Performance Tools and Holistic HPC Workflows

Karen L. Karavanic

Portland State University

Work Performed with:

Holistic HPC Workflows: David Montoya (LANL)

PSU Drought Project: Yasodha Suriyakumar (CS), Hongjiang Yan (CEE),

PI: Hamid Moradkhani (CEE), co-PI: Dacian Daescu (Math)

PPerfG PSU Undergraduate Programmers: Jiaqi Luo, Le Tu

What is an HPC Workflow ?

Holistic View

- One science effort across a period of time/campaign, or for 1 specific goal – may include multiple platforms or labs
- Track resource utilization, performance, and progress, data movement
- Includes System Services – power, resource balance, scheduling, monitoring, data movement, etc.
- Includes Data Center – power, cooling, physical placement of data and jobs
- Informed by & Interfaces with the Application and Experiment Views
- Includes hardware, system software layers, application

Foundational Work: All Layers of Workflow and their Relationships

Layer 0 – **Campaign**

- Process through time of repeated Job Runs
- Changes to approach, physics and data needs as a campaign or project is completed - Working through phases

Layer 1 – **Job Run**

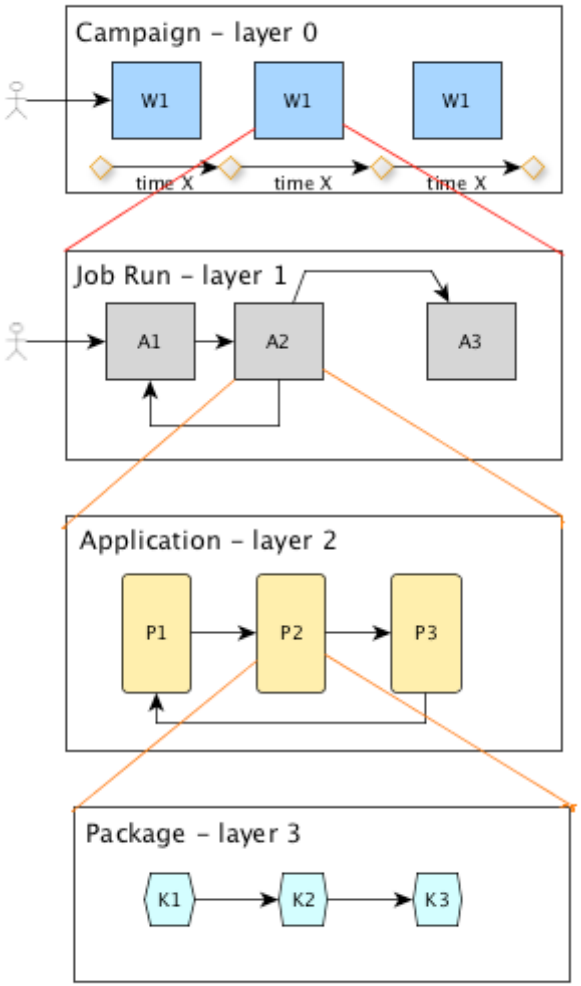
- Application to application that constitute a suite job run series
- May include closely coupled applications and decoupled ones that provide an end-to-end repeatable process with differing input parameters
- User and system interaction, to find an answer to a specific science question.

