10TH INTERNATIONAL PARALLEL DATA SYSTEMS WORKSHOP
NOVEMBER 17, 2025

SUPPORTING SCIENCE THROUGH DATA MANAGEMENT SOFTWARE

ROB ROSS
Senior Computer Scientist
Mathematics and Computer Science Division
Argonne National Laboratory

THIS TALK

OUR TEAM FOCUS

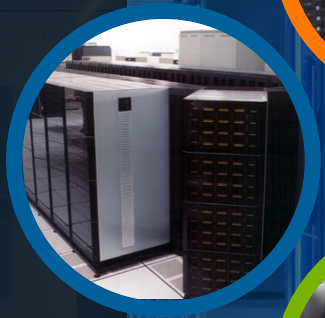
Research, development, and deployment of software for managing data on HPC platforms

- HPC platforms are moving targets
- Codes on HPC platforms push the limits of what is possible
- Data is key to successful scientific computing

Two software case studies

- Successes and challenges

Trends and future needs in data management



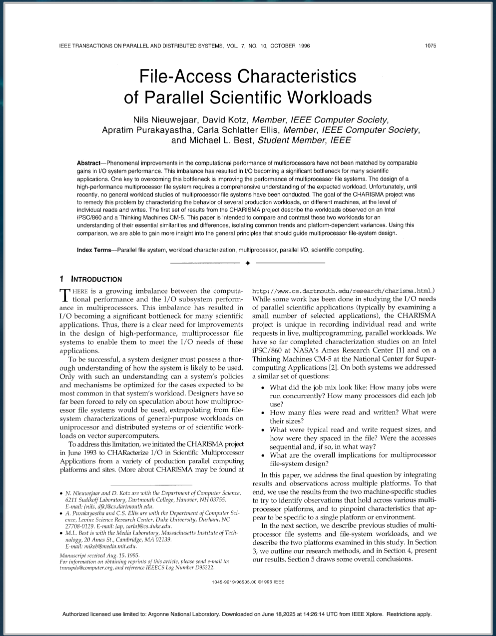
OBSERVATION

WHAT ARE SCIENTISTS DOING?

“To be successful, a system designer **must possess a thorough understanding of how the system is likely to be used.**”

“Designers have so far been **forced to rely on speculation about how multiprocessor file systems would be used …**”

“To address this limitation, we initiated the CHARISMA project in **June 1993** to **CHARacterize I/O in Scientific Multiprocessor Applications from a variety of production parallel computing platforms and sites.**”



IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 7, NO. 10, OCTOBER 1996 1075

File-Access Characteristics of Parallel Scientific Workloads

Nils Nieuwejaar, David Kotz, *Member, IEEE Computer Society*,
Apratim Purakayastha, Carla Schlatter Ellis, *Member, IEEE Computer Society*,
and Michael L. Best, *Student Member, IEEE*

Abstract—Phenomenal improvements in the computational performance of multiprocessors have not been matched by comparable gains in I/O system performance. This imbalance has resulted in I/O becoming a significant bottleneck for many scientific applications. One key to overcoming this bottleneck is improving the performance of multiprocessor file systems. The design of a high-performance multiprocessor file system requires a comprehensive understanding of the expected workload. Unfortunately, until recently, no general workload studies of multiprocessor file systems have been conducted. The goal of the CHARISMA project was to remedy this problem by characterizing the behavior of several production workloads, on different machines, at the level of individual reads and writes. The first set of results from the CHARISMA project describe the workloads observed on an Intel iPSC/860 and a Thinking Machines CM-5. This paper is intended to compare and contrast these two workloads for an understanding of their essential similarities and differences, isolating common trends and platform-dependent variances. Using this comparison, we are able to gain more insight into the general principles that should guide multiprocessor file-system design.

Index Terms—Parallel file system, workload characterization, multiprocessor, parallel I/O, scientific computing.

