据信息流进行评估。这些时间属性由 DFA 的度量标准定义，详见第 4.2 节。需要注意的是，某些时间属性需要分析进入顶点的数据流随时间/迭代的统计量。

定时信息传播。 定时标签可在运行时沿着边E网络物理系统中常见的时序标签传播模式有两种（如图 7 所示）：

- *在单个数据流中传播定时标记。* 时序信息沿着单个数据流传播（图 7(a)），其中边沿继承了前一个边沿的时序标签，除非数据流来自新的传感器读数。由于时序标签代表的是物理世界进行观察的时间，因此，数据流中的

网络空间不会改变标签。这是最

一个案例。在实际操作中，开发人员会在程序中添加

根据从前置任务接收到的数据，为变量添加时间戳。图 8(a) 展示了百度 Apollo 自动驾驶汽车项目（Apollo-Pull-8503）中的一个示例。预测任务从物体检测任务中继承了障碍物的时间戳。然后，该时间戳被用于计算当前感知环境的数据年龄，并以此为基础进行预测。

- *从多个数据流中合并定时标记。* 此类别

图 7(b)）涉及在一个顶点合并多个数据流。这通常是融合来自不同传感器的信息的任务所需要的。在这种情况下，语句所产生的内存 SSA 继承了来自不同传感器的时间戳。

其传入边，并保持$t_{p\text{栋}y} = f_v(t^1 \quad , t^n \quad)$、
其中f_v是顶点v的合并函数。虽然图中只显示了两个数据流，但可能不止两个。请注意，在如何合并时序标签问题上有一个放之四海而皆准的解决方案，因为这实际上是在合并

从不同的时间位置对物理世界进行观测。一种常见的方法是保留时间标签范围。图 8(b) 中的代码片段描述的是 Apollo-Pull-5459 Apollo 中添加的 融合任务。由于该任务融合了激光雷达、摄像头和雷达的探测结果，还包含了它们的时间戳，以便稍后检查时间对齐情况。

4.2 TDFG 的时间限制

在时序错误研究的基础上，我们根据网络物理控制回路抽象概念提出了三个基本的时序正确性：新鲜度、一致性和稳定性。下面，我们将展示如何使用 TDFG 以*时序正确性的*形式来捕捉它们。

*新鲜度*关注的是进行物理观测的时间与控制系统使用该观测数据的时间之间差异。在网络物理系统中，这一时间差通常必须有一定的界限，因为任何延迟都会加大网络世界与物理世界之间的时间差，这一点在第 3 节中已经讨论过。因此，给定一条边e ，其最大可容忍时间阈值为θ_f，其新鲜度的计算公式为

$$C_f = \theta_f \quad -(t_{-} - t_{\text{phy}}) \tag{1}$$

其中t_{-}是当前时间。

*一致性*涉及到
不同数据流中的时间标记会汇集到一个顶点，直观地显示不同时间物理世界状态的差异。一般来说，顶点越小，时间标记越接近，物理世界的观测结果就越一致。考虑具有相同出口顶点的n 边：$T_{\text{def}} = \langle t^1, \ldots, t^n \rangle$。它们的时间一致性可以通过以下方法检查：

$$C_c = \theta_c - \max_{ij \le n}(t^i \quad \dot{t} \quad) \tag{2}$$
捍卫 捍卫 植物 植物

其中θ_c是可容忍阈值（或范围）。

*稳定性*捕捉了
数据按时间流入/流出顶点。实时系统中的许多任务都是以周期性工作负载的形式实现的，因此一些基础算法/模型在设计时假定了周期性，这就要求数据使用具有周期性，如输入（如传感器输入）或输出数据（如执行命令）[71]。对于属于w

的边

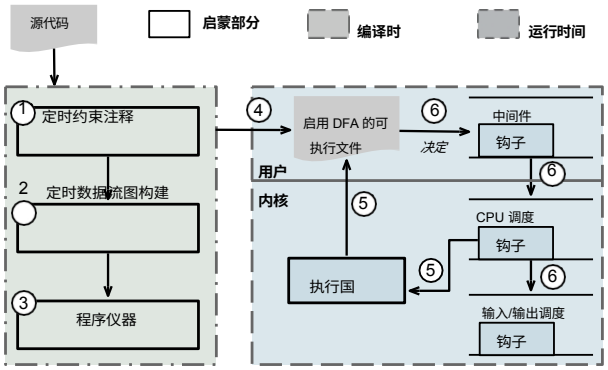


图 9：Kairos 的工作流程。

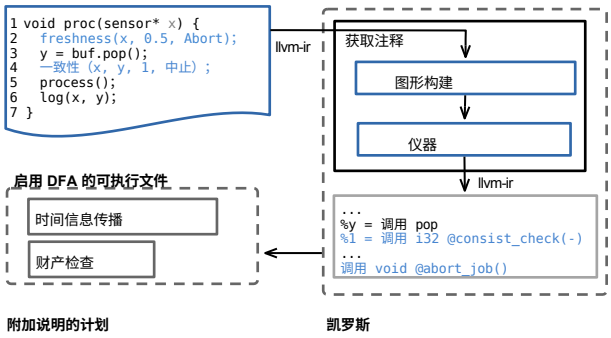


图 10：支持 DFA 的应用程序构建流程。

通过同一程序点从顺序循环中流出的一组数据流，它们有$T=\langle t^1,\ldots,t^w\rangle$。检查稳定性的一种典型方法是测量抖动：

$$C_s=\Theta_s-\max_{i,j\le w-1}|D_i-D_j^{物},\ D_i=\Delta_i^{物}I,\ \Delta_i=t^{i+1}-t^i \quad (3)$$

植物　植物

其中，Δ 代表两次迭代之间的间隔，I是预期间隔。在实践中，稳定性的形式可以根据目标系统的设计而变化，其替代方案可能是新鲜度的变化。

一条边e 在程序执行到其出口顶点时进行评估，如果所有附加指标都满足$C>0$，即 $(C_f>0\wedge C_c>0\wedge C_s>0)$，则认为该边符合时间正确性。

5　启明星的设计与实施

Kairos 是数据流可用性的概念验证。它由两个主要部分组成：使用 TDFG 的时态策略定义和策略违规缓解。Kairos 由编译器扩展和运行时系统组成。图 9 概述了其主要组件和工作流程。在编译时，Kairos 利用程序分析和用户注释/自动注释 1（来自剖析）来构建目标应用的 TDFG 2；利用 TDFG，Kairos 利用代码来执行时序信息传播 3；在运行时，任务更新时序信息并评估时序正确性 4；一旦违反时序约束，任务会触发一个处理程序来执行预定义策略 5；然后，处理策略的调度决策会与不同层的调度程序共享 。

5.1　支持 DFA 的应用程序

如图 10 所示，构建 DFA 应用程序有两个主要步骤，一是构建 TDFG（定义应用程序必须遵循的时间属性），二是对应用程序进行工具化，以便检测和减少违反属性的情况。

TDFG 建设。TDFG 捕捉了开发人员预期的时间属性。在提取值流图（value-flow graph）[49, 86]后，TDFG 中的时序约束可通过开发人员注释手动表达，或通过动态剖析自动表达。

表 2：启明星应用程序接口

功能名称	论据　研究				说明
	目标	宽容	窗户	处理政策	
新鲜感	变异	门槛	-	中止	检查预期属性。
一致性	var, ...		-	优先跳过	如果违反，则触发处理策略功能。
稳定性	变异		尺寸	下一个	

为便于手动注释，我们提供了三个应用程序接口，用于注释源代码，以表达对第 3 节中讨论的三个关键属性（新鲜度、一致性和稳定性）的时序约束。如表 2 所示，这些函数接受四类参数：目标变量、容限阈值、窗口大小（仅适用于稳定性）和处理策略。阈值和窗口大小参数用于检查时序正确性。处理策略参数指定了时序正确性检查失败时要调用的函数。

