**存储产品系统测试实践之探索未知领域**

戴尔科技集团中国研发集团 中端存储团队 主管软件工程师 肖攀

# 前言

存储产品承载着各行各业用户的关键数据，数据越来越多地成为用户的关键资产。对于用户而言，数据的不可访问或者丢失都是极其严重的事故。系统测试是存储产品开发周期里重要的一环，它往往是把控产品质量的最后一道关卡。任何在系统测试阶段漏测的问题，都有可能在客户的使用过程中爆发，其可能带来的后果，这里就无需多加描述了。在本文中，我们将借此机会梳理存储产品系统测试中的挑战、目标、方法和实践，以探索产品中的未知领域。

# 系统测试的挑战

存储产品的开发是一个大型的、复杂的软硬件工程项目。系统测试也面临着极大的挑战。这里我们来用玩游戏打个比方：很多游戏采用的是地图探索模式，一开始整个游戏世界是一个还没有展开的地图。玩家从起始点出发，开始探索。世界在玩家的探索过程中逐渐展开。玩家在这个过程中开始逐步了解整个游戏世界的布局，并且可以找到各种道具和宝贝，逐渐升级。

游戏的过程往往是：一开始地图可以快速地展现开，玩家可以比较容易地找到各种宝贝。随着游戏的发展以及地图区域的扩大，后续的探索会越来越具有挑战，并且往往是要花费比游戏初期更多的时间和精力后，才能在地图上再找到一件宝贝。游戏后期找到宝贝的难度越来越大，但每件宝贝的回报也越来越高。

测试的过程和这个玩游戏的过程很类似：我们可以把产品里存在的所有bug看作是一副地图里所有隐藏的宝贝。在地图的起始地点，埋藏的宝贝比较集中、易于寻找。接着在地图的扩展部分，宝贝在哪里，都是随机分布的，分布的区域也越来越疏散。测试的早期阶段是功能测试：功能测试具有高度的自动化，对测试中的代码运行时序和状态可以做精确的定义和控制，高效的功能测试执行流程可以快速地发现产品bug，让产品达到可用的程度。这就好比游戏玩家在游戏的开始阶段迅速地探索出游戏世界的初步布局，宝贝可以一个不少地被找到。

功能测试之后就开始进行系统测试，也就是游戏的难度提升了一个档次。鉴于产品的复杂程度，究竟bug会隐藏在什么地方是未知的。有经验的测试人员可以通过设计的测试用例有效地发现产品问题，但是对于大型的软件项目，发现所有的bug几乎是不可能的。这就导致产品实际是带着一些隐藏问题发布的，用户在使用过程中，迟早会触发到这些问题。



图 1 测试的几个阶段

以戴尔公司新一代的中端存储产品PowerStore为例，它不仅提供了对最尖端的存储硬件支持，同时在支持数据存储功能的基础上，为用户提供了直接运行程序的AppsON功能。PowerStore的研发团队更是在数据的高效存储上进行了创新的架构设计，其中获取的相关专利超过百个，将产品的存储性能提升到了新的高度。产品的复杂度可想而知，测试团队面临的挑战也是前所未有的。



图 戴尔PowerStore中端存储

总结一下，系统测试的挑战在于：需要探索的未知领域过于广阔，探索的难度会逐步增加。对于系统测试里没有能力去探索的领域，其隐藏的问题最终会演变成用户使用过程中的DUDL事故（Data Unavailable and Data Loss）。

# 系统测试的目标

通过前面的论述我们已经知道，系统测试越到后面的阶段，测试的难度，或者说发现bug的难度会越来越大，耗费的精力会越来越多。那么，我们可以把系统测试的目标定义为：

***“在资源有限的情况下，用有效的测试方法，最大化地探索到系统的未知领域。”***

这个定义揭示出了系统测试中需要考虑的因素：

**资源有限** —— 这里的资源包括测试需要的软硬件资源、人力资源、时间资源；

**有效的测试方法** —— 系统测试的方法论、实践；

**最大化地探索** —— 系统测试的目标和动力。不断改进测试方法，更多地发现系统潜在问题。

# 探索未知领域的方法论和依据

由于未知领域的广阔和复杂性，依靠测试初期阶段的方法，即枚举和穷尽被测试模块的状态和逻辑分支来达到测试覆盖率，这几乎是不可能的。对于系统测试的方法，我们总结出以下三点：

**重复性（Repetitiveness）**：在测试的整个过程里反复地重复某些操作。例如对于文件系统的压缩选项，定期地设置为打开和关闭；

**随机性（Randomness）**：在测试过程里引入随机性。例如在给系统进行外部IO的压力测试中，随机改变IO的模式，包括IO的寻址模式、IO block的大小、data pattern的压缩率、重复性等。随机性也包括测试里随机模拟的用户操作，包括读写文件、删除文件、创建文件夹等；