Layer 2 – **Application**

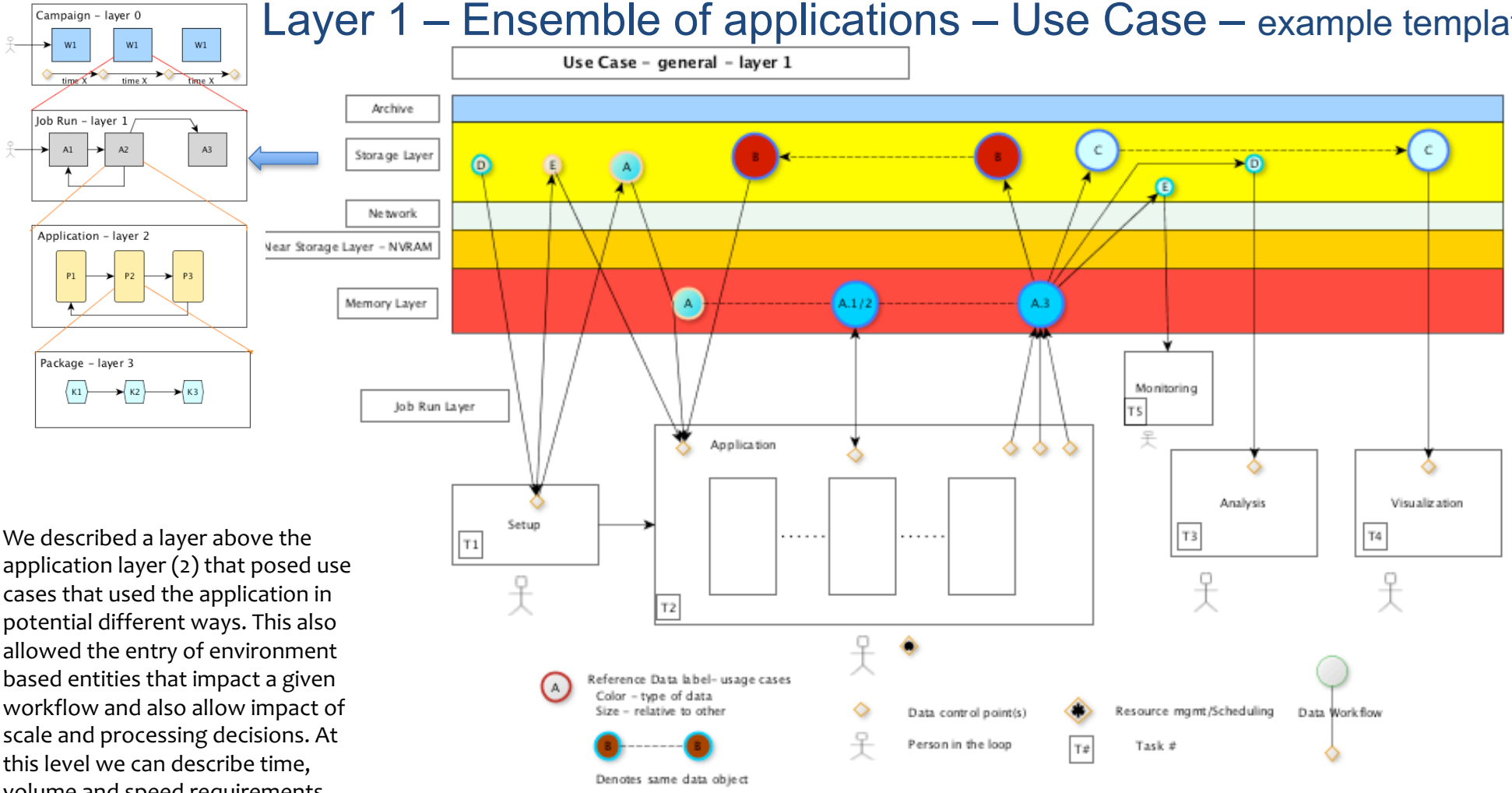
- One or more packages with differing computational and data requirements Interacts across memory hierarchy to archival targets
- The subcomponents of an application {P1..Pn} are meant to model various aspects of the physics

Layer 3 – **Package**

- The processing of kernels within a phase and associated interaction with various levels of memory, cache levels and the overall underlying platform
- The domain of the computer scientist



Layer 1 – Ensemble of applications – Use Case – example template



We described a layer above the application layer (2) that posed use cases that used the application in potential different ways. This also allowed the entry of environment based entities that impact a given workflow and also allow impact of scale and processing decisions. At this level we can describe time, volume and speed requirements.

Our Goal

Measurement infrastructure in support of Holistic HPC Workflow Performance Analysis and Validation