1 Introduction

There is a growing imbalance between the computational performance and the I/O subsystem performance in multiprocessors. This imbalance has resulted in I/O becoming a significant bottleneck for many scientific applications. Thus, there is a clear need for improvements in the design of high-performance, multiprocessor file systems to enable them to meet the I/O needs of these applications.

To be successful, a system designer must possess a thorough understanding of how the system is likely to be used. Only with such an understanding can a system's policies and mechanisms be optimized for the cases expected to be most common in that system's workload. Designers have so far been forced to rely on speculation about how multiprocessor file systems would be used, extrapolating from file-system characterizations of general-purpose workloads on uniprocessor and distributed systems or of scientific workloads on vector supercomputers.

To address this limitation, we initiated the CHARISMA project in June 1993 to CHARacterize I/O in Scientific Multiprocessor Applications from a variety of production parallel computing platforms and sites. (More about CHARISMA may be found at `http://www.cs.dartmouth.edu/research/charisma.html`.) While some work has been done in studying the I/O needs of parallel scientific applications (typically by examining a small number of selected applications), the CHARISMA project is unique in recording individual read and write requests in live, multiprogramming, parallel workloads. We have so far completed characterization studies on an Intel iPSC/860 at NASA's Ames Research Center [1] and on a Thinking Machines CM-5 at the National Center for Supercomputing Applications [2]. On both systems we addressed a similar set of questions:

- What did the job mix look like: How many jobs were run concurrently? How many processors did each job use?
- How many files were read and written? What were their sizes?
- What were typical read and write request sizes, and how were they spaced in the file? Were the accesses sequential and, if so, in what way?
- What are the overall implications for multiprocessor file-system design?

In this paper, we address the final question by integrating results and observations across multiple platforms. To that end, we use the results from the two machine-specific studies to try to identify observations that hold across various multiprocessor platforms, and to pinpoint characteristics that appear to be specific to a single platform or environment.

In the next section, we describe previous studies of multiprocessor file systems and file-system workloads, and we describe the two platforms examined in this study. In Section 3, we outline our research methods, and in Section 4, present our results. Section 5 draws some overall conclusions.

- *N. Nieuwejaar and D. Kotz are with the Department of Computer Science, 6211 Sudikoff Laboratory, Dartmouth College, Hanover, NH 03755. E-mail: {nils, dfk}@cs.dartmouth.edu.*
- *A. Purakayastha and C.S. Ellis are with the Department of Computer Science, Levine Science Research Center, Duke University, Durham, NC 27708-0129. E-mail: {ap, carla}@cs.duke.edu.*
- *M.L. Best is with the Media Laboratory, Massachusetts Institute of Technology, 20 Ames St., Cambridge, MA 02139. E-mail: mikeb@media.mit.edu.*

Manuscript received Aug. 15, 1995.
For information on obtaining reprints of this article, please send e-mail to: transpds@computer.org, and reference IEEECS Log Number D95222.

1045-9219/96$05.00 ©1996 IEEE

Nieuwejaar, Nils, et al. "File-access characteristics of parallel scientific workloads." *IEEE TPDS* 7.10 (**1996**): 1075-1089.