不过，手动注释通常不仅需要物理系统的丰富领域知识，还需要通信栈的丰富领域知识，而这些知识可能并不总是可用的。为了解决这个问题，Kairos 还提供了一个选项，利用动态分析的性能曲线来提取时序一致性。，Kairos 需要两个关键组件：第一，确定软件时序行为是否需要修正的神谕（crite- ria）；第二，系统仪器输入，以便观察所有潜在行为。在甲骨文方面，Kairos 借鉴了 CPS 评估中的现有做法，即使用安全性（通常以控制状态偏差来衡量）作为衡量标准。当物理安全（如车辆撞上行人或无人机从天而降）因违反特定的时间属性而受到损害时，Kairos 就会认为该时间属性至关重要，必须在运行时进行监控和检查。测试系统的系统输入（即物理场景）是 CPS 测试中的一个公开挑战[72]。在 Kairos 中，除了

除了依靠用户提供可能揭示违反时序特性的场景外，我们还使用性能互证工具[65]来探测系统在不同时序下可能受到的影响。为了尽量减少剖析系统对软件时序行为的影响，我们使用了硬件性能监控器和调试功能。在所有导致违反相同属性的流程中，Kairos 只对首次出现的流程进行检查。值得注意的是，动态剖析寻找可接受的约束范围方面要比寻找在何处添加约束更有效（后者的搜索空间要大得多）。

嵌入 TDFG。在构建 TDFG 之前，Kairos 会对源代码进行分析，以识别接收传感器输入的语句，并自动对其进行检测，以从传感或输入有效载荷中提取时间戳t_{py}。为了开始构建 TDFG，Kairos 利用 LLVM-IR [63] 在 SVF [86] 工具的值流分析基础上进行构建，然后使用一组 python 脚本添加时序约束和注释。此外，还开发了一套 LLVM 编译器传递程序，以检测时序信息传播和检查所需的代码。Kairos 还利用几种启发式方法来降低性能开销。首先，为避免对每条指令进行定时元数据传播，Kairos 会自动绕过具有相同时间标签$t_{p\text{惊}y}$（传感器时间戳）的定时数据流。在不失一般性的前提下，选择一个顶点进入 TDFG 有三个标准：*(i)* 它是物理输入顶点或物理输出顶点；*(ii)* 它合并了多个数据流；或 *(iii)* 被注释为具有时序约束的相关顶点。

5.2 时间限制违规缓解

时序约束违规处理策略。缓解违反时序约束的情况通常需要考虑物理组件，没有放之四海而皆准的解决方案。Kairos 从我们的漏洞研究、实时计算领域的先前研究成果 [39, 46, 69, 91] 以及当前的工业实践 [29] 中汲取灵感，提供了三种策略：*中止*、*优先处理*和*跳过下一个*。更具体地说，*"中止"*会丢弃违反定时约束的任务。*"优先"会将*系统切换到不同的任务模型集，通常包括提高任务的优先级。最后，*skip-next* 允许延迟的任务继续执行，但跳过其下一个实例以恢复。值得注意的是，这些策略可能会在级联效应中导致更多违反时间限制的行为。例如，优先处理已错过截止日期的任务可能会阻止其他任务取得进展，从而导致后续任务错过截止日期。不过，如果这些策略组合得当，就能支持现有的自适应实时调度范例，例如弹性调度 [4647] 和混合调度 [] 。临界调度 [90]。

在*弹性调度*下，任务利用率会降低（通常是通过增加调用任务的周期）。

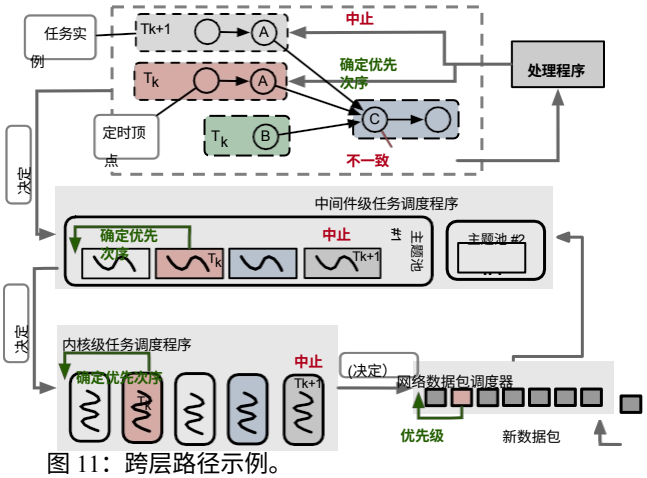


图 11：跨层路径示例。

以避免错过截止日期。弹性调度最初是在文献[46]中作为一种适应系统过载的机制而提出的，后来发展成为一种系统适应意外的任务执行时间过长[45]或其他任务干扰[85]的手段。针对违反时间限制的情况，Kairos 可以使用 [84] 中的算法快速重新计算任务周期，然后通过多重*优先*策略来执行，从而相应地改变任务优先级或 `SCHED_DEADLINE` 属性。

在*混合临界系统*中，非关键任务实例可能会因关键任务的时间异常而被放弃。具有虚拟截止期的最早截止期优先（EDF）调度（EDF-VD）是一种针对非千里眼混合临界系统（即无法事先预测时序异常，只能在异常发生时才能确定的系统）的最优调度算法[39]。在 EDF-VD 下，每项关键任务都根据其虚拟截止时间确定优先级，虚拟截止时间被分配为一个常量参数。当关键任务超过其预期执行时间时，非关键任务实例会被丢弃，以保持对关键任务的保证，而关键任务会根据其绝对截止时间重新排序 [39]。Kairos 的处理程序通过*中止*策略和*优先级*策略（分别应用于非关键任务和关键任务）的组合来支持这种模式切换。虽然开发更复杂的策略带来了令人感兴趣的研究机会，但这仍有待未来的探索。

*执行。*处理程序可以在单个层或多个层中实现。在我们的原型中，我们将其作为对内核调度程序的修改来实现，在内核调度程序之前调用缓解机制，以进行概念验证。对于任务中止，我们的原型会编写代码以实现提前返回。不过，需要注意的是，资源去分配和不一致状态消除通常需要复杂的管理[82]。在中间件中，下一个任务调用信息被丢弃在 ROS 中，从而演示了跳过下一个任务的功能。

跨层调度协会。虽然处理
违反时序约束的策略相对来说还不够完善。
在实时任务问题的表述下，对"……

任务层。现有的软件生态系统在不同的架构抽象层都有可编排的实体

从 I/O 层（如网络数据包）和操作系统

从抽象层（实时进程）到中间件（ROS 组件），再到应用程序（特定于应用程序的调度器）。由于抽象层之间语义的缺失，这给实现处理程序策略的一致性带来了非同小可的系统挑战。

我们从 13 个错误 [18, 38] 中发现了这个问题。

先前的错误研究。，在处理一致性违规时，往往需要调整传感器处理的线程。

而不是融合过程，因此有必要将
TDFG 的子图到相应的可调度实体
这样，时序约束违规处理程序就知道该对哪一个进行干预 [41]。另一个常见于对时间敏感的网络中的例子是，需要优先处理以下问题

由于任务优先级的调整，网络队列中的特定项目会发生变化 [95]，这可能是处理流程的一部分。

为了缓解这一问题，我们建议通过将可调度实体与 TDFG 中的数据流关联起来来弥合语义鸿沟。这不仅能让处理程序知道要操作的可调度实体，还能让其他抽象层更有效地响应处理机制。为了确保关联的完整性，Kairos 从 Scout 系统[73] 的 Path 概念中汲取了灵感，Path 用于跟踪网络设备系统中数据包经过的组件（如网络设备或协议层）。如图 11 所示，在 Kairos 中，调度更新 Path，以反映导致应用程序沿特定路径执行可调度实体链。

*实施。*我们的原型修改了本地调度实体的数据结构，以存储它们所属的路径。该信息在用户空间、内核、中间件和网络协议栈四层可访问的共享缓冲区中更新。路径增量的方法各不相同：在内核和网络协议栈中，它发生在创建新任务或数据包时；而在 ROS 中间件中，它发生在调度线程执行回调函数时。

6 评估

本节试图回答以下问题： *(i)* 什么是"……"？

表 3：评估平台

平台	软件栈	计算	内核	内存	内核
Autoware	Autoware.Auto [36, 61]	AMD 9 3900X RTX 3070 Ti	12	128GB	Linux 5.11
豺狼	制图师 [47] & Navigation [5]	英特尔 Nuc 8	4	16GB	Linux 5.11
乌龟宝宝3	导航 [5]	RPi 4B	4	4GB	RPi 5.15
微基准测试	ORB-SLAM3 [4]	Intel i9-12900K	12	128GB	Linux 5.11