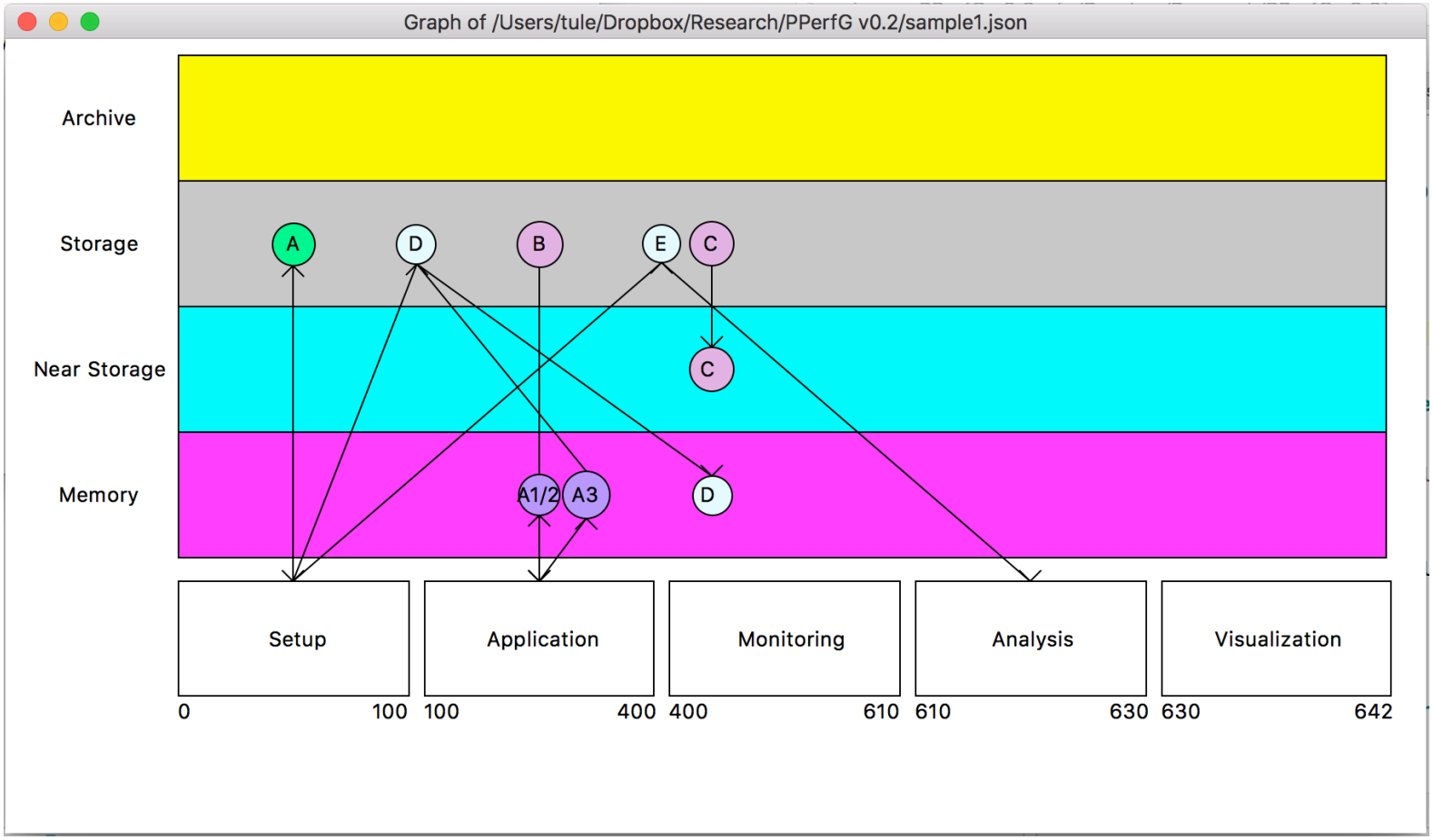
Goal #1: PPerfG

- Motivation: How can we automatically generate the workflow layer diagrams?
- Initial Focus:
  - Layer 2 (Application): One or more packages with differing computational and data requirements Interacts across memory hierarchy to archival targets
- Approach:
  - Implement simple prototype using python and TkInter
  - Investigate data collection options
  - Evaluate with a case study

