基于AUTOSAR自适应平台的监控与容错方案研究

朱元[1] 赵乾翔[1] 张彪[2] 毕承鼎[2]

（1.同济大学汽车学院，上海 201804；2.一汽解放汽车有限公司商用车开发院，长春 130011）

【欢迎引用】朱元, 赵乾翔, 张彪, 等. 基于AUTOSAR自适应平台的监控与容错方案研究[J]. 汽车文摘, 2024(6): 11-23.

【Cite this paper】ZHU Y, ZHAO Q X, ZHANG B, et al. Research on Monitoring and Fault-Tolerance Scheme Based on AUTOSAR Adaptive Platform[J]. Automotive Digest (Chinese), 2024(6): 11-23.

【摘要】为了解决面向服务的AUTOSAR自适应平台（AP）缺乏有效监控与容错机制的问题，并确保软件系统在故障情况下的高度稳定性和安全性，以汽车基础软件平台AP为研究对象，通过自动代客泊车（AVP）软件系统设计了一套完备的监控与故障容错机制，通过分析传统软件架构与其他面向服务架构的监控方案和AP的特性，设计了AP的监控方案，实现对平台基础设施（处理器、网络、内存）和服务状态（响应时间）的监控；基于Qt开发的数据显示模块通过LT协议实现实时采集与显示监控数据；提出了一种支持SOME/IP协议的服务调用链追踪方法，实现对面向服务架构中复杂服务调用关系的分析。结果表明该方案能够在AVP软件系统中监控和分析服务故障，并在发生故障时采取容错机制，提升系统可靠性。

关键词：AUTOSAR；面向服务架构；平台监控；容错机制

中图分类号：TP311.131　　**文献标志码：**A　　**DOI:** 10.19822/j.cnki.1671-6329.20240021

Research on Monitoring and Fault-Tolerance Scheme Based on AUTOSAR Adaptive Platform

Zhu Yuan[1], Zhao Qianxiang[1], Zhang Biao[2], Bi Chengding[2]

（1. School of Automotive Engineering, Tongji University, Shanghai 201804; 2. Commercial Vehicle Development Institute, FAW Jiefang Automobile Co., Ltd., Changchun 130011）

【Abstract】To address the lack of effective monitoring and fault tolerance mechanisms in the service-oriented AUTOSAR Adaptive Platform (AP) and to ensure high stability and safety of the software system in case of faults, this study takes the automotive basic software platform AP as the research object. A complete monitoring and fault tolerance mechanism is designed through the Automated Valet Parking (AVP) software system. By analyzing traditional software architectures, other service-oriented architecture monitoring solutions, and the characteristics of AP, a monitoring scheme for AP is designed to supervise the platform infrastructure (processors, networks, memory) and service states (response times). A data display module developed with Qt, using the LT protocol, implements the collection and display of real-time monitoring data. Furthermore, a service call chain tracing method supporting the SOME/IP protocol is proposed, enabling analysis of complex service call relationships within service-oriented architectures. The results indicate that the scheme can monitor and analyze service failures within the AVP software system and implement fault tolerance mechanisms in case of failures, thus enhancing system reliability.

Key words: AUTOSAR, Service-oriented architecture, Platform monitoring, Fault tolerance mechanism

0 引言

电动化、智能化、网联化是汽车工业未来的三大发展趋势，三者彼此关联，协同发展[1-7]。在新的汽车工业发展趋势下，诸如高级驾驶辅助系统和车载信息娱乐系统等新技术不断应用于汽车领域[8-11]。为了满足汽车日益增长的高计算需求，多核高性能微处理器正逐步应用于汽车控制系统[12-13]。这些微处理器被用作域控制器，能够整合原本分散在不同小型控制器中的功能，同时承担更复杂的智能驾驶算法[14-15]。

在这种背景下，2017年AUTOSAR联盟推出了AUTOSAR自适应平台（Adaptive Platform，AP）。AP作

为电子控制单元(Electronic Control Unit, ECU)的标准化集成平台,构建在POSIX操作系统之上,致力于满足汽车领域的最新发展趋势需求[16-17]。

与此同时,随着车载软件功能需求的不断增长,通过增加相应的监控手段与容错机制,保证汽车软件可靠性[18-20]。然而,在AP领域尚未存在有效的监控与容错方案。

本文以自动代客泊车(Automated Valet Parking, AVP)系统为例,针对AP领域缺少对系统的有效监控手段和故障容错机制的问题,设计了平台监控方案完成对系统的实时监控,实现开发过程中的故障定位分析与运行时发现故障,并设计了故障容错机制,保证故障发生后系统能够正常运行。

1 AUTOSAR自适应平台数据监测方案

1.1 设计方案

针对AP平台,设计了数据监测系统(图1),包括分布式数据采集模块及其在ARA(AUTOSAR Runtime for Adaptive Applications)上运行的软件应用组件(SoftWare Compenent, SWC)、数据存储与分析模块,数据显示模块。

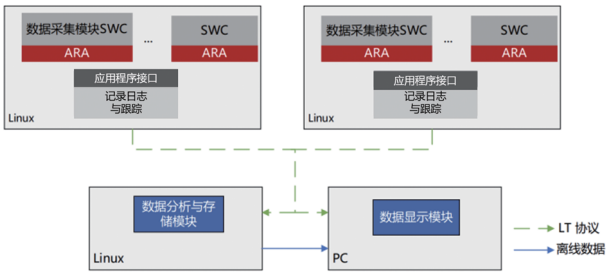


图1 AUTOSAR自适应平台监测系统

数据采集模块负责采集基础设施相关的数据,分布式地部署到每一个需要监视的操作系统中,对中央处理器(Central Processing Unit, CPU)、内存相关数据进行监测,这些数据主要通过操作系统提供的接口或者文件进行获取。数据采集模块以固定的周期对这些数据进行采集,并通过LT(Log and Trace)协议发送至数据存储模块和数据显示模块。

数据存储与分析模块负责对所有采集到的数据进行汇总,包括采集模块的数据,以及应用主动上报的数据,如Method的响应时间等,并将这些数据进行存储,以离线文件的形式提供给数据显示模块。该模块会对数据进行统计分析,通过配置文件确定判断故障的条件,从而根据已有数据统计故障数量。

数据显示模块负责可视化监测数据,通过表格、曲线图、柱状图等方式显示数据。数据显示模块并非安装在汽车内,而是以上位机的形式安装在个人计算机(Personal Computer,PC)中,既能够实时通过LT协议接收监测数据,也能将数据存储与分析模块的离线监控数据进行解析和显示。

方案使用AUTOSAR中的Dlt模块实现数据的收集工作,Dlt模块是AUTOSAR标准中使用的日志模块,通过LT协议传输日志,无需在AP上安装其他组件即可实现Dlt日志的发送与控制。

LT协议报文格式包含3部分:基础头部(Base Header)、扩展头部(Extension Header)和有效载荷(Payload),见图2。

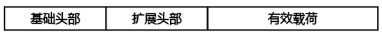


图2 LT协议报文格式

1.2 监测数据选择

数据监测模块中数据指标的选取需要确保该指标确实与服务状态直接或间接相关,并在记录时标上监测的时间信息,形成数据在时间上的关联。

在AP中,与SOME/IP(Scalabe service-Oriented MiddlewarE over IP)服务相关的直接数据指标包括:

(1)Method响应时间

在AP中,通过SOME/IP Method实现远程过程调用时服务调用方与服务提供方之间常见的交互方式。Method的响应时间是衡量SOME/IP服务状态的重要指标,在一些对于时间要求比较高的场景中,如果响应时间超过约定指标,即认定本次交互失败。

Method响应时间有2种计算方式,一种是从服务提供方角度进行计算,即服务提供方接收到Method请求和完成Method计算的时间间隔;另一种是从服务调用方角度进行计算,即服务调用方发起Method和接收到结果的时间间隔。与前者相比,后者计算出的响应时间包括了中间件对服务报文的处理时间和服务报文在网络中的传输时间。本设计中,这2种Method响应时间都应被监测。

