JG|U

JOHANNES GUTENBERG
UNIVERSITÄT MAINZ

Towards Dynamic Scripted pNFS Layouts

Matthias Grawinkel, Tim Süß, Gregor Best,
Ivan Popov, André Brinkmann

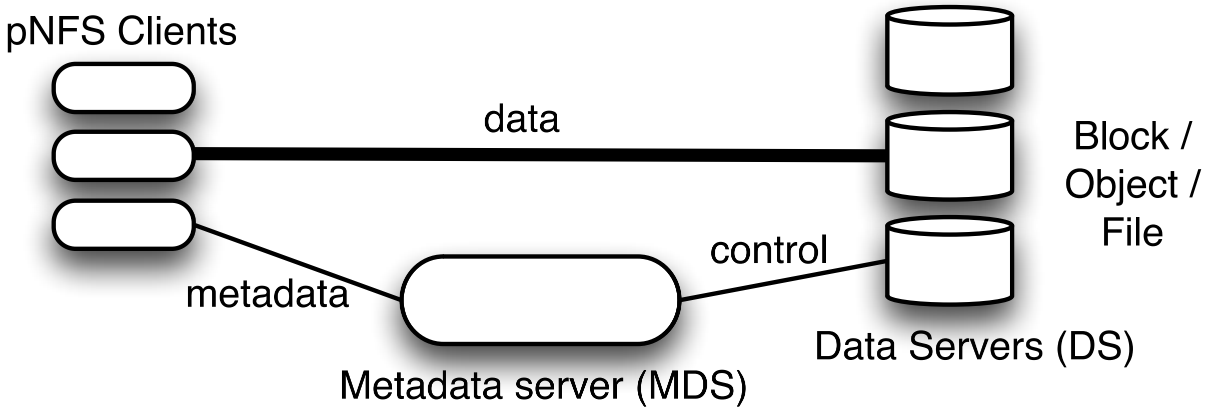
JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Motivation

- Massive amounts of data
- Huge variety in:
  - Storage system architectures
  - Storage media (Ram / Flash / Disk / ... + RAID)
  - Storage protocols
  - Application's access patterns / requirements
- Mismatch of access pattern and storage system can have severe impact on performance!

- Ideas to improve this situation:
  - Shift some responsibility to clients
  - Extend application's hints on resource usage
  - Use reconfigurable, script based file layout descriptors

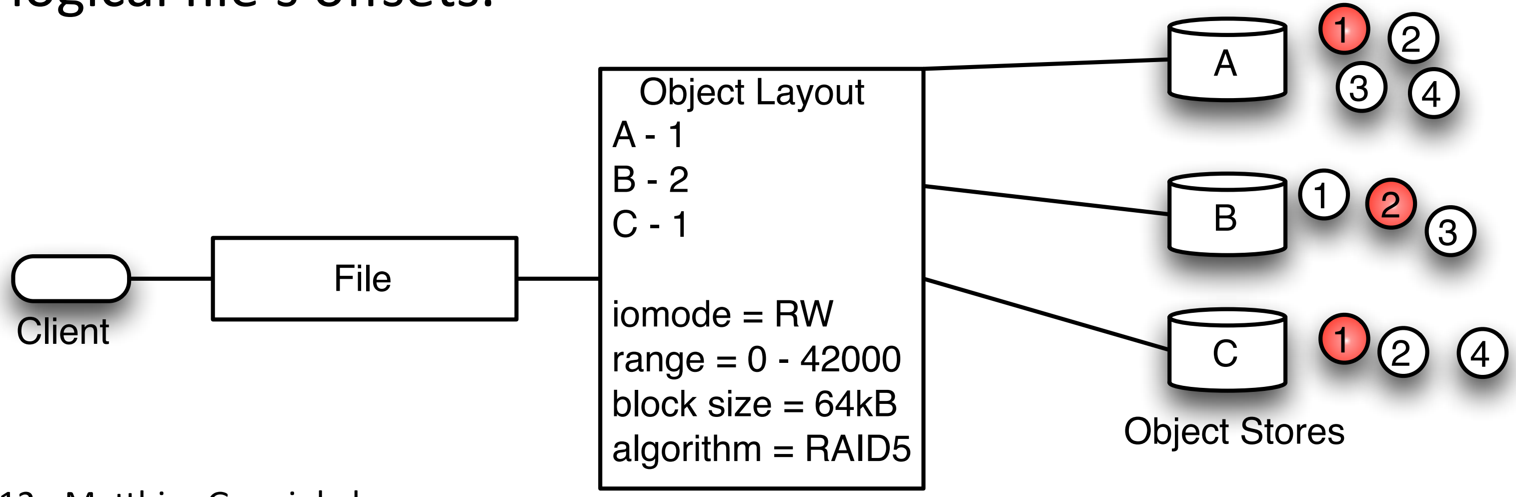
NFSv4.1 / pNFS



- NFSv4.1 extension for parallel and direct data access
- Namespace and metadata operations on MDS
- Direct data path to data servers (Block, Object, File layouts)

