**Kubernetes在存储产品极限测试中的应用**

*Kubernetes是近年来最为火热的容器管理技术，而极限测试则在存储产品测试中扮演着至关重要的角色，为产品的质量保驾护航。这二者的结合将会对存储产品的研发带来怎样的创新？本文将介绍极限测试的背景和策略，极限测试所面临的技术难点和挑战，然后会介绍如何创新性地利用Kubernetes架构设计来攻克极限测试的难点，以及设计出更加灵活和高可扩展性的测试框架。*

# 存储产品极限测试介绍

Unity系列是Dell EMC主打的中端存储产品。Unity系列支持全闪存和混合闪存磁盘阵列，实现了对Block和File两类主流存储访问方式的统一维护和管理，同时提供了丰富的数据保护和压缩去重等功能，在全球拥有庞大的用户群。

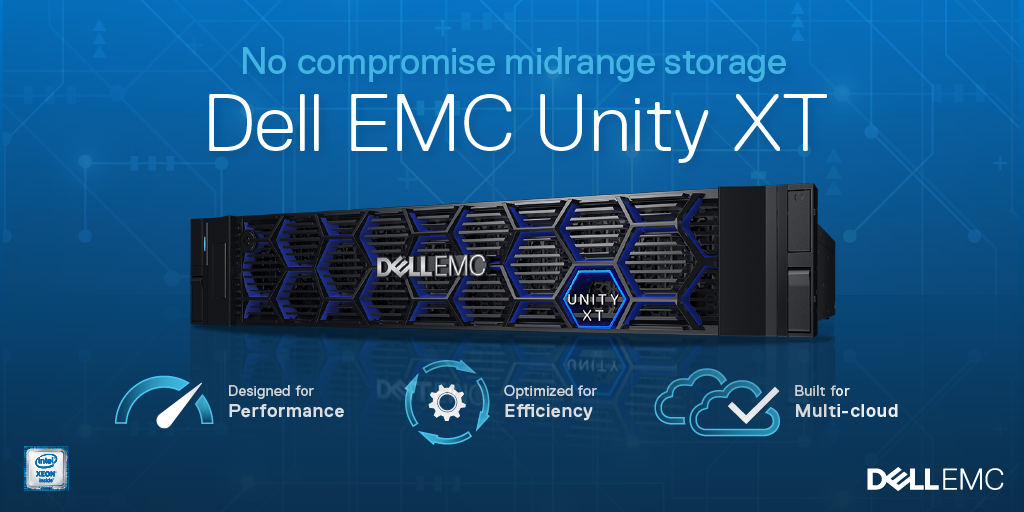


图 1 Unity中端存储

由于Unity在各行业和政府部门里所承担的数据承载业务，可想而知其高可靠性是用户所关注的产品特性里的重中之重。通过近几年我们对客户使用Unity产品的情景调查和总结，我们发现越来越多的客户会在实际使用时将Unity的处理性能推至极限：客户希望Unity可以支持创建更多的文件系统，处理更多的IO请求，及时响应更多的用户访问。。。因此，极限测试成为了保障产品质量的最后一道关卡。极限测试 -- Extreme Test，是通过测试来验证当产品的系统资源被推至接近或达到极限时，产品是否可以依然稳定地对外提供可靠的服务。这里所说的 “极限” ，除了通常服务器都会涉及到的CPU利用率，内存使用以外，对于存储产品，还包括以下三个方面：

1. 硬件资源极限。例如，系统支持的最大磁盘数量，可提供的最多FC光纤端口个数等；

2. 软件资源极限。例如，系统支持创建的最大文件系统个数，单个文件系统的最大容量，创建的文件快照（Snapshot）的最大个数，系统可以创建的远程复制（Replication）的session数量等；

3. 系统内部资源极限。例如，系统可以支持接受的最大TCP连接数量，系统的NFS或者CIFS协议最大用户访问量等。

对于第一和第二类资源，在产品说明文档里会有明确的标称，而对于第三类资源，用户在使用时并不知道明确的限制。极限测试在实施时，会对所有这三类资源的使用都进行覆盖。

极限测试对存储设备的软硬件配置要求高，配置过程复杂且漫长，测试用例的执行通常要求大数量的测试终端以模拟规模巨大的外部客户对存储设备上文件系统的访问，以及大并发的IO请求。

# 存储产品极限测试的难点和挑战

由上面的介绍可知，极限测试耗时耗力，并且对测试终端的数量要求也远高于其他测试类型。在测试资源数量有限，测试时间有限的情况下，极限测试需要高效合理地利用测试资源来达到全面的测试用例覆盖。经过我们多年的探索和总结，已经对极限测试的策略和执行形成了一套系统性的测试框架。然而，有一个测试分支领域，却一直没有能够很好的进行测试覆盖 -- 即对于系统承载负荷波动性变化的模拟和测试。