A DECADE PASSES

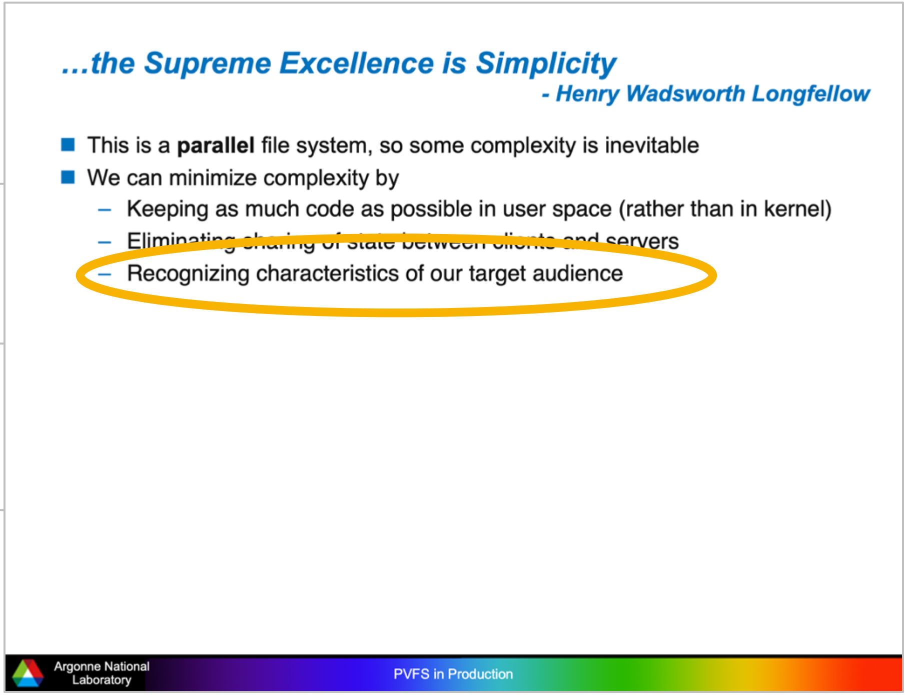
(no, seriously)

We’re busy putting PVFS on a Blue Gene system.

I’m at Livermore giving a talk about this work.

I’m still thinking about observation and understandability.

…



R. Ross. ”PVFS in Production”. Presented at LLNL. February 2007.

A DECADE PASSES

(no, seriously)

We’re busy putting PVFS on a Blue Gene system.

I’m at Livermore giving a talk about this work.

I’m still thinking about observation and understandability.

Talking with Jeff Vetter, I learn about a tool called mpiP…

Statistical Scalability Analysis of Communication Operations in Distributed Applications

Jeffrey S. Vetter   Michael O. McCracken

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, California, USA 94551

{vetter3,mccracken6}@llnl.gov

ABSTRACT

Current trends in high performance computing suggest that users will soon have widespread access to clusters of multiprocessors with hundreds, if not thousands, of processors. This unprecedented degree of parallelism will undoubtedly expose scalability limitations in existing applications, where scalability is the ability of a parallel algorithm on a parallel architecture to effectively utilize an increasing number of processors. Users will need precise and automated techniques for detecting the cause of limited scalability. This paper addresses this dilemma. First, we argue that users face numerous challenges in understanding application scalability: managing substantial amounts of experiment data, extracting useful trends from this data, and reconciling performance information with their application’s design. Second, we propose a solution to automate this data analysis problem by applying fundamental statistical techniques to scalability experiment data. Finally, we evaluate our operational prototype on several applications, and show that statistical techniques offer an effective strategy for assessing application scalability. In particular, we find that non-parametric correlation of the number of tasks to the ratio of the time for communication operations to overall communication time provides a reliable measure for identifying communication operations that scale poorly.

1 INTRODUCTION

Current trends in high performance computing suggest that users will be running their applications on scalable clusters of multiprocessors with hundreds, if not thousands, of processors in the near future [3, 15]. This unprecedented availability of computing resources motivates the need for precise and meaningful scalability analysis of these applications. By scalability, we mean *the ability of a parallel algorithm on a parallel architecture to effectively utilize an increasing number of processors* [6, 7, 13]. Undoubtedly, this new, high degree of concurrency will expose scalability limitations of applications that, at lower levels of concurrency, might have been shrouded by other application or system characteristics. Furthermore, perpetual improvements in single node performance will continue revealing the scalability limitations of communication operations in their distributed applications.

Although metrics like execution time, speedup, and efficiency [14] help quantify scalability on an abstract level, users need precise information about poorly scaling communication operations in their application. In addition, for any analysis to help users understand their application’s scalability, the technology should be able to explain scalability phenomena in terms of decisions a user makes while designing their application.