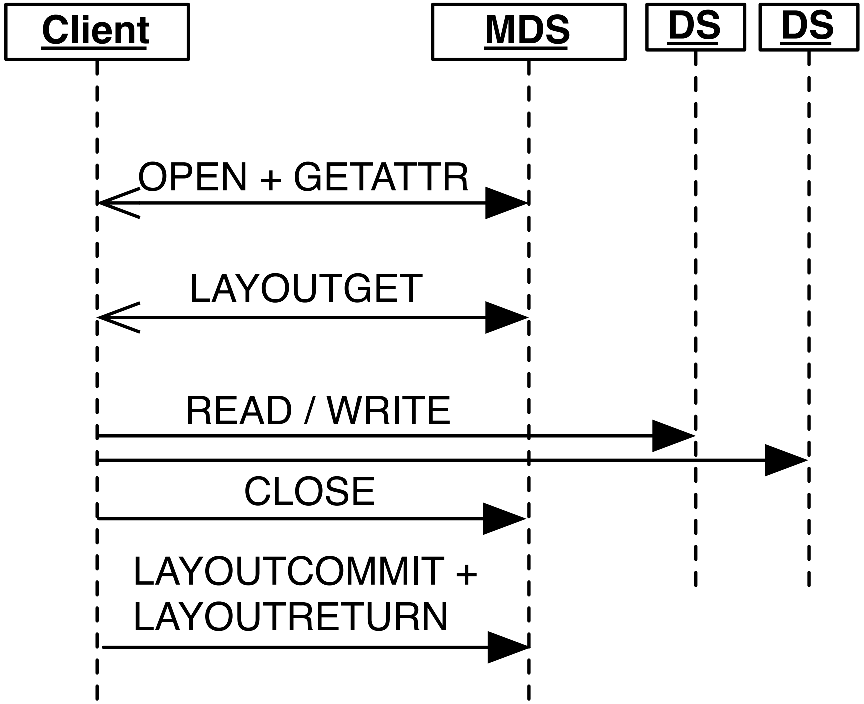
Data Access

- In pNFS the file‘s content is organized in a **layout**
- Organized by MDS, client calls GETLAYOUT for a file handle
- Layout contains:
  - Locations
    - Map of files, volumes, blocks that make up the file
  - Parameters
    - iomode (R/RW), range, striping information, access rights, ...
- Current layouts define fixed algorithms to calculate target resources for logical file‘s offsets.



Layout Semantics

- MDS knows who holds which layouts
- Conflicting layouts are prevented by MDS
  - Ask client to return layouts
  - Calls back invalidated layouts
- Layout is valid for full file or a range
- Overlapping read-layouts possible
- RW-layout is exclusive for a range
- RW-layout‘s content can be updated (LAYOUTCOMMIT)
- Layouts have to be returned to the MDS (LAYOUTRETURN)



Layout Hints

- How does the MDS create layouts?
  - open (...,O_CREATE, ..., layout_hint, ...)
- Application can provide a *layout_hint* on file creation
- Goal: Applications can express their requirements

- We argue for more verbose hints
- Introduce **storage classes**
  - Characterized by metrics: Throughput, latency, reliability, ...
    - Gold, Silver, Bronze?
  - Application can send a wish list for storage resources
    - I.e. 2 x Gold on two servers for RAID1,
      10 x Silver on ten servers for RAID6
- Application provides algorithm to interpret layout
  - E.g. map some file regions to Gold, others to Silver

Scripted Layouts

- Introduce scripting engine to pNFS stack

- Layout uses script instead of fixed algorithm
  - Flexible placement strategies
    - RAID 0/1/4/5/6, Share, CRUSH, Clusterfile, ...
  - Flexible mapping to storage classes

- Application can:
  - Provide own layout script
  - Reconfigure storage driver
  - Update layout script, parameters (LAYOUTCOMMIT)
  - Move storage resources between layouts