对于Method响应时间这项数据指标,在具体实现上,应当由服务提供方和服务调用方主动上报,上报格式见图3。

N bytes	1 byte	N bytes	1 byte	N bytes	1 byte	N bytes	1 byte
Context	'\20'	Service Instance Id	'\20'	Method Name	'\20'	Time	'\20'

图3 Method响应时间的Dlt日志结构

其中Context用来区分Method响应时间的计算方式,“Call”表示服务调用方计算的响应时间,“Impl”表

示服务提供方计算的响应时间。

(2)Event发送/接收时间间隔

在AP中,Event可以被用来传输周期性的数据(例如周期性采集的雷达数据)。这种应用场景下,该周期必须是确定的,如果服务调用方在周期间隔内未收到Event数据,可能会导致服务调用方故障。

Event发送时间是服务提供方的Event发送时间点,Event接收时间是服务接收方接收到Event的时间。

对于Event发送/接收时间这项数据指标,在具体实现上,应当由服务提供方和服务调用方主动上报,上报格式见图4。

N bytes	1 byte	N bytes	1 byte	N bytes	1 byte
Context	'\20'	Service Instance Id	'\20'	Event Name	'\20'

图4 Event发送/接收时间的Dlt日志结构

其中Context用来区分Event发送/接收时间,"Send"表示Event发送时间,"Recv"表示Event接收时间。

除了服务状态的直接数据指标外,数据监测模块还应当采集与SOME/IP服务相关的基础设施数据指标。

(3)CPU利用率与CPU负载

CPU资源是硬件平台中非常重要的系统资源,如果CPU资源耗尽,将造成硬件平台中的大部分进程得不到调度,引起进程饥饿问题。汽车中,CPU负载过高的原因有很多,如程序设计失误造成应用进入死循环,系统超过CPU处理能力。

CPU使用率是一段时间内CPU运行时间占总时间的比值,CPU负载是一段时间内处于可运行状态和不可中断状态的进程平均数量。

在CPU处于高负载状态时,CPU负载更能反应CPU的真实状况,由于高负载状态时CPU使用率已经接近100%,无法得出更多CPU的负载信息,CPU负载能够反映出此时正在运行和等待运行的进程数。

(4)网络延迟

延迟通常采用网络数据包从发送端到接收端再回到接收端所经历的时间。分布式架构对底层网络十分依赖,网络的运行状况直接影响服务的健康状况。当网络发生拥堵时,会造成服务的响应时间变长,进而通过服务间的依赖引起连锁反应。

(5)内存使用率

内存使用率,即已使用的内存占总内存的比重。AP应用在启动后,会将可执行文件中的代码和数据放入内存中,等待CPU访问。Linux等支持虚拟内存的操作系统在内存资源耗尽时,会将一些内存中的数据转移到容量更大的外部存储器中,以缓解内存不足问题。

数据采集模块负责周期性地对基础设施数据指标进行采集,并将采集到的数据日志形式上报至数据分析与存储模块以及数据显示模块,基础设施数据的Dlt日志结构见图5。

N bytes	1 byte	N bytes	1 byte	N bytes	1 byte	N bytes	1 byte	N bytes	1 byte
Machine Name	'\20'	Round-Trip Time	'\20'	CPU Usage	'\20'	CPU Load	'\20'	Memory Usage	'\20'

图5 基础设施数据的Dlt日志结构

1.3 数据显示模块

数据显示模块是安装在PC中的上位机程序,能够对监控数据进行可视化的展示,监控数据可以使用离线数据,也可以通过LT协议对监控数据进行采集(图6)。

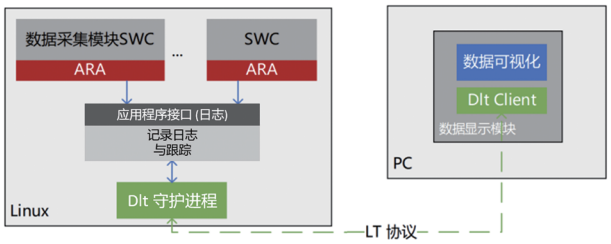


图6 数据显示模块

数据显示模块包含2个子处理模块——Dlt Client和数据可视化部分。AP中Dlt守护进程通过LT协议将日志信息发送至Dlt Client,LT协议分为Server和Client,Dlt守护进程为Server,数据显示模块是Client。Dlt Client收到日志信息后需要对内容进行过滤和筛选,将监控方案需要的信息存储下来,通过数据可视化功能进行显示。

数据可视化部分负责对收到的监控数据进行显示,在设计的监控方案中,数据可视化采用Qt实现。Qt是图形用户界面程序开发框架,支持数据表格、折线图、柱状图、自定义图形绘制等功能。监控数据中采集的上报数据显示到Qt数据表格中,对于服务响应时间,绘制响应时间分布图和随时间变化的曲线图,对于系统基础设施,绘制各项监控数据随时间变化的曲线图。

2 服务调用链追踪

汽车软件系统中存在大量的服务和服务节点,同时服务节点之间会存在复杂的服务调用关系,如:A服务调用B服务,B服务调用C和D服务。在这样的情况下,如果C服务出现故障,那么B服务和A服务将也

会出现故障。

如果选用传统的故障定位方式，通过日志模块将每个节点调用服务和被请求服务的行为记录到日志中，逐一分析日志来排查故障。但是这些服务节点日志分散，彼此没有很强的联系，无法从中整理出C服务故障是B服务与A服务故障的根本原因。为了将这些服务节点的内部行为和相互关系加以整理，本章引入服务调用链动态追踪技术。通过此技术将一条完整的服务调用链信息从离散的服务节点日志中提取，此调用链信息包含了所有服务节点、服务调用关系、服务处理时间、时延和故障信息。

因此需要设计一种基于AP、支持SOME/IP协议的服务调用链追踪方法。旨在解决上述面向服务架构中故障定位困难和SOME/IP协议不支持调用链追踪的问题，以此帮助开发人员梳理服务节点之间的动态调用关系，监控方案在运行时的服务调用情况，更加容易地观察服务依赖关系，定位故障所在的服务节点。

2.1 设计方案

在服务调用方安装调用链追踪工具Client组件，在服务提供方安装调用链追踪工具Server组件，在PC上安装调用链追踪工具。服务调用链追踪系统结构见图7，服务调用链追踪系统时序图见图8。

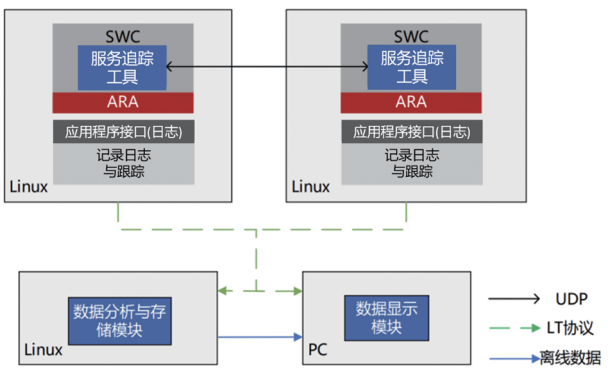


图7 服务调用链追踪系统结构图

当AP应用A作为服务调用方发起SOME/IP服务调用时，被视为一条服务调用链的开始，调用链追踪工具Client组件会生成唯一的调用链Trace Id。将该Trace Id和该应用的节点信息、调用服务的Service Id与Method Id等信息组成调用链信息报文，通过UDP多播协议发送至作为服务提供方的AP应用B，同时将这些调用链信息通过LT协议以日志的形式发送到服务调用链追踪工具。当作为服务提供方的AP应用B接收此次SOME/IP服务调用时，也会从接收到与本次SOME/IP服务调用对应的调用链信息，在完成服务调用基本功能的同时，会同时将这些调用链信息通过LT协议以日志的形式发送到服务调用链追踪工具。

