A Generic RDMA API based on VFIO

Christoph Lameter

December 9, 2013

Basic Ideas

- Simplification and base on existing APIs and tools
- A provider interface is an implementation of a series of prescribed functions and is available either as
  - Open source: Header file and a set of source files
  - Proprietary: Header file and a linkable .o file.
- Provider uses VFIO and other Linux system calls to implement the functionality. System calls are expanded if generic functions are missing to realize certain functionality. No kernel dependencies aside from the use of new features that may be required by a driver.
- Should current kernel functionality not be sufficient then we may have to add what is missing.

Directly linking to a provider

- Provider header files are used by the application via #include “provider.h” allowing the use of inline functions and other optimizations.
- Resulting binary is provider specific.

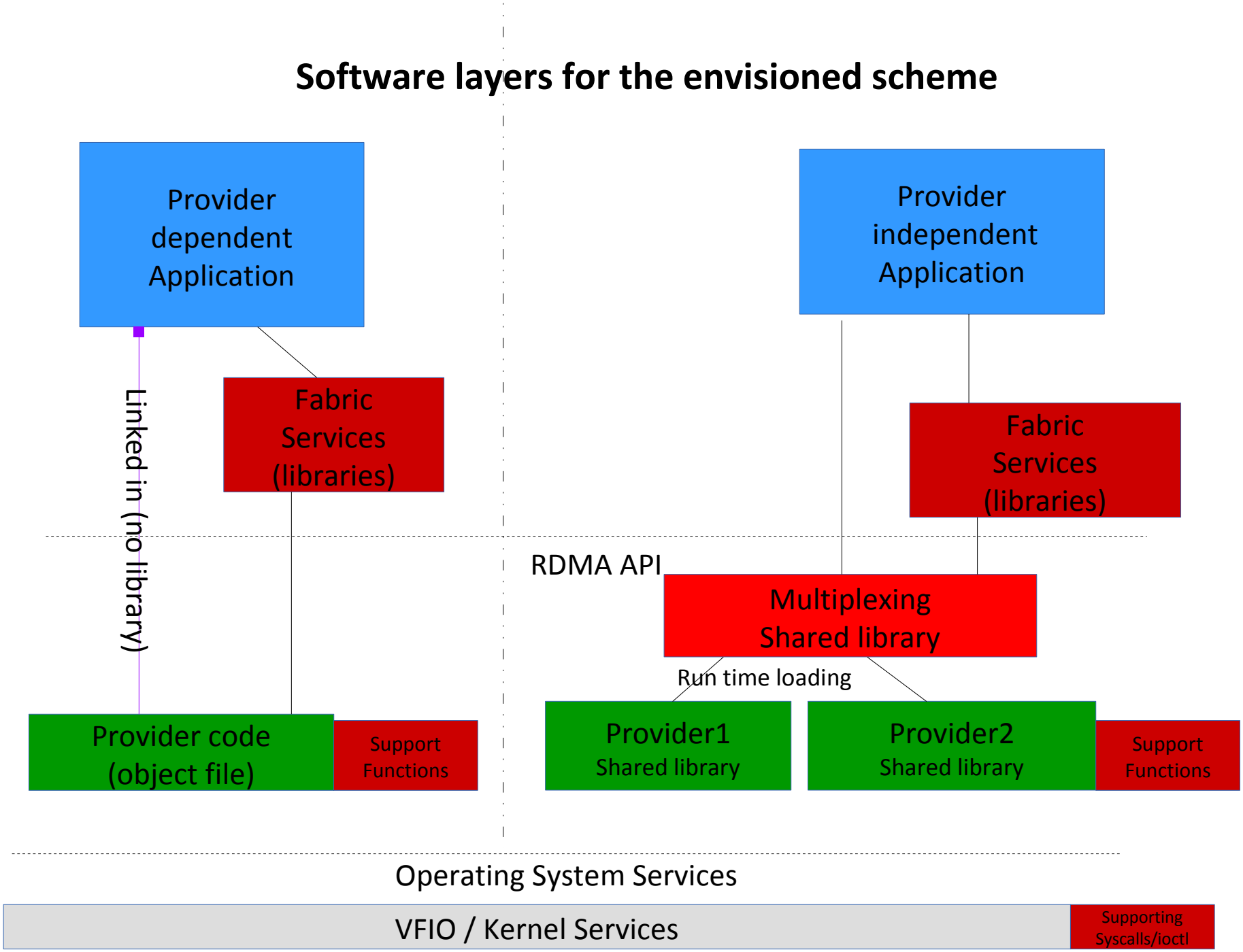
Sean Nunan

- The provider can provide extensions that are directly usable by the application (but result in the application source being provider dependent).
- The app can directly inspect and modify the hardware status if needed (also results in non portable applications).

Use of a multiplexing layer

- A multiplexing layer takes the .h and the .o file to produce an .so object that does not include inline functions.
- The multiplexing layer provides a way to select providers at run time. Which means that the application has to sacrifice some performance performance.
- The multiplexing layer exports an API that is closely compatible with the provider API. Ideally applications can be build/linked against the multiplexing layer without source code changes (unless vendor specific extensions have been used).
- The multiplexing layer provides a .so file and a header file to be used to compile and link into applications.

Software layers for the envisioned scheme



Higher level fabric APIs

- Build on top of the provider/multiplexing APIs.
- The APIs are provided using the libraries supported by the development environment of Linux
- Symbol versioning, API extensions etc etc are all supported.
- Easy debugging
- Large base of developers familiar with writing libraries.

Advantages for Applications

A) An application can be linked to a series of libraries that provide additonal functionality

B) The resulting code is kernel version independent and executes fully in user space.

C) Applications are portable and only depending on the stability of the Linux kernel API.

D) A highly developed toolchains exists for debugging, code analysis etc etc.

E) Applications can use the noise reduction approaches to limit OS noise. There are no kernel threads from the IB subsystem that may interfere.