GWDG
Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

Governance-Centric Interaction Including Data Management in HPC

Hendrik Nolte

Task-Driven Projects

- Historically HPC was used to serve single, highly parallelizable tasks
  - One task was parallelized across many nodes (Task Parallelism)
  - Required that one overarching task can be split down into smaller parts
    - Weather simulation
    - Molecular dynamics simulations
    - Algebra
  - The required resource was mostly computation
  - Software could be (easily) optimized for parallel filesystems

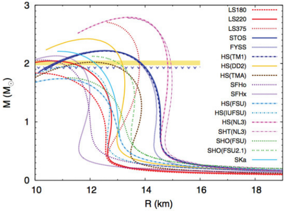


Figure: Stone, J.R. Nuclear Physics and Astrophysics Constraints on the High Density Matter Equation of State. Universe 2021, 7, 257.

Data-Driven Projects 1

- Data-driven methods had a lot of success in the last years
  - Just think of the recent impact of ChatGPT
  - Sepsis Prediction based on a multitude of streaming data
  - Self-driving cars
- These methods can be used to run independent jobs in parallel
  - On different input data
  - On different hyperparameters
- Generally these data-driven methods are characterized by
  - large data sets
  - random IO
  - potentially unoptimized file formats

Data-Driven Projects 2

- This success has lead to an increased adoption in new domains
- Which increased the share of data-driven projects on HPC systems
  - These projects also require large computational power
- This excarcabates the problems from dd-methods on HPC infrastructure
  - Large data sets require large storage systems
  - These large data sets often consist of millions of small files
  - Which are organized in flat namespaces to encode their target

Challenges Data-Driven Projects

- In particular we have identified 4 challenges with this new user group:
  - Storage Performance and Efficiency
  - Data Management
  - Integration of Compute and Data Handling
  - Reproducibility

Storage Performance

- Iterative procedures read in small files
  - This can quickly overload the metadata servers
- These small files are organized in flat namespaces
  - Explosion of tree depth due to indirect inodes
- Users use parallel filesystems for WORM-workloads
  - PFS often implement POSIX-IO semantics
  - The statefullness kills read performance

Figure: Structure of an inode. Source: Wikipedia

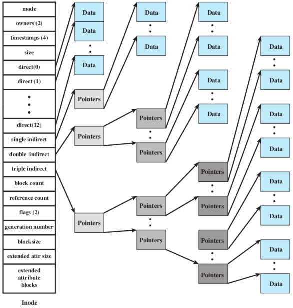


Figure 12.16 Structure of FreeBSD inode and File

Figure:
https://www.usna.edu/Users/cs/crabbe/SI41

Data Management

- There is an increasing demand for proper data management in science
  - One famous standard are the FAIR principles
  - Is sometimes required by project funders
    - Sometimes in the form of a data management plan
  - or by journals when publishing a paper

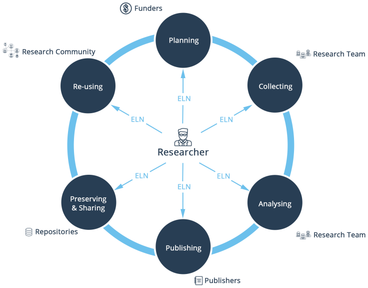


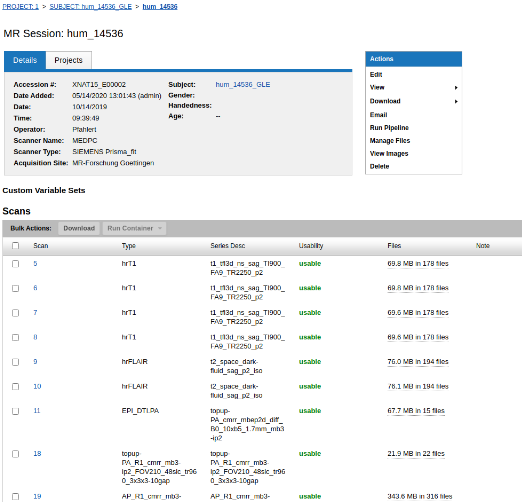
Figure: https://www.labfolder.com/guide-research-data-management/

