Deficiencies in I/O and Future?!

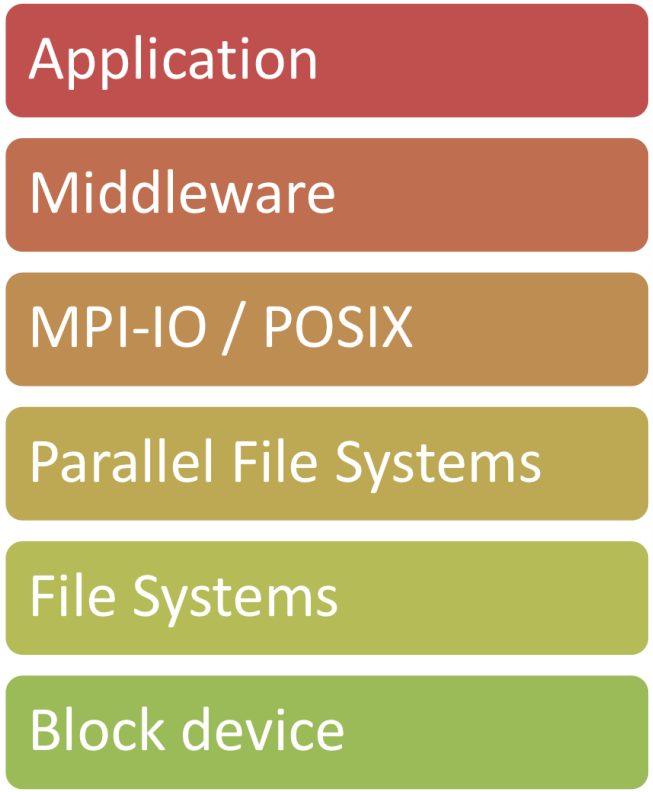
Julian Kunkel

International Supercomputing Conference

25. June 2014

(Some) Challenges of the Current I/O Stack

- Coexistence of access paradigms
  - File (POSIX, ADIOS, HDF5), SQL, NoSQL
- Semantical information is lost through layers
  - Suboptimal performance
- Reimplementation of features across stack
  - Unpredictable interactions
  - Wasted resources
- Restricted (performance) portability
  - Optimizing each layer for each system?



Example I/O stack

User Perspective: Accessing Data

Multitude of data models

- POSIX File: Array of bytes
- HDF5: Container like a file system
  - Dataset: N-D array of a (derived) datatype
  - Rich metadata, different APIs (tables)
- Database: structured (+arrays)
- NoSQL: document, key-value, graph, tuple

Choosing the right interface is difficult.

A workflow could involve different data models.

Properties / qualities

- Namespace: Hierarchical, flat, relational
- Access: Imperative, declarative, implicit (`mmap()`)
- Concurrency: Blocking vs. non-blocking
- Consistency semantics: Visibility and durability of modifications

Semantical Gap of File Access

Information hidden from file systems

- Data types
- Data semantics
- Concurrent access
- Value of data: Checkpoint, computed, original, logfile
- Data lifecycle: production, usage, deletion

Characteristics can even vary within a file, e.g. for metadata

Storage systems could use this information for

- Improving performance: Automatic tiering, caching, replication
- Simplifying management: ILM, offering alternative data views
- Correctness: Ensuring data consistency

Performance Tweaks

- There are many options to tune the I/O-stack
  - API: `posix_fadvise()`, HDF5 properties, open flags, cache size
  - Via command line: `lfs setstripe`
  - Setup/initialization of a storage system
- Many options are of technical nature
  - Performance gain/loss depend on hardware, software
  - Specific to file system, API (MPI, POSIX, HDF5)
  - Many types of hints/tweaks are not portable
- Performance loss forces us to use these optimization

Critical Discussion

Questions from the users' perspective

- Why do I have to organize the file format?
  - It's like taking care of the memory layout of C-structs
- Why do I have to convert data between storage paradigms?
  - Can't I just store it like an HDF5 file and access it with SQL?
- Where are my experiment results with these properties?
  - OK I will organize all my data with a (fixed) scheme to find it
- Why must I provide system-specific performance hints?
  - It's like telling the compiler to unroll a loop exactly 4 times
- Why is a file system not offering the consistency model I need?
  - My application knows the required level of synchronization

