BDUS: Implementing Block Devices in User Space

Alberto Faria, Ricardo Macedo, José Pereira, João Paulo
INESC TEC & University of Minho

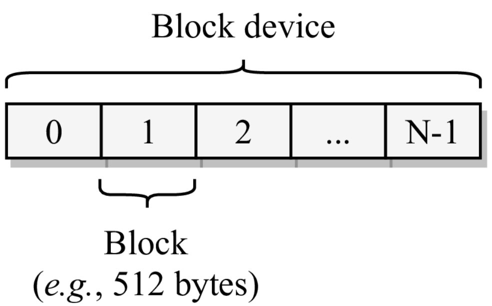
Systor2021
June 14-16 Virtual

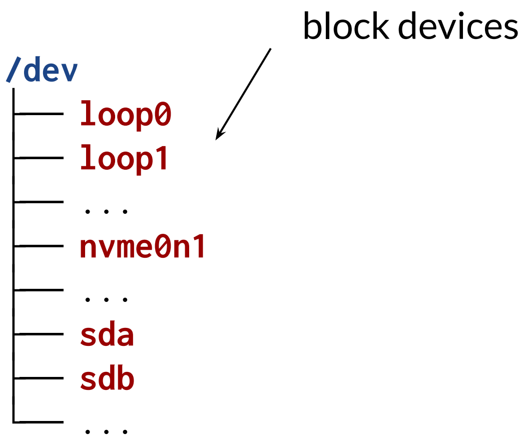
Motivation

- **Storage services** are typically implemented in the **kernel**
- **More performant** than user-level services
  - Less context switches, memory copies
- Kernel-level development is **complex**
  - Operating system-specific, limited environment
- **User-space** services have several **advantages**
  - **Easier** development and maintenance
  - Greater **portability**
  - Access to more and **higher-level languages** and **libraries**
  - Improved **reliability**, **fault tolerance**, and **security**
- *E.g.*, **FUSE**: https://github.com/libfuse/libfuse

Block devices

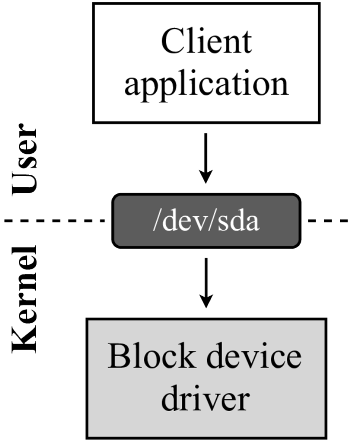
- We consider **user-level** development at the **block layer**
- ***Block devices*** expose storage devices/systems
  - Contiguous sequences of fixed-size blocks
- Used by a wide range of **applications**
  - Either **directly** or through **local file systems**





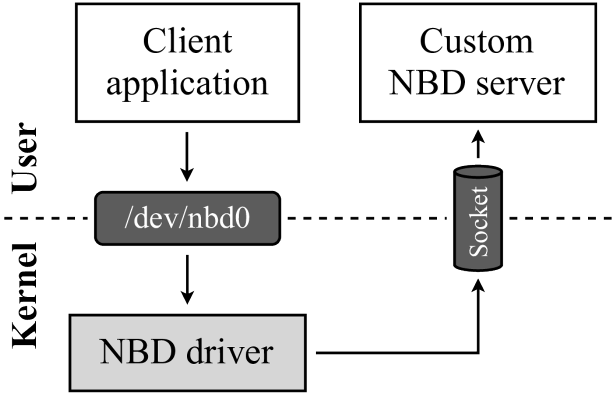
Block device drivers

- ***Block device drivers*** implement the **behavior** of block devices
- Are typically included in the **kernel**
  - Or separately as loadable **kernel modules**
- Can be **implemented** in **user space** by leveraging existing operating system subsystems



Network Block Device (NBD)

- Provides access to **remote storage** through **block devices**
- Client-server architecture
  - In-kernel **block device driver** as client
  - **User-space process** as server
- Communication through **TCP** or **Unix Domain Sockets**
- Can create **custom** NBD servers
  - Using frameworks like **BUSE**, **nbdcpp**, **nbdkit**
- Effectively allows building **drivers** in **user space**
  - Create **custom server** with desired logic
  - Deploy in **same host as client**