Integration of Compute and Data Handling - Admin View

- Usually there a multitude of storage tiers available
  - HOME
  - WORK / SCRATCH
  - Local SSD’s
  - Burst Buffers
  - tmpfs
  - Archive
  - And probably more, specific to each center
- These different systems differ in
  - performance
  - durability
  - cost
  - volume
  - semantics

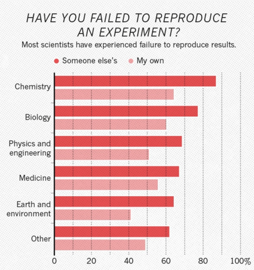
Integration of Compute and Data Handling - User View

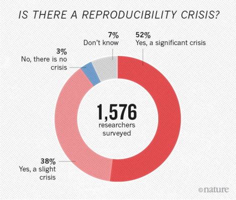
- External Domain-Specific RDMS
  - e.g. XNAT, or Viking
  - Have high market share
  - Users expect support
  - HPC is only "necessary evil"
- Users are familiar with Cloud
  - Started by Hadoop
  - Data Access via
    - HDFS
    - YARN
  - Moved to Jupyter Notebook
    - Limited virtualization
    - Manual data handling



Reproducibility

- There is a general reproducibility crisis
- For HPC one needs to distinguish
  - Deterministic execution of a job
  - Proper provenance auditing
- Deterministic execution is hard
- Proper lineage recording shouldn't be
  - Due to insufficient data management
- Specififc HPC tools are often not used
  - e.g. *PASS*, *LPS*, or *ReproZIP*
  - Domains developed own standards
  - Integrated into remote DMS





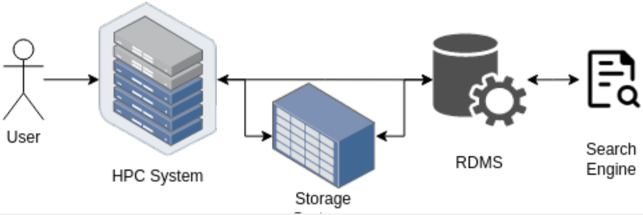
Nature 533, 452–454 (26 May 2016) doi:10.1038/533452a

HPC Interaction Paradigms

- Generally, there are three different interaction Paradigms:
  - Traditional
  - Compute-Centric
  - Use-Case, or DMS-centric
- **Traditional Interaction Paradigm**
  - Users log in via *ssh*
  - Users manually have to prepare their code
  - Users manually manage data, processes, and resources
  - Users manually have to map input/output data to storage targets

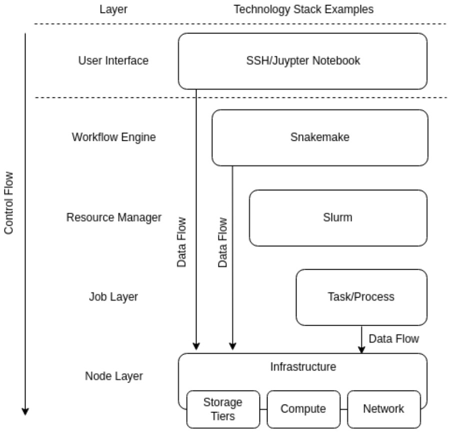
Compute-Centric Interaction Paradigms

- Users connect to a HPC system as in the traditional paradigm
- Different degrees of sophistication
- In the simplest form:
  - Users maintain a data catalog and select based on semantic metadata
  - Data is loaded into the running code
    - Either synchronously or asynchronously
    - asynchronous staging has to be explicitly defined by users
  - Provenance auditing is the response of the users
- Example implementation with iRODS, Python and Snakemake