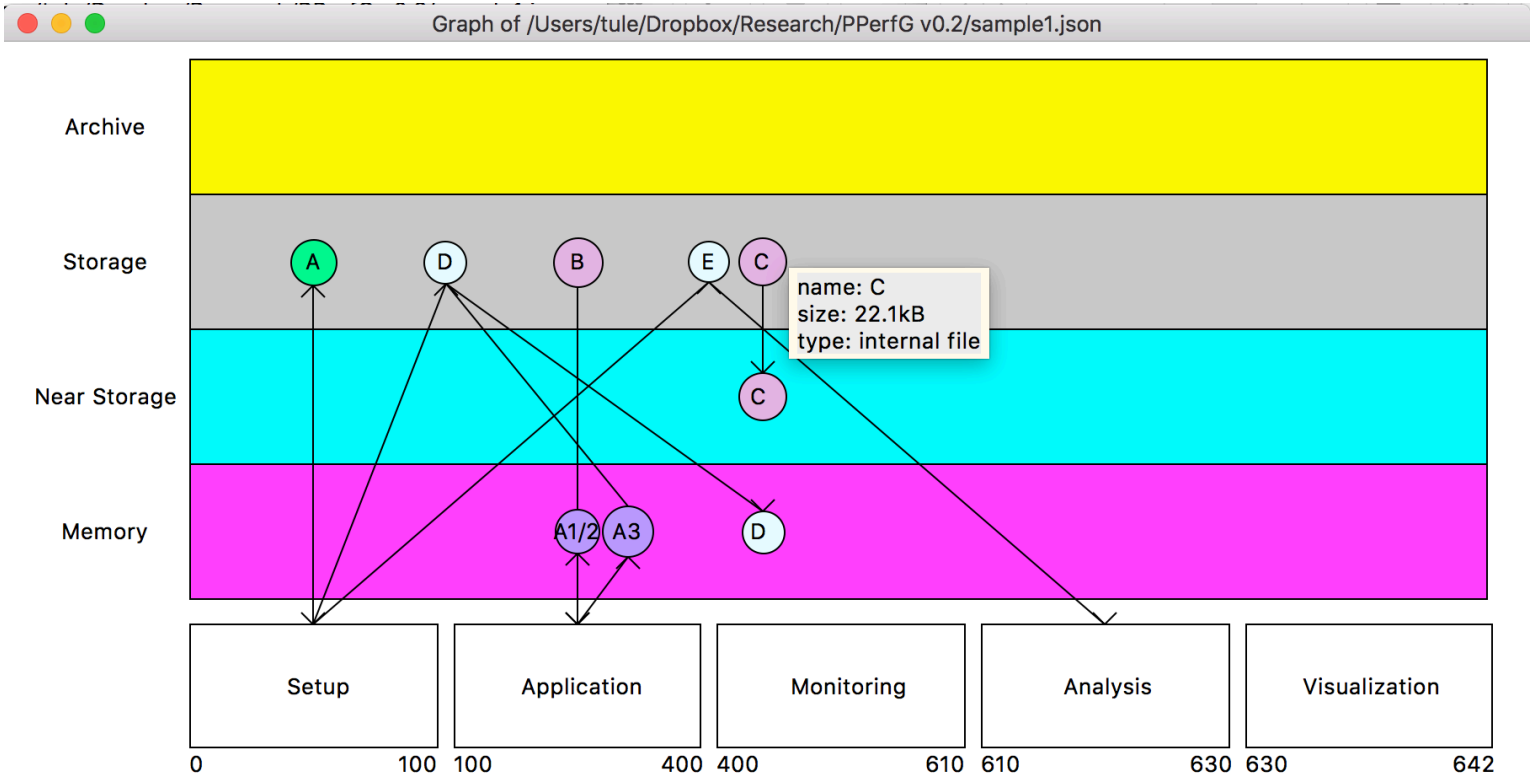
PPerfG

- PPerfG: A Visualization Tool for Holistic HPC Workflows for use in both performance diagnosis and procurement
- Captures the data movement behavior between storage layers, and between different stages of an application
- Challenges: Measurement and Data integration to generate the display
- Initial prototype developed with Python and TkInter

PPerfG Prototype



PPerfG Prototype



PPerfG Prototype: simple json input file

```
{
    "layers": [
        "Archive", "Storage", "Near Storage", "Memory", "Job Run"
    ],
    "tasks": [{
            "name": "Setup",
            "runtime": 100,
            "collectionTool": "darshan"
        },
        {
            "name": "Application",
            "runtime": 300,
            "collectionTool": "oss-man"
        },
        {
            "name": "Monitoring",
            "runtime": 210,
            "collectionTool": ""
        },
        {
    ],
    "datasets": {
        "D": {
            "size": 10,
            "type": "binary file"
        },
        "E": {
            "size": 8,
            "type": "binary file"
        },
        "A": {
            "size": 15,
            "type": ".txt file"
        },
    "events": [{
            "method": "r",
            "dataset": "D",
            "origin": "Storage",
            "destination": "Setup"
        },
        {
            "method": "w",
            "dataset": "E",
            "origin": "Setup",
            "destination": "Storage"
        },
        {
```

