基于 Docker 的数据库微服务系统设计与实现

吴坤安 黄文思 韩泽华 黄屏发 李 芸 王荣烨 陈伟仑
(国网信通亿力科技有限责任公司 福州 350001)

摘 要:针对大数据时代数据库的读写速度以及移植性能急需得到提升的问题,开发了基于 Docker 的数据库微服务系统。首先,分析了 Docker 容器的技术原理、Docker 容器的架构、以及内部模块的功能,并与传统虚拟机相比,在速度、效率、管理等方面具有很大提升;其次,针对需要开发的数据库微服务系统所需要包含的功能,分别在 Redis、MongoDB 和 MySQL 3 种数据库中进行需求分析,并建立了上层人机交互界面包括数据层、管理层、功能层的架构,并根据需求以及架构进行了数据库微服务系统的开发;最后,采用实际计算机系统对开发的数据库系统进行功能以及性能测试,分析系统的读写能力,数据库功能,以及人机交互界面的测试结果,验证了本文所开发系统在实际应用中的有效性以及可靠性。

关键词:Docker; 数据库; 微服务; 系统
中图分类号: TP311.13 TN0 **文献标识码**:A **国家标准学科分类代码**: 520.99

Design and implementation of database micro-service system based on Docker

Wu Kun′an Huang Wensi Han Zehua Huang Pingfa Li Yun Wang Rongye Chen Weilun
(Grid Info-Telecom Great Power Science and Technology Co.,Ltd., Fuzhou 350001, China)

Abstract: The database micro-service system based on Docker is developed for the reading and writing speed of Database in big data times and the problem of the need to improve the migration performance. First of all, the paper analyzes the Docker container technology principle, Docker container architecture, as well as the function of the internal module, and compared with traditional virtual machines, in speed, efficiency, management has improved; Secondly, to analyze the requirements of the database micro-service system that needs to be developed, in Redis, MongoDB, and MySQL. The interface of the upper computer interface includes the data layer, the management layer, and the functional layer structure. And the development of database micro-service system based on demand and architecture. Finally, the development of the database system is performed by the actual computer system. The analysis system′s reading and writing ability, database function, and the test result of human-computer interaction interface verify the validity and reliability of the system in the application.

Keywords: Docker; database; Micro service; systems

0 引 言

目前,随着互联网、大数据以及计算机技术的飞速发展,采用 Docker 容器实现数据库微服务系统,可实现快速读写,提高启动速度,增强系统的移植性以及扩展性能,对于实现数据库的便捷管理与操作具有重要作用[1]。

近些年,已有部分学者针对 Docker 容器的优良性能进行了相关的研究。文献[2]首先对 Docker 的工作原理进行了分析,然后研究了其隔离技术的实现方式。文献[3]采用 Docker 轻量级虚拟化的移动特性,设计了一款开源信号具有私下交换功能的软件,用以防止窃听以及拒绝黑客的攻击,并通过实验验证了此开发系统的有效性。文献[4]提出了在 openstack 中创建 Docker 容器,提升了虚拟机的启动速度,CPU 以及资源使用率。文献[5]采用 Docker 容器技术,开发了 PaaS 平台,实际效果显示,该平台运行速度较高,资源使用率得到了很大提升。文献[6]基于 Docker 容器构建了 Web 集群,并通过实际测试,验证了此系统达到了扩容要求并能完成部署的工作。文献[7]采用 Docker 构造的容器,实现了进程监控,网络监控以及文件度量等功能,对系统进行功能测试,验证了设

收稿日期:2017-10

计的容器具有极高的安全性。

虽然针对 Docker 容器有了上述研究，但是目前还鲜有采用 Docker 容器开发的数据库存储微服务系统。由于 Docker 容器相比于传统虚拟机有更明显的优势，本文根据数据库微服务系统所需要的功能，设计了基于 Docker 的数据库架构，并根据相应的功能进行了开发，最后，根据实际实验测试了所开发系统在功能完备、处理速度、以及资源使用情况等各个方面的性能，结果验证了本文所开发系统的有效性以及可靠性。

1 Docker 容器技术

Docker 是一种容器引擎，只需要编译一次，就能很方便的在其他平台上进行移植操作[8]。Docker 的底层核心是采用 LXC(Linux Container)技术进行操作的，采用虚拟机运行程序使进程更便捷。Docker 使用 control groups 对 CPU、内存等进行管理，并规划管理和使用共享资源[9]。围绕 Docker 的核心技术实现的功能和相关项目逐步完善，本身的功能和易用性也逐渐增加，其生态系统如图 1 所示。

