Choosing Your Runtime

By Rasmuss Graaf, Enea

A trend in today's telecom systems is to move functionality that previously used dedicated hardware towards a cloud environment with a common operating system and common hardware. In cases were the level of abstraction admits it, create virtualized network functions (VNF) that can be freely instantiated or migrated within the cloud environment.

For server like applications (e.g. HLR/HSS, MSC, RNC, and BSC), this transition has in many cases already been achieved or is ongoing. The potential benefits of centralized resources, such as increased hardware utilization, on demand capacity, etc. is pulling L2 and L3 functionality from edge devices (typically digital units for LTE, WCDMA and GSM) to centralized servers. There is however a contradicting need to be able to place functionality locally in the edge devices, e.g. in cases when the capacity of the communication infrastructure is insufficient.

There are several challenges finding a maintainable solution that allows features to exist both in a cloud and in an edge environment. The scope for this document is primarily around the migration challenges for applications and operating systems. The aim is not to give an answer to how to allow the migration of a certain use-case, it is to induce questions and considerations that are applicable for many.

While the complexity of the transition for an individual application may differ, there are a set of common main areas that are worth considering:

- Code migration
  Can legacy applications, fully or partly, be migrated to a new system or is a reimplementation needed?

- System characteristics
  Are there environmental expectations from the system and application that need to be considered?

- Performance requirements
  What should the system be capable of in e.g. upstart/restart time, internal/external communication, interrupt response time, jitter?

The relevance and content of each main area is likely to differ from application to application and its usage, e.g. a high overhead low effort migration solution for a scarcely used component may not be fitting for a frequently used component, as it may have large impact on needed hardware resources and power consumption.

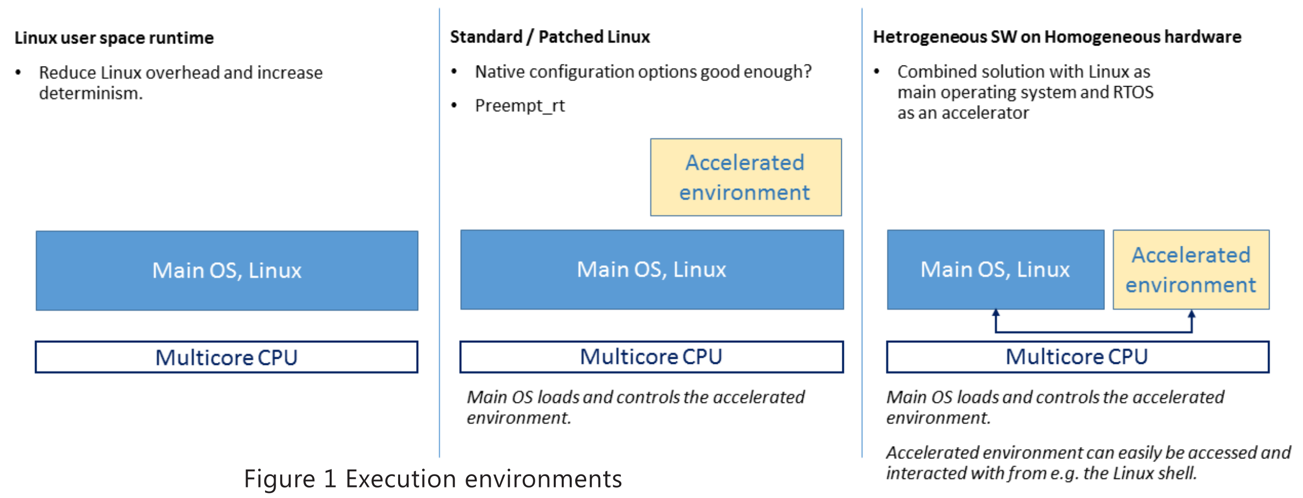
Linux user space runtime
Reduce Linux overhead and increase determinism.
Main OS, Linux
Multicore CPU
Standard / Patched Linux
Native configuration options good enough?
Preempt_rt
Accelerated environment
Main OS, Linux
Multicore CPU
Main OS loads and controls the accelerated environment.
Hetrogeneous SW on Homogeneous hardware
Combined solution with Linux as main operating system and RTOS as an accelerator
Main OS, Linux
Accelerated environment
Multicore CPU
Main OS loads and controls the accelerated environment.
Accelerated environment can easily be accessed and interacted with from e.g. the Linux shell.

Figure 1 Execution environments

The execution environment - overview

Many systems facilitating today's telecom infrastructure face a challenge with cloud migration. Often they have a large base of legacy software, designed for a hardware and software platform that have a very different set of characteristics and capabilities compared to the new environment. If moving to a cloud environment, the first step is to consolidate the various applications to be able execute in that environment.

Modern cloud operating system environments are typically Linux based and basing new applications on native Linux concepts is a natural approach. Consideration of the execution environment should be taken with the applications' requirements as input (see Figure 1).

- Standard Linux
  This is likely the favorable choice as using the mainstream kernel decreases future maintenance cost, though code migration and performance aspect should be considered.