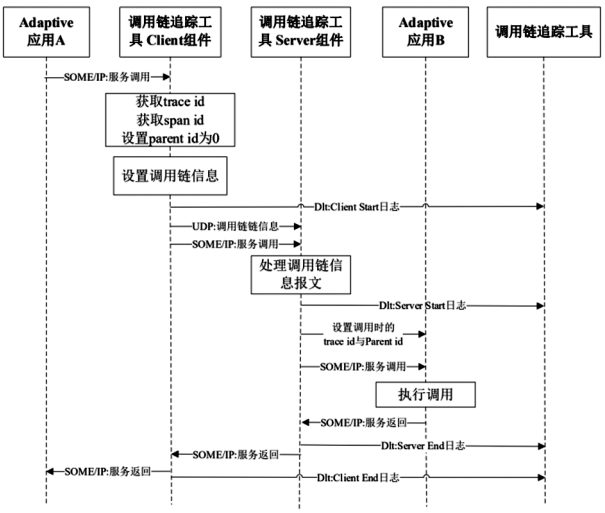


图8 服务调用链追踪系统时序图

如果AP应用B服务继续调用其他服务时，会将自身的信息加入调用链信息，继续重复上述操作。

数据显示模块会收集上述所有的日志信息，通过Trace Id将所有调用链信息日志加以区分，最终分析出每一条调用链的信息，并以可视化形式展现。

2.2 调用链信息报文设计

调用链追踪工具通过UDP多播发送调用链信息报文实现调用链信息的传输，在调用链信息报文中加入SOME/IP报文的信息（Client Id、Session Id、Service Id和Method Id），实现与SOME/IP报文的唯一对应。此方法不需要修改SOME/IP报文信息和ARXML配置文件，并且能够与未安装调用链追踪工具的AP应用实现SOME/IP通信。调用链信息报文见图9。

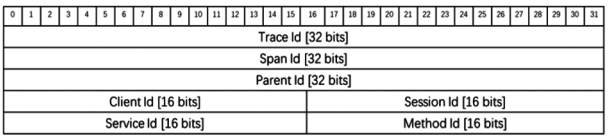


图9 调用链信息报文结构

（1）Trace Id [32 bit]

调用链Trace Id是调用链的唯一标识，因此必须保持该标识的唯一性。

Trace Id使用当前时间加随机数的方法来获取（见图10）。同一时间的服务调用链发起数量有限，对于开始时间完全相同的服务调用链，通过加入随机数的方式，避免Trace Id冲突的可能。

图10 Trace Id结构

(2)Span Id [32 bit]

在服务调用链中,服务调用的服务调用方和服务提供方都被认为是一个节点,节点的唯一标识符即为Span Id。在一条服务调用链中,同一个服务可能被反复调用,所以对于同一服务被多次调用的情况,服务提供方的Span Id不能相同,服务调用方的Span Id也不能相同。因此Span Id被设计为节点IP地址主机号+服务Id +当前时间(见图11),来实现上述需求。

IP host Id [8 bits]	Service Id [16 bitss]	Time [8bits]

图11 Span Id结构

(3)Parent Id [32 bit]

当服务调用方向服务提供方调用服务时,服务调用方的Span Id即为服务提供方的Parent Id。通过每一个调用链信息中Span Id和Parent Id即可分析出该条调用链的服务调用关系。

(4)Client Id [16 bit]和Session Id [16 bit]

即为服务调用方的Client Id和服务调用方调用服务时的Session Id。在同一时间,可能有多个服务调用方并发访问服务提供方提供的同一服务,这些SOME/IP服务请求报文的到达时间与调用链信息报文的到达时间并不是有序的。通过比对SOME/IP服务请求报文中的Client Id与Session Id与调用链信息报文中的Client Id与Session Id,能够将调用链信息报文与SOME/IP报文唯一关联。

(5)请求服务的Service Id与Method Id

一个AP应用会同时作为多个服务的服务提供方,在调用链信息报文中加入Service Id和Method Id,能够帮助调用链追踪工具Server组件准确的识别该条调用链信息报文是与哪个服务相关联的。

2.3 Dlt日志设计

调用链信息由调用链追踪工具Server和Client组件通过Dlt模块以日志的形式,发送至调用链追踪工具,调用链追踪工具通过收集和处理这些包含了调用链信息的日志分析调用链的服务调用关系和其他信息。为了与其他日志进行区分,服务调用链日志将LT协议的Extended Header中Context Id设置为“Trace”作为标记。

LT协议的Payload用来存储调用链信息,结构如图12所示。为了使日志具有可读性,对于上报日志中的每一项信息采取不定长的字符串加以记录,以‘\20’(空格)作为间隔加以区分。

其中Call Context表示当前上报信息的时机,包括“Client Start”、“Client End”、“Server Start”和“Server End”4种状态。“Client Start”表示服务调用方发起服务调用时,“Client End”表示服务调用方完成服务调用时,“Server Start”表示服务提供方接收服务调用时,“Server End”表示服务提供方处理完成服务调用时。

N bytes	1 byte	N bytes	1 byte	N bytes	1 byte	N bytes	1 byte
Trace Id	'\20'	Span Id	'\20'	Parent Id	'\20'	Service Name	'\20'
N bytes	1 byte	N bytes	1 byte	N bytes	1 byte	N bytes	1 byte
Service Id	'\20'	Method Name	'\20'	Method Id	'\20'	Call Context	'\20'

图12 调用链信息Dlt日志结构

2.4 调用链信息报文处理与传输的实现

调用链信息报文的处理与传输由调用链追踪工具Client组件与Server组件完成。

调用链追踪工具Client组件的流程图见图13。AP应用通过调用链追踪工具Client组件接口发起服务调用时,组件会读取AP应用发起服务调用时上下文中是否有Trace Id信息。如果有,读取上下文中Trace Id,赋值到调用链信息的Trace Id,将上下文中的Span Id赋值到调用链信息的Parent Id。如果没有,则生成新的Trace Id,并将Parent Id设置为0,即为调用链根节点。根据服务调用时的SOME/IP报文,设置调用链信息中的Client Id、Session Id、Service Id和Method Id信息。通过LT协议向调用链追踪工具发送调用链信息日志,通过用户数据报协议(User Datagram Protocol,UDP)协议向服务提供方传输调用链信息报文,通过基于IP的可扩展面向服务的中间件SOME/IP协议发起对服务提供方的服务调用。

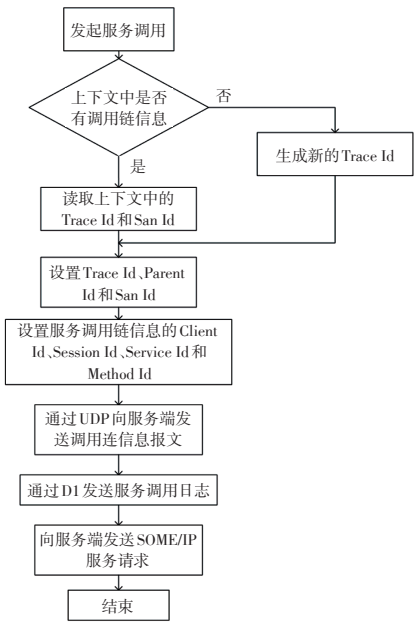
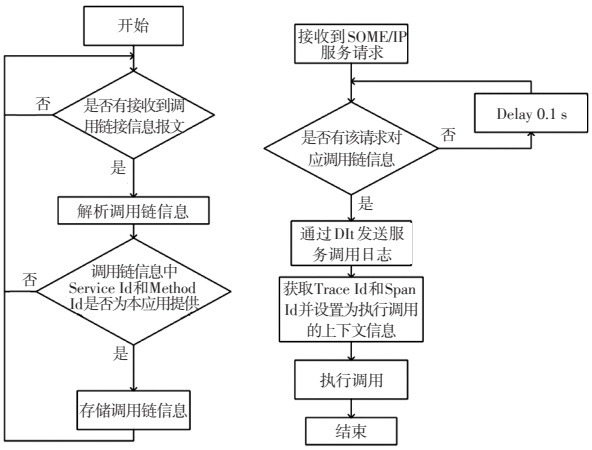


