**Microsoft Makes rDSN Open Source**

Developed by System Group of [Microsoft Research Asia](http://www.msra.cn/zh-cn/default.aspx), Robust Distributed System Nucleus (rDSN), is now open source on [GitHub.com](https://github.com/). rDSN provides an open framework for those developers, students, and researchers, who work on distributed applications and frameworks, to quickly build high-performance and robust distributed systems.

rDSN leverages and enhances the existing RPC code generators such as Google Protocol Buffer and Apache Thrift for programmability; it adopts the event-based architecture to fully utilize the computation resource to achieve high performance, similar to nginx, Node.js, and lighttpd. The difference of rDSN, is that it also addresses the other issues important to robustness that occur throughout the whole lifetime of a distributed system's development and operation, such as test, debug, deployment, scale up/out, and high-availability. Developers usually don't pay enough attention to these problems when the projects start (e.g., lack of resource); however, they surface when the projects go and often lead to significant damage (e.g., service unavailable for hours or even days); even worse, the post-programming fix often does half the result with double the effort. rDSN therefore provides a holistic development framework, targeting at helping developers achieve all these goals systematically and (semi-)transparently.

The idea of this new framework arises during our past effort to automating the process of test, debug, optimization, operation, replication, composition etc. for the legacy distributed systems, with which we encountered lots of obstacles, mostly about the code continuously presents surprises to the automation tools and break them. We summarize the common requirements of all these tools and turned out we are going to answer the very basic question: what are the key challenges the distributed systems impose and how to deal with them (semi-)transparently. Compared to a single thread program, we believe the challenges are the new system complexities, such as concurrency, asynchrony, network delay, message lost, machine crash, and all kinds of different faults. These non-determinisms and their combination decide that it is much more difficult to build a robust distributed system compared to a single thread one, so are the automation tools. rDSN proposes two major techniques to address this challenge.

First on a single node, through disciplined design of its programming model and execution model, rDSN is able to monitor and manipulate these non-determinisms. rDSN provides two APIs: a Service API and a Tool API. The former is for developing high-concurrent service based on the event-driven architecture, and the latter exposes the non-determinism around the events to help developers write various tools. For instance, a tool can simulate the time so that timeout won't happen during debugging - an annoying pain-point developers usually have; or introduce various faults and their combinations systematically, for better exposing the possible bugs when the system is deployed in the real environment later; or capture the non-determinism on one node and enforce the same on the others, to replicate the state consistently on multiple nodes to achieve high availability even when machine crashes.

Second for the distributed part, rDSN also tries to reduce the system complexities by advocating a simple and consistent architecture via its programming model. In rDSN, a distributed system is composed of some self-contained services, combined with some workflows atop of them to handle end-to-end system inputs (e.g., user requests). Developers uses IDL (interface definition language) to define the contract of all self-contained services, and a SQL-alike declarative language to compose workflows atop. A code generator produces the code for both atop of an event-driven architecture. Developers fill in the business logics for the services as simple events and it is required that all cross-event operations (when needed) are implemented using rDSN's service API. A set of distributed system frameworks such as load balancer, partition manager, replication, and workflow controller are adopted to support the practical distributed execution of the generated code. By this means, rDSN takes full awareness and control of the application-level dependencies across threads and machines, enabling reliable and advanced tools/frameworks/policies to make the applications robust.

Although rDSN introduces pre-defined execution and programming model, it sticks to the common programming practice and porting legacy packages to rDSN is proved to be easy (code change for five popular packages is between 200~500 LOC). On the other hand, the framework is open in that it follows the "microkernel" architecture so the underlying components can be easily customized. For example, all tools developed using the tool API, can be seamlessly integrated with the applications atop of rDSN. This brings several goods. Above all, it makes easy for developers share the tools, and the application developers can benefit from all - creating a network effect. Second, rDSN considers the native runtime libraries used in real deployment, such as network, file and lock libraries, also tools, which means that developers can easily integrate their favorite libraries and/or develop new libraries for new scenarios, as new tools. The current version of rDSN already contains a set of development tools, some native runtime tools, and certain distributed frameworks for bootstrapping - all can be easily reused and/or replaced.

The original version of rDSN has already been successfully implemented and used inside Bing. With feedbacks from product teams, the updated version improves greatly. It is going to bring much more agility to those developers, students, and researchers who are going to develop, operate and/or study distributed systems.