表 4：错误修复能力的根本原因分析

类别		说明	数量
可修复	重构	不重构 时序信息/约束/传播不足	104
		删除内置冲突逻辑	5
		与软件语义相适应	7
无法修复	范围外	与硬件有关的定时故障	6
		与算法有关的定时错误	8
		基础设施错误（如调度程序崩溃）	41
	有限公司	并发性错误	12
		性能问题	6

终端实时自主系统。(2) Jackal UGV--一种无制导地面车辆，代表中端自主系统。它使用 Google 制图器 [47] 进行车辆定位，使用 ROS 导航 [6] 进行路径规划和控制。

(3) Turtlebot3 - 一种低端室内机器人，依靠 ROS 导航[6]进行定位、规划和控制。鉴于每个系统的软件栈都需要不同的计算能力，我们使用了与官方建议一致（或相似）的三种不同计算单元，以更好地模拟异常计时情况。表 3 列出了实验硬件设置。Autoware 和 Jackal UGV 是通过硬件在环模拟进行评估的，而 Turtlebot3 也是通过真实机器人进行评估的。

6.1 DFA 在解决现实世界错误中的应用

在评估 Kairos 的功效时，需要回答的一个关键问题是其解决时序问题的能力。为此，我们的评估利用了从错误研究中收集到的资料，分析了 Kairos 是否能够（通过 TDFG 中定义的时序策略）检测到时序问题并（使用跨层时序策略违规处理程序）缓解时序问题。由于需要使用物理系统或仿真来演练系统，因此本次评估的大部分结果都是不准确的。

Def ult Handle

第 6.1 节；(ii) DFA 解决现实世界时序错误的能力如何？- 第 6.1 节；*(ii)* Kairos 的成本和功效如何？- 第 6.2 节；*(iii)* DFA 和 Kairos 如何提高异常时序情况下的性能/安全性？- 第 63 .节

实验设置。评估是在合成工作负载和三个真实世界通用系统的工作负载上进行的：（1）Autoware.Auto [36] - 一个开源的全栈式自动驾驶项目，该项目提出了一个高

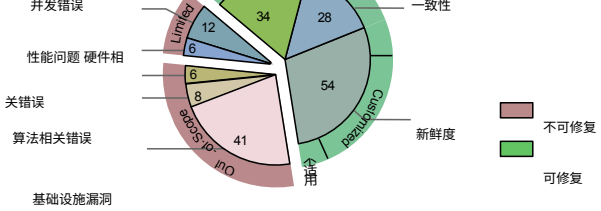
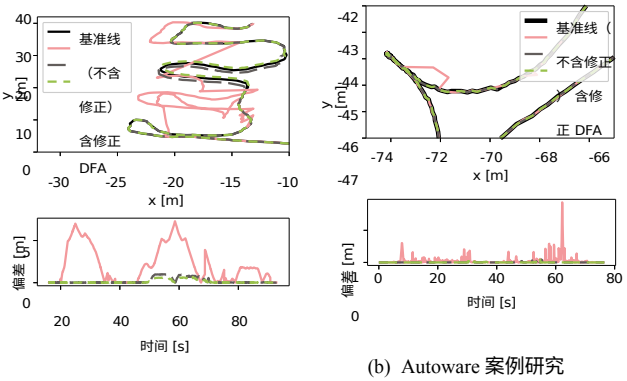


图 12：关于错误可修复性和根本原因的统计数据。

修正项目是由三位分别拥有 6 年、10 年和 17 年经验的网络物理系统开发人员通过人工检查获得的。如果这三位开发人员都认为可以用 Kairos 中的基元来表达漏洞的修复，那么该漏洞就被认为是可修复的。此外，我们还进行了两项案例研究，以证明 Kairos 可用于检测和缓解违反关键时间属性的行为。

人工检查的结果如图 12 所示。根据我们之前的案例研究，共有 189 个时序错误。其中 116 个可以通过 DFA 检测到，23 个无法检测到，因为它们是由底层基础架构错误、设计缺陷或硬件问题（超出了 Kairos 的范围）引起的。
76 个错误可以直接使用默认的缓解处理策略来缓解，而 35 个错误只能使用定制的时间违规处理程序来缓解。有 5 个错误即使使用定制的处理程序也无法缓解，因为它们需要对底层设计模型或算法进行调整，因此必须重新设计整个软件。表 4 总结了 Kairos 无法解决的所有错误的原因。为了进一步了解如何利用 Kairos 解决现实世界中的错误，我们重现了两个分别违反稳定性和一致性的错误，因为新鲜度通常更容易处理。

*案例-1：Cartogra-pher 中的激光雷达点云时序异常。*本案例研究展示了 Kairos 在识别和减少违反稳定性时序约束方面的有效性。具体来说，我们对 Cartogra- pher [56]（一款广泛使用的定位软件包）中的 Cartographer- Issue-242（如图 5 所示的代码片段）问题对 Kairos 进行了评估。根据最初的问题报告，假定以周期性间隔到达的激光雷达点云数据有时会比预期更快到达，从而违反了系统的稳定性时序约束。为了重现该问题的相同影响，我们修改了驱动程序代码，以诱发相同的异常定时模式，特别是将两个点云之间的时间控制在 1 毫秒以下。如图 13(a)所示，这种计时模式会导致车辆偏离基线最多 10.3 米。为了解决这个问题，代码库中的补丁检查了每个点的时间戳，并删除了时间间隔小于 1 毫秒的异常点。删除后，生成的定位结果与基线（偏差为 0.20 米）相当。在使用 Kairos 时，我们在 *ScanMatch()* 语句中指定了稳定性时序约束，该语句会消耗任务 *HandleLaserScanMessage* 中的变量 *msg*，该任务接收带有注释 API *stability() 的*点云。API 中的容差阈值参数设置在 22 至 33 毫秒之间，是通过对不会产生不利控制结果的时间间隔范围进行动态剖析获得的。这一过程需要 18.3 分钟。我们将*中止*作为默认策略，以减少违规行为。如图 13（a）所示，生成的定位结果与基线一致，为 0.19 米，与官方补丁相



(a) 豺狼案例研究。

(b) Autoware 案例研究

当。

*案例 2：更新位置的延迟。*本案例研究显示

图 13：使用 DFA 修复定时错误的案例研究。

我们对 Kairos 在检测和减少违反一致性时序约束方面的有效性进行了评估。具体来说，我们在 Auto- ware 中的 Autoware-Issue-458 问题上对 Kairos 进行了评估（代码片段如 14 所示）。根据最初的问题报告，旨在执行定位的优化程序（第 31 行）所使用的生成的激光雷达数据和里程表数据的时间戳应在阈值范围内，但有时会错位，从而违反了系统一致性时序约束。由于在最初提交的问题中没有讨论触发错误的机制，我们在运行定位相关任务的内核上使用 stress-ng [62]注入了 60% 级别的间歇性 CPU 过载，以引入激光雷达和里程表数据之间的不一致性。如图 13（b）所示，这种不一致性导致车辆产生的轨迹偏离地面实况 1.88 米。为了解决这个问题，代码库中的补丁用时间戳标记了激光雷达和里程测量数据。然后比较这些时间戳。如果两者之间的时间差超过 1 秒，则会丢弃结果。为了用 Kairos 解决这个问题，我们使用注释 API *consistency()*（图 14 中第 31 行）来指定 *transform_tree* 和 *msg_ptr 的*一致性时间约束。API 中的容差阈值参数设置在 1.2 秒之间，它是通过动态*计算两代变量的时间戳之差来*获得的，不会产生不利的控制结果。这一过程需要 8.5 分钟。我们使用*优先*策略作为默认策略，以减少违反时间限制的情况。生成的定位结果与基线一致，为 0.091 米与官方补丁相当。