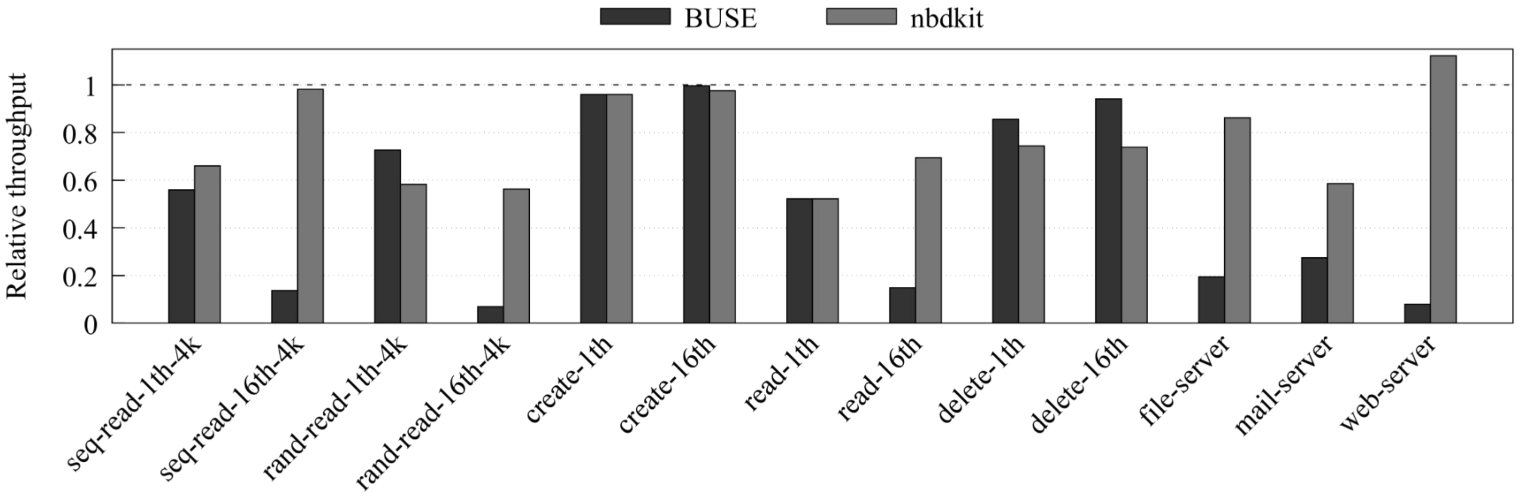


Evaluation methodology

- Built **"pass-through"** drivers with **BUSE**, **nbdcpp**, **nbdkit**
  - Redirect all requests to **underlying hardware device**
- **Measured** throughput, latency, CPU utilization
  - When operating on **underlying device**
  - When operating on **each pass-through device**
- **16 workloads** performing operations **directly** on block device
- **25 workloads** performing operations on ext4 **file system** backed by block device
  - **Data-intensive** micro workloads
  - **Metadata-intensive** micro workloads
  - **Macro** workloads
- (Full results in the paper)

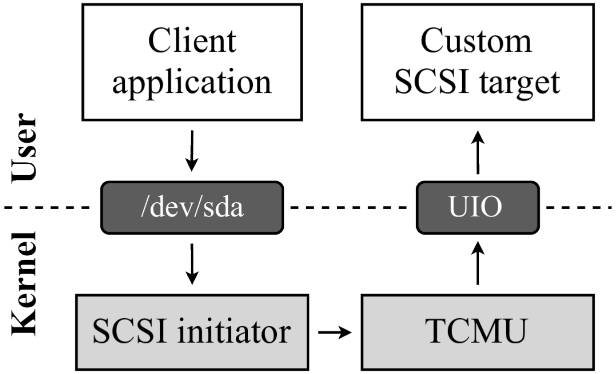
NBD: Performance

- Throughput of **pass-through** devices, **relative** to underlying device:
  - nbdcpp (not shown) **never outperforms** BUSE
  - BUSE (like nbdcpp) processes requests **sequentially**; nbdkit in **parallel**
  - **Sockets** impose an additional **memory copy**