图13 调用链追踪工具Client组件流程

调用链追踪工具Server组件包括主流程和报文接收与处理两部分。报文接收与处理部分的流程见图14a所示，当接收到服务调用链信息的UDP报文时，解析该报文中的所有信息，如果Service Id和Method Id与本应用提供的服务匹配时，存储本条服务调用链信息，否则丢弃该报文。

当接收到SOME/IP服务请求时，触发调用链追踪工具Server组件的主流程，如图14b所示。根据SOME/IP服务请求中的Client Id、Session Id、Service Id和Method Id信息查找对应的服务调用链信息。因为网络传输的不确定性，SOME/IP报文和服务调用链信息的报文到达服务提供方的先后顺序未知，如果此时服务调用链信息的报文未到达，则等待，直到完成查找。服务调用链信息查找完成后，通过Dlt模块向调用链追踪工具发送包含调用链信息的日志。设置执行调用的上下文信息，即Trace Id和Span Id，并执行此次服务请求的调用。服务调用执行完成后，通过Dlt模块向调用链追踪工具发送包含调用链信息的日志。



(a)报文接收与处理线程流程　(b)调用链追踪工具Server组件主流程

图14　调用链处理信息流程

3　AUTOSAR自适应平台服务容错机制

3.1　汽车面向服务架构的服务容错概述

在面向服务架构中，服务错误可能无法完全避免，表现为服务无响应或对于时间要求严格的服务未能在规定时间内响应。对于汽车环境内时间敏感的软件，服务未及时响应意味着服务不可用。

这些服务错误主要由以下因素引起：

(1)基础设施故障：例如，CPU负载过高、内存不足、存储器访问速度过慢等，导致服务执行时间超出规定，无法按时响应。

(2)网络故障：例如网络拥塞、数据包丢失、网络中断，导致服务底层网络报文无法或无法按时到达服务。

(3)服务提供方故障：如程序异常退出、服务逻辑错误、限流策略等，导致服务提供方无法正常处理请求。

当服务调用方调用的服务出现错误时，没有合适的容错机制，错误可能会扩散，导致整个服务链路出现问题，造成更严重的“服务雪崩”现象，见图15。

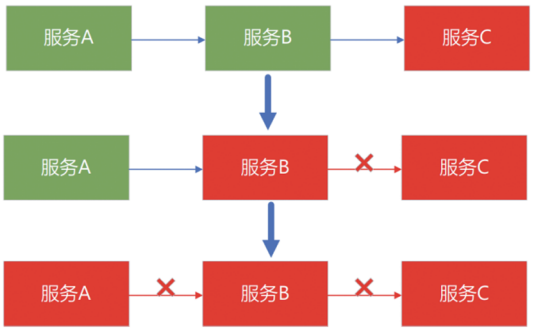


图15　服务雪崩

因此，针对服务故障，必须采取相应的容错措施以提高服务的可靠性。针对不同模块对服务可靠性的需求，采用不同的容错策略，以保持资源利用率与服务可靠性之间的平衡。

3.2　失败重试

失败重试是一种策略，指的是在服务调用出现错误时，会在一定的时间间隔后重新尝试调用该服务，直到服务能够正常响应，见图16。

图16　服务重试

这种方法在汽车软件领域，特别适用于智能座舱等对服务响应时间要求不高的场景。它最大限度地确保服务的可用性，但不能保证服务的响应时间能够满足要求。

3.2.1　服务重试风暴问题

服务重试能够应对网络拥堵等问题导致的服务调用错误，但重试的频率和策略必须慎重设计，否则可能引发系统中的服务重试风暴灾难。

服务失败重试的前提是故障可能在一段时间内得以恢复，但通常恢复时间不会特别快。如果服务调用方在短时间内频繁发起大量服务重试，可能会加剧故障。例如，如果网络拥塞导致服务超时，若持续进行大量服务重试，将进一步加剧网络拥堵，进而造成更大规模的服务故障。

此外，在多级服务调用场景下，设计不良的服务失败重试策略可能导致服务重试在服务链中迅速蔓延。例如，A服务调用B服务，而B服务进一步调用C服务。如果C服务的故障无法在短时间内恢复，A和B服务均采用服务失败重试策略以应对服务调用错误，当B服务在多次重试调用C服务后返回异常，A服务仍然持续尝试多次重试调用B服务，最终造成C服务的重复调用达到了多次。

3.2.2　等待时间随机指数补偿与限制链路失败重试

针对产生服务重试风暴的第一种场景，本方案采用了等待时间指数补偿策略，即服务发生错误后，等待时间为重试次数的指数幂：

$$t_1 = b^c \tag{1}$$

式中：t_1为服务错误后的等待时间，b为等待时间的基数，c为失效重试的次数。

然而，当网络在特定时刻发生拥堵时，大量服务超时，并在同样的等待时间后发起服务重试，极有可能再次引发网络拥堵，此种现象称为“服务重试碰撞”。

为了避免这种碰撞现象的发生，必须保证每个服务调用方的服务重试时间是不同的，因此将服务重试时间调整为：

$$t_2 = b \times k,\ k \in [0.2^c - 1],\ c < c_{max} \tag{2}$$

式中：t_2为避免“服务重试碰撞”的重试时间，重试时间t_2为等待时间基数b的k倍，其中k为$[0.2^c - 1]$区间内的任意整数，并且重试次数c小于最大重试次数c_{max}。

以等待时间基数为2 ms，最大重试次数4次为例：

a. 如果服务调用方首次调用服务超时，则在等待0 ms或2 ms发起服务重试。

b. 如果仍然超时，则在等待0 ms、2 ms、4 ms或者6 ms后发起服务重试。

c. 当重试4次后仍然无法正常调用该服务，则重试终止。

针对产生服务重试风暴的第二种场景，本方案采用了限制链路失败重试，即当一条调用链路中的A服务在尽力尝试服务失败重试后仍无法完成服务调用后，调用A服务的服务不应继续采用服务失败重试。

针对SOME/IP通信协议的具体实施方案是：

a. 设计统一的服务调用失败错误码，用来表示本服务已经采用服务失败重试，仍发生错误。

b. 任意一级服务在采用服务失败重试后，仍发生错误，将该错误码写入SOME/IP报文的Return Code，返回至上一层服务调用者。

c. 上一层服务调用者收到该错误码后，不会继续执行服务重试，直接向更上一层传递该错误码。

3.3　失败转移机制

3.3.1　失败转移策略

失败重试虽然可以在一段时间内可恢复故障情况下确保服务的可用性，但并不适用于故障无法短时间内恢复或对于响应时间要求严格的场景，如感知服务和路径规划服务。在这些情况下，长时间无响应可能带来严重后果。

为了解决这些问题，才引入失败转移策略，见图17。在这种策略下，在同一个服务部署多个提供方，这些提供方具备相同的服务功能，但位于不同的硬件平台。当单一硬件平台发生故障时，冗余的服务提供方可以确保服务的可用性。

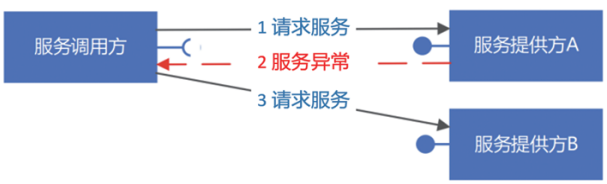


图17　失败转移

对于服务调用方而言，如果某个服务提供方在服务调用时发生错误，可以立即切换至其他可用的服务提供方，无需等待故障服务的恢复。这种方法有效提高了服务调用方在服务错误发生时的响应速度。