编程实例。我们以案例-2（Autoware-Issue-458）为例，展示 Kairos 如何减少时序约束编程的工作量。图 14 显示了 Autoware 本地化组件的简化代码片段。在该组件中，传入的激光雷达点云、高清地图和变换树（根据过去的信息推断出的姿态）被联合使用，因此它们的时间戳应该是

```
01 void observation_callback(typename ObservationMsgT::ConstSharedPtr
msg_ptr){
02   // 获取新收到的激光雷达信息的时间戳
03    缁  auto observation_time= get_stamp(*msg_ptr);
04   //获取 全局变量转换树
05    缁  auto & transform_tree= xxx;
06   //获取 全局变量映射
07    缁  自动和地图= xxx;
08a   缁  auto &initial_guess= m_pose_initializer.guess(
08b                              tree, observation_time) ;

10    if (m_external_pose_available){
- 00 initial_guess.header.stamp= transform_tree.stamp;
11      initial_guess= m_external_pose;
- 12      initial_guess.header.stamp= get_stamp(*msg_ptr);}
13    // 指定时间戳
- 14    const auto message_time= msg.header.stamp;
15    // 验证时间戳（地图不应比测量值更新）
- 16    if (message_time< map.timestamp()){
- 18      return ERROR;}
20    // 指定时间戳
- 21 const auto guess_scan_diff= initial_guess.header.stamp - message_time
;
- 22    const auto stamp_tol= m_config.guess_time_tolerance();
24 // 验证时间戳（不支持向后外推法）
- 25 if (initial_guess.header.stamp< message_time){
- 26      return ERROR;}
28    // 验证时间戳
- 29    if (guess_scan_diff.count()> std::abs(stamp_tol.count())){
- 30      return ERROR;}
+31 CONSISTENCY (guess、transform_tree、THRESHOLD、PRIORITIZE) ;
+32 CONSISTENCY (map、msg_ptr、THRESHOLD、ABORT) ;
33 NDT_optimizer.solve(initial_guess, msg, map);...}
```

图 14：Autoware 中时间一致性检查的简化代码。-"（红色）代表内置检查，而 "+"（绿色）是通过 Kairos 的应用程序接口进行的检查。

检查是否对齐。标准检查机制（标有"-"的红线）要求开发人员识别数据出处、标注时间戳，并在使用前进行验证。这通常需要在不同的上下文中频繁跳转到其他函数，因此必须深入了解数据流关系，这既耗时又容易出错。相比之下，通过使用 Kairos 的应用程序接口，用户可以省略所有时间戳分配和校验，只需在使用激光雷达点云和地图前添加两条语句即可（图中两条带 "+"的绿线）。

总的来说，Kairos 从三个方面简化了有时间限制的编程。首先，它消除了在程序中为变量指定时间戳的要求。其次，它避免了不必要或重复的时间戳检查。第三，它不要求开发人员彻底理解源代码中不同数据流之间的时间关系。

6.2 凯罗斯的成本和功效

实际应用的运行时间开销。Kairos 的运行时间开销来自三个方面：时序信息传播、时序正确性检查和调度程序中的附加逻辑。我们在每个平台的五个代表性任务（或功能）上分别测量了每个方面的开销，并取 100 次运行的平均执行时间。结果如图 15 所示，包括原始时间和按比例增加的时间。

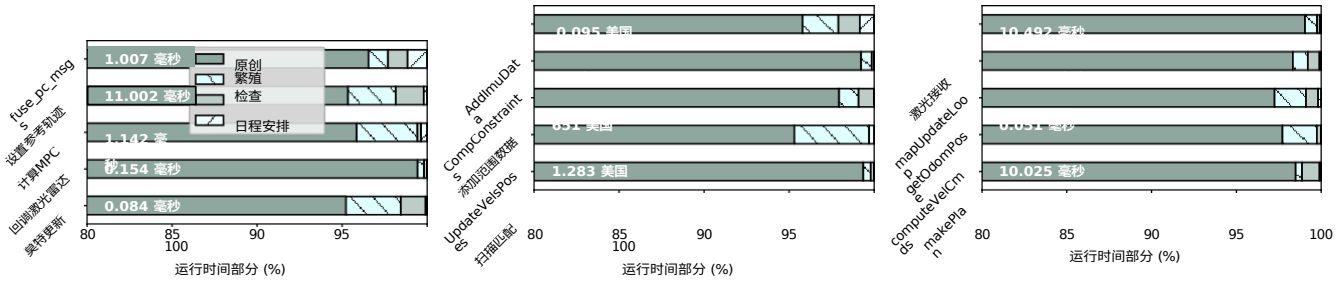
在这些任务中观察到的最大开销是 *MOTUp-*

Autoware 中的 *date*（4.77%）、Jackal UGV 中的 *UpdateVelsPoses*（4.69%）和 Turtlebot3 中的 *getOdomPose*（2.74%）。总体而言，执行时间百分比的增加与任务依赖图（TDFG）中的边数量密切相关。通常情况下，涉及更多传感器输入的任务会引入更多的边。例如，*MOTUpdate* 任务的开销百分比较高，因为它是 Autoware 中的多对象跟踪任务，可融合多个点云输入。此外，维护更多历史时序状态的任务也会产生更高的开销。Jackal UGV 中的 *AddImuData* 任务就是一个例子，它将数百个惯性数据帧存储在队列中，造成了 4.21% 的开销。细分下来，大部分开销来自沿边缘传播时序信息，最高可达 4.39%。我们发现，Kairos 在调度器上的附加逻辑带来的开销微乎其微，其中最大的开销是来自任务 *AddImuData 的* 0.84%。除了单个执行时间外，我们还测量了从传感器输入读取到执行输出的端到端延迟。Autoware、Jackal UGV 和 TurtleBot3 的端到端延迟开销分别为 3.24%、2.44% 和 2.75%。

可扩展性分析。传感器读取率、TDFG 中的边数量以及定时约束检查中的定时标签数量都会影响 Kairos 的可扩展性。因此，我们通过测量与这三个因素相关的运行时间开销来评估 Kairos 的可扩展性。我们通过修改 ORB-SLAM3 的原始工作负载，创建了具有不同可扩展性影响因素的合成工作负载。具体来说，(1) 为了模拟不同的传感器读取率，我们改变了原始 ORB-SLAM3 工作负载中记录的传感器数据的重放速度。(2) 为了调整边的数量，我们复制任务并在其共享数据流上注释定时约束。(3) 由于只有*稳定性*在多个时间戳上执行定时约束检查，因此我们调整其窗口大小，以评估定时标签大小的影响。

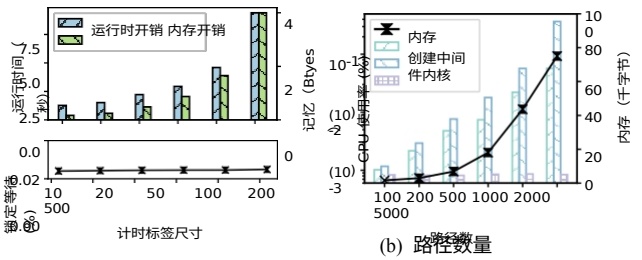
*传感器读取率。*传感器读取率会影响运行时间开销。图 16(c) 显示，随着输入频率的提高，更新和检查定时标签的执行时间呈线性增长。单个顶点的执行时间从频率为 20Hz 时的 37.537μs 增加到频率为 500Hz 时的 4087.95μs，主要原因是定时检查。更高的传感器读取速率也会显著增加锁定等待时间，从 10 Hz 到 500 Hz 增加了 100 倍。然而，即使在高输入频率下，单个内核的 CPU 使用率仍然只有 0.23%。

*定时传播中的边数。*我们用每秒创建的路径数来表示 TDFG 的大小。图 16(b) 显示了运行时间和内存开销。重复任务是 ROS 中间件中的回调工作者，因此增加了中间件中的任务数量，但对内核调度器影响不大。我们观察到，中间件调度时间随着任务数量的增加而增加。单个内核的 CPU 使用率峰值为 0.42%，任务数量越多，CPU 使用率越高。

