**Arrakis: The Operating System is the Control Plane**

*Simon Peter, Jialin Li, Irene Zhang, Dan R. K. Ports, Doug Woos, Arvind Krishnamurthy, and Thomas Anderson, University of Washington; Timothy Roscoe, ETH Zürich*

**Link:**<https://www.usenix.org/conference/osdi14/technical-sessions/presentation/peter>

**Summary:**

The main motivation behind the research was to build an Operating System suitable for data centers. In traditional operating systems such as operating systems with a monolithic kernel like Linux, the kernel mediates the communication between the application server and the input / output device that leads to additional overhead thus delaying the whole process. Directly using I/O devices delivers much higher performance. Arrakis is basically an attempt to decrease the kernel overhead for most type of operations by eliminating the need for the application server to always use the kernel to facilitate communication with the input / output device. It divides the kernel functions into two parts, a control plane and a data plane which increases the performance substantially.

Most servers execute operating system code such as delivering interrupts, de-multiplexing, copying network packets and maintaining file systems meta-data. The operating system overhead when processing a network packet may be due to network stack cost, scheduler overhead, kernel crossing and copying of the data packet from the kernel to the user buffer and vice versa. By speeding up the I/O much of the server side processing can be completed at a faster rate. Key-value stores, lock managers etc. are measured principally by the throughput and latency at which they carry out the client operations. Server applications perform very simple functions but for each client request made they need to traverse through the kernel to complete the operations. The paper studies the implications of removing the kernel entirely from the data path while conduction I/O operations and it demonstrates how it is still possible to achieve the same security that the traditional operating systems provide while improving the performance by extending the computer base to incorporate the application code. It demonstrates that by trying to achieve a higher accuracy we certainly do not have to compromise on the security the kernel provides. It is still possible to achieve that by making minor modifications in the application server side and using advanced I/O devices and storage controllers that support virtualization.

The functions of a traditional Linux kernel include Multiplexing, Resource limits, I/O Scheduling, Copying, Naming, Access Control, I/O Processing, Protection, and API. Kernel mediation is a heavy weight process in comparison to the performance increase we get by directly invoking I/O devices as done in data centers. The goals of Arrakis go are:

* To be able to skip the kernel and deliver I/O directly to applications thereby reducing operating systems overhead.
* Keep the traditional OS features such as security, process protection, resource limits, I/O protocol flexibility and global naming.

This can be achieved with the help of hardware by making use of hardware virtualization which is already supported by many of the standard NIC cards and is emerging on storage controllers such as the RAID controllers. Hardware technologies such as the SR-IOV (Single root I/O virtualization) are already capable of multiplexing and de-multiplexing operations. They are intended to support high speed I/O for multiple virtual machines that share a single physical machine. SR-IOV allows PCI devices to provide multiple virtual copies of the hardware each with their own set of registers, queues etc. to individual applications. Protection can be achieved by using services such as the IOUMM that restricts device access to the application’s virtual memory. Some of the hardware devices such as the network interface cards are also capable of providing kernel level functions such as packet filtering, logical disks etc. for protection and rate limiters and packet schedulers for I/O scheduling. The Arrakis kernel depends on the I/O devices for protection, multiplexing and I/O scheduling. With the API and I/O processing support provided to server applications as libraries, the Arrakis operating system attempts to remove the kernel from the data path of most I/O operations. It provides a zero copy environment since now the I/O devices map directly to the applications virtual memory. Naming, access control and resource limits are separated out into a control plane which is handled by the Arrakis kernel. Thus, the kernel is only responsible for operations related to the control plane and all the I/O can be done directly in the data plane providing applications with their own dedicated virtual instance of the I/O device. When following this architecture, the access control is carried out only once during the initial configuration of the data plane. According to the I/O operations that are needed by the application, the NIC filters and logical disks can be used to provide the appropriate access control in the data path. The I/O schedulers are programmed according to the system administrator’s resource policy to enforce the resource limits. Global naming can still be carried out since the virtual file system is still in the kernel but the implementation of storage I/O is done within applications. All persistent data that are created within the application is immediately written to a storage device. This technique eliminates marshalling, early allocation of metadata and spatial locality. Evaluations conducted on a Linux operating system with disabled receive side scaling and on the Arrakis operating system show that the read and write operation latency of a Redis persistent NoSql store is 2-5X better than that of the well-tuned Linux OS and the write throughput of the Arrakis OS is about 9X faster than that of Linux.

In my opinion this type of architecture would prove to be very useful in situations where we are typically dealing with less number of applications running on the server each of which are mostly I/O intensive; since each of the application would have its own virtual copy of the I/O device, the total number of applications that can be run on the system will be limited according to the virtualization capacity of the underlying hardware. Also, I believe stand-alone applications would give the best throughput on such architectures since functionalities, such as global naming, still require the server application to communicate with the kernel thus adding some overhead. The overhead incurred may definitely be less than the overhead incurred by using the regular non-optimized Linux I/O stack but it is still not completely eliminated. It would be interesting to study if the same architecture of separating out the control plane and the data plane can be applied without using virtualization at the hardware level and yet achieving comparable performance of I/O as the Arrakis operating system but with increased performance where global naming is involved. Also, it would be interesting to see how much performance boost we would get in the network stack by utilizing the performance enhancing features of the network hardware as used by the Linux OS and what changes would that require to the current Arrakis architecture.