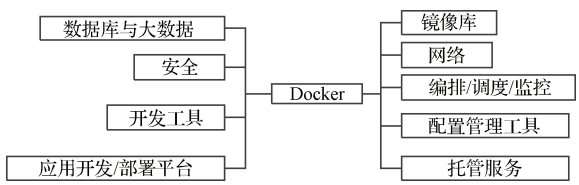

图 1 Docker 生态系统

1.1 Docker 的技术原理

Docker 属于分布式运行平台，应用 LXC 实现基本功能，LXC 采用虚拟机技术可以确保程序的隔离操作。Docker 主要包括 Namespaces 和 Cgroups 两个方面[10]。

Namespaces 又称命名空间，其可以实现各个进程互不干扰，都有自己的运行空间，保证每个名称下的进程都不会对其他的进程产生干扰[11]。

Cgroups 又称为群组控制，它的功能是对每个 Docker 的进程进行 CPU 和内存的分配。还可以对每个进程进行监控，实现资源的合理利用。其还能够根据各个进程的重要程度不同，制定控制顺序，保障资源分配与进程之间不发生冲突[12]。Docker 采用 client-sever 架构模式，其架构如图 2 所示。

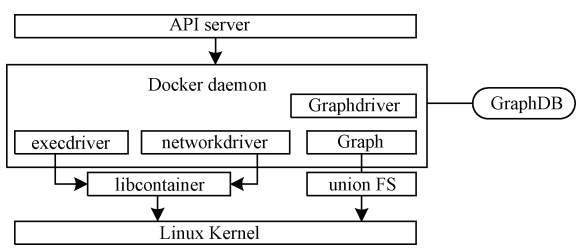

图 2 Docker 架构

由图 2 可以看出，Docker daemon 在 Docker 中处于核心地位。用于接收从外界输入的指令信息，并对传送进来的命令进行处理，分别送到相对应的模块当中、执行命令，还用于实现内部各个模块之间的通信，保障信息传递的有序性[13]。

Docker Daemon 中的 driver 用于制定运行环境。以 Graph 方式将 Docker 存储为镜像的模式，用于构建 Docker 容器。通过配置网络驱动以及运行环境实现 Docker 的网络环境构建。当制定 Docker 的使用权限和分配内存资源的时候，使用 execdriver 模块实现[14]。Libcontainer 用于实现各个 Docker 的分离运行以及对每个容器所占资源的控制。每个 Docker 属于一个独立的系统，当给予运行命令的时候，就可实现容器的运转。

Docker 包括 Docker Daemon，Docker client，Graph，GraphDB，Driver。

1) Docker Daemon

该模块处于容器后台程序的核心地位。当外界 API 指令传入该模块之后，执行指令，实现操作。

2) Docker Client

用于传送容器操作指令到 Daemon 模块。可以通过 Docker 命令行或者通过远程控制指令实现操作指令的传递。

3) Graph

用于保存镜像信息。对镜像进行各种处理和操作。

4) GraphDB

用于保存容器以及容器之间的信息。属于数据库操作模式，可以实现容器的增、删、遍历等过程。

5) Driver

将外界命令转换成系统调用，用于构建和管理容器。系统调用包括：容器(execdriver)、网络(networkdriver)、文件(graphdriver)。

1.2 Docker 的技术优势

Docker 由于本身特有的技术原理，使其优势明显。1)Docker 的启动仅需几秒钟，为系统的操作带来了极大的便利。2)每台主机可构建上万个 Docker 容器。使系统的资源得到了合理的分配，从而提高了系统的运行效率。具体体现在如下 5 个方面。

1) 节约时间和成本

Docker 可对宿主机的内核直接操作，从而使执行程序的时候所需要的资源更小，运行更快捷。当使用传统的虚拟机进行操作时，所需要的运行空间是 Docker 容器的上千倍。这样在一台主机上，几秒钟内可以实现上万个 Docker 同时执行[15]。减少了运行时间，并且单机运行的程序大大提高，降低了运行成本。