Case Study: The DroughtHPC[1] Project Goals

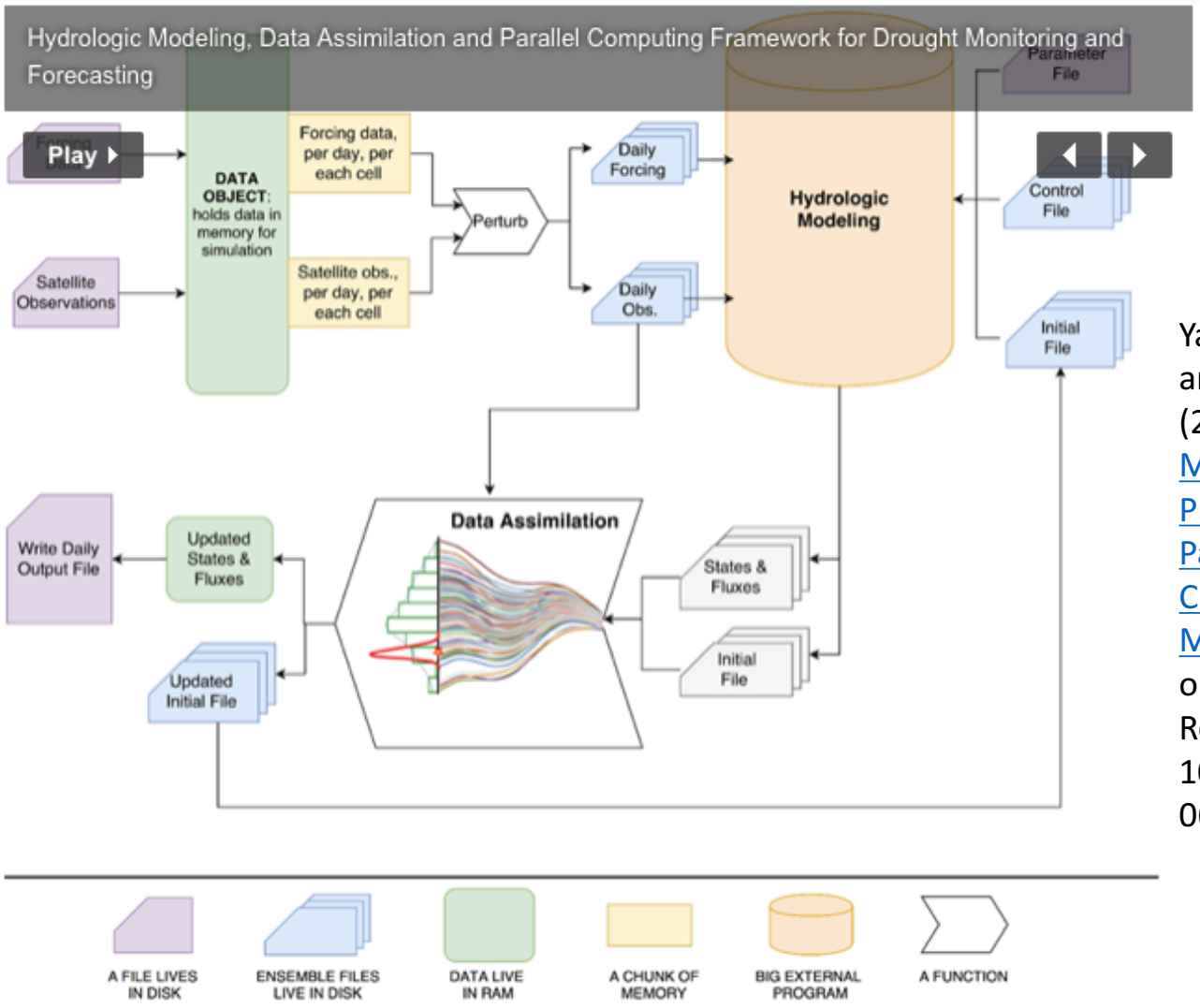
- Develop a performant implementation of DroughtHPC, a novel approach to drought prediction developed at Portland State University
- Scale the application to do finer-grained simulations, and to simulate a larger geographical area
  - DroughtHPC
    - improves prediction accuracy for a target geographical area
    - uses data assimilation techniques that integrate data from hydrologic models and satellite data
    - Uses Monte Carlo methods to generate a number of samples per cell
    - Inputs span a variety of data: soil conditions, snow accumulation, vegetation layers, canopy cover and meteorological data
    - Uses Variable Infiltration Capacity (VIC) Macroscale Hydrologic Model [2]

[1] *https://hamid.people.ua.edu/research.html*

[2] Liang, X., D. P. Lettenmaier, E. F. Wood, and S. J. Burges (1994), A simple hydrologically based model of land surface water and energy fluxes for general circulation models, *J. Geophys. Res.*, **99**(D7), 14415–14428, doi:10.1029/94JD00483