- Patched Linux
  When the capabilities of a standard Linux solution does not fulfill the requirements, changes may be made to accommodate specific needs. Patched Linux solutions, however, come with certain drawbacks.

- User-space runtime in Linux
  Running an operating system API runtime in Linux user-space allows greater freedom to e.g. maintain legacy environment characteristics and increase performance for some use-cases.

- Linux as front end of a hybrid runtime
  Linux as the front end of a high performance heterogeneous environment allows more Linux feature and configuration freedom of choice, without sacrificing performance.

Standard Linux

There are many advantages with using a standard Linux solution, a few prominent aspects are:

- It leverages the open source community.
  Gaining access to common standards/de-facto standards.

- There is a large set of open source tools available.
  It leverages directly of the open source community.
  Gaining access to common standards/de-facto standards.

- Second source.
  Building a native Linux solution reduces software vendor lock-in effects.

The Linux environment configuration options are abundant and the performance focus can be tuned to match a number of use-cases., though the tuning suitable for some applications can decrease the performance of others. A few examples are:

- Latency vs. throughput:
  Having a highly responsive low latency system will increase the number of context switches and amount of rescheduling. The increased overhead and jitter can be very costly from a throughput oriented use-case.

- Performance vs. functionality:
  For high performance use-cases, superfluous functionality in the OS, or the features it requires, may have a negative performance impact.

- Scalability and jitter tolerance:
  Multicore scalability from the OS level basically boils down to the degree of cross core interaction, amount of shared resources and handling thereof. Cross-core interactions may cause jitter effects that ripples over to cores that are not involved in the direct interaction, e.g. by cache effects or handling of system wide interrupts like ticks.

Patched Linux

A patched Linux kernel may be applicable in cases were the capabilities of the native option do not fulfill the requirements.

A commonly known approach is the preempt_rt patch set, which allows the Linux kernel to act more like an RTOS by sacrificing performance in other aspects (e.g. throughput and overall system overhead). Although many patches are minimalistic, the preempt_rt patch is very large both in number of lines and impact on the system.

A patched Linux solution may resolve the need at hand, but the impact of using a patched solution should be considered. A few high level considerations are:

- Linux kernel version(s) availability for a patch may be limited.
  Patches are not maintained by the community, enablement for a new Linux version is up to the maintainer or the user to do. Depending on the patch, the upgrade path to a newer Linux kernel version can be a very cumbersome task.

- Patches may be incompatible with standard Linux kernel configuration options and features.

- Patches may improve the system's characteristics for some use-cases at the cost of others, e.g. system real-time capabilities vs. throughput.

User-space runtime

The user-space runtime allows for handling the application processes in user-space and reduce the non-productive cycles spent in the OS. This is achieved by minimizing or eliminating Linux kernel involvement, thereby achieving better determinism, reducing system overhead and enabling a number of other benefits a user-space runtime provides.

A multicore environment is preferred for maximum efficiency, allowing the user-space runtime to execute uninterrupted. Examples of use-case could be high determinism/low jitter tolerance, low interrupt latency, different/less costly scheduling, faster inter process communication, support for different programing environments, instantiation of legacy systems, etc.

A User-space runtime typically has a more limited scope than compared to the broad Linux environment and has a dedicated maintainer/owner which either has an existing roadmap or can be consulted for new functionality in future releases.

The amount of available development tools and features are dependent on the maturity of the user-space runtime ecosystem.

Hybrid runtime

The hybrid runtime provides the user with execution environments that are disconnected from each other, allowing requirements to be fulfilled that may otherwise be conflicting or mutually excluding.

One fundamental aspect of a hybrid runtime is the partitioning of system resources. This can be done in a more or less permanent manner by:

- Virtual machines:
  Partition the system's resources to fulfill the need of the virtual machine but keep the control of the virtual machine and its resources in the main OS.

- Bare metal:
  Partition the systems resources during the startup of the system. Typically the resources are partitioned during the main OS' initialization. Thereafter a second OS, or self-contained application, takes full control of its dedicated resources. Depending on the scope, reclamation/redistribution of resources can either be done by the applications in run-time or require a restart. As there is no main OS that owns the resources, care has to be taken to crash and recovery scenarios.

Virtual machines

The main operating system provides an emulated hardware environment for a guest operating system to execute in, basically setting up a customizable sandbox environment. The freedom of virtual machine content provides the user with a number of appealing options:

- Co-existance of applications with conflicting requirements on the same CPU:
  The guest operating system can be configured to fulfill the needs of specific applications without jeopardizing the performance of others.

- Minimize migration impact on legacy software and tools:
  The guest operating system can be the operating system used in the legacy system, preserving system characteristics and APIs. The legacy system can run in a hardware environment that resembles the legacy set up, e.g. number of cores, interfaces, memory configuration, etc.

- Instantiation and consolidation of legacy systems:
  Multiple instances, or types of systems, can co-exist with minimal, or no, disturbance. For guests containing applications with high performance requirements, a suitable approach is to dedicate hardware resources, e.g. core(s) to avoid the guest being swapped out.