2) 高效性

Docker 只需要设置一次，就可以在平台之间移植运行。降低了开发和测试时间。Docker 可以直接访问操作系统，不需要中间的管理程序，使系统的利用率提高[16]。针对程序所需的资源，合理分配 CPU、内存，确保程序之

间的独立运行。

3）简洁的管理

Docker从外界接收的指令信息，在API上可以进行查看。当发生故障的时候，可以通过查看正在执行的命令，来确认故障点，实现故障的快速修复[17]。版本更新的时候，只需要对一小部分进行修改。

4）标准化应用发布

Docker使用镜像仓库实现任务的发布。在进程运行的时候，不需要虚拟机的安装及配置过程[18]。

5）Docker与传统虚拟机对比

Docker传统虚拟机对比如表1所示。

表1 Docker与虚拟机对比

特征	Docker	虚拟机
开启时间	s	min
硬盘容量	MB	GB
效率	接近原生	慢
系统支持数目	每个计算机上万个Docker	每个计算机最大几十个

2 基于Docker的数据库设计与实现

2.1 基于Docker的数据库功能需求

针对现在存储数据的类型有很多种，尽量使开发的存储服务系统能交互更多类型的数据。本文开发的系统分别使用了Redis、MongoDB和MySQL 3种数据库。分别都实现了数据库的备份、部署、以及管理功能，具体的功能需求如图3所示。

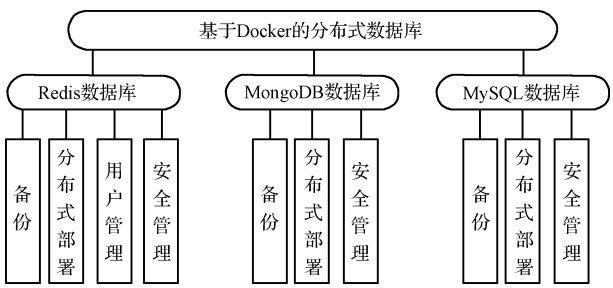

图3 基于Docker的数据库功能需求

在使用数据库的时候，人机交互界面所包含的功能要便捷，齐全。人机交互界面包含的功能主要包括：用户注册功能、系统登录、密码修改、退出系统、个人信息设置、创建数据库以及查看数据库信息等。

2.2 基于Docker的数据库架构设计

底层的Docker数据库系统包括平台层和应用层。其中，平台层的ZooKeeper用于故障诊断，排除故障干扰，继续工作[19]。Mesos和Marathon分别为资源管理和任务管理。应用层为Redis、MongoDB、MySQL数据库，通过平台层发送指令信息给应用层，实现功能[20]。

上层人机交互界面包括数据层、管理层、功能层。其中数据层用于管理数据库信息。比如，工作人员的信息注册、登陆、退出、功能使用记录等。管理层接收操作人员的命令信息执行相关的指令任务。功能层主要是对数据库的相关操作，包括新建、删除、保存等。人机交互界面架构如图4所示。

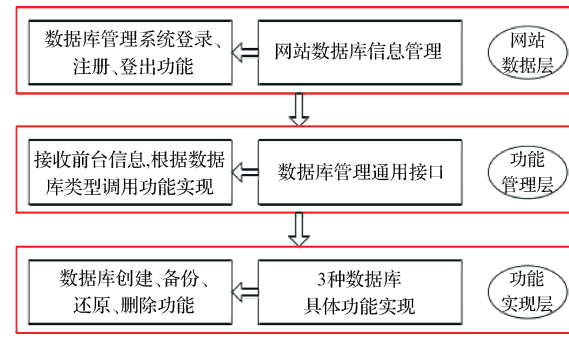

图4 数据库微服务系统人机交互界面架构

2.3 数据库微服务系统的开发

开发的基于Docker的数据库系统，人机接口界面主要包括用户注册、登陆、退出、个人信息填写、以及对数据库的相关操作。

当用户注册的时候，用户名要唯一，密码要求数字加字母混合才能注册成功。注册成功后，进入登陆界面，输入正确的账号、密码才能登陆系统。设置用户登陆后对系统能进行操作的权限。开发的Docker数据库微服务系统，人机交互界面如图5～9所示。