Case Study: DroughtHPC Code

- Application is written in Python, and uses two hydrologic models VIC [2] written in C, and PRMS [3] written in FORTRAN and C
- The modeling codes are treated as "black boxes" by the domain scientists
- Land surface of the target geographical area is modeled as a grid of uniform cells, and simulation divides it into jobs, with group of 25 cells in each job
- Data is Small by our standards: For a job that simulates 50 meteorological samples and one month time period:
  - input data size : 144.5 MB
  - satellite data : 132 MB
- Runtime for 1 job (25 cells) on single-node is approximately two hours with the initial Python prototype



Yan, H., C.M. DeChant, and H. Moradkhani (2015), Improving Soil Moisture Profile Prediction with the Particle Filter-Markov Chain Monte Carlo Method, IEEE Transaction on Geoscience and Remote Sensing, DOI: 10.1109/TGRS.2015.2432067

Initialization Overheads

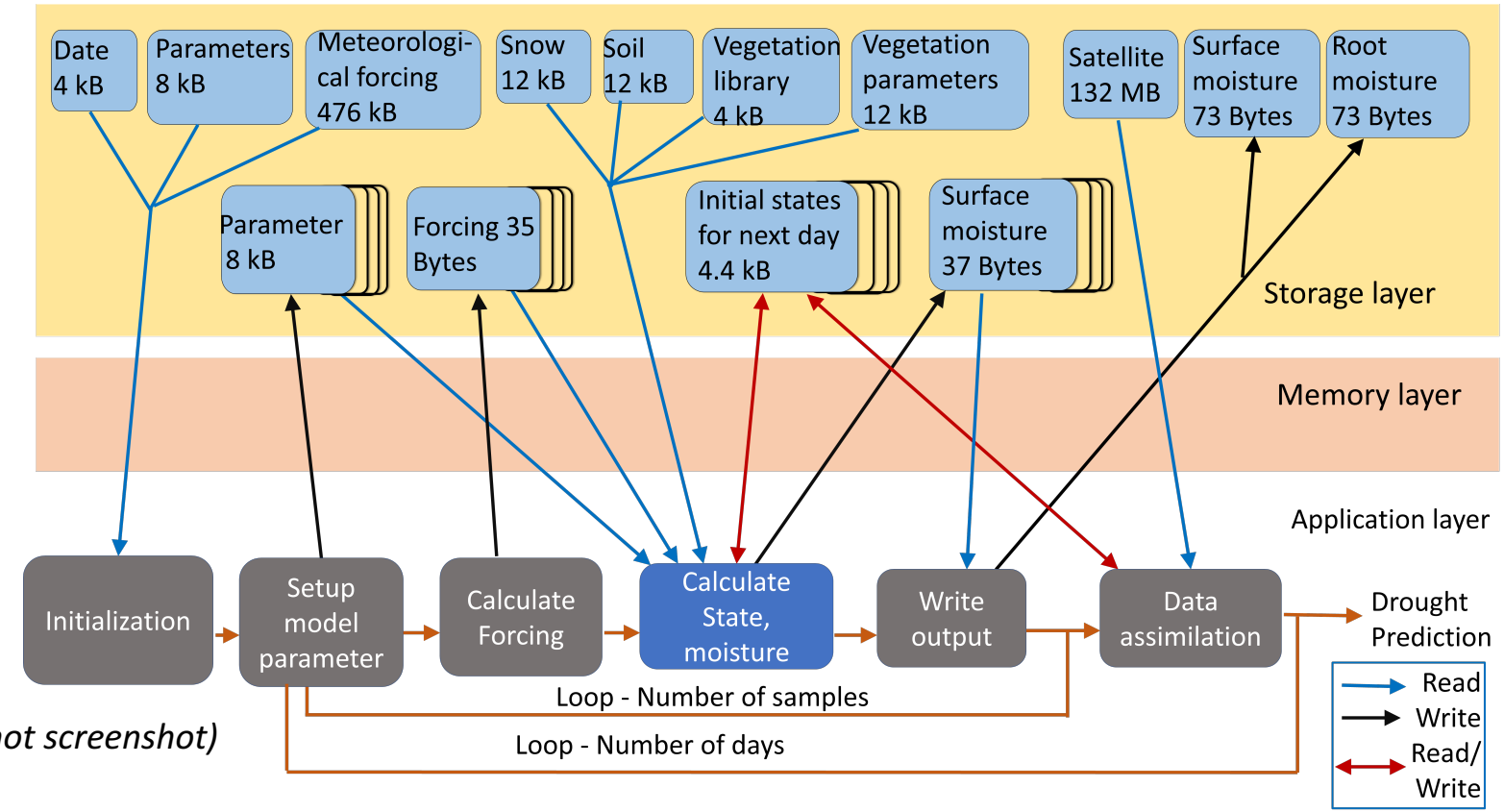
Model	Data	Initialization (Milliseconds)	Work (Milliseconds)	Write Output (Milliseconds)	Total
VIC 4 – ASCII text files	Sample – single cell	177.592 (99%)	0.241	0.144	177.977
	CRB 25 cells	4,079.126 (98%)	70.990	10.774	4,170.89
VIC 5 – NetCDF files	Sample – Stehekin data – 20 cells	19,088.990 (99%)	196.116	29.065	19,314.171
	CRB – 11280 cells	26,277.904 (47%)	29,001.285	80.398	55,359.587