**并发性（Concurrency）**：通过多线程和多测试主机来模拟出多用户对于系统的并发使用。并发性也体现在系统内多个功能的并发使用中，例如在打开文件系统的压缩和去重功能时，同时为文件系统配置定期的快照（Snapshot）创建，并且配置远程复制（Replication）将文件系统备份到其他存储系统。

由于以上三点的应用会带来太多的系统测试组合变化，因此我们需要为它们增加一个约束条件，或者说是一个设计测试场景的依据，即：

***“以用户使用场景为根，由此生发出测试场景和测试用例。”***

领域虽然是未知的，但是并不是在探索时到处瞎跑瞎撞。尽可能去了解用户实际的使用需求和使用方式，这个一定是我们设计系统测试场景的依据。以用户场景作为约束，用重复性、随机性、并发性作为扩展，在足够大的系统压力和丰富的配置下进行测试，这样在探索过程中发现的问题，才最有可能是用户在未来使用产品中遇到的问题。

# 探索未知领域实践 —— TRICK

我们在以往的文章里曾经提出过采用Kubernetes来设计测试框架的想法，请参见[《Kubernetes在存储产品极限测试中的应用》](https://zhuanlan.zhihu.com/p/335742775)。这是我们对于系统测试目标的一个回应。简单来说：

* 通过将测试程序容器化，极大地简化系统测试中多主机的配置和管理工作；
* 利用Kubernetes的容器管理能力，实现对于单一测试程序可定制化其测试压力；
* 综合管理和调度多个Kubernetes clusters，实现测试场景中的并发性和重复性。

通过一年多的实践，我们设计和开发出了一套基于Kubernetes的测试框架，命名为TRICK——**Test Running in Container Kubernetes**。

TRICK的系统框架如下图所示：

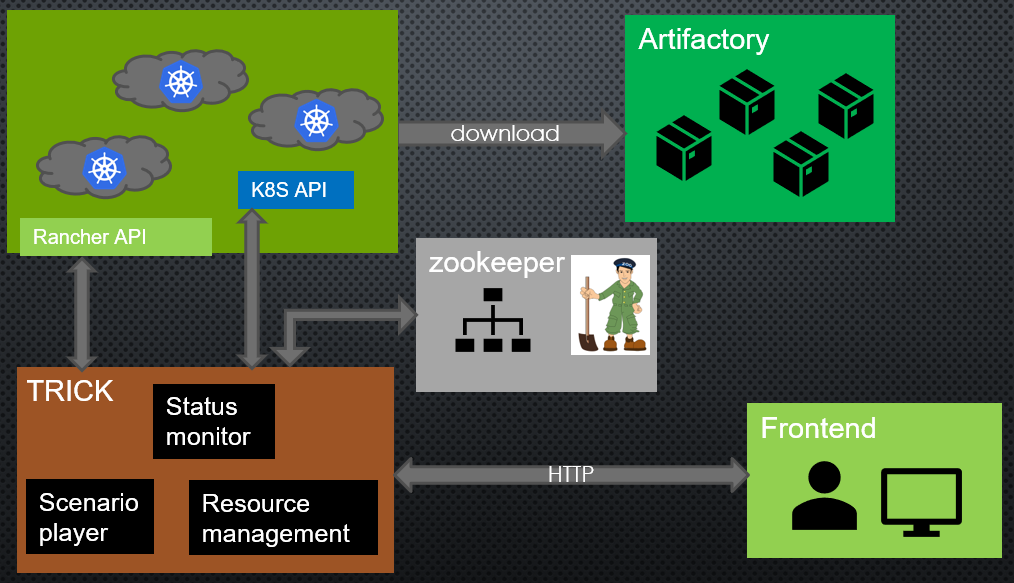


图 3 TRICK系统框架

TRICK由以下几个部分构成：

## Frontend

提供一个基于web的用户界面。测试人员选择需要执行的测试程序和需要测试的存储对象。同时一个重要的功能是选择测试程序Scaling的模式。Scaling模式包括三部分：

