Arrakis: The Operating System is the Control Plane

Nitish Paradkar, Andrew Quinn

EECS 589 Paper Review

1 Paper Reference

Peter S. *et al.* "Arrakis: The Operating System is the Control Plane," *Proc. of the 11th USENIX Symp. on OSDI*, pp. 1-16, 2014.

2 Paper Summary

The paper presents Arrakis, a new operating system that splits the duties of the kernel into two different parts. Traditional operating system kernels mediate all interactions between the application software and the hardware. As hardware has gotten faster, device I/O time has decreased significantly. As of today, most of the time spent in the interaction between hardware and the application software is in the kernel. Arrakis fixes this by allowing applications direct access to virtual I/O devices, thus bypassing the kernel on most hardware accesses. Arrakis also provides some of the same process isolation and rate limiting features provided by traditional kernels, but does this through the control plane with techniques such as packet filtering and rate limiting. The authors then evaluate Arrakis on four different workloads which include a read-heavy load pattern in a memcached system, a write-heavy load pattern to a Redis NoSQL store, an HTTP load balancer from various clients, and finally as an IP-layer middlebox. They show that Arrakis achieves up to 3x less latency on the memcached workload, up to 9x less latency on the Redis workload, 2x less latency on the HTTP load balancer workload, and up to 5x less latency on the IP-layer middlebox workload.

3 Highlights

The first thing we appreciate in this paper is that the authors take the time to highlight the amount of time spent in the kernel on most common I/O operations. They do this by running a UDP echo server to stress test the packet processing algorithms in the kernel. By doing this, they are able to show that around 70% of the time spent processing each packet is spent in the kernel network stack. The authors then also add a nice touch by showing how Arrakis can get rid of time spent in the kernel by directly delivering the packet to the user space (so they bypass the scheduler and kernel crossing delays experienced by the traditional Linux kernel). This then helps show they can achieve 3-9x less latency using Arrakis compared to a traditional Linux kernel. We feel like the addition of this data included helped make a strong case for Arrakis as it highlighted the problems with the traditional Linux kernel, and helped show how Arrakis could mitigate some of these problems.

We also appreciate how the authors used some of the concepts from SDN in the development of Arrakis. This can be seen as the authors separate the tasks of the traditional kernel into two separate parts. They re-engineer the kernel to perform as the control plane to perform tasks such as managing file naming, associating each application with certain directories and files, and installing rate limiters and packet filters. The data plane in the Arrakis system can then perform the tasks of multiplexing, filtering and I/O scheduling as a virtual I/O device. They use the separation of these kernel tasks to then show that the kernel can be bypassed on many common operations, and that the new kernel (control plane) only needs to be involved in the setup of these virtual I/O devices. Without the separation of the kernel into a control and data plane, the core benefits of Arrakis could not be achieved.

We also appreciate how the authors add POSIX bindings to the Arrakis system. This is especially useful as this version of the Arrakis system still provides the main benefits of the system while still maintaining an API that is familiar to a lot of programmers. This helps make a stronger case for the adoption of Arrakis as more developers could still port their applications over to Arrakis with little to no modification to their code base.

We were also impressed by the performance improvements provided by the Arrakis system. It provided significant performance improvements for every single test they performed and made a very strong case for using Arrakis over a traditional Linux OS. Arrakis was originally forked from the Barrelfish operating system, and something interesting thing to note is that Arrakis has officially been merged back into the Barrelfish as of August 2015 [1]. This makes it easier to pull and use Arrakis, and helps make it slightly easier to deploy.

We also appreciate that Arrakis ensures some of the protection provided by traditional operating systems. In Arrakis, these are provided through filters and rate specifiers. These can help ensure that each application only receives information from other applications it is interested in. The rate specifier also ensures that the OS can throttle certain applications or give priority to certain applications. Both of these mechanisms are managed through the control plane. As a result of this, none of the management or protection services offered by a traditional Linux kernel are sacrificed when moving over to Arrakis. This helps create a stronger case for the adoption of Arrakis over a traditional Linux kernel.

4 Possible Improvements and Extensions

It is often extremely challenging tease out the novelty in a paper such as Arrakis, because the work is building on a ton of prior ideas. For example, nearly two decades before Arrakis, Engler et al. proposed the exokernel; their paper's opening line is: "We describe an operating system architecture that securely multiplexes machine resources while permitting an unprecedented degree of application-specific customization of traditional operating system abstractions" [2]. In a similar vein, Balay et al. proposed Dune in 2012, which similarly provides applications with the ability directly access hardware in order to improve performance, while not having to perturb the entire operating system [3].

Arrakis is very similar to these projects, and I wish the related work section had done a better job differentiating Arrakis. For what it's worth, the improvements in Arrakis come from doing multiplexing and security checks for I/O in hardware; Dune and the exokernel improve application performance by allowing them to only implement the os features that are needed for a given application. One way to think about Arrakis is as an application of the exokernel given the rich feature set of modern networking hardware.

Another interesting question is the applicability of the Arrakis approach. They built Arrakis on top of Barrelfish, which is fine for a research prototype, but is a severe limitation for the adoption of Arrakis. One of the major advances in Dune is that it was built on top of linux; adopting Dune is extremely simple.

Finally, the hardware requirements in Arrakis may not be reasonable. They had to emulate the storage stack (VSIC), and in certain cases they had to emulate the network stack (when there are too many connections for hardware to multiplex). This emulation requires a separate core per hardware interface. If future hardware does not support the Arrakis requirements, this emulation may prevent adoption.

5 References

[1] https://lists.inf.ethz.ch/pipermail/barrelfish-users/2015-August/001398.html

[2] Engler, Dawson R., and M. Frans Kaashoek. *Exokernel: An operating system architecture for application-level resource management*. Vol. 29. No. 5. ACM, 1995.

[3] Belay, Adam, et al. "Dune: Safe user-level access to privileged CPU features." *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. 2012.