- Sandboxed crash domains:
  If a fatal error occurs in a guest, it should only affect the content sandboxed environment. The main OS can thereafter take appropriate action, e.g. collect the error report, clean up and restart the guest.

- Security domains:
  The decoupling of the sandboxed environment and the main OS adds an extra security barrier, though it is up to the system design and configuration this barrier remains intact.

The benefits of a virtual machine based system are compelling, but need to be balanced by the drawbacks:

- Additional maintenance tracks:
  Resources must be spent on keeping maintainability and support.

- More complex configuration:
  Which resources should the guest be able to request?

- Tools availability and support:
  Depending on the system in the virtual machine, the availability of tools may be limited and the tool platform may differ from the one used for the host system.

- Resource inefficiency:
  A virtual machine adds an OS layer on top of the main OS, requesting a set of resources as it boots, e.g. memory, network, core(s) etc. The main OS knowledge of the system's resource usage is obfuscated, the resource usage by a guest is typically known only by the guest. Once a virtual machine has been given a resource, it is difficult to redistribute excess amounts.

- Hardware consideration:
  Hardware virtualization support is essential for the virtual machine approach to be viable without too much performance loss, unless it is acceptable to reduce the level of virtualization and tie certain functionality in the virtual machine with the host OS.

For edge devices running applications with conflicting requirements, the virtual machine approach can be a way forward. Though without hardware virtualization support, the impact on the host system to fulfill the needs of the guests may become too severe. In such case, a bare metal approach may be more suitable.

Bare metal
A definition of “bare metal” is in place, as the term has slightly different meaning and interpretation depending on the audience.

- Classic meaning of "bare metal":
  An application having exclusive, or high degree of, direct access to physical hardware resources without going through an OS as a resource administration layer. Allocation of physical resources can be achieved by a minimalistic hypervisor, e.g. Jailhouse.

- Cloud environment "bare metal":
  The need for direct physical hardware access becomes more apparent, as the cloud migration reaches further out into high performance systems. This is addressed in OpenStack has Ironic, enabling the cloud orchestrator to provide physical hardware access in addition to the typical virtualized environments. Thus providing “bare metal” access for the executing system, reducing overhead and allowing greater freedom in tuning the system.

The benefits and drawbacks for a “bare metal” system are to a large extent the same as for virtual machines but taken to the extremes. For system with hardware virtualization support, the virtual machine is, with exception for a few privileged instructions, running in a “bare metal” environment.

One scenario for hardware without hardware virtualization support is to divide the available hardware resources between a native Linux runtime with a high performance OS, or accelerated, runtime. In such case, Linux can boot up and initiate the hardware, then load the accelerated environment on the applicable cores before finally starting them. One aspect that must be considered here is the impact and handling of shared hardware resources and caches, as these typically have large performance impact if unfavorably configured. Another important aspect is crash and recovery scenarios, e.g. if it is possible to isolate a restart to one execution environment or if both need to be restarted.

During runtime, the operating systems will have a set of available interaction points, e.g. a file system accessible from both environments, cross core communication, shell access. If Linux is the main interaction point for the user, access to the accelerated environment can be made to have the touch and feel of a native Linux environment, as this is basically what it is, though with the accelerated environment executing performance critical applications.

Typically, adding new features in the kernel or additional functionality will negatively affect the Linux system’s performance and determinism. In high performance use-cases, the performance degradation tolerance will limit, or prohibit, the addition of new features. As the high performance use-cases are executing in their own environment, additional features or even changing the Linux kernel version could be done without jeopardizing the performance in the accelerated environment.

Partitioning of the system with a Linux part and an accelerated part will allow much more freedom in regards of Linux features and functionality. If choosing a proven in use single- or multicore RTOS as base for the accelerated environment, this will enable typical RTOS characteristics for the high performance partition, e.g. low overhead, high determinism, multicore scalability with no or minimal jitter, etc.

The execution environment – Enea’s presence

Enea has a long history of creating robust and high performing software solutions. With a prominent presence as a supplier to major telecom equipment manufacturers, Enea has proven knowledge and experience from situations where performance, availability, scalability, software maintainability and demanding support levels are valued.

Enea’s products are found in software platforms that enable executing applications to fully utilize the underlying hardware capabilities. Within telecom, Enea’s products are typically found in the connectivity and control layer.

A wide range of use-cases can be solved with one or more of Enea’s products, both exclusively or interacting with other vendor or community software. At the bottom line, what matters is which use-cases should be fulfilled, Enea can provide a robust base to fulfill those use-cases:

- Standard Yocto Linux (optionally with RT patch):
  Enea offers a Yocto based Linux distribution that can be tuned in accordance to the needed performance and functionality.

- User-space runtime in Linux:
  Enea can provide based user-space runtime solutions in for use-cases were the performance of the native Linux environment is not sufficient.

- Linux as front end of a hybrid runtime:
  Enea have much experience with virtualization, both from having developed hypervisors and researched higher level virtualization on top of Linux, e.g. KVM and XEN. In addition to virtualized environments, Enea can also provide “bare metal” solution for systems with requirements out of reach for a pure Linux solution.