- *Mean of 30 runs, simulation of 24 hours (one hour time steps)*
- *Columbia river basin (CRB) has 5359 cells in VIC 4 dataset, but it has 11280 cells in VIC 5 data set. The data used in the meteorological forcing is different between the two versions. VIC 5 data includes precipitation, pressure, temperature, vapor pressure, and wind speed. VIC 4 data specifies maximum temperature, minimum temperature, precipitation and wind speed.*

DroughtHPC / VIC calling patterns

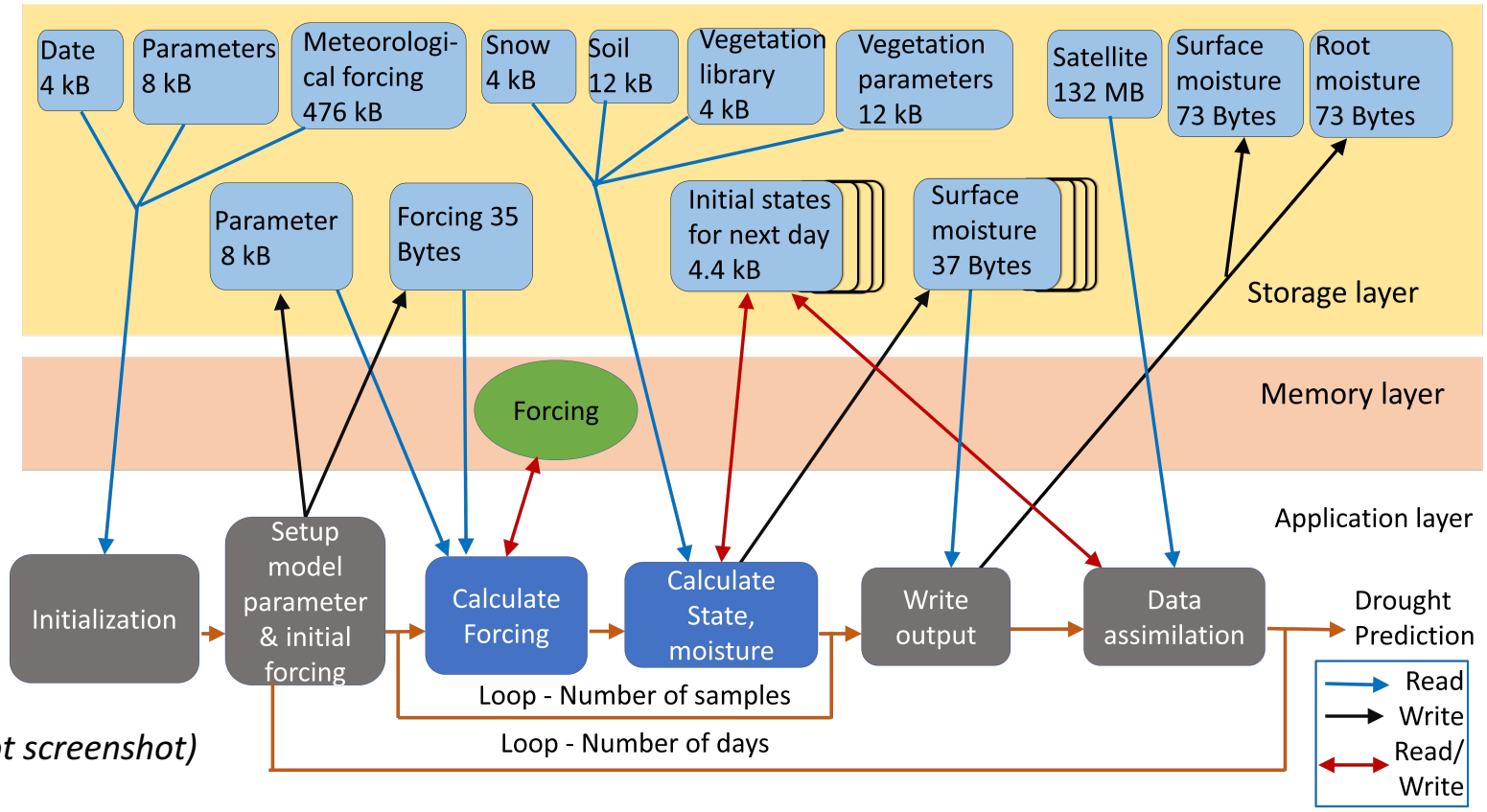
- Initial DroughtHPC prototype code (python) called VIC version 4 (“classic driver”):
    - For each grid cell
        - For each simulation time step
            - For each probabilistic sample
                - Call VIC
                - Use results to compute inputs for next time step
- VIC 4 is Time-before-space
- New VIC 5 “image driver” is Space-before-time, designed for call-once
    - Uses MPI, embarassingly parallel model (each cell computation is independent)
    - Single call to VIC can now compute over all data, reducing call overhead
- Our solution: add extensibility to VIC, inject our code into the model

