A Closer Look at Android RunTime – ART. (Began with Android L)

With the latest I/O conference, Google has finally publicly announced its plans for its new runtime on Android. The Android RunTime, ART, is the successor and replacement for Dalvik, the virtual machine on which Android Java code is executed on. We've had traces and previews of it available with KitKat devices since last fall, but there wasn't much information in terms of technical details and the direction Google was heading with it.

Contrary to other mobile platforms such as iOS, Windows or Tizen, which run software compiled natively to their specific hardware architecture, the majority of Android software is based around a generic code language which is transformed from "byte-code" into native instructions for the hardware on the device itself.

Over the years and from the earliest Android versions, Dalvik started as a simple VM with little complexity. With time, however, Google felt the need to address performance concerns and to be able to keep up with hardware advances of the industry. Google eventually added a JIT-compiler to Dalvik with Android's 2.2 release, added multi-threading capabilities, and generally tried to improve piece by piece.

However, lately over the last few years the ecosystem had been outpacing Dalvik development, so Google sought to build something new to serve as a solid foundation for the future, where it could scale with the performance of today's and the future's 8-core devices, large storage capabilities, and large working memories.

Thus ART was born.

First, ART is designed to be fully compatible with Dalvik's existing byte-code format, "dex" (Dalvik executable). As such, from a developer's perspective, there are no changes at all in terms of having to write applications for one or the other runtime and no need to worry about compatibilities.

The big paradigm-shift that ART brings, is that instead of being a Just-in-Time (JIT) compiler, it now compiles application code Ahead-of-Time (AOT). The runtime goes from having to compile from bytecode to native code each time you run an application, to having it to do it only once, and any subsequent execution from that point forward is done from the existing compiled native code.

Of course, these native translations of the applications take up space, and this new methodology is something that has been made possible today only due to the vast increases in available storage space on today's devices, a big shift from the early beginnings of Android devices.

This shift opens up a large amount of optimizations which were not possible in the past; because code is optimized and compiled only once, it is worth to optimize it really well that one time. Google claims that it now is able to achieve higher level optimizations over the whole of an applications code-base, as the compiler has an overview of the totality of the code, as opposed to the current JIT compiler which only does optimizations in local/method chunks. Overhead such as exception checks in code are largely removed, and method and interface calls are vastly sped up. The process which does this is the new "dex2oat" component, replacing the "dexopt" Dalvik equivalent. Odex files (optimized dex) also disappear in ART, replaced by ELF files.

Because ART compiles an ELF executable, the kernel is now able to handle page handling of code pages - this results in possibly much better memory management, and less memory usage too. I'm curious what the effect of KSM (Kernel same-page merging) has on ART, it's definitely something to keep an eye on.

The implications to battery life are also significant - since there is no more interpretation or JIT-work to be done during the runtime of an app, that results in direct savings of CPU cycles, and thus, power consumption.

The only downside to all of this, is that this one-time compilation takes more time to complete. A device's first boot, and an application's first start-up will be much increased compared to an equivalent Dalvik system. Google claims that this is not too dramatic, as they expect the finished shipping runtime to be equivalent or even faster than Dalvik in these aspects.

The performance gains over Dalvik are significant, as pictured above; the gains are roughly a 2x improvement in speed for code running on the VM. Google claimed that applications such as Chessbench that represent an almost 3x increase are a more representative projection of real-world gains that can be expected once the final release of Android L is made available.