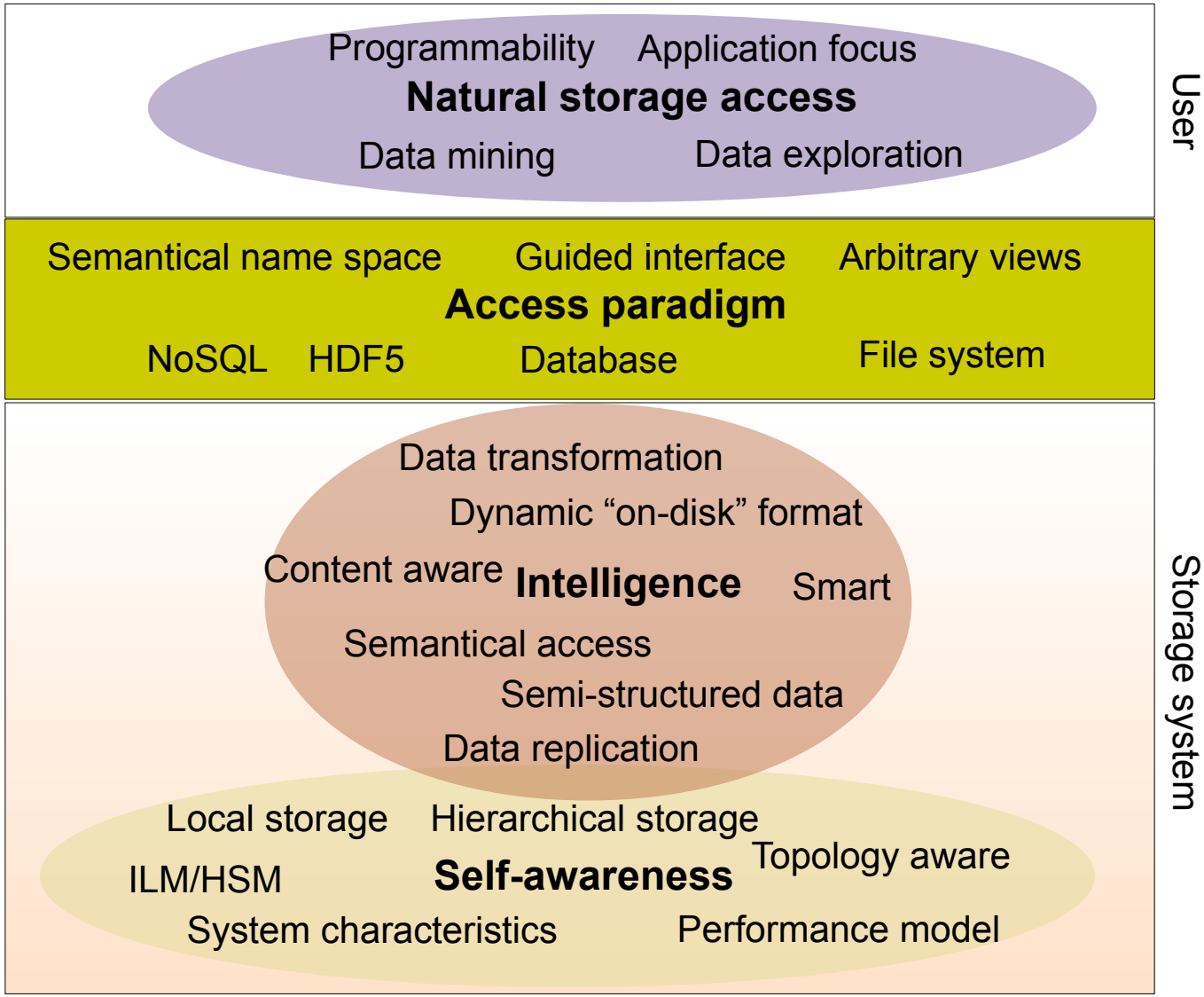
I would rather like to code my application ...

Storage in the Future with E10

- E10 will become a modular allround storage system
  - Incremental deployment on top of existing infrastructure
- E10 will overcome limitations of the current architecture
  - Data-type aware storage
  - Multiple “views” to the same data (BigData)
  - Guided interfaces instead of technical hints
  - Data “format” handled by storage system
  - Multi-tiering support
  - Intelligent monitoring
  - Feedback to optimization and “what-if” analysis
  - Integrates active storage concept
  - Post-processing handled by file system
- Huge effort to (d|r)efine design in a community
- A first high-level draft of the E10 architecture is available

Backup Slides

Personal Vision of Future Storage Systems

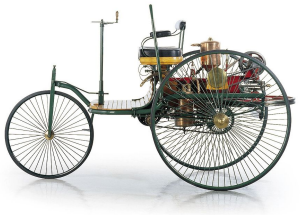
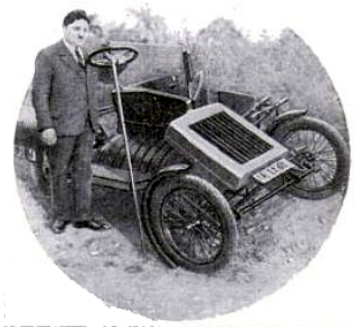


User-Perspective and Software Deficiencies

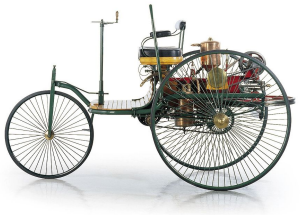
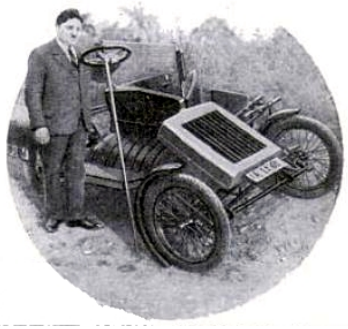
If Storage APIs and File Systems Would be Vehicles

Use-Case: travel from Hamburg to Barcelona.

Flavor of Storage APIs and File Systems


Flavor of Storage APIs and File Systems


How about traffic jams?

Can’t we go the direct path?

What I rather like to see

(If I dream) Yes we can