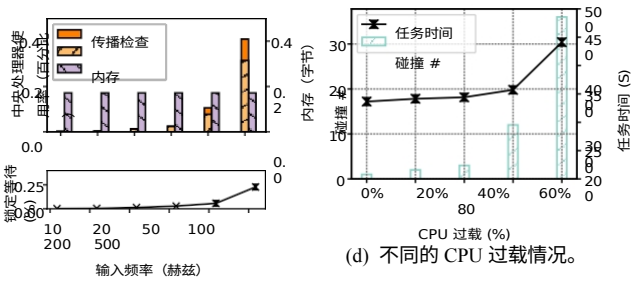


(a) Autoware (b) Jackal UGV (c) 乌龟宝宝3

图 15：运行时间开销细目。图中显示了原始任务的执行时间、记录计时信息、在线检查计时正确性和额外调度所花费的时间。



(a) 定时标签尺寸。 (b) 路径数量



(c) 输入频率 (d) 不同的 CPU 过载情况。

图 16：可扩展性分析（16(a)、16(b) 和 16(c)）和 CPU 过载的控制影响 16(d)。

5000 条路径，而内存开销仅为 75 KB，鉴于目标平台通常拥有超过 10 GB 的内存，这一数字相对较低。

*时序标签数量。*图 16(a) 显示了在时序约束检查过程中使用不同数量时序标签的单个边引起的运行时间和内存开销。我们可以看到，检查时间随定时标签数量的增加而成正比增加。标签大小为 100 时，平均开销为 2.945μs 运行时间和 896 字节内存；增加到 500 时，分别达到 9.4μs 和 4096 字节。鉴于边的数量通常保持在几百条以下，因此总开销较低。该图还显示，随着定时标记数量的增加，锁等待时间略有上升，但仍低于一个内核 CPU 使用量的 0.05%。

执行决策的调用延迟。在这个实验中，我们比较了 Kairos 和 ROS [80] 的功效、

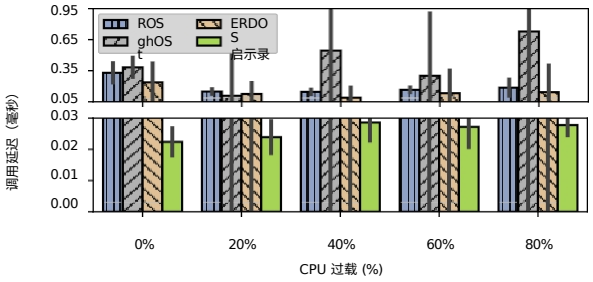


图 17：不同系统中处理程序的调用延迟。

ghOSt[57]和 ERDOS[55]提供调度决策。ghOSt[57]是一个用户空间知情的节点调度系统，允许从用户空间做出调度决策。EDROS 是一个机器人中间件，它提供了编程接口，用于部署错过截止日期的手持设备。

延迟。我们测量的是从做出决定到目标任务执行的延迟时间。实验是在不同的 CPU 过载情况下进行的。我们使用 *stress-ng* [62] 工具注入 0% 至 80% 的过载，然后在图 17 中比较延迟的增加和变化。

在 80% 的过载情况下，ROS、ghOSt、ERDOS 和 Kairos 的响应时间分别为 0.186 毫秒、0.72 毫秒、0.14 毫秒和 0.027 毫秒。Kairos 在执行决策时的响应时间最快，在系统过载严重的情况下，比其他系统至少快 2.16×公平地说，这些系统的目标并不是在高过载情况下实现性能。Kairos 在跨调度层确定目标任务优先级方面具有合作性。相比之下，ROS 和 ERDOS 只在中间件层执行调度决策。同样，ghOSt 仅通过自己的调度器执行调度决策，该调度器是一个优先级低于 Linux CFS 调度器的子调度器。在 CPU 空闲时，所有四个系统的调用时间都很稳定，变化幅度分别为 0.095 毫秒、0.094 毫秒、0.18 毫秒和 0.004 毫秒。不过，我们注意到，随着系统开销的增加，ghOSt 和 ERDOS 的调用延迟和变化显著增加（在 CPU 负载过重 80% 的情况下高达 0.81 ms）。这是因为它们

表 5：不同平台上的 DFA

平台	位置	# 输入	# 任务	# 顶点	偏差（米）		# 碰撞	
					本地	DFA	本地	DFA
Autoware	92 k	8	16	14	0.67	0.13	45	16
豺狼	68 k	4	6	6	0.27	0.09	12	4
Turtlebot3	34 k	3	4	3	0.87	0.21	35	13

~~受 Linux 底层本地调度程序的影响很大~~(CFS)。我们的结论是，Kairos 在缓解时序违规时采取的对策更快、更稳定。

6.3 改善安全的效果

本节评估了 DFA 模型和 Kairos 在异常时序情况下提高性能/安全性的能力。我们在三个平台上进行了动态剖析，以确定哪些代码区域需要注释时序约束，并确定预期的时序属性。关于处理策略，还需要了解目标程序的任务模型和语义。为了减轻这种主观性的影响，我们采用了一种自动策略，相应地为这些时序约束应用三种默认策略。具体来说，我们将目标程序的任务建模为有向图。我们对关键路径上的任务采用*优先*策略，因为中止这些任务会大大增加端到端响应时间。对于非关键路径上的任务，我们采用具有*新鲜度*限制的边，并对其余任务采用*中止*策略。这是因为违反*一致性*和*稳定性*通常会导致错误的计算结果，因此应防止其传播给下游任务。表 5 显示了三个平台上的输入和任务数量，以及在 TDFG 中注释了时序约束的边的数量。在为这些时序约束生成时序阈值时，我们观察到平均差异为 8.82 毫秒。

我们在每个场景（停车场[34]中的 Autoware、办公室[48]中的 Jackal UGV 和房屋[81]场景中的 Turtlebot3）中生成了 100 条轨迹供车辆跟踪。在导航过程中，我们使用 *stress-ng* 工具[62]注入 CPU 过载，以模拟异常时序。我们选择了 60% 的过载，因为这种情况通常会导致控制性能显著下降。图 16(d) 显示了 Jackal UGV 在过载情况下运行 100 次的碰撞次数。我们观察到碰撞次数在 60% 时明显增加。此外，随着 CPU 过载，任务时间也会增加，因为过载越高，往往会触发故障安全，使车辆在任务期间停止运行。过载率达到 80% 时，车辆通常会停止运行，需要人工干预才能继续。

控制性能改进。控制性能通过以下指标进行量化：(1) 飞行器偏离参考任务轨迹的距离；(2) 碰撞次数。结果如表 5 所示。我们观察到结果表明，在所有三个平台上，Kairos 都能显著减少控制偏差。偏差最小的改进是 2.97× 。至于碰撞，Kairos 在 Autoware、Jackal UGV 和 Turtlebot3 上分别减少了 64.4%、83.3% 和 62.9%。进一步研究发现，这种改进主要是由于主动放弃了错误的计算结果，以防止车辆输出错误的执行命令。不过，这种方法会降低飞行器的速度，从而增加任务时间。

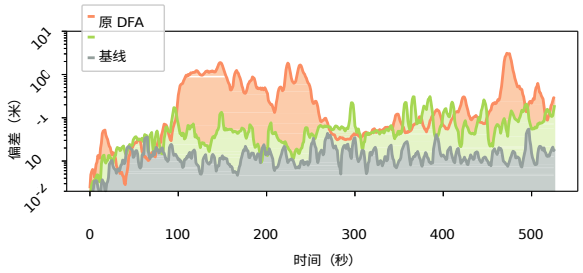


图 18：Jackal UGV 的控制性能比较，10 个窗口的平均偏差。

图 18 显示了 Jackal UGV 一次测试运行的定位误差随时间变化的情况。我们看到，Kairos 的异常定时检测和处理机制可以显著降低间歇性计算误差的幅度，从~ 1 米的水平降低到~ 10 厘米的水平。在这种情况下，近三分之一的帧会在系统超负荷的情况下被丢弃，从而防止软件使用时序异常的数据，避免出现错误的计算结果。放弃或跳过数据也会减慢机器人的移动速度，降低碰撞风险，但会增加 56.8% 的任务时间。这种策略可能不适合有严格期限要求的硬实时系统。不过，它能有效减少软实时系统中的不利控制结果。