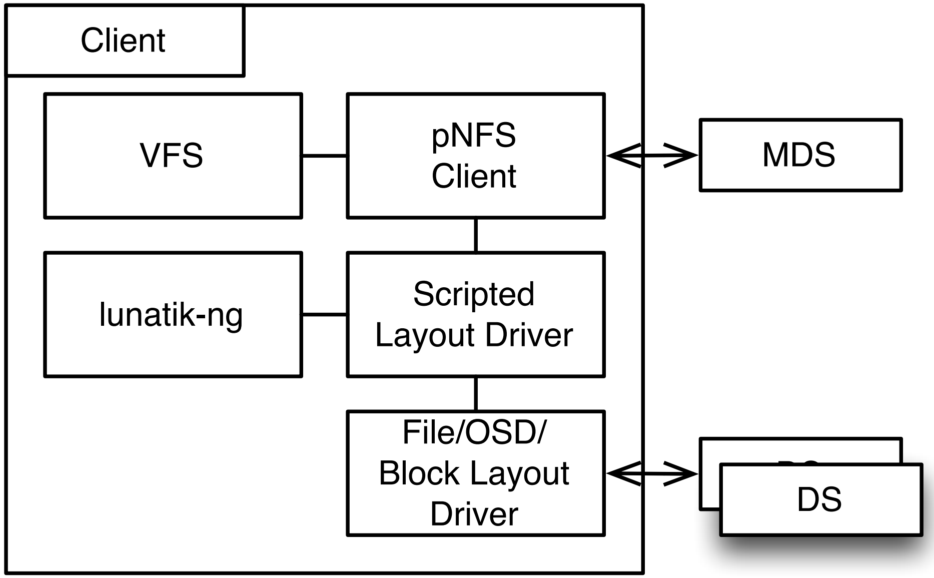
Scripting Engine

- Lua
  - Very fast scripting language
  - Embeddable with bindings for C/C++
- In-kernel scripting engine - lunatik-ng [1]
  - Stateful: Can hold functions, tables, variables
  - Callable from kernel code
  - Syscall for applications
    - Administrators / Applications can get/set (global) variables and functions
  - Extendable by bindings
    - kernel crypto API
    - pNFS

[1] http://github.com/lunatik-ng/lunatik-ng

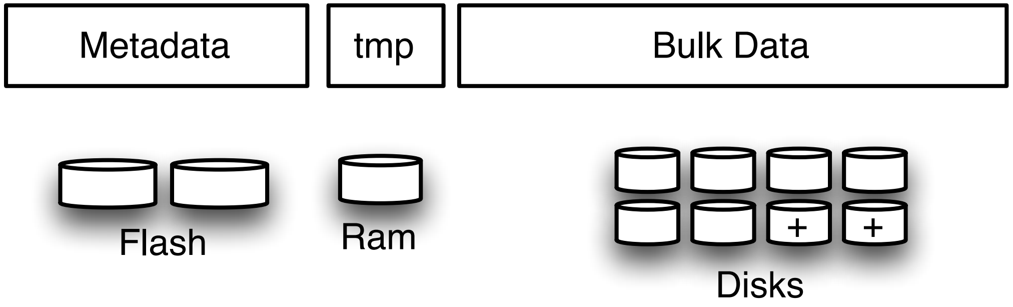
Scripted Layout Driver

- Meta layout driver that uses existing drivers
  - On data access, the layout's script is evaluated
  - Existing drivers can be reused



Examples

- NetCDF, HDF5 like data structures can be spread to multiple locations that match the files internal structure and access patterns



  - Application can adapt on file regions, algorithms, mappings, ...

- Pseudo randomized data placement strategies
  - Layout contains (link to) list of storage resources and a script to calculate the actual targets

Evaluation

- No implementation of scripted layout driver (yet)
- lunatik-ng scripting engine
  - With bindings for pNFS kernel objects, crypto API
- Performance tests for relevant scripts

- Tests conducted on
  - Linux Kernel 3.6 - git://linux-nfs.org/~bhalevy/linux-pnfs.git
    - pnfs-all-latest branch
  - lunatik-ng – http://github.com/lunatik-ng/lunatik-ng
  - Intel(R) Xeon(R) CPU E3-1230 V2 @ 3.30GHz with 16 GB Ram

Results

- Calculate stripe unit index from file_layout: **0.87µs** / call (±0.03)
- Creating a new file_layout object: **2.18µs** / call (±0.05)

```
function lua_create_filelayout (buf)
  rv = pnfs.new_filelayout()
  rv.stripe_type = "sparse"
  rv.stripe_unit = buf[1] + buf[3]
  rv.pattern_offset = buf[2] + buf[4]
  rv.first_stripe_index = buf[5] + buf[6]
  return rv
end
```

- Calling kernel.crypto.sha1(20 bytes): **1.25µs** / call (±0.02)
- Creating new file_layout with sha1() calculation: **3.25µs** / call (±0.02)

Conclusion

- It would work!
  - Proposed hints and script based layouts are compatible with pNFS protocol
  - Scripting capabilities look promising
- Opens up:
  - New possibilities for optimizations, self-adapting applications
  - Field for experimentation on placement strategies
- Problems:
  - Usage scenarios? Who will provide the scripts? User / Developer / Admin?
  - Scripts are dangerous! “while(true) {}” - signed building blocks?
  - MDS looses control / consistent view on files
  - Performance overhead of “static” scripts vs code
  - The killer app?

Questions!?

grawinkel@uni-mainz.de

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

JG|U