Arrakis - The Operating System is the Control Plane

Aris Paphitis
CUT

1 Introduction - Problem Statement

The authors recognize that as the datacenter hardware keeps accelerating its performance, the network server operating system is becoming an I/O bottleneck. Traditionally, the OS kernel mediates access to device hardware by server applications, to enforce process isolation as well as network and disk security. Server applications often perform simple functions, such as key-value table lookup and storage, yet they spend a great deal of time traversing the OS kernel multiple times per request.

2 Proposed Solution

The solution the authors propose is virtualization[1]. They explore the implications of removing the kernel from the data path. First they propose a new architectural design for the separation of the control and data planes in the kernel. Then they implement this new design by creating a new operating system, Arrakis. Fianlly, their output prototype is used to quantify potential belefits.

3 Background

3.1 Sources of kernel overhead

Networking Stack Overheads, including mainly packet processing at the hardware, IP and UDP layers.

Storage Stack Overheads stem from data copying between user and kernel space, parameter and access control checks, block and i-node allocation, virtualization (VFS layer), snapshot maintenance (btrfs filesystem), as well as metadata updates, in many cases via a journal. Modern hardware devices have significantly reduced write times, by means of on device flash-backed DRAM, rendering OS storage overhead a bottleneck in the system.

Application Overheads are attributed to socket operations and I/O latency.

3.2 Using Virtualization

The Arrakis OS leverages the Single-Root I/O Virtualization(SR-IOV) technology, the IOMMU, and supporting adapters to provide direct application-level access to I/O devices.

4 Arrakis Design

Arrakis was designed with the following goals in mind: a) to minimize kernel involvement for data-plate operations by routing I/O requests to/from the application's address space. b) transparency to the application programmer so that applications written to the POSIX API do not need modification c) appropriate OS/hardware abstractions, flexible enough to support efficiently a broad range of I/O patterns, scale well, and support application requirements fro locality and load balance. Applications, through dedicate I/O libraries, request network and storage jobs from the respective devices via virtual network/storage interface controllers provided by the devices themselves. Through this architecture the kernel is bypassed and each device provides virtualized instances. Such virtual interface cards provide queues, filters rate limiters and virtual storage areas.

5 Implementation and Evaluation

The Arrakis operating system is based upon the Barrelfish[2] multicore OS code base. The base OS is extended with drivers to support the virtualization capable hardware used by the developing team. A user-level network stack and storage API were also developed. Arrakis is evaluated on four typical cloud applications workloads and is compared to a Linux kernel version 3.8. Two versions of Arrakis are used; a POSIX compliant version and Arrakis using its native interface (implementing a zero-copy policy). It is demonstrated that Arrakis performs better in terms of throughput during read/write

operations and scales better than Linux when multiple cores are added to the system. Arrakis design leverages hardware support and removes the kernel control from the data path. It can therefor eliminate entirely certain overheads and minimize the effect of others. Arrakis removes scheduling and kernel crossing overheads since packets are delivered directly to user-space. Network stack is greatly simplified without the need for (de-)multiplexing[3] because each application has a separate network stack. Additionally, dut to the fact that packets are delivered to the cores where the application is running, lock contentions and cache effects are reduced.

Notes

[1]Hardware virtualization is the virtualization of computers or operating systems. It hides the physical characteristics of a computing platform, showing an abstract platform instead.

[2]Barrelfish is a 'multikernel' operating system where each kernel runs on each core. The rest of the OS is structured as a distributed system of single-core processes atop these kernels. Kernels share no memory. Applications however can use multiple cores and share address space between cores; this facility is provided by user-space runtime libraries.

[3]Multiplexing: an idea for non-blocking I/O. Provide system calls (like select() , poll() or epoll()) to inform the kernel as to which system device an application is interested in. The kernel then "polls" the device to see if it is ready to do I/O without blocking. If not the kernel puts the process (application) to sleep until the device is ready.