Play Mode：

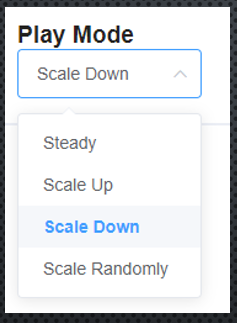


图 4 Play Mode选择

测试人员可以选择在测试过程中保持测试压力不变，或者逐步scale up、scale down，或者随机的改变测试压力。

Range and Interval：

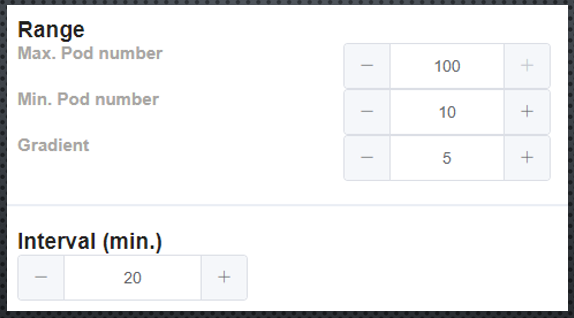


图 5 Range和Interval选择

Kubernetes以pod为一个调度单元来管理容器化程序。通过设定pod的数量，可以调节一个测试程序同时运行的实例，也就可以调整其测试的压力。测试人员可以指定pod的数量范围，每次改变pod数量的粒度，以及调整pod的时间周期。

Playback：

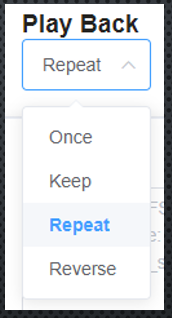


图 6 Playback选择

类似于歌单管理，当一次完整的测试场景执行完毕后，测试人员可以选择把测试压力保持在最后的状态，或者直接结束测试。也可以选择从初始状态开始，重新执行一遍，或者按照相反的scale模式执行一遍（scale up结束后做scale down，或者scale down结束后做scale up）。

通过以上三个选项的组合，我们可以在测试过程中设计出丰富的场景。三者的组合就是对重复性、随机性、并发性的实践。以下是TRICK中实现的几种scale模式的例子：

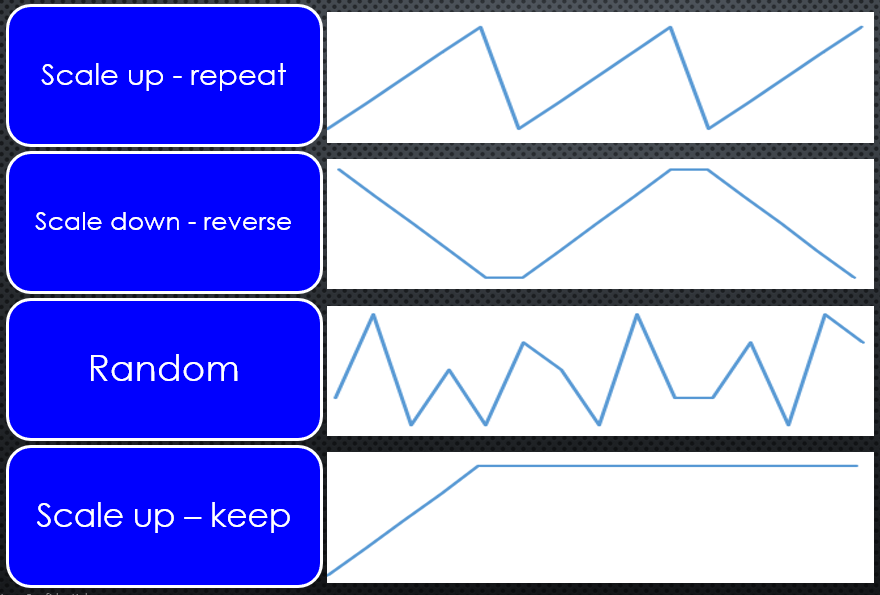


图 7 TRICK Scale模式示例

## Artifactory

用来保存容器化的测试程序image。测试开始时TRICK会下载并运行容器image。

## Rancher clusters

TRICK采用Rancher解决方案来管理多个Kubernetes cluster。

## ZooKeeper

ZooKeeper是Apache的一个开源项目，我们利用它来持久化测试中的数据，供各模块之间通信和调用。

## TRICK

框架的核心部分。TRICK利用Rancher和Kubernetes的API接口对多个cluster进行资源监控，在测试中负责将用户启动的测试程序合理地分布到各个cluster中。TRICK在测试运行期间会根据测试人员设计的场景，动态调正pod数量从而改变测试压力。例如在Scale up模式中，TRICK会周期性地增加pod数量，从而模拟出系统压力逐渐增大的场景。在Random模式中，TRICK会在程序运行一段时间后随机调整pod数量，从而模拟出存储产品使用中外部IO压力未知，随机波动的场景。