服务可用度的计算方式可以用于衡量服务的稳定性和可靠性。这个指标可以用数学公式来描述，其计算方式为：

$$A = \frac{T_D}{T_S} \tag{3}$$

式中：A为服务可用度，T_D为服务正常工作时间，T_S为服务工作总时间。

或者，可以使用以下公式来表示：

$$A = \frac{MTBD}{MTBD + MDT} \times 100\% \tag{4}$$

式中：$MTBD$为服务的平均无故障时间，MDT为服务的平均故障修复时间。

增加服务的平均无故障时间或减少服务的平均故障修复时间都能提高服务的可用度。引入服务冗余是提高服务可用度的有效方法。当服务采用1∶1的冗余时，只有当两个服务同时失效时，系统才会发生故障。因此，加入服务冗余机制后的服务可用度A_r为：

$$A_r = \left(1 - (1 - A)^2\right) \times 100\% \tag{5}$$

举例说明，假设初始时服务可用度A为90%，在引入服务冗余机制后，服务可用度A_r为99%。可见，合

理的服务冗余机制能够显著提高服务的可用性。

3.3.2　失败转移实现

失败转移策略代表着同一个服务有多个服务提供方(多个服务实例),服务调用方可以使用所有这些服务实例,并且可以自由地切换到另一个可用的服务实例。

通常,为了实现这种多重绑定,可以在服务的代理(Proxy)中使用多个服务实例句柄,见图18。每个服务实例句柄都对应不同服务提供方提供的服务实例。当服务调用方使用查找服务的方法(如FindService())时,需要指定查找所有服务实例,而不是特定实例ID的服务实例。在基于SOME/IP SD的服务发现中,可以通过将多个服务实例绑定到同一个AP应用的端口(Port)上,这样在调用FindService()方法时,只需提供相应的端口,就可以找到与该端口绑定的所有服务实例。

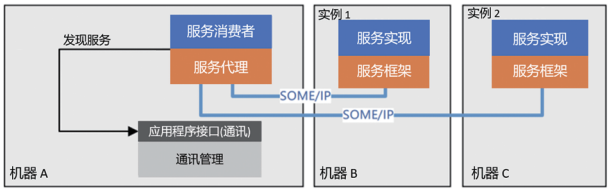


图18　服务多重绑定

这种方法允许系统中的服务调用方灵活地使用可用的服务实例,从而实现了失败转移策略,提高了系统的可用性和稳定性。

3.4　聚合调用机制

聚合调用是一种容错策略,与失败转移有所不同。在聚合调用中,服务调用方会同时向系统内所有可用的服务实例发起服务调用。只要有一个服务实例正常响应,系统就会立即返回结果给服务调用方,见图19。这与失败转移策略的最大不同在于,聚合调用不会等待单个服务实例的失败或恢复,而是只要有任意一个服务实例处于正常运行状态,即使其他服务实例发生故障,系统也能正常提供服务。

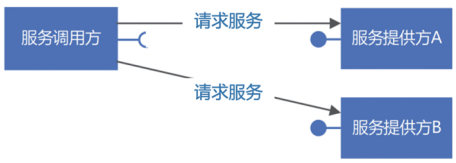


图19　聚合调用

这种策略适用于对服务响应时间要求非常严格的场景。通过同时发起多个服务调用,可以显著缩短服务的响应时间,因为只需等待任意一个服务实例的响应即可立即返回结果。这样可以提高系统的可用性和性能,并确保即使部分服务实例出现问题,系统仍能继续工作。

然而,聚合调用也可能会增加系统的负载和复杂度。同时对多个服务实例发起调用可能需要更多的资源,并且在某些情况下,可能会导致重复的、不必要的服务调用。因此,在选择容错策略时,需要根据实际场景和需求权衡不同的方案。

聚合调用在提高系统可靠性和服务时效性方面带来了明显的优点。它确实能够在系统中存在可用服务实例时,确保服务请求不会发生故障,并保持较高的响应速度。然而,这种策略也存在一些潜在的问题。

除了消耗更多的操作系统内存资源外,聚合调用可能会对CPU和网络资源造成更高的负载。饱和式的服务请求可能会消耗更多的CPU资源和网络资源,增加系统负担,对系统整体性能产生影响。在汽车软件中,这对于执行模块等对服务响应时间要求较高的模块来说,可能会对系统性能和稳定性产生影响。

在实施聚合调用策略时,需要平衡提高可靠性和时效性所带来的额外资源消耗,必须谨慎考虑系统的资源限制以及负载情况,确保所使用的策略能够在提高可靠性的同时,不会对系统整体性能造成严重影响。

聚合调用、失败重试与失败转移是三种服务容错机制,他们在消耗系统资源和代码入侵程度上各不相同,同时最终实现的容错效果也有所差异(见表1)。因此,在选择适当的服务容错机制时,应根据AP应用的重要程度进行权衡和选择。

表1　服务容错机制对比

	失败重试	失败转移	聚合调用
容错时间	慢	中	快
系统占用资源	低	中	高
应对故障类型	网络故障 短时可恢复的基础设施故障 短时可恢复的服务提供方故障	基础设施故障 服务提供方故障	基础设施故障 服务提供方故障
代码入侵性	高	高	低

4　测试与分析

本章将通过Pilot3.0控制器、PC机,搭建测试环境,对基于AP的AVP软件系统进行测试。并通过AP

监控方案，实现对AVP软件系统的监控。并对AVP软件系统中路径规划应用注入故障，以验证不同容错机制在不同故障场景下的表现。

4.1 测试环境

测试系统包含Pilot 3.0控制器、PC机、CANoe测试工具。Pilot 3.0控制器基于地平线征程三代芯片（Journey 3，J3）的自动驾驶控制器（图20），包含三颗J3芯片（下文分别称为J3A、J3B、J3C）和一颗英飞凌TC397芯片，每颗J3芯片拥有独立的存储器和外围电路，能够独立工作。控制器支持以太网通信，算力能够满足AVP软件系统的算力要求。具体参数见表2。

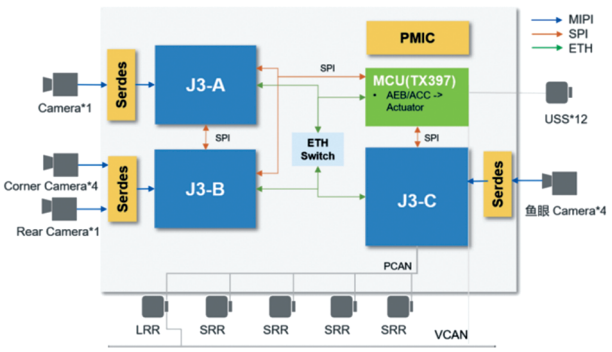


图20 Pilot 3.0控制器硬件框架

表2 Pilot 3.0控制器硬件参数

处理器	Journey3.0（4核A53） TC397/ASIL-D/TriCore CPUs UP to 300MHz
以太网	支持1路1000Base-T，标准RJ45接口 支持2路1000Base-T1两片SJA1105Q以太网交换机，支持1000Mbit/s以太网传输
存储器	每颗J3的存储器资源4GB LPDDR4 up to 3200 Mbps 64GB eMMC 64MB No.RFlash
操作系统内核	Linux version 4.14.87

PC机运行AP应用的测试工具CANoe软件和AP应用监控数据显示模块，参数见表3。

表3 PC机硬件参数

处理器	Intel i7-11800H16核心/32线程
以太网卡	Realtek PCIe GbE 1000M
内存	SAMSUNG DDR4 8GB×2 3200 MHz
硬盘	SAMSUNG SSD 1024 GB×1
操作系统	Windows 10