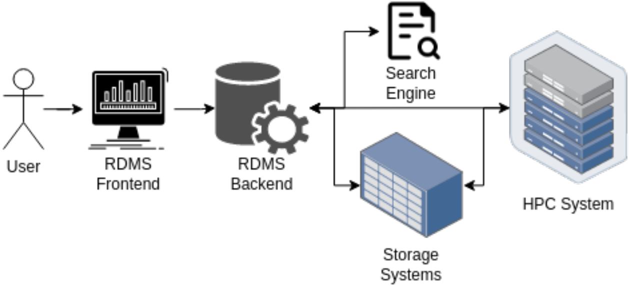
Compute-Centric Paradigm - Control and Data Flow

- Data is accessed explicitly
  - using library functions
- or implicitly
  - as an input parameter
- storage-tier aware data placing
  - Data locality
  - Data Availability
  - Data Durability
- Control flow managed by the user layer



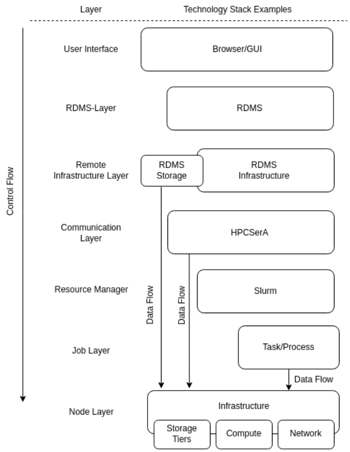
DMS-Centric Interaction Paradigms

- Web frontend is used to
  - query and select input data
  - define a compute task
  - and submit it to the HPC system
- Users expect efficient and transparent data transfers
  - Data placement should also be transparently handled



DMS Paradigm - Control and Data Flow

- User not interested in data access
  - Complexity should be abstracted away
- Communication Layer is mandatory for
  - Data transfer, depending on topology
  - Control flow for job dispatch
  - reingest of artifacts and metadata
- Data placement and process management done by DMS
  - Data is located outside of HPC



Analysis of Paradigms

Characteristics	Traditional	Compute-Centric	Use Case-Centric
Resources (Compute)	Auto	Auto	Auto
Resources (Storage)	Manual	Manual	Auto
Res. Mgmt (Compute)	Semi-Auto	Semi-Auto	Auto
Res. Mgmt (Storage)	Manual	Manual	Auto
Job spec	Manual	Semi-Auto	Auto
Program	Manual	Manual	Auto
Software land	Provided	Provided/User-Container	Provided
Workflow spec	Manual	Semi-Auto	Auto
DMP	Manual	Manual	Tool-specific
User interface	SSH	SSH+Web	Web
User interface (Data)	SSH	SSH	Web
Client	SSH	SSH+Browser	Browser
Performance	User-specific	User-specific	+
Data management	-	-	Tool-specific
Integration	–	0	++
Reproducibility	–	+	Tool-specific
Flexibility	++	++	–

Analysis of Paradigms

- Metrics for user experience basically boil down to
  - Where is my data located?
  - How is my data linked?
- To answer this question one has to come from two directions
  - Integration Layer above the resource manager
    - Provide a unified namespace for users
  - **Actionable** information flow in the opposite direction as the control flow
    - It must be actionable in order to guide a user to a predefined state
    - Not create another piece of data a user has to manage as well
- To solve this problem, we propose the **Governance-Centric Paradigm**

