Arrakis: The Operating System is the Control Plane

CS380L

> Applications have direct access to virtualized I/O devices, allowing most I/O operations to skip the kernel entirely, while the kernel is re-engineered to provide network and disk protection without kernel mediation of every operation.
> *Peter, Li, Zhang, Ports, Woos, Krishnamurthy, Anderson, Roscoe,*
> *"Arrakis: The Operating System is the Control Plane"*

1 Movitation

- The authors make a classic efficiency argument: servers usually perform concpetually simple operations, but in practice this results in too much operating system overhead.
- The new secret sauce is that hardware device virtualization allows user-level programs to get efficient access to I/O without compromising protection.

1.1 Measurements

What a lovely and convincing display of quantitative information.

1. Study Table 1 (networking). What is Arrakis improving and how is it doing it?
2. How can Arrakis eliminate scheduling overheads? What if the machine has more user programs that want to process network input then cores?
3. How is Arrakis/P and Arrakis/N similar to the idea in the exokernel of POSIX-compliant applications and exokernel-optimized applications?
4. Study Table 2 (storage). What is Arrakis improving and how is it doing it?
5. Why does Arrakis measure its storage overheads with an in-memory file system?
6. Why do CPU overheads matter for the storage stack?
7. What is `fsync` good for?

SR-IOV is the hardware secret sauce. It allows software to setup flexible hardware multiplexing. One area to read critically is to compare and contrast how well SR-IOV is tailored for networking and for storage.

2 Design

"The user-level I/O stack can be tailored to the application as it can assume exclusive access to a virtualized device instance, allowing us to remove any features not necessary for the application's functionality."

- Trust yields efficiency.
- Give an example of this from Exokernel.
- Still can't trust user-level code with direct control of hardware. How does Arrakis address this problem? Exokernel?

2.1 Programming API

An interface card is the thing you stick into your computer's I/O slot that "talks" to a device for you. The ethernet jack (the network) plugs into your ethernet interface card. Sometimes this card is built into the motherboard.

Arrakis virtualizes these cards to provide an abstract programming interface that will remain portable across hardware implementations, while presumably allowing increased performance as hardware improves.

- **Network.** Queues. These specify memory regions to DMA (transfer) to and from the card.
- **Network.** Transmit and receive filters. Transmit filters prevent spoofing and receive filters allow hardware demultiplexing. Filters are flexible and can represents sets of machines and port numbers. What makes this safe?
- **Storage.** Virtual storage areas. This is a "large" area that can store multiple files and directories (or their equivalent). The large size allows easy protection and makes it possible for applications to control the layout of their own data and metadata.
- **Both.** Bandwidth allocators. Rate limiters, traffic shapers.

Discussion points.

1. Will these abstractions form a complete basis for programming? Geting better at thinking through this kind of question is what it means to get better at designing computer systems.
2. How will the control plane validate bandwidth allocation? What if one network flow wants to build up "credit" for a while, then use a lot of bandwidth? What if average bandwidth differs greatly from peak bandwidth?
3. Storage: What are the advantages and disadvantages of allow application-defined storage layouts? Will these be used to store a configuration file? What happens if a program has a bug and we need to edit it's custom-layout configuration? What if the bug is in the code that controls the virtual storage interface card?
4. Protection requires virtualized hardware, an IOMMU, and some elements of software policy (e.g., maximum allocations).

Access to file system data.

- Application library exports an RPC interface. How is this similar to microkernels? Exokernel?
- Per-VSA access (e.g., to virus scanner or backup system).
- Kernel-provided file system.

→ But the paper focuses on persistent data structures when it talks about virtualized storage areas. Why?

Asynchronous notification of events.

- "Doorbells are delivered directly from hardware to user programs via hardware virtualzied interrupts when applications are running and via the control plane to invoke the scheduler when applications are not running" How is this similar/different from ASHes in Exokernel?
- Doorbells implemented via events on file descriptors multiplexed via `select`.

3 Evaluation

One thing to look for in a paper is "workload engineering." This is the process of finding a workload that happens to show off the good case for your system. There is nothing wrong with a microbenchmark or two that shows off how your system has an advantage. But such evidence is flimsy until it is paired with evidence that these cases happen in practice.

The Arrakis paper is an example of good workload engineering (in my opinion). Many of their benchmarks are not complex (GET/SET key value store, HTTP load balancer, IP-layer middlebox), but they are important and the authors show significant performance wins for reasonable configuration. One can still reasonably question how Arrakis would perform for more substantial workloads, e.g., a full-featured web server. Serving a static 1,024 byte file out of main memory is NOT representative of modern web workloads.

The authors tune their Linux baseline using the latest device driver and disabling receive side scaling on a single kernel (improving throughput by 10%). As Ron Burgundy would say, "Stay classy San Diego."

Limit study. In section 4.1, the authors conduct a limit study. A standard quantitative evaluation likes to claim, our system X is 30% or 5× better than system Y. This can be useful, especially if system Y is widely deployed, and the measurement is on a useful program. But an important question (that can be difficult to answer), is how close is my system X to the best possible system? Trying to bound that

Sensitivity study. Another concept in this domain is the sensitivity study. Many systems have "magic numbers" or parameters that must be set in order to achieve good performance. Sometimes a system having these numbers is described as a good thing (e.g., "it allows greater control for applications"). Sometimes it really is nice to have some control knobs, especially if reasonable default values are easy. But managing systems is complex enough as it is, most administrators do not want to be required to actively tune their system.

A sensitivity study allows an author to say, yes we have this parameter, but look, the exact value you set it too doesn't have a huge impact on performance. That might make the existence of the parameter easy to swallow because it is easy to set a default and the default won't have to be changed for every successive hardware generation.