CANoe是VECTOR公司发布的专门用于汽车ECU开发、测试与分析的综合工具，其功能包括汽车分布式网络设计、仿真与测试，单个ECU的功能仿真与测试，通信协议的分析，支持CAN、以太网、SOME/IP、LIN等汽车网络协议。测试使用CANoe工具实现SOME/IP节点仿真和SOME/IP报文抓取，实现对AP应用的仿真与测试，版本为CANoe 15.0 (SP1)。

AP MICROSAR是VECTOR公司发布的用于AP应用开发的工具，主要功能包括了系统设计，代码生成，代码编译。且所有AP应用均在AP MICROSAR工具中完成开发，AP MICROSAR工具版本见表4。

表4 AP MICROSAR工具版本

基础代码包版本	r3.20.15
DaVinCi版本	2.3.41.r93818
AP规范版本	19-03

4.2 基于AP的AVP软件系统测试

在本试验中，在Pilot 3.0控制器和PC的虚拟中安装了AP必需的系统模块，并在Pilot 3.0控制器的J3A中安装了信息采集应用和感知应用，J3B中安装了行为规划应用和路径规划应用，PC中的虚拟机安装了控制应用，其中J3A、J3B与PC通过以太网交换芯片相连（图21）。为了对通信报文进行捕获，在PC中安装了CANoe软件。

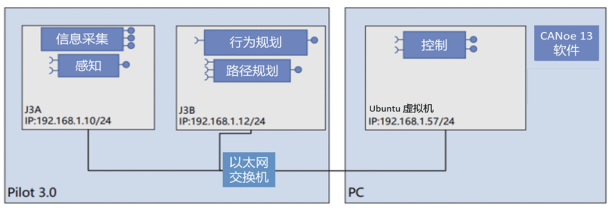


图21 AVP软件系统网络拓扑

4.2.1 AVP例程基本通信功能测试

Pilot 3.0控制器上电后，J3A、J3B、J3C分别进入操作系统，EM模块作为操作系统启动后的第一个AP进程运行，并在运行后将系统中应当启动的AP系统模块（SOME/IP守护进程等）和AVP软件系统的各个模块。PC中通过CANoe软件对SOME/IP报文进行捕获，如图22所示，显示了SOME/IP通信的部分报文，即AVP软件系统中所有AP应用发送的Offer Service报文，报文中包含了服务Id、服务实例Id、服务实例通信地址等信息。

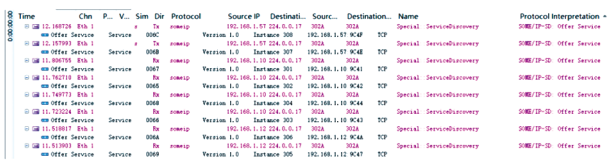


图22 CANoe捕获SOME/IP SD报文

J3A启动信息采集应用与感知应用，对外提供相应的服务。感知应用周期性发送代价地图的信息，代价地图的数据与Simulink仿真中的数据保持一致，以

验证部署到AP的路径规划模块与Simulink仿真中路径规划应用的结果一致性。J3B的行为规划应用和路径规划应用启动后，发送查找所需服务的SOME/IP报文，以定位服务所在的地址，订阅需要的Event。路径规划应用在接收到代价地图、当前位置等信息后，会根据部署的路径规划算法计算出到达下一目标位置的所有路径点和到达时运动方向，并将计算结果以RefPoses和Directions Event的方式发送到所有Event订阅者。PC中的控制应用启动后，订阅路径规划应用提供的RefPoses和Directions Event。控制应用对收到的RefPoses与Directions数值进行分析与Simulink算法仿真的计算结果一致。

通过上述试验结果可以验证，AVP中各个AP应用通过SOME/IP协议实现面向服务的通信功能正常，并且通过Simulink进行算法建模，生成C++代码，集成到AP应用的开发方式是可行的。

4.2.2　基于AP监控方案的AVP例程测试

将开发的AP监控方案的数据采集模块安装到Pilot3.0控制器中，虚拟机中安装监控数据分析与存储模块，PC中安装数据显示模块如图23所示。

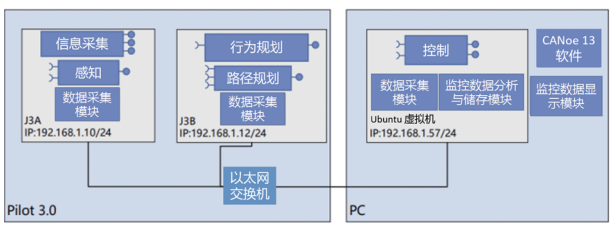


图23　AP应用监控安装方案

数据显示模块使用Qt工具开发UI界面（图24），支持通过LT协议与AP应用相连，或者从分析与存储模块获取离线数据，并以图形化的形式显示AVP例程中服务和平台相关的监测数据。

图24　AP监控数据显示模块UI界面（PC端）

AP服务监控功能实现了对Method响应时间、Event发送/接收时间间隔的信息采集以及显示功能，列表中显示了所有Method、Event相关的上报信息，并将信息加以处理，显示出Method响应时间和Event发送/接收时间的时间间隔的分布图，折线图中显示了上述两个变量随时间变化的曲线。

以路径规划模块提供的PathService为例，该模块中部署了路径规划算法，控制应用调用了PathService中GetRefPoses Method 200次后，该Method的响应时间分布见图25，从图中可以看出响应时间主要集中在232 ms左右。

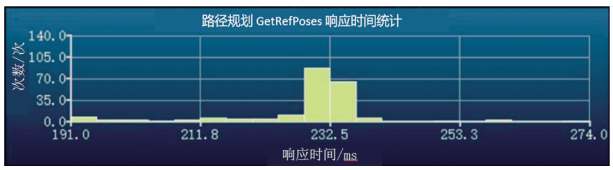


图25　GetRefPoses Method响应时间分布

基础设施监控功能显示了AP相关基础设施的工作状态，包括了网络延迟、CPU使用率、CPU负载和内存使用率的时间变化曲线。

通过对J3B的基础设施进行监控，从监控数据能够看出AP启动前后基础设施状态的对比。网络延迟在AP启动后未发生较大改变（图26），因为当网络中报文数量没有发生拥堵时，网络延迟不会有明显上升。CPU使用率从0%上升到了8.5%左右（图27），反映了AP和AVP应用启动后使用的CPU。CPU负载未出现明显的上升（图28），根据3.2节分析，当CPU使用率过高时才会引起CPU负载的上升，CPU使用率维持在8.5%水平时，不会造成明显的进程排队（CPU负载）。内存使用率反映了AP和AVP应用需要消耗的内存，从内存使用率曲线图中可以看到（图29），路径规划应用、行为规划应用以及AP总共使用了J3B 0.6%的内存，即24.5 MB左右的内存。