7 讨论和限制

使用 DFA 表达实时计算结构。DFA 从数据流的角度处理时序保证问题，提供了一种更直观的机制来表达、检测和缓解违反时序约束的情况。然而，为了使这一系统与现有的实时系统结构协调一致，DFA 必须能够表达传统的实时基元。这样，开发人员就可以利用 DFA 表达的实时基元，在过去几十年来实时理论取得的巨大进步基础上再接再厉。实时计算基元分为两类。第一类是执行时间约束，它规定了程序中两条语句之间的执行时间界限。利用 Kairos，开发人员可以使用语句 *freshness(var,delay,abort)* 和 *freshness(var,delay,policy)* 作为确定和软截止时间规范，其中 *var* 是任务的输出，*delay* 是截止时间。第二个

同步原语指定了多个实时线程之间的共享数据访问。这可以通过使用 *consistency(write_var, read_var, 0, policy)* 等语句对两个数据流的时序进行严格排序来表达，其中 *read_var* 和 *write_var* 是由不同并发线程读写的 SSA 变量。该注解将确保写操作发生在读取操作之前。

人工操作。虽然 Kairos 提供了通过动态分析进行剖析的工具，但要在何处以及包含哪些时间限制条件，搜索空间往往大得令人望而却步。，自动工具可能需要很长时间才能确定适当的时间策略。开发人员的指导和一些手动注释可以快速缩小搜索空间。此外，一旦找到策略，将其部署到安全关键型系统中可能需要对目标系统进行重新验证甚至重新认证。

违规处理中的多个系统组件。Kairos 要求软件仪器与不同调度层的多个组件无缝协作，以有效缓解违规问题。这种相互依赖性带来了两个限制。首先是可靠性，因为一个部件的故障会影响整个系统。其次是可维护性，因为迁移到不同平台可能需要大量的工程设计工作。不过，时间违规检测和缓解的模块化设计允许 Kairos 与其他现有的检测或缓解技术集成。此外，弥合不同抽象层之间语义鸿沟的基础设施也减少了构建跨层时序缓解所需的工程工作量。

DFA 的通用性。虽然 DFA 是为网络物理系统设计的，但对数据流施加时间预期的概念通常适用于更广泛的网络类别，包括数据中心等传统的网络环境。例如，DFA 对数据使用的时序约束可适用于具有非确定性的系统，以确保逻辑正确性，如分布式系统中输入事件的顺序[74]。还可以利用 DFA 通过分布式工作负载中的数据流来跟踪计算进度。

8 相关工作

编程模型中的定时语义。在数据流系统中，人们一直在努力将时序信息纳入编程模型，以表示逻辑点，如逻辑时间戳或水印 [30,74,88,92]。这有助于协调分散节点之间的计算。数据流图上时序信息的这种扩展启发了我们的设计。不过，这些系统是为大规模并行数据处理而设计，而不是网络物理时序对齐。

在实时计算领域，已经提出了几种编程模型来应对违反定时的情况 [42、44、55、76、87]。其中，Timed C [76] 是 C 语言的一种方言，允许指定软性和硬性实时约束。然而，与这些著作相比，DFA 引入了一种设计方法，重点关注数据流的时间策略，它建立在网络物理控制回路抽象之上，允许检测和缓解网络物理状态（数据）错位。

跨层调度。目前大量研究集中于跨层调度。然而，现有的工作通常针对特定的硬件[40, 58, 78, 95]，如网卡。此外，许多目标服务器平台拥有强大的计算能力，因此这些解决方案可能无法很好地应用于资源受限的嵌入式系统。值得注意的是，与 Kairos 类似，Syrup [60] 为自定义调度策略提供了可编程抽象和接口。不过，它的重点是快速部署定制调度程序，而不是实现跨层调度操作。

跨层调度也是实时社区在组合调度背景下进行研究的内容 [51, 83, 93]。然而，这些技术的部署往往要求目标系统被严格建模并部署为实时任务，这可能并不总是适合 CPS 的某些现有软件架构。

9 结论

在中，我们提出了数据流可用性这一概念，旨在定义实时安全关键网络物理系统中数据流的时间策略。通过对 7 个具有代表性的网络物理系统软件的 189 个问题进行错误研究，我们提取了有关网络状态和物理状态在时间上的一致性的三个关键时间属性。为了具体表达时间预期，我们用 TDFG 捕捉到的时序约束增强了数据流。为了在系统中实现这一概念，我们设计并开发了 Kairos，通过在应用程序中嵌入作为检查的策略来检测时间违规行为，并通过跨层调度基础设施来缓解这些违规行为。最后，我们在三个 CPS 平台上对系统的可行性进行了评估。

鸣谢

我们对匿名审稿人和 Shepherd 的深刻反馈表示感谢。我们还要感谢 Sanjoy Baruah 和 Ron Cytron 的宝贵讨论。这项工作得到了国家自然科学基金（CNS-2238635）和英特尔公司的部分

支持。

参考资料

[1] Google cartographer ros for the hsrgoogle-cartographer-ros-for-the-toyota-hsr. toyota . https:// readthedocs.io/en/latest/。访问日期: 2024-04-18.

[2] Moveit. https://github.com/ros-planning/moveit。访问日期：2023-04-15。

[3] Orb-slam2. https://github.com/raulmur/ORB_SLAM2.访问日期：2023-04-15。

[4] Orb-slam3 github. https://github.com/UZ-SLAMLab/ORB_SLAM3。访问日期：2023-04-15。

[5] Ros navigation. https://github.com/ros-planning/navigation。访问日期：2023-04-15。

[6] Ros 导航堆栈。https://github.com/ros-planning/navigation。访问日期：2023-10-04。

[7] Ros rcl. https://github.com/ros2/rcl.访问日期：2023-04-15。

[8] 制图员 #153 计算每个轨迹的共同起始时间 每条轨迹的共同起始时间。https://github.com/cartographer-project/cartographer/pull/153,2016. 访问时间：2023-5-13。

[9] 制图师 #8 添加速率计时器。https://github.com/cartographer-project/cartographer/pull/ 8, 2016.访问日期：2023-05-22。

[10] 制图师 #242 改进 二维速度 https://github.com/cartographer-project/ cartographer/issues/242，2017。 已访问：2023-05-24.

[11] 阿波罗 #4492 Conti_radar 中的时间戳更正。https://github.com/ApolloAuto/apollo/issues/4492, 2018.访问日期：2024-05-20。

[12] Cartographer #1033 在子地图*d 中存储最新范围数据的时间戳。https://github.com/cartographer-project/cartographer/pull/ 1033，2018 年。访问日期：2023-05-22。

[13] 制图师 #1275 添加指标：实时 ra- tio 和 CPU 时间比。https://github.com/ cartographer-project/cartographer/pull/ 1275, 2018.访问时间：2023-05-13。

[14] 制图师 #1495 为时间添加序列化--tampedtransform。https://github.com/ cartographer-project/cartographer/pull/ 1495, 2019.访问日期：2023-05-13。

[15] Moveit #1299 在一个控制器中止时抢先执行轨迹。https://github.com/ros-planning/moveit/issues/1299，2019。访问日期：2023-05-22。

[16] Ros 2 rclcpp #694 fixup time. https://github.com/ROS2/RCLCPP/PULL/694，2019 年。访问时间：2023-11-22.

[17] Autoware.auto #1002 添加预测/更新功能接收时间戳到卡尔曼 https://gitlab.com/autowarefoundation/ autoware.auto/AutowareAuto/-/issues/1002, 2020.访问日期：2023-05-22。

[18] Autoware.auto #65 玫瑰 2 和 实时。https://gitlab.com/autowarefoundation/autoware.auto/AutowareAuto/-/issues/65,2020。访问日期：2023-05-22。

[19] Moveit #232 修复更新 plan- ningscene 时的竞赛条件。https://github.com/ros-planning/moveit/pull/232，2020。访问日期：2023-05-22。

[20] Moveit #2395 修复姿势跟踪竞赛条件：//github.com/ros-planning/moveit/pull/2395, 2020.访问日期：2023-05-22。

[21] https://github.com/raulmur/ORB_SLAM2/issues/946, 2020。已访问：2023-05-22.