Required Degree of Automation

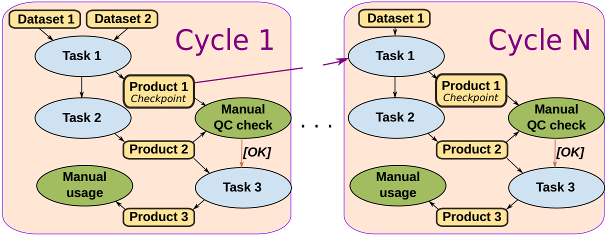
Characteristics	Traditional	Compute-Centric	**Governance-Centric**	UseCase-Centric
Resources (Compute)	Auto	Auto	Auto	Auto
Resources (Storage)	Manual	Manual	Auto	Auto
Res. Mgmt (Compute)	Semi-Auto	Semi-Auto	Auto	Auto
Res. Mgmt (Storage)	Manual	Manual	Auto	Auto
Job spec	Manual	Semi-Auto	Semi-Auto	Auto
Program	Manual	Manual	Semi-Auto	Auto
Software land	Provided	Provided/User-Container	Provided/User-Container	Provided
Workflow spec	Manual	Semi-Auto	Semi-Auto	Auto
DMP	Manual	Manual	Semi-Auto	Tool-specific
User interface	SSH	SSH+Web	Web+SSH	Web
User interface (Data)	SSH	SSH	Web+SSH	Web
Client	SSH	SSH+Browser	Browser+SSH	Browser
Performance	User-specific	User-specific	++	+
Data management	-	-	++	Tool-specific
Integration	–	0	++	++
Reproducibility	–	+	++	Tool-specific
Flexibility	++	++	+	–

Architecture

- Scientists define an experimental description at the beginning, containing
  - a high-level workflow description linking data with tasks
  - a data management plan for all input/output data
- Moving from a DMP as an abstract Plan towards an enforced entity
  - Requires an machine-readable DMP, where users can specify
    - the data flow
    - the data sets
    - access and backup policies
    - the data life cycle
    - IO intensity (if known)
  - Each task, e.g. Slurm job, has to be linked to a workflow step

Use in Data-Intensive Projects

- Specifically in data-intensive projects a workflow is being executed
  - as opposed to one large, expensive single task
- This workflow description states when which data should be where



Gathering Rich Metadata

- To improve findability users provide domain specific metadata
  - User annotated side car file
  - Automated process to extract metadata from a dataset
- Metadata modeling is done in the DMP
  - Ensures continuous, automatic quality control
- Indexing in an external DB
  - Unifies compute-centric and DMS-centric paradigms

Modifying Tasks and Resource Manager

- Each compute job has to be prepared, annotated, and linked with the DMP
- Linkage has to be done by the resource manager, i.e. Slurm
  - Data becomes another resource
  - Users specify the task of the workflow/DMP
  - Users specify the input data set
    - Instead of working with explicit filenames
    - Required mappings are provided by the integration layer
  - Explicit data path is exported via environment variables
  - Data staging strategies and storage targets are determined by DMP tool
    - Can be done based on generalized heuristics implemented by admins
    - Users can hint expected IO intensity

Reproducibility

- At least enough provenance information for retrospective comprehensibility
- Automatically recording data lineage is a nested problem
  - Job Script
  - Resolve ambiguities of running commands, e.g.: *python myScript.py*
    - Check if file is part of a git repository, create sidecar file with git commit hash
    - Parse job script and identify untracked, userspace dependencies
    - Use known provenance tools and write results in side car file
    - Prompt users to write sidecar file themselves

Implications

- Abstractions of files towards data sets will **integrate** storage and compute
  - Users don't need to know specifics about storage tiers
  - Pushes users to proper metadata management and cataloging
- **Performance** increase since users work with data sets, not with filepaths
- Increased **reproducibility** by linking datasets to tasks during scheduling
  - Tight integration with containers and auditing tools can be provided by admins
- The proposed methodology is **enforcable/actionable**
  - Compare the is-state against the DMP-defined state
  - Automatically using cron jobs to detect
    - files outside of data sets
    - insufficient/missing sidecar files

How to Start?

- Looks like a big paradigm shift, how can we convince users?
- We suggest to make it part of the application process for Tier-2 systems
  - Just having short, and simple section will raise awareness for the user
  - It will have a simple structure to make it machine-readable
- Incentivice by offering more resources?
- **What do you think? Let’s have a discussion!**