所谓波动性变化，是指存储系统的IOPS，用户访问量在推至系统处理极限，以及普通负荷之间做无规律的变化。这种场景下系统内部的运作机制和所触及到的代码执行路径，有别于系统长时间停留在极限压力或者普通压力之下，对于此类场景的漏测会导致潜在的系统问题在客户实际使用中暴露从而导致数据不可访问或其他严重问题。

通常的极限测试流程，是在测试中通过配置逐渐增加所要测试对象的数量，同时启动测试终端设备，模拟大规模的IOPS或者用户访问请求。因此系统的负荷是线性增加的，或者会长时间停留在极限负荷之下。而对于波动性负荷测试，一个基本要求是系统负荷不能保持不变，而是要在系统可承受的负荷之内可以做灵活的调整。这就要求测试框架做相应的调整：测试管理程序需要调度一次测试执行中所用到的测试终端，对运行中的终端或者进程可以灵活地配置其数量，监控其运行状态，随时可以进行数量的变化，即关闭或杀死运行的测试进程，以及随机地再启动不同数目的测试进程。整个测试过程中还需要监控每次停止或启动测试进程的状态，对异常场景或状态不正常的测试终端做容错处理。由于极限测试用到的测试资源较多，异常状态的发生并不少见。

除此之外，由于极限测试所有覆盖的用户场景广泛，因此测试中会使用到不同的测试脚本。这既有Dell EMC内部开发的测试程序，也有开源软件以及商业版的测试软件。对于每种测试脚本的系统配置要求，启动或终止的方式都有所差别。

由于以上的原因，波动性测试在之前一直没有很好地开展起来。其根本原因就在于现有的测试框架只适合于模拟系统负荷线性增加和单一停留在极限负荷状态，而波动性测试导致测试流程，调度方式的复杂性增加，现有的测试方式不能对其进行有效的支持。在有限的测试资源和紧张的版本进度要求之下，极限测试里的负荷波动性测试往往被忽略掉了。

# 基于Kubernetes而设计的极限测试框架

怎么来突破极限测试在面对波动负荷模拟时遇到的困境？答案就在于对Kubernetes的利用。K8S火遍全球已经有很多年了，本文不再对其原理和架构做过多描述。简单来说，K8S里业务的基本调度单元是Pod：一个Pod是K8S Cluster里一个基本运行实例，Pod里运行的单个容器化程序，或者多个紧耦合的容器化程序。K8S通过对Pod数量的控制从而实现业务的可扩展性。

Diagram

Description automatically generated

图 2 Kubernetes架构

对于K8S的典型运用场景，是将Cluster里的Pod作为承载业务的处理单元。例如，业务后端是Jenkins服务器，用户接收用户端发送的Job运行请求。Jenkins server可以将这些Job运行工作灵活分发到Cluster的Pod里执行。Pod实例随着客户端Job运行请求量的变化而作相应的调整。因此，K8S大多作为服务器端的后台业务承载来使用，而客户端则是具体的用户业务请求。

当我们利用K8S来设计一个全新的极限测试架构时，则是反其道而行之：用Pod实例来模拟客户端的用户请求，而服务器端则是我们需要测试的存储产品。我们发现，利用K8S的调度特性，非常贴切于系统极限测试的测试目标。具体表现为：

* 容器化技术以及Pod的调用，可以极大地简化客户端模拟测试程序的部署和规模调度。在一个Cluster里，模拟终端IO请求可以很轻易地推至存储系统承载极限；
* K8S提供了对于Pod运行实例的管理调度，Pod数量可以实时调整，出现故障的Pod或者工作结点会被及时侦测到并被新的运行Pod代替。而我们恰好可以运行这个特性来实现波动负荷的测试；
* K8S负责运行实例的调度，而与具体运行实例的内容无关。现有的测试程序，只需要稍作调整，负荷容器化程序的运行标准，就可以适配到K8S Cluster里，使得已有的各个测试用例，都可以升级成为波动负荷的新测试用例。