图5 数据库微服务系统登陆界面

图6 数据库微服务系统管理界面

图 7　数据库微服务系统个人信息界面

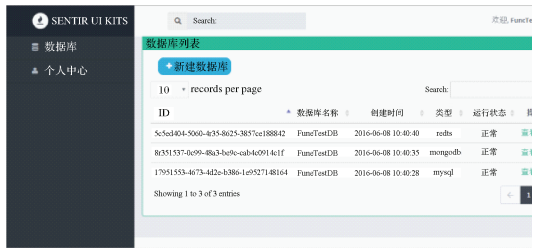

图 8　数据库微服务系统数据库操作界面

图 9　数据库微服务系统数据库信息界面

3　系统测试

3.1　测试环境

为了对开发的数据库微服务系统进行性能测试，本文对所开发的系统每个功能模块进行了实际实验验证。检测其功能是否满足设计要求。针对此开发系统的测试计算机共有 10 台，配置如表 2 所示。虚拟机的系统是 Ubuntu-Sever16.10。

表 2　数据库微服务系统测试环境

服务器	CPUs/个	内存/GB	硬盘/B
master1	24	32	1 T
Master2	24	32	1 T
Master3	24	32	1 T
Slave1	24	64	1 T
Slave2	4	16	500 G
Slave3	4	16	500 G
Slave4	16	64	1 T
Slave5	24	32	1 T
Slave6	4	16	500 G
Slave7	24	32	1 T

3.2　系统读写能力测试

3.2.1　实验测试

对开发的系统进行内存密集型测试创建基于 Docker 的 Redis 应用。设计其进程所占内存 1 GB，CPU 占用 0.5 个。虚拟机的内存设置为 1 GB，CPU 为 1。向系统发送 100 Bytes 的数据测试。测试结果如表 3 所示。对开发的系统进行 IO 密集型测试，测试结果如表 4 所示。

表 3　内存密集型读写速度测试结果对比

	Docker	Ubuntu
CPU/个	0.5	1
内存/GB	1	1
启动时间/s	＜0.2	＞30
写入速度/(条/s)	58 000	43 000
读取速度/(条/s)	61 000	45 000

表 4　IO 密集型读写速度测试对比结果

	Docker	Ubuntu
CPU/个	0.5	1
内存/GB	2	2
启动时间/s	＜2	＞30
写入速度/(条/s)	41	54.3
读取速度/(条/s)	2 971	1 936

3.2.2　结果分析

通过上述实验结果可以发现，无论在处理内存密集型数据，还是 IO 密集型数据，在外界环境配置相同的情况下，Docker 比 Ubuntu 的启动时间降低了很多，读写速度有了很大提升。主要是 Docker 直接对主机内存进行操作，而 Ubuntu 采用中转地址增加了处理时间。

3.3　数据库功能测试

3.3.1　实验测试

针对 3 种数据库是否能够正常接收执行任务进行功能测试。测试任务如表 5 所示，测试结果如图 10～12 所示。

表 5　3 种数据库功能测试任务

Redis	Master1	Redis-master
	Master2	Redis-slave
	Master3	Redis-sentinel
Mongo DB	Master1	Mongo DB-Primary
	Master2	Mongo DB-Secondary
	Master3	Mongo DB-Hidden
My SQL	Master1	Mysql-Master
	Master2	Mysql-Slave

将本文开发的系统与传统的 Ubuntu 的服务器进行

```
# Sentinel
sentinel_masters:1
sentinel_tilt:0
sentinel_running_scripts:0
sentinel_scripts_queue_length:0
master0:name=master1,status=ok,address=172.29.152.185:6379,slaves=1,sentinels=1
```

图 10 Redis 数据库监控信息

```
"primary" : "172.29.152.185:27017",
                "slaveDelay" : 86400,
"hidden" : true,
"me" : "172.29.152.187:27017",
```

图 11 MongoDB 数据库监控信息

Msater_Host	172.29.152.185
Master_Port	3306
Master_Log_File	master-bin.000001
Slave_IO_Running	Yes
Slave_SQL_Running	Yes

图 12 MySQL 数据库监控信息

性能对比测试，测试前后对比结果如表 6 所示。测试服务器每秒能够处理的信息数目，以及服务器每秒接收到的信息数目。

表 6 性能测试对比结果

实例数	Docker 容器		Ubuntun 虚拟机	
	数目/s	KB/s	数目/s	KB/s
1	9.6	1.6	5.2	0.6
2	12.9	2.1	7.8	1.2
3	64.7	11	26.3	5.2
4	65.9	12.8	27.6	5.6
5	76.8	13.7	29.1	6.1

