**微软开源rDSN分布式系统开发框架**

今天，由微软亚洲研究院系统组开发的分布式系统开发框架——Robust Distributed System Nucleus（rDSN）通过[GitHub](https://github.com/microsoft) 平台开源。rDSN旨在为广大分布式系统的开发者，学生，和研究人员提供一个开放式的框架，用于快速搭建和运维高性能和高鲁棒的分布式系统。

rDSN的思想来源于系统组过去在分布式系统开发和运维各个阶段进行的各种自动化项目。这些项目尝试帮助开发者（半）自动地测试，调试，优化，监控，扩展(Scale-out)，复制(Replicate)，组合(Compose)，甚至推理(Reason)。这些项目遇到了很多的困难，很多是由于开始的代码工作没有考虑这些需求，导致后期任务的实施事倍功半。rDSN提供了一个开发框架使得开发人员一开始也不需要考虑这些问题，但是代码会符合一定的原则，使得后期可以用很小的甚至零代价升级他们的代码来实现上面提到的需求。rDSN的一个早期版本在Bing里面得到了使用和验证。根据产品组的反馈，rDSN进行了改进并希望通过开源的方式对社区有所贡献，特别是对那些分布式系统的相关开发人员，学生和研究者有所帮助。以下是一些示例(相关教程在这里)，当然这些用途并不局限于相关的角色。

rDSN可以帮助开发者改善开发和运维体验，来提高系统的编程效率，性能和鲁棒性。从最简单的用法而言，rDSN可以看成是一个和主流RPC框架（比如Apache Thrift）等兼容的升级版RPC框架，或者是一个采用基于事件编程的高性能任务库。开发者设置rDSN为测试模式，来自动模拟各种各样的调度决策和系统错误，来测试系统在各种实际部署环境中的鲁棒性。当错误发生时，rDSN能重放发生的错误，并把分布式系统所有节点的状态放在一个进程里调试，并

来帮助开发者更快地找到问题的根

**For developers**, rDSN enhances development and management experience for system programmability, performance, and robustness. As its simplest form, rDSN can be used as an enhanced RPC library compatible to many others (e.g., Apache Thrift), or a task library where event-driven programming is adopted for high throughput. Developers can also configure rDSN into “test” mode, which tests the systems against various failures and scheduling decisions systematically, exposing early the possible bugs. Once a bug is exposed, you can switch to a “debug” mode to reproduce it. When it is online, rDSN provides automatic flow tracing and performance monitoring. If you are not satisfied with the default libraries in rDSN and want to use your own (e.g., logging or networking library), rDSN is open and you can easily integrate them. Even further, when you need to scale your service and make it reliable, rDSN replicates it with minor further development cost. In summary, rDSN provides and allows tools/frameworks to be seamlessly integrated with your system for good, and together tries to create an ecosystem as they are growing.

**For students**, rDSN provides a platform where you can easily understand and manipulate a distributed system. For instance, as rDSN adopts event-driven architecture, it generates a so-called “event matrix” which records the invocation count among named events, revealing the dependencies with weight inside the system. When learning distributed protocols, you can easily implement one atop of rDSN, and test it on its simulator. The simulator can abstract away many practical difficulties initially, and you can add them back gradually to evolve your protocol, such as from single-thread to multiple-thread, from constant message delay to variant ones, even with message lost. And there are a lot more potentials.

**Researchers** usually want to find and build something common to many distributed systems, such as runtime policies and diagnosis tools. rDSN provides a dedicated Tool API for that purpose. The API provides virtualization of all low level components, and exposes all non-deterministic behaviors from the upper applications at the event granularity. With this API, it is much easier to build reliable and effective runtime tools and/or policies. The current release contains a handful set of examples. Even better, rDSN ensures that those tools can always be seamlessly integrated with the upper applications - a big bonus for the research work to make real impact.

With all these possible benefit, it is hoped that the community can together build better distributed systems easily, by not only adopting rDSN, but also contributing back whatever built with rDSN to help the others. Visit the project now at <https://github.com/Microsoft/rDSN>.