[22] Ros 2 rclcpp #1121 lock-order-inversion（潜在死锁）。https://github.com/ros2/rclcpp/issues/ 1121，2020。访问日期：2023-05-23。

[23] Autoware.auto #605 记录 replay_planner 不连续 更新 轨迹。https://gitlab.com/autowarefoundation/ autoware.auto/AutowareAuto/-/issues/605, 2021.访问日期：2023-05-22。

[24] Autoware.auto #821 detect when nodes' in- coming messages are skipped. https://gitlab.. com/autowarefoundation/autowareauto/AutowareAuto/-/issues/821, 2021. 讵 cessed: 2023-05-22.

[25] Autoware.auto #980 更新了 ne raptor 接口以定期发送信息。https://gitlab.com/autowarefoundation/autoware.auto/AutowareAuto/-/merge_requests/980/diffs,2021.访问时间：2023-05-22。

[26] Ros 2 rclcpp #1679 动作客户端反馈回调不能可靠触发。https://github.com/ros2/ rclcpp/issues/1679，2021。访问日期：2023-05-22。

[27] Ros 2 rcl #967problems with arguments in rcl_timer_exchange_period api. https://github.com/ros2/rcl/issues/967、 2022. 访问日期：2023-05-22。

[28] Ros actionlib. http://wiki.ros.org/actionlib.访问日期： 2022-10-10。

[29] Benny Akesson、 Mitra Nasri、 Geoffrey Nelissen、 Sebastian Altmeyer 和 Robert I Davis。基于经验调查的实时系统行业实践研究。 In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 3-11.IEEE, 2020.

[30] Tyler Akidau、 Robert Bradshaw、 Craig Chambers、 Slava Chernyak、 Rafael J Fernández-Moctezuma、 Reuven Lax、 Sam McVeety、 Daniel Mills、 Frances Perry、 Eric Schmidt：《数据流模型：在大规模、无约束、无序数据处理中平衡正确性、延迟和成本的实用方法》。2015.

[31] 亚马逊 airprime. https://www.aboutamazon.com/news/transportation/amazon-prime-air-drone-delivery-mk30-photos。访问日期：2023-11-30。

[32] 亚马逊 astro。 https://www.aboutamazon.com/news/devices/meet-astro-a-home-robot-unlike-any-other.访问日期：2022-01-10。

[33] 比约恩-安德森、桑乔伊-巴鲁阿和扬-琼森。多处理器上的静态优先级调度。 In *Pro- ceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001)(Cat. No. 01PR1420)*, pages 193-202.IEEE, 2001.

[34] Autoware 基金会。 自主代客泊车演示。 https://autowarefoundation.gitlab.io/autoware.auto/AutowareAuto/avpdemo.html，2020 年。访问日期：2023-12-05。

[35] Autoware 基金会。 Autoware 的过去、现在和未来。 https://autoware.org/past-present-and-the-future-of-autoware/,2023. 访问日期：2024-04-18。

[36] Autoware.auto 项目。 https://autowarefoundation.gitlab.io/autoware.auto/AutowareAuto/。访问日期：2022-08-15。

[37] 百度。阿波罗自动驾驶项目。https://github.com/ApolloAuto/apollo。访问日期：2022-08-15。

[38] 百度。Apollo #9433: Cyberrt, coroutine to thread map- ping, 2019.访问时间：2023-05-22。

[39] S.Baruah、V. Bonifaci、G. DAngelo、H. Li、A. Marchetti- Spaccamela、S. van der Ster 和 L. Stougie。混合关键性隐式截止日期零星任务系统的预空闲单处理器调度。*2012 年第 24 届欧洲实时系统大会*，第 145-154 页。

[40] Adam Belay、George Prekas、Ana Klimovic、Samuel Grossman、Christos Kozyrakis 和 Edouard Bugnion。{IX}：高吞吐量、低延迟的受保护数据平面操作系统。*第 11 届 USENIX Sympo-操作系统设计与实现研讨会（OSDI 14）*，第 49-65 页，2014 年。

[41] Tobias Blass、Arne Hamann、Ralph Lange、Dirk Ziegen- bein 和 Björn B Brandenburg。ros 2 的自动延迟管理：优势、挑战和未决问题。In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 264-277.IEEE, 2021.

[42] Gregory Bollella 和 James Gosling.Java 的实时规范》。*计算机*，33（6）：47-54，2000。

[43] 艾伦-伯恩斯混合临界系统--综述。

[44] Alan Burns 和 Andrew J Wellings.*实时系统和编程语言：Ada 95、实时 Java 和实时 POSIX*。培生教育，2001 年。

[45] 乔治-布塔佐和卢卡-阿贝尼通过弹性调度实现自适应工作负载管理。*实时系统*》，23:7-24，2002 年。

[46] Giorgio C. Buttazzo、Giuseppe Lipari 和 Luca Abeni。自适应速率控制的弹性任务模型。*电气和电子工程师协会实时系统研讨会*，1998 年。

[47] 谷歌制图师。https://github.com/cartographer-project/cartographer。访问日期：2022-11-21。

[48] Clearpath Robotics。其他模拟世界。Jackal Tutorials 0.6.0 文档，2020 年。已访问：2023-12-05.

[49] Ron Cytron、Jeanne Ferrante、Barry K Rosen、Mark N Wegman 和 F Kenneth Zadeck。高效计算静态单一赋值形式和控制依赖图。*ACM 编程语言与系统交互（TOPLAS）*，13（4）：451-490，1991.

[50] davetcoleman.Moveit #294 isvalidvelocitymove() 用于检查两个机器人状态之间的最大速度。https://github.com/ros-planning/moveit/pull/294, 2016.访问日期：2023-11-22。

[51] Arvind Easwaran、Madhukar Anand 和 Insup Lee。使用 edp 资源模型的组合分析框架。*第 28 届 IEEE 国际实时系统研讨会（RTSS 2007）*，第 129-138 页。IEEE, 2007.

[52] Facebook 工程。大满贯：通过技术将艺术融入生活。https://engineering.fb.com/2017/09/21/virtual-reality/slam-bringing-art-to-life-through-technology/、2017 年 9 月。访问日期：2024-04-18。

[53] Autoware 基金会。Autoware.auto.访问时间：2023-04-15.

[54] 加施勒 制图师 #936 优雅地处理时间重叠点云。https://github.com/ cartographer-project/cartographer/pull/936,2018.访问日期：2023-11-22。

[55] Ionel Gog、Sukrit Kalra、Peter Schafhalter、Joseph E Gonzalez 和 Ion Stoica。D3：构建自动驾驶汽车的动态截止日期驱动方法。*第 17 届欧洲计算机系统会议论文集》*，第 453-471 页，2022 年。

[56] 谷歌。Cartographer。https://github.com/googlecartographer/cartographer。访问日期：2023-04-15。

[57] 杰克-蒂加-汉弗莱斯（Jack Tigar Humphries）、尼尔-纳图（Neel Natu）、阿什温-乔古勒（Ashwin Chaugule）、奥菲尔-韦斯（Ofir Weisse）、巴雷特-罗登（Barret Rhoden）、乔什-唐（Josh Don）、路易吉-里佐（Luigi Rizzo）、奥列格-隆巴赫（Oleg Rombakh）、保罗-特纳（Paul Turner）和克里斯托斯-科兹拉基斯（Christos Kozyrakis）。*ACM SIGOPS 第 28 届操作系统原理研讨会论文集》*，第 588-604 页，2021 年。

[58] Stephen Ibanez、Alex Mallery、Serhat Arslan、Theo Jepsen、Muhammad Shahbaz、Changhoon Kim 和 Nick McKeown。纳秒堆栈：数据中心的纳秒网络堆栈。*第 15 届{USENIX}操作系统设计与实现研讨会（{OSDI} 21）*，第 239-256 页，2021 年。

[59] Jackal ugv. https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/。获取时间：2021-07-30。

[60] Kostis Kaffes、Jack Tigar Humphries、David Mazières 和 Christos Kozyrakis。糖浆：跨堆栈的用户自定义调度。*ACM SIGOPS 第 28 届操作系统原理研讨会论文集》*，第 605-620 页，2021 年。

[61] 加藤真平、德永翔太、丸山裕也、前田诚也、平林万里、北川由纪、亚伯拉罕-蒙罗伊、安藤友人、藤井雄介和拓也。

阿苏米车载 Autoware：利用嵌入式系统实现自动驾驶汽车。In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pages 287-296.IEEE, 2018.