3.3.2 结果分析

如图 10 所示为 Redis 数据库测试结果，测试结果为监控的 mastet 个数是 1。其他的是 master 数据的 IP 端口以及相应的节点数目。测试结果显示该数据库的读写功能正常，功能稳定。图 11 与图 12 的测试结果同样证明了 Mongo DB 与 MySQL 数据库读写正常。上述实验证明了本系统数据库功能的正常部署。从表 3、表 4，以及表 6 的实验测试对比结果中，可以看出，相同条件下，无论是内存密集型数据还是 I/O 密集型数据，Docker 容器的读入写出速度远远高于 Ubuntu 服务器，使用相同数量的服务器时，Docker 容器处理信息的速度和每秒接收的数据速度比 Ubuntu 虚拟机的速度具有明显优势。所以本文所开发 Docker 容器具有良好的性能，适于大数据时代，进行数据的高速处理。

3.4 测试结果分析

通过将 Docker 数据库微服务系统与 Ubuntu 的性能对比发现，本文开发的 Docker 容器无论在读写速度还是启动速度上，性能比传统虚拟机更加优良。测试的数据库人机交互系统，功能完善，能实现设置的各种功能，3 种数据库均能正常部署，实现读写操作。上述测试结果验证了此开发系统的性能可靠，功能完备，可以使操作员对数据库的操作更加便捷，方便管理。

4 结 论

基于 Docker 容器在处理速度、效率、资源使用率等方面具有的优良性能，针对数据库存储系统的功能需求，为了增强其移植性以及扩展性，本文开发了基于 Docker 容器的数据库微服务系统。该系统可以实现 Redis、Mongo DB、My SQL 3 种数据库的操作。开发的系统读写速度快，可移植性和扩展性较高、功能齐全，具备登陆、退出、数据库的建立删除等操作功能，安全性较高。开发的 Docker 系统由于快速，扩展性高等优势，为当前大数据的处理提供了可靠的技术保障。

参 考 文 献

[1] TIHFON G M, PARK S, KIM J, et al. An efficient multi-task PaaS cloud infrastructure based on docker and AWS ECS for application deployment[J]. Cluster Computing, 2016, 19 (3):1585-1597.

[2] 刘思尧，李强，李斌. 基于 Docker 技术的容器隔离性研究[J]. 软件，2015，36(4)：110-113.

[3] CHA B R, KIM J W, MOON H M, et al. Global experimental verification of Docker-based secured mVoIP to protect against eavesdropping and DoS attacks[J]. EURASIP Journal on Wireless Communications and Networking, 2017, 2017 (1):1-14.

[4] 张忠琳，黄炳良. 基于 openstack 云平台的 docker 应用[J]. 软件，2014，35(11)：73-76.

[5] 王亚玲，李春阳，崔蔚，等. 基于 Docker 的 PaaS 平台建设[J]. 计算机系统应用，2016，25(3)：72-77.

[6] 刘熙，胡志勇. 基于 Docker 容器的 Web 集群设计与实现[J]. 电子设计工程，2016，24(8)：117-119.

[7] 王鹃，胡威，张雨菡，等. 基于 Docker 的可信容器[J]. 武汉大学学报(理学版)，2017，63(02)：102-108.

[8] 马越，黄刚. 基于 Docker 的应用软件虚拟化研究[J]. 软件，2015，36(3)：10-14.

[9] 于烨，李斌，刘思尧. Docker 技术的移植性分析研究[J]. 软件，2015，36(7)：57-60.

[10] 张建，谢天钧. 基于 Docker 的平台即服务架构研究[J]. 信息技术与信息化，2014(10)：131-134.

[11] 陈清金，陈存香，张岩. Docker 技术实现分析[J]. 信息通信技术，2015，9(2)：37-40.

[12] 伍阳. 基于 Docker 的虚拟化技术研究[J]. 信息技术，2016(1)：121-123，128.