具体来说，基于K8S的极限测试框架如下：

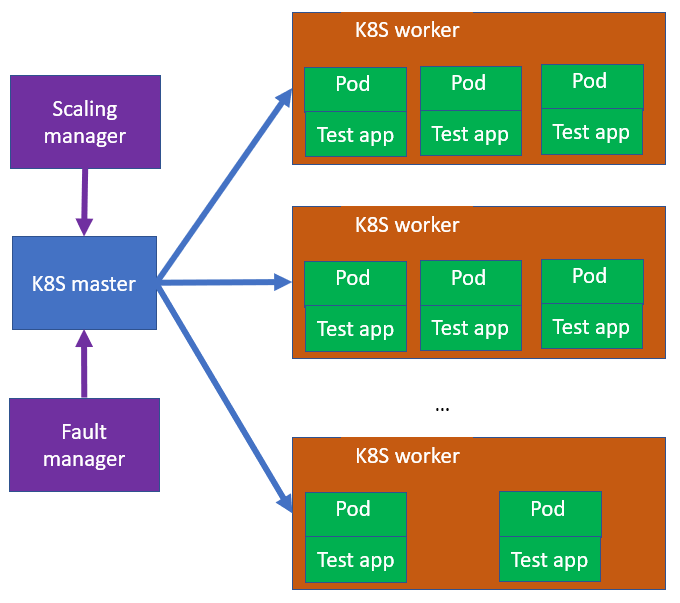


图 3 基于K8S的极限测试框架

可以看出，整体的设计框架是基于K8S的Cluster调度来实现的。框架里需要单独实现的模块如下：

**Scaling manager**： 负责测试中客户端负荷规模的调整。Scaling manager通过K8S的访问接口，根据测试场景的需求来调整Pod的数量，它可以在测试运行中定期地，或者随机地改变Pod数量 -- 大数量的Pod代表大规模的客户端IO请求或者文件访问请求，从而给被测试的存储系统增加了负荷。相反，小数量的Pod则带面客户端的访问请求下降。Scaling manager的实现只需要专注在如何调整Pod的数量，而不需要了解具体运行的测试脚本内容；

**Fault manager**: 负责在测试中注错。Fault manager在测试中可以定期或者随机的终止一部分Pod的运行。由于K8S的高可靠性，它会侦测到这些错误，然后重新启动新的Pod实例。而这正好实现了在对存储产品的访问当中，客户端意外终止，新的客户接入的场景。

**Test app**：具体运行的测试程序。除了需要符合容器化程序运行的一些基本要求外，针对极限测试， 测试程序需要有能力可以覆盖到最大的用户场景。例如，在一些大型跨国公司使用Unity产品的实际案例中，能够访问文件系统的用户数可以上万甚至十万。而测试在模拟用户对文件系统的访问时，由于Pod可能会被终止或者重新调度，因此测试程序在初始化时，可以以随机的方式来生成此次运行所使用的用户账号。这样当测试框架将测试程序进行调度并且长时间运行后，可以生成较大规模的用户访问记录，从而对存储产品端的相关功能，例如用户Quota设置，做大规模的测试。

在具体实施这个测试框架的过程中，我们也发现了由于我们对于K8S的使用方式有别于典型的使用场景，即我们是用Cluster里的Pod来模拟客户端，而把存储服务器作为要测试的目标，因此带来了一些新的问题。问题的描述和解决方案如下：

极限测试里的一个测试场景是在存储产品上创建大数量的文件系统直至系统极限，然后在客户端发起大量的文件系统访问请求，以测试产品是否可以在大规模客户端文件访问，且访问的文件系统数量接近系统极限时提供稳定的服务。而K8S对于在Cluster里挂载外部存储有比较严格的规定，在测试程序里通过NFS或者CIFS协议动态地挂载到一个文件系统是不被允许的。K8S只允许通过配置YAML文件，静态地配置外部存储挂载点，这显然不利于我们对大数量的文件系统的测试。我们的解决方案是：引入一个辅助的测试引导脚本，测试人员可以在此指定想要测试的文件系统数量和挂载点信息。引导脚本可以据此生成YAML配置文件，将所有挂载信息配置到文件里。这样当一个Pod实例启动后，其运行的测试环境会自动挂载所有的外部文件系统，测试程序根据测试场景，选择其中的一个或者部分挂载点进行IO读写和其他文件访问操作即可。

通过这套新的测试框架，我们现在克服了之前在设计波动负荷测试时所遇到的技术障碍。在对Unity存储端的系统资源的监控中，我们发现改测试方法有效地模拟了负荷的波动变化，从而弥补了之前测试中的缺漏，对于系统极限测试有了更好的覆盖。

总结一下，新的极限测试框架充分利用了K8S在容器化程序调度里各种优势，将其各项调度功能很好地匹配到测试需求里。对于终端模拟用户的规模变化和调整都交由K8S管理，而测试设计则把精力集中到具体测试程序的实现，以及规模调度的方式上。同时，由于使用Pod运行实例访问和测试外部存储系统是一种 “非主流”的K8S使用场景，新的测试框架也增加了对一些K8S使用限制的应对机制，从而让系统极限测试和K8S很好地结合在一起，让新的测试框架更好的为产品质量服务。