PPerfG: Visualizing Data Patterns Across Separate Codes



drawing (not screenshot)

PPerfG: Illustrating the change in calling pattern



drawing (not screenshot)

PPerfG Data Collection

- Performance Data was collected with a variety of performance tools
  - No single performance tool provides all of the data we need
  - No tool characterizes the calling pattern / interactions between Python and VIC

- PerfTrack performance database[1] used to integrate the data postmortem but some integration was done manually
  - Interface over PostGreSQL relational database
  - Multiple runs for different measurement tools

- Json file was generated manually

[1]*Karen L. Karavanic, John May, Kathryn Mohror, Brian Miller, Kevin Huck, Rashawn Knapp, Brian Pugh, "Integrating Database Technology with Comparison-based Parallel Performance Diagnosis: The PerfTrack Performance Experiment Management Tool," SC2005.*

PPerfG Future Work

- How to ease comparison of different versions with PPerfG?
- Slider to move forward over time from start to finish?
- Can we generate the json automatically from PerfTrack?
- How to integrate application/developer semantics with measurement data?
  - How to link data structures in memory with files?
  - How to label the phases?
  - How to collect the loop information at the bottom?
- How to show scaling behaviors?
  - Number of files per simulation day?
  - Size of files per simulation cell?
  - Traffic Map idea: use edge colors to show data congestion

Conclusions and Future Work

- We propose a new performance metric: Workflow Critical Path
  - WCP: What part of the *Entire Workflow* to focus on?
  - DroughtHPC case study: pattern of file activity, calling pattern, overhead of VIC initialization
- We have designed PPerfG, a visualization for Workflow Layer 2: Application
  - Workflow Layers: different perspectives of an HPC workflow used in Holistic HPC Performance Diagnosis
  - Data Collection is challenging
  - Need to integrate Layer 3 (Package) – drilling down into DroughtHPC and VIC
- PerfTrack is useful to gather and do some integration of data
  - Currently the generation of json files is mostly manual

Acknowledgments

- PSU students Henry Cooney, Tu Le, Jiaqi Luo , and Kristina Frye contributed ideas and discussions and implemented software used in this project. Students in Karavanic's Accelerated Computing and Introduction to Performance courses performed analysis and parallelization of the VIC modeling code.
- This material is based upon work supported by the National Science Foundation under Grant No.1539605. *Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.*
- This work supported in part by a generous gift from Intel Corp.
- *Part of this work was conducted at the Ultrascale Systems Research Center (USRC) supported by Los Alamos National Laboratory under Contract No. DE-AC52-06NA25396 with the U.S. Department of Energy. The U.S. Government has rights to use, reproduce, and distribute this information. This work supported in part by Portland State University and by the New Mexico Consortium.*

Contact: karavan@pdx.edu