[13] 赵乐乐，黄刚，马越. 基于 Docker 的 Hadoop 平台架

构研究[J]. 计算机技术与发展,2016,26(9):99-103.
[14] 卢胜林,倪明,张翰博. 基于Docker Swarm集群的调度策略优化[J]. 信息技术,2016 (7):147-151,155.
[15] 陈存香,陈清金,张岩. Hadoop与Docker技术的融合[J]. 邮电设计技术,2015(5):5-8.
[16] 刘琳羽,南凯. 一种基于Docker的开发者服务平台设计[J]. 科研信息化技术与应用,2015,6(5):65-72.
[17] 杨鑫,吴之南,钱松荣. 基于Macvlan的docker容器网络架构[J]. 微型电脑应用,2016,32(5):58-60,64.
[18] 陈军相,李桂杰. 虚拟化及Docker轻量容器技术在高校图书馆中的应用[J]. 图书馆研究与工作,2016(5):52-56.
[19] 杨文林,谭曦,郭俊廷,等. Docker脆弱性分析与安全增强[J]. 信息安全与技术,2016,7(4):21-23,55.
[20] 彭勇,谢剑,童遥,等. 一种基于Docker的数据中心云平台实现方法及系统[J]. 中兴通讯技术,2017,23(2):60-62.

作 者 简 介

吴坤安,1987年出生,硕士研究生,主要研究方向为云计算和机器学习。
E-mail:2490333532@qq.com

NI推出基于Xilinx Kintex UltraScale技术的全新PXI FlexRIO架构

全新的PXI FlexRIO示波器与PXI FlexRIO协处理器模块相结合,无需通过自定义设计,即可提供自定义硬件解决方案。

2017年12月5日,NI(美国国家仪器公司,National Instruments,简称NI) 作为致力于为工程师和科学家提供基于平台的系统解决方案来应对全球最严峻工程挑战的供应商,近日宣布推出集成Mezzanine I/O模块与Xilinx Kintex UltraScale FPGA的全新PXI FlexRIO架构硬件平台。基于此全新架构推出的第一批产品,包含两款高分辨率PXI FlexRIO示波器、3款专用PXI FlexRIO协处理器模块,以及一款可协助进行自定义前端开发的模块开发包。

FlexRIO产品线将可自定义的I/O与用户可编程FPGA组合到可重配置的高性能仪器,用户可使用LabVIEW FPGA模块对其进行编程。采用Kintex UltraScale FPGA后,相较于先前基于Kintex-7架构的FlexRIO模块,全新的FlexRIO架构能提供更多可编程资源。此外,全新的Mezzanine架构可兼容单个3U集成式PXI模块中的I/O模块与FPGA后端。这些全新的FlexRIO模块采用PCI Express Gen 3×8连接,具有高达7 GB/s的数据传输带宽,因此能与机箱内的其他模块进行高速通信。

“FPGA与高性能数据转换器是不断突破探索与创新过程中不可或缺的技术;但要部署到自定义设计环境中,却又困难重重。”NI模块化仪器研发部门副总裁Steve Warntjes表示,“一直以来,FlexRIO都帮助工程师与科学家运用最新FPGA与转换器技术等现成解决方案,更快速将各种想法变成现实。全新的FlexRIO架构,进一步加速了这一愿景。”

全新的FlexRIO模块运用最高性能的FPGA与A/D转换器技术,非常适用于远程感应、信号情报、通信与粒子物理等高级应用。全新模块包括:

1)数字化仪模块-全新的PXI FlexRIO数字化仪无需牺牲动态范围即可提供高速采样率与高带宽。PXIe-5763与PXIe-5764分别提供500 MS/s与1 GS/s的采样率。两款模块皆可提供16位分辨率,而4个通道总共可提供400 MHz的带宽。

2)协处理器模块-提供专属Kintex UltraScale PXI FlexRIO协处理器模块,将实时信号处理功能纳入系统。内含这类模块的机箱,可为每个NI系统内的每U机架空间提供最高密度的计算资源。

3)模块开发包-先从LabVIEW可编程的FlexRIO FPGA后端着手,再设计自定义I/O模块,以满足更独特的应用需求。

FlexRIO是NI平台和生态系统的重要组成部分,可帮助工程师构建更智能的测试和测试系统。这些系统将受益于从直流到毫米波等不同工作频率范围的600多个PXI产品。它们采用PCI Express第3代总线接口,具有高吞吐量数据移动,同时具有亚纳秒级同步以及集成的定时和触发。NI平台受到一个由合作伙伴、附加IP和应用工程师组成的活跃生态系统的支持,可帮助工程师大幅降低测试成本,缩短上市时间以及确保测试装置能够适应未来需求,解决未来挑战。

阅读本技术白皮书,了解全新的FlexRIO架构如何帮助工程师建构更智能的测试与测量系统。