缺乏一个类似于rDSN这样的开发框架，因此代码缺乏

符合一定的原则，

关于（半）自动化的去帮助分布式系统测试，调试，优化，运维，多机扩展，项目

rDSN首先利用和扩展了已有的RPC框架生成工具(如Google Protocol Buffer, Apache Thrift)来保证**可编程性**；和很多高并发系统一样(如nginx, Node.js, lighttpd)，采用事件驱动架构来达到对系统资源的充分利用来实现**高性能**。rDSN的不同之处在于，它同时还尝试解决分布式系统开发和运维的整个生命周期里的其他一些和鲁棒性相关的问题，比如**测试**，**调试**，**部署**，**自动扩展**，和**高可用**等等。开发者一开始通常对这些问题缺乏整体规划(如缺乏资源)，但是到后期它们会暴露并造成比较大的损失(如几个小时甚至几天服务不可用)，而开发者在系统代码基本完成后(Post-Programming)修复这些问题也经常事倍功半。rDSN因此提供了一个统一的开发框架，在尽量不增加开发者负担的前提下，帮助开发者**系统和（半）自动地解决**这些问题。

开发rDSN这个框架的想法来源于我们过去基于已有分布式系统的各种自动化工作，包括自动测试，调试，优化，运维，复制(Replication)，组合(Composition)等。在这其中我们遇到了很多困难，很多都是由于我们的工具没办法处理所有已有代码的状况 - 导致没法真正投入生产使用。我们尝试总结这些工具的共同需求，结果发现我们需要回答一个很基本的问题: **分布式系统带来什么样的困难，如何(半)透明地解决这些困难**。相比单机单线程系统，我们认为分布式系统的困难之处在于它引入了各种各样的系统复杂度，比如并发(Concurrency)，异步(Asynchony)，网络延迟(Network Delay), 丢包(Message Lost)，宕机(Machine Crash), 以及其它形式各异的故障(Fault)等。这些不确定性和它们的叠加决定了开发一个鲁棒的分布式系统相比单机单线程系统要困难许多，开发如上这些自动工具则更加困难。rDSN于是引入了两个方法来解决这个问题。

首先在单机方面，通过仔细定义编程模型和执行模型，rDSN能够**观察和操纵这些系统不确定性**。rDSN提供了两个开发API: 服务API(Service API)和工具API(Tool API)，前者用开发基于事件驱动的高并发应用，而后者则在事件驱动模型的基础上暴露各种系统不确定性来开发各种工具。比如，为了解决分布式系统调试超时的问题，我们有一个工具来模拟时间。另外一个工具模拟各种各样的调度决策和系统错误，来测试系统在各种实际部署环境中的鲁棒性。当错误发生时，它能够实施相同的不确定性决策，重播发生的错误，来帮助开发者更快地找到问题的根源。再比如，我们的自动复制工具(Replication)录制了在一台机器上发生的不确定性并通过复制协议把不确定性在另外几台机器上重播，保证了系统状态的一致性和发生故障时的高可用性。

第二在分布式方面，rDSN定义了一个一致的系统架构来**简化系统的复杂度**。在rDSN中，分布式系统由一组自包含的基础服务，以及在它们之上的用来处理端到端系统输入(比如用户请求)的一组工作流组成。开发者使用IDL(Interface Definition Language, 比如Google Protocol Buffer) 来描述基础服务的接口，和另外一个类SQL的语言来描述如何调用这些接口组成工作流。rDSN提供了工具来自动翻译这两个语言并生成基于事件的代码框架。开发者需要简单地把具体的应用逻辑写成一个个简单的事件，并使用rDSN提供的服务API来实现所有跨事件的以及不确定的操作，以保证相关工具能够正常工作。rDSN另外还提供了一组分布式系统框架，比如load blancer, partition manager, replication, 和workflow controller，来支持上层应用的分布式执行。这些方法使得rDSN对上层应用在跨事件，线程，和机器的所有依赖都有足够的了解和控制，这使得我们有可能实现可靠的和高端的开发和运维工具/框架/策略，来进一步保障应用的鲁棒性。

尽管提供了预定义的框架和编程模型，rDSN尽量和现有的常用分布式系统编程框架保持一致。我们移植了5个比较流行的库，发现代码的改动都在200~500行之间。另外，rDSN采用了"微内核“的架构，使得整个系统非常开发。比如， 所有基于工具API开发的工具，都可以和基于服务API开发的应用**无缝集成**。这带来了几方面的好处：一是开发者可以很方便地把自己开发的工具共享给社区，帮助其他应用变得更加鲁棒和高效。二是由于在rDSN中，实际部署使用的运行时库(比如网络，文件，锁等)也是一个特殊的工具，开发者因此可以很容易地集成他们自己喜欢的库或者在新的硬件平台上开发新的运行时插件(如基于RDMA的网络库)。rDSN目前已经包含了一组开发工具，一些运行时工具，和一些分布式系统框架。

此外，由于早期rDSN版本已经在Bing内成功使用和上线，现在的新版本也在产品组反馈的基础上进一步做了很多改进。相信一定会为正在开发，学习和研究分布式系统的人员带来很大的便利！