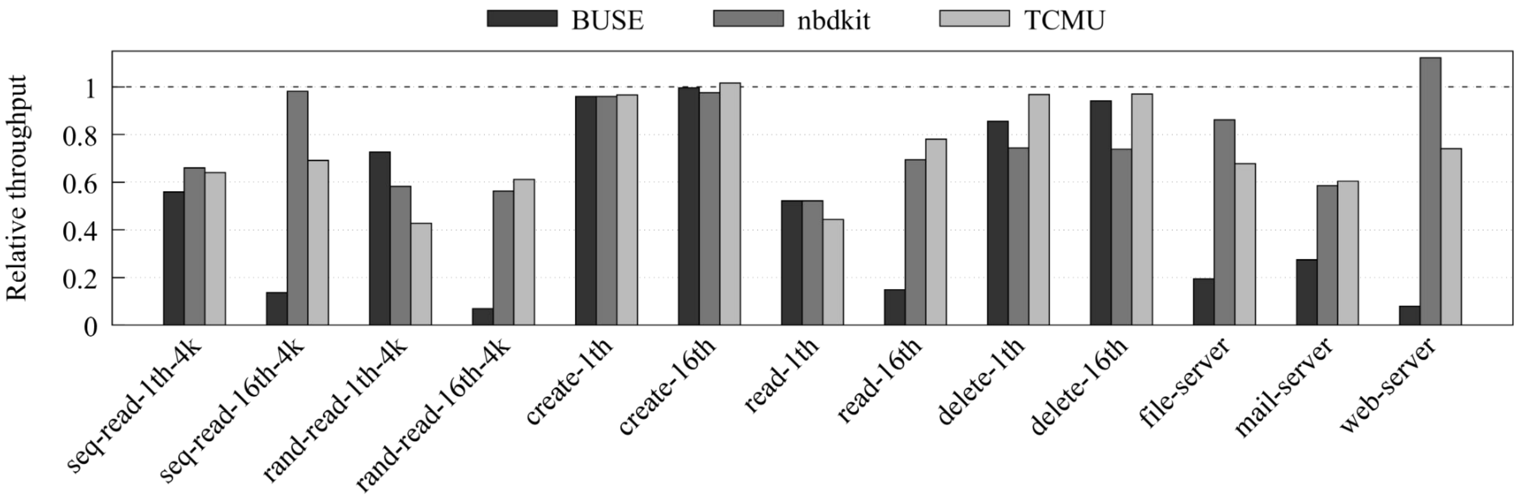
Target Core Module in User space (TCMU)

- **SCSI**: Standards for computer ↔ storage device data transfer
  - ***Target***: service that handles SCSI commands
  - ***Initiator***: client that submits SCSI commands
- **Linux**'s SCSI subsystem includes **TCMU**
  - Enables **user-level** processes to act as SCSI **targets**
  - Communicates with kernel through the **UIO** framework
- Can be used to create **user-level block device drivers**
  - Implement **SCSI target** using **TCMU** with desired logic
  - Deploy target in **same host as client**
  - Configure **initiator** to expose **block device** backed by deployed target



TCMU: Performance

- Same plot as before, now with **TCMU:**
  - **Overhead** on throughput of up to **57%**
  - **Better** than NBD-based solutions under some workloads, **worse** under others



A new solution is needed

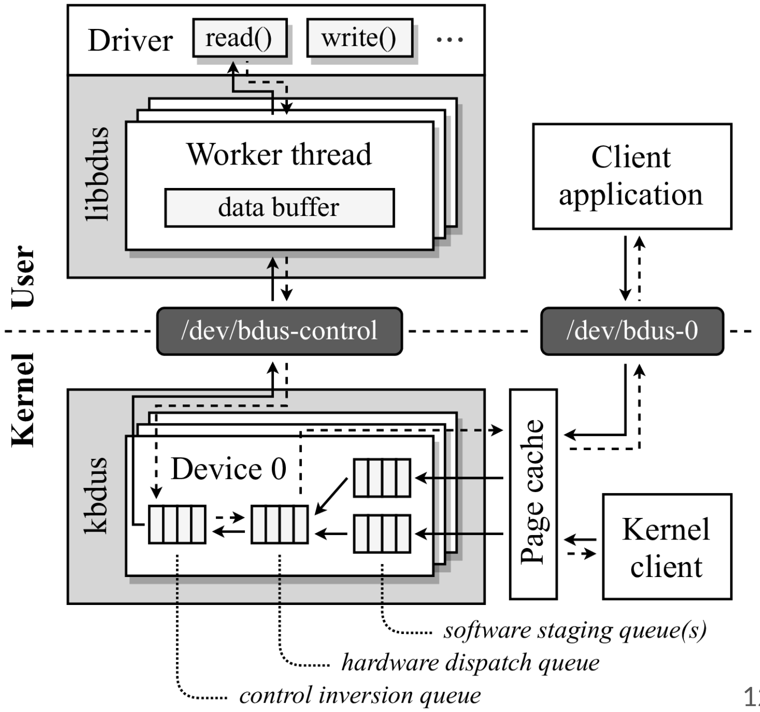
- Existing solutions have significant **performance limitations**
- Should *not* rely on subsystems designed for **other purposes**
  - Inherent **limitations** of implementations targeting **networked access**
  - Less room for **specialized** optimizations and improvements
- Can do better with a **purpose-built framework**
  - Improve **performance**
  - **Unlock** further performance and functionality **improvements**