# TRICK的优势

在我们近一年的实践中，TRICK充分体现出了它的优势。为什么说TRICK可以帮助我们更好地去探索未知领域？这体现在以下几个方面。

## 化繁为简，测试环境的极简化管理

系统测试有别于功能测试，测试环境的搭建更加复杂。举个例子，为了充分模拟多用户对于文件系统的读写访问，以及产生足够量的IO压力，我们在一个测试用例中使用了超过20台主机作为IO客户端。对于测试人员来说，管理20台主机绝对是件恼火的事情。这里不仅有各类网络环境问题要解决，同时每台主机的测试程序版本管理、依赖包管理、路径设置、参数设置等等，都会耗费我们大量精力。使用TRICK后，测试程序采用容器化方式，启动几乎不需要花费任何额外的精力。让我们可以真正把精力集中在测试探索上，而不是花费在工具的配置上。

## 如愿以偿，探索方式从理论走向现实

在多年的测试总结中，如何更好地探索未知领域一直是我们在不断思索的问题。以往的测试框架并不能帮助我们去实现这一目的。测试框架在调度测试程序时，往往只能通过增加线程数量，或者配置多台主机来达到加大压力的目的，Scaling的方式极其单一。这就等于在未知领域中，我们始终迈不开脚步去触及到更多的地方。

使用TRICK让我们终于可以实现以往无法设计出的测试场景。例如，系统压力的逐渐降低或者随机波动。以往的测试程序和测试框架没有办法来达到的这样的场景。使用TRICK后，测试程序的开发者只需要专注于单一测试场景的开发，而测试程序的使用者则专注于不同测试场景的组合，以及定制每个测试程序的运行方式即可。

利用TRICK我们可以快速而轻松地搭建起一个复杂的测试运行环境。我们可以针对被测试的存储对象并发运行多个测试程序，并且对每个测试程序的运行模式尝试多种组合。测试程序的并发性和运行模式的多样性，带来的是对于产品代码更多逻辑路径的覆盖，从而有效地扩大了系统测试的覆盖率。

## 行云流水，探索之旅变得轻松有趣

TRICK的使用让测试过程变得非常轻量级。启动程序往往只需要一两分钟的时间，而过去一个重型的系统测试程序光是启动和各类自检就要花上很长时间。Kubernetes的使用还使得测试人员可以根据需要一键实现测试的暂停和继续。而以往的测试程序在开发时往往是没有考虑暂停功能的，程序只能从运行到结束，再到运行，这又要耗费大量的时间。

使用TRICK，测试人员可以尝试并快速地实现自己的设计思路，观察测试效果。测试人员不再被纠缠于各种环境和配置问题上，而是专注于对产品的思考，对测试方法的设计上。探索成了一件有趣的活动，而不是一个负担。

# 总结

在很多科幻小说里，对于未知，作者们都有着相似的看法：人们渴望生活在有序中进行，渴望拥有预测未来的能力，梦想着做每件事的决定都来自于对未知世界的预测。然而，对未知的预测实际成了对人类的诅咒。因为去除掉未知性其实就锁死了人类继续探索的动力和能力，人类文明反而会走向停滞和衰落。

我认为系统测试也存在类似的道理。我们使用自动化和其他各类方法，让测试更加高效、有序。而系统的未知领域，则是让测试人员能够不断进步的动力所在。我们始终要思考，怎么样可以再触及到以前测试中没有触及到的领域，从而发现更多系统的潜在问题。

TRICK是我们依据这样的目标设计而出的一个测试框架。通过近一年的使用，它展示出了它的优势，帮助我们在测试中发现了多个隐藏很深的关键问题，这些问题用传统的测试方法很难在短时间里得到暴露。它让我们对于产品的未知领域有了更多的探索，产品的质量也相应得到了提升。



图 8 TRICK是系统测试的改进和升级

测试方法的不断演进和创新，使得我们的存储产品在不断突破性能瓶颈的同时，能够实现对数据有效性和完整性的高可靠度承诺。在戴尔新一代中端存储PowerStore中，我们提出了6个9的可靠度目标：

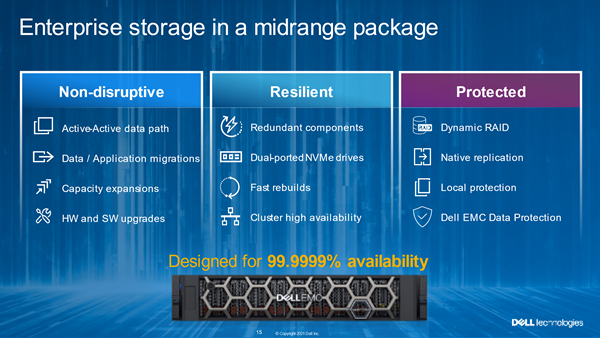


图 9 PowerStore的高可靠性

测试即思考，测试即探索。相信在探索领域的不断扩展中，更多的产品问题将会暴露出来，系统测试也就实现了自身的目的。