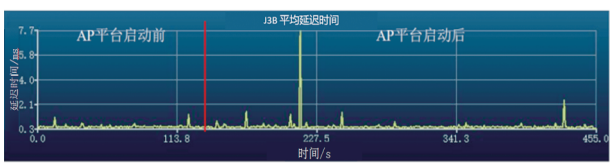


图26　J3B网络延迟

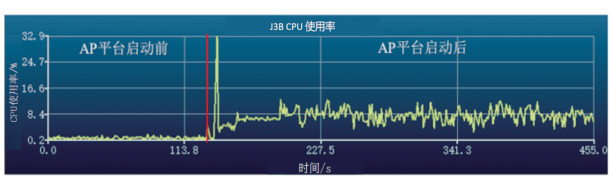


图27　J3B CPU使用率

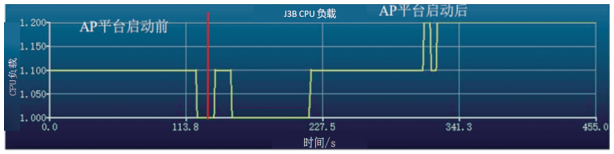


图28　J3B CPU负载

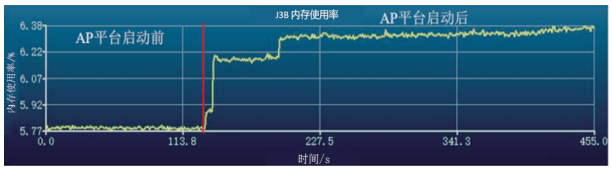


图29　J3B内存使用率

调用链追踪功能够根据调用链组件上报Dlt日志，分析出服务调用链信息。图30显示了调用链追踪相关的信息列表，包含了时间、Trace Id、Method Id等信息。图31显示了其中一条调用链的调用关系，即控制应用调用路径规划模块的GetRefPoses Method时的服务调用关系。从图中可以看到，控制应用调用该方法时，共计用时304 ms。GetRefPoses共调用了其他服务中的4个Method，其中RefCostmapGetter方法耗时最多，消耗了165 ms，原因是代价地图的数据比较大，数据传输时消耗时间较长。

	Time (s)	Trace Id	Span Id	Parent Id	Method Name	Method Id	Call Context
1	11.2868	624c5b7174b...	19495cff0000...	0	root	20019	ImplStart
2	11.2869	624c5b7174b...	19495cff0000...	0	RefPosesGetter	20019	CallStart
3	11.2871	624c5b7174b...	643c9869000...	19495cff0000...	RefPosesGetter	20019	ImplStart
4	11.2872	624c5b7174b...	643c9869000...	19495cff0000...	CostmapGetter	20013	CallStart
5	11.289	624c5b7174b...	643c9869000...	643c9869000...	CostmapGetter	20013	ImplStart
6	11.331	624c5b7174b...	643c9869000...	643c9869000...	CostmapGetter	20013	ImplEnd

图30　调用链追踪工具信息列表

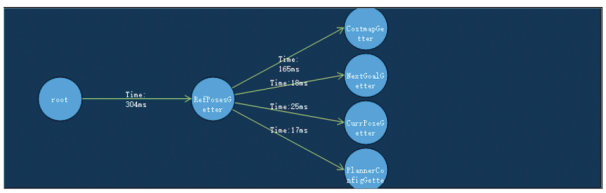


图31　调用链追踪工具显示的调用关系

4.3　AP服务容错测试

AP服务容错机制的测试中，将以路径规划应用为例，测试不同容错机制的表现将冗余的路径规划应用安装到J3C中（图32），AVP软件系统中存在了两个提供PathService的服务实例。从CANoe中抓取的报文中可以看到（图33），PathService（服务ID为006A）有两个不同的服务实例，服务实例ID分别为306和320。

AP服务容错测试通过对AP路径规划应用注入故障的方式进行测试。本试验对路径规划应用与冗余的路径规划应用，在一个故障注入周期内注入连续的服务故障时间，并且连续的服务故障时间会随机的出现在一个故障注入周期中。失败重试中最大重试次数设置为3次，等待时间的基数设置为5 ms。该试验中，因为路径规划应用和冗余应用有概率同时出现故障，失败转移和聚合调用也会采用失败重试，参数与单独使用失败重试时相同。当在1 s故障注入周期内注入100 ms服务故障时间时，试验结果如下：未采取任何容错措施时（图34），故障率为10%，符合本试验注入故障的时间比例；当采用重试的容错机制时（图35），故障率降至0，平均响应时间为228.767 ms，最长响应时间为599.253 ms，能够显著减少故障率；采用错误转移时（图36），平均响应时间为229.904 ms，最慢响应时间586.102 ms；。采用聚合调用机制时（图37），平均响应时间为221.808 ms，最慢响应时间为423.519 ms。

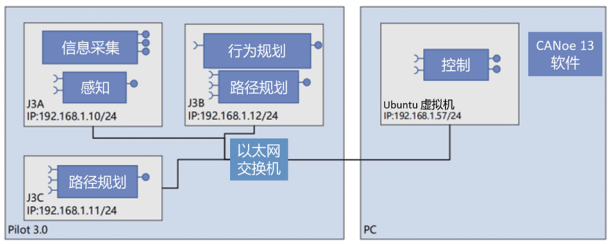


图32　加入PathService冗余服务实例的AVP软件系统

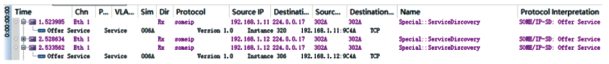


图33　两个PathService服务实例的Offer Service报文

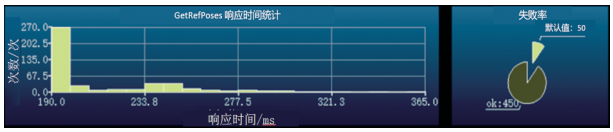


图34　未采取容错机制服务响应时间

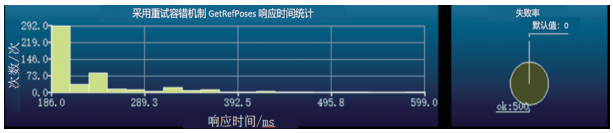


图35　采用重试的容错机制服务响应时间

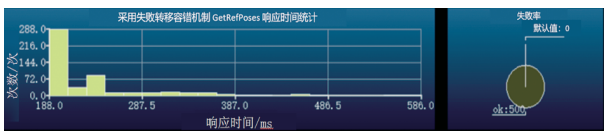


图36　采用失败转移的容错机制服务响应时间

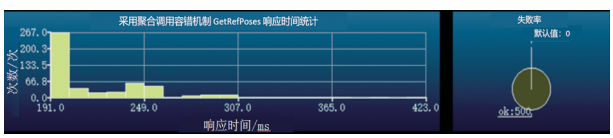


图37　采用聚合调用的容错机制服务响应时间

对PathService采用聚合调用的容错策略，需要新启用一个微处理器做冗余，并且每次都需要所有的路

径规划应用做运算，资源消耗很大。然而根据上述试验结果，聚合调用策略并没有起到特别优异的效果。通过对基础设施的监控发现，当采用聚合调用的容错策略时，会对网络造成非常大的拥堵。如图38所示，937.8 s后采用了聚合调用的容错策略，此时的网络延迟相比于其他容错策略高了很多，这是造成聚合调用策略Method响应时间很高的原因。

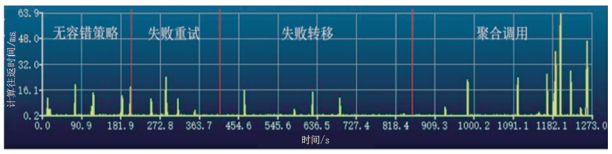


图38　不同容错策略的网络延迟对比

造成网络延迟升高的主要原因是，采用容错策略，服务调用方的每次服务调用都会对2个服务实例发送服务请求，网络负载将是未启用该策略时的2倍。CANoe软件捕获的PC上网卡的收发数据量表明，当采用聚合调用策略后，数据量提升了一倍。当路径规划应用与冗余的路径规划应用收到服务请求后，均会请求更多的服务，尤其是获取代价地图的方法（RefCostmapGetter），会传输大量的数据，当这些代价地图数据需要传输两次时，网络就有可能产生拥堵的情况。

为了对短时间不可恢复故障下不同容错机制的表现进行测试，本试验分别在1 s故障注入周期内注入100 ms服务故障时间；1 s故障注入周期内注入200 ms服务故障时间；5 s故障注入周期内注入500 ms服务故障时间；5 s故障注入周期内注入1 s服务故障时间4种情况下，对比不同容错机制的表现。结果如图39所示，通过采用服务容错机制，能够有效地减少调用服务时的故障率。