The BDUS framework

- Built specifically to enable the development of **block device drivers in user space**
- Design curtails **memory copies** and **system calls**
- Fully-functional, **open-source** implementation for Linux
  - https://github.com/albertofaria/bdus
- Driver replacement and recovery with **no downtime**
  - **Hot-swap** the driver of an existing device
  - **Recover** from a **failed driver** without interruption of service
- **Less overhead** and **resource utilization** than existing solutions

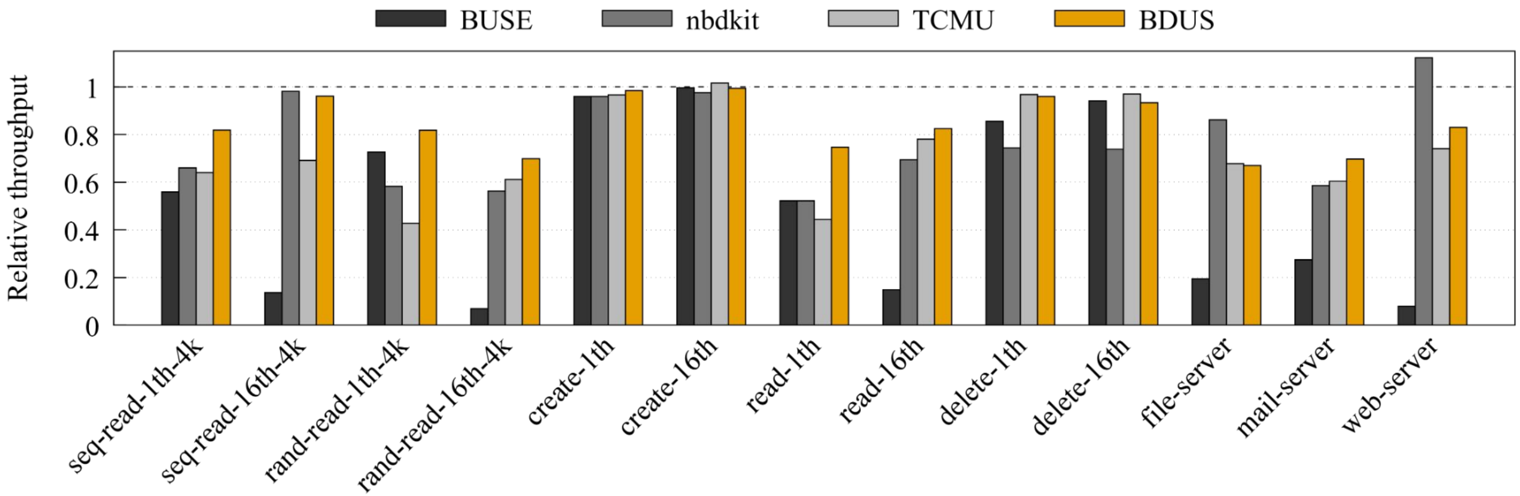
BDUS: Design and implementation

- Two main **components**:
  - ***kbdus*** kernel module
  - ***libbdus*** user-space library
- **Drivers** are user-level C programs
  - Implement **handlers** for each request type
  - Specify **block size,** total **device size**, …
  - Link against *libbdus*
- **Run** compiled driver to **create device**
  - Appropriate **handler** called for every request
- **Kernel** ↔ **user** communication uses **`ioctl()`**
  - Through *character device* `/dev/bdus-control`
  - Average of 1 system call per request