[62] Colin King. stress-ng. ubuntu.com/Kernel/Reference/stress-ng。访问：2022 年 5 月 20 日。

[63] Chris Lattner 和 Vikram Adve。Llvm：用于终身程序分析和转换的编译框架。In *International Symposium on Code Genera- tion and Optimization, 2004.CGO 2004.*，第 75-86 页。
电气和电子工程师学会，2004 年。

[64] Tanakorn Leesatapornwongsa、Jeffrey F Lukman、Shan Lu 和 Haryadi S Gunawi。Taxdc：数据中心分布式系统中的非确定性并发错误分类法。*第 21 届编程语言和操作系统架构支持国际会议论文集*》，第 517-530 页，2016 年。

[65] Ao Li、Marion Sudvarg、Han Liu、Zhiyuan Yu、Chris Gill 和 Ning Zhang。多节奏：针对定时干扰的多通道攻击模板自适应调整。*2022 年 IEEE 实时系统研讨会（RTSS）*，第 225-239 页。IEEE, 2022.

[66] Ao Li、Jinwen Wang、Sanjoy Baruah、Bruno Sinopoli 和 Ning Zhang.性能干扰实证研究：定时违规模式及影响。*2024 年实时和嵌入式技术与应用研讨会（RTAS）*。IEEE, 2024.

[67] Shan Lu、Soyeon Park、Eunsoo Seo 和 Yuanyuan Zhou。从错误中学习：现实世界并发错误特征综合研究》。*第 13 届编程语言和操作系统架构支持国际会议论文集*》，第 329-339 页，2008 年。

[68] Steven Macenski、Tully Foote、Brian Gerkey、Chris Lalancette 和 William Woodall。机器人操作系统 2：设计、架构和野外使用》。*科学机器人*》，7（66）：eabm6074，2022。

[69] Martina Maggio, Arne Hamann, Eckart Mayer-John, and Dirk Ziegenbein.连续期限错过约束下的控制系统稳定性。*第 32 届欧洲微实时系统会议（ECRTS 2020）*。Schloss Dagstuhl-Leibniz-Zentrum für Informatik，2020。

[70] Claire Maiza、Hamza Rihani、Juan M Rivas、Joël Goossens、Sebastian Altmeyer 和 Robert I Davis。多核实时系统时序验证技术概览。*ACM Computing Surveys (CSUR)*, 52(3):1-38, 2019.

[71] Pau Marti、Josep M Fuertes、Gerhard Fohler 和 Krithi Ramamritham。实时控制系统的抖动补偿*第 22 届电气和电子工程师学会实时系统研讨会（RTSS 2001）论文集（目录编号：01PR1420）*，第 39-48 页。IEEE, 2001.

[72] Till Menzel、Gerrit Bagschik 和 Markus Maurer。自动驾驶汽车的开发、测试和验证方案。In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1821-1827.IEEE, 2018.

[73] 大卫-莫斯伯格和拉里-彼得森。在侦察兵操作系统中明确路径。*OSDI* 第 96 卷，第 153-167 页，1996 年。

[74] Derek G Murray、Frank McSherry、Rebecca Isaacs、Michael Isard、Paul Barham 和 Martín Abadi。Na-iad：及时数据流系统。*第二十四届 ACM 操作系统原理研讨会论文集*》，第 439-455 页，2013 年。

[75] Picknik 机器人公司赢得太空部队、美国国家航空航天局的支持。https://www.therobotreport.com/picknik-robotics-wins-space-force-nasa-contracts/。访问日期：2024-04-18。

[76] Saranya Natarajan 和 David Broman。定时 C：一个实时系统 C 编程语言的扩展。在 *2018 IEEE 实时和嵌入式技术与应用研讨会（RTAS）*上，第 227- 页。239.IEEE, 2018.

[77] José Carlos Palencia 和 M González Harbour。具有静态和动态偏移的任务的可调度性分析。*第 19 届 IEEE 实时系统研讨会论文集（目录号 98CB36279）*，第 26-37 页。IEEE, 1998.

[78] George Prekas、Marios Kogias 和 Edouard Bugnion。Zygos：实现微秒级网络任务的低尾部延迟。*第 26 届操作系统原理研讨会论文集*》，第 325-341 页，2017 年。

[79] 制图师项目 修复 按 速度。https://github.com/ cartographer-project/cartographer/commit/b4b83405ce4009ea0c1ac22c7ab9edeeb9d48a42,2017. 提交 b4b834。

[80] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system.在 *ICRA 开源软件研讨会上*，第 3 卷，第 5 页。日本神户，2009 年。

[82] Utsav Sethi、Haochen Pan、Shan Lu、Madanlal Musuvathi 和 Suman Nath。系统中的取消：任务取消模式和失败的实证研究。*第 16 届 USENIX 操作系统设计与实现研讨会（OSDI 22）*，第 127-141 页，2022 年。

[83] Insik Shin 和 Insup Lee.合成实时调度框架。*第 25 届 IEEE 国际实时系统研讨会*，第 57-67 页。IEEE, 2004.

[84] Marion Sudvarg、Chris Gill 和 Sanjoy Baruah.弹性调度的线性时间准入控制。*实时系统*》，57（4）：485-490，10 2021。

[85] Marion Sudvarg, Ao Li, Daisy Wang, Sanjoy Baruah, Jeremy Buhler, Chris Gill, Ning Zhang, and Pontus Ekberg.谐波任务系统的弹性调度。*2024 年实时和嵌入式技术与应用研讨会（RTAS）*。IEEE，2024 年。

[86] Yulei Sui 和 Jingling Xue.Svf：程序间静态

llvm 中的值流分析。*第 25 届编译器构造国际会议论文集*》，第 265-266 页，2016 年。

[81] ROBOTIS. Turtlebot3 模拟。https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/#gazebo-simulation。访问日期：2024-05-20。

[87] Milijana Surbatovich、Limin Jia 和 Brandon Lucia。在间歇系统中自动执行新鲜一致的输入。*第 42 届 ACM SIGPLAN 国际编程语言设计与实现大会论文集》*，第 851-866 页，2021 年。

[88] Peter A. Tucker、David Maier、Tim Sheard 和 Leonidas Fegaras。在连续数据流中利用标点符号语义。*IEEE 知识与数据工程论文集*，15(3):555-568, 2003.

[89] Turtlebot3. https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/。访问日期：2022-09-10。

[90] Steve Vestal.具有不同执行时间保证程度的多关键性系统的抢占式调度。*第 28 届 IEEE 国际实时系统研讨会（RTSS 2007）*，第 239-243 页。IEEE, 2007.

[91] Nils Vreman、Anton Cervin 和 Martina Maggio。受突发截止时间错过影响的控制系统的稳定性和性能分析。*第 33 届欧洲实时系统大会（ECRTS 2021）*。Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[92] 王国章、陈磊、阿尤斯曼-迪克希特、杰森-古斯塔夫森、陈博阳、马蒂亚斯-J-萨克斯、约翰-罗斯勒、索菲-布莱-戈德曼、布鲁诺-卡当纳、阿普尔瓦-梅塔、

一致性与完整性：重新思考 Apache Kafka 中的分布式流处理。In *Proceed-ings of the 2021 International Conference on Manage- ment of Data*, pages 2602-2613, 2021.

[93] 王金文、李敖、李浩然、吕晨阳、张宁。Rt-tee：使用 arm trustzone 实现网络物理系统的实时系统可用性。In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 352-369.IEEE, 2022.

[94] Waymo 无滴水服务在凤凰城 https://blog.waymo.com/2020/10/waymo-is-opening-its-fully-driverless.html。访问日期：2022-01-10。

[95] Chuanyu Xue、Tianyu Zhang、Yuanbin Zhou、Mark Nixon、Andrew Loveless 和 Song Han。802.1qbv 时敏网络（tsn）的实时调度：系统回顾与实验研究。In *2024 IEEE Real-Time and Embedded Technology and Appli- cations Symposium (RTAS)*.IEEE, 2022.