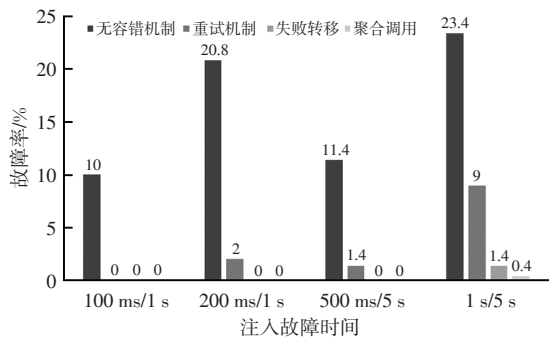


图39　容错机制故障率对比

5　结论与展望

5.1　结论

对AUTOSAR自适应平台采取监控与容错机制，能够提升故障分析效率，降低故障发生率，提高软件系统可靠性。研究了AP的工作原理与开发流程，以及适用于AP的监控方案和服务故障发生时的容错机制。总结如下：

（1）针对AP缺少监控机制、面向服务架构中故障定位困难的问题，以AVP软件系统为案例，提出了AP的监控方案。

（2）针对AP没有提供故障容错机制的问题，提出了基于AP的失败重试、失败转移和聚合调用3种容错机制。

5.2　展望

采用面向服务架构的AUTOSAR自适应平台为研究对象，对其工作原理和整体架构进行了分析，并以AVP软件系统为例，针对AP缺少监控机制、面向服务架构中故障定位困难问题以及运行阶段可能出现服务故障问题，设计了监控方案和容错机制，通过AVP软件系统验证了上述功能的可行性，但在下述方面仍有欠缺：

（1）监控方案采集了当前系统基础设施的状态以及服务的响应时间，能够分析出当前的系统状态。换言之，只有出现故障时，监控方案才能发现。进一步工作可以根据当前采集的数据，预测出未来是否可能发生故障，这样将能够在故障发生前采取相应措施。

（2）容错机制中采取服务冗余的方式能够提高服务的可靠性，然而冗余服务造成的数据量增加有可能导致网络拥堵。进一步工作可以考虑对网络结构进行优化，避免所有报文都需要从同一交换芯片进行转发，以避免交换芯片满负载可能会造成通信网络整体堵塞。

参　考　文　献

[1] 初士雨, 安涛, 王秋锋. 中国新能源汽车产业发展展望[J]. 科研, 2016, 17(7): 209-210.

[2] 刘佳熙, 丁锋. 面向未来汽车电子电气架构的域控制器平台[J]. 中国集成电路, 2019, 28(9): 82-87.

[3] 稀土信息编辑部. 国家发布《智能汽车创新发展战略》智能汽车正加速驶来[J]. 稀土信息, 2020(3): 33-35.

[4] 刘宗巍, 匡旭, 赵福全. 中国车联网产业发展现状, 瓶颈及应对策略[J]. 科技管理研究, 2016(4): 121-127.

[5] 贾文伟, 徐匡一, 王海波, 等.智能汽车电子架构分析与研究[J]. 时代汽车, 2020(4): 43-46.

[6] 孙娅苹, 田慧蓉.车联网网络安全标准化研究进展[J]. 电信网技术, 2017 (6): 18-21.

[7] 李兵, 赵磊, 林方圆.车载信息娱乐系统软件开发流程研究与应用[J]. 汽车实用技术, 2019(20): 3.

[8] 吴锴. 智能自动泊车系统研究[D]. 南京: 南京理工大学, 2008.

[9] BAYYOU D. Artificially Intelligent Self-Driving Vehicle Technologies, Benefits and Challenges[J]. International Journal of Emerging Technology in Computer Science and Electronics, 2019, 26(3): 5-13.

[10] TRAUB M, MAIER A, BARBEHÖN K L. Future Automotive Architecture and the Impact of IT Trends[J]. IEEE Software, 2017, 34(3): 27-32.

[11] FÜRST S, BECHTER M. AUTOSAR for Connected and Autonomous Vehicles: The AUTOSAR Adaptive Platform [C]//2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W), 2016.

[12] STOLZ W, KORNHAAS R, KRAUSE R, et al. Domain Control Units-the Solution for Future E/E Architectures? [J/OL]. SAE Technical Paper, [2010-04-12](2024-05-09). https://doi.org/10.4271/2010-01-0686.

[13] NAVALE V M, WILLIAMS K, LAGOSPIRIS A, et al. (R)evolution of E/E Architectures[J]. SAE International Journal of Passenger Cars-Electronic and Electrical Systems, 2015, 8(2): 282-288.

[14] CHEN S, HU J, SHI Y, et al. Vehicle-to-Everything (V2X) Services Supported by LTE-Based Systems and 5G [J]. IEEE Communications Standards Magazine, 2017, 1 (2): 70-76.

[15] YAQOOB I, KHAN L U, KAZMI S M A, et al. Autonomous Driving Cars in Smart Cities: Recent Advances, Requirements, and Challenges[J]. IEEE Network, 2019, 34(1): 174-181.

[16] 刘聪, 陈敏. 面向服务的电子电气架构研究与应用[J]. 北京汽车, 2021(6): 34-40.

[17] BARRY D K. Web Services, Service-Oriented Architectures, and Cloud Computing[EB/OL]. 2003[2024-05-11]. https://www.service-architecture.com.

[18] 于磊, 林宗楷, 郭玉钗, 等. 多服务器系统中的负载平衡与容错[J].系统仿真学报, 2001, 13(3): 325-328.

[19] 童蕾. 事件驱动的消息发布/订阅研究和实现[D]. 北京: 中国科学院研究生院(软件研究所), 2005.

[20] DONOVAN P. Payback Time for Network Management[J]. Telecommunications, 2000, 34(1): 71.

（责任编辑　**明慧**）

《汽车技术》征稿启事

《汽车技术》杂志是中国第一汽车集团有限公司主办的国内外公开发行的汽车前瞻与应用技术类月刊，为我国高质量科技期刊分级目录入选期刊、中国科学引文数据库（CSCD）来源期刊、中文核心期刊、中国科技核心期刊、RCCSE中国核心学术期刊（A）、俄罗斯《文摘杂志》（AJ）收录期刊、日本科学技术振兴机构数据库入选期刊、EBSCO学术数据库收录期刊、欧洲学术出版中心（EuroPub）数据库收录期刊。

《汽车技术》杂志以报道汽车整车及其零部件设计、研究、试验等方面的前瞻与应用技术为主，并兼有理论研究内容，是中国汽车行业核心学术和知识传播与共享的平台。

《汽车技术》将在国家提出的“创新、协调、绿色、开放、共享”发展理念的指引下，把握《节能与新能源汽车技术路线图》和“低碳化、信息化、智能化”的汽车技术主流发展趋势，努力在传统内燃机汽车高效动力系统、轻量化、低阻力领域，新能源汽车和互联智能汽车技术领域，大力吸收优质稿源，为广大科研和工程技术人员服务，为我国汽车工程技术创新能力提升贡献力量。

《汽车技术》欢迎高等院校师生、研发工程技术人员、技术管理人员及相关人员不吝赐稿，反映国家重点扶持项目、自然科学基金项目和其他重点项目等研究成果的稿件将被优先选择刊登。

投稿要求：

1.文章字数最好控制在6 000~8 000字范围之内；

2.请按科技论文要求撰写文章摘要，摘要中文字数控制在180字左右；

3.文章必须附有公开发表的、体现本领域最新研究成果的参考文献，且在文中应标注文献引用处；

4.文章主要作者应提供其简介，包括出生年、性别、职称、学历、研究方向及技术成果等；

5.来稿的保密审查工作由作者单位负责，确保署名无争议，文责自负；

6.请勿一稿多投；

7.本刊使用网站投稿，请先登陆网站注册成功后投稿，详细投稿要求见本刊网站中“下载中心”栏的“作者指南”，网址：http://qcjs.cbpt.cnki.net。

《汽车技术》编辑部