BDUS: Performance

- Same plot as before, now with BDUS:
  - **Degrades** throughput by at most **33%**
  - **Improves** throughput over existing solutions by up to **43%**
  - Outperformed by nbdkit under *file-/web-server* due to **unfair configuration** (see paper §6.2)

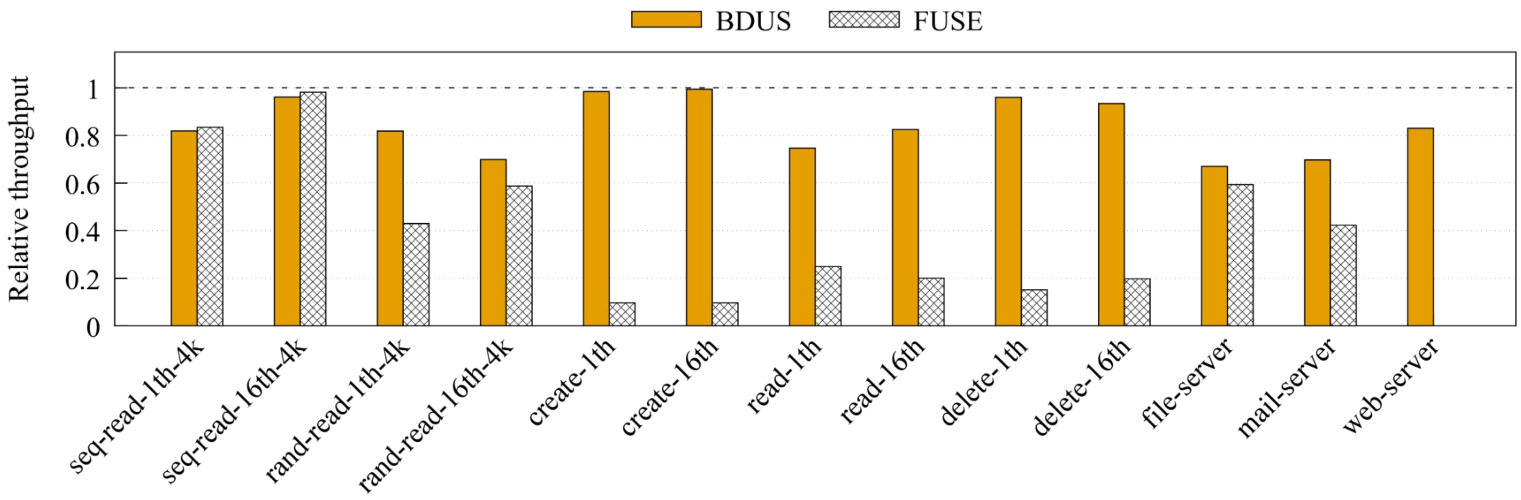


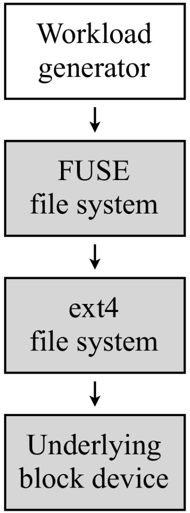
BDUS and FUSE

- **FUSE**: Enables the implementation of **file systems** in user space
- **Similar objective** as BDUS, **different layer** of the storage stack
  - BDUS and FUSE are **orthogonal** and **complementary** to each other
- But many **storage functionalities** can be implemented at **both layers**
  - Compression, deduplication, thin provisioning, encryption, erasure coding, replication, …
- May have to **decide** between using BDUS or FUSE
  - Must have knowledge of **performance** advantages/disadvantages

BDUS and FUSE: Performance

- Relative throughput of **FUSE pass-through** file system:
  - Same workloads as before, compared with previous BDUS results
  - **BDUS outperforms FUSE** significantly under many workloads
  - Most noticeable under **metadata-intensive** workloads





Summary

- Existing solutions exhibit **limited performance**
- **Restricted** by dependency on existing subsystems
  - Also limits the introduction of **specialized functionalities** and **optimizations**
- **BDUS** follows a **clean-slate** approach
  - **Improved** performance and resource utilization
  - Additional features for **driver replacement** and **recovery**
  - **Unlocks** further performance and functionality improvements
- **Outperforms FUSE** in file system stacks
  - Particularly under **metadata-intensive** workloads
  - BDUS is thus a useful alternative over FUSE when a storage solution can be built using either

BDUS is open source!

https://github.com/albertofaria/bdus