To this end, we propose an automated technique that uses familiar statistical techniques to direct a user's attention on poorly scaling communication operations in their application. Our method digests the results of multiple application experiments and suggests communication operations whose growth has a positive correlation with the number of tasks. We empirically evaluate the usefulness of these techniques on nine applications with both fixed and scaled problem sizes. Our results show that, in every case, our method quickly identifies the communication operations that grow to dominate the application's execution time during highly parallel experiments. More importantly, our technique selects operations that a user might not normally locate when using simpler methods.

1.1 Background

The analysis of scalability is not a new concept [6, 14]. Yet many users find scalability analysis of their applications difficult, time-consuming, and inconclusive. Despite the fact that investigators have proposed numerous metrics, such as speedup, scaled speedup, efficiency, and iso-efficiency, these metrics provide only an abstract and broad view of application scalability behavior. They do not provide specific evidence that allows users to understand and optimize their applications. Worse, the experimental process of measuring application scalability, in practice, can generate an intractable amount of data. This fact alone can hinder the effort, because users are basically inundated with lots of uninteresting, redundant data.

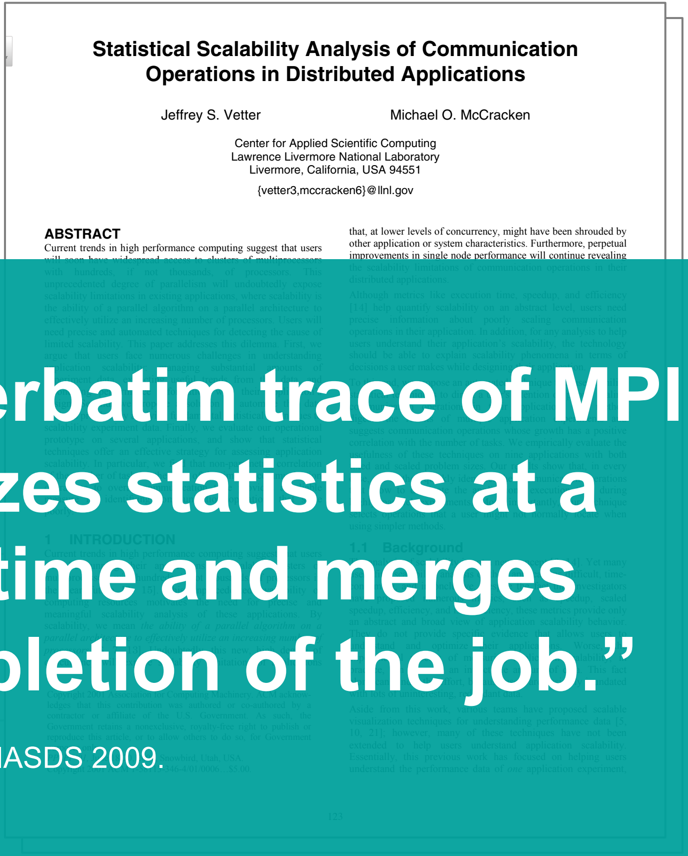
Aside from this work, various teams have proposed scalable visualization techniques for understanding performance data [5, 10, 21]; however, many of these techniques have not been extended to help users understand application scalability. Essentially, this previous work has focused on helping users understand the performance data of *one* application experiment,

PPoPP’01, June 18-20, 2001, Snowbird, Utah, USA.

J. S. Vetter and M. O. McCracken, “Statistical scalability analysis of communication operations in distributed applications,” SIGPLAN Notices, vol. 36, no. 7, pp. 123–132, 2001.

A DECADE PASSES

(no, seriously)


Statistical Scalability Analysis of Communication Operations in Distributed Applications

Jeffrey S. Vetter Michael O. McCracken

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, California, USA 94551

{vetter3,mccracken6}@llnl.gov

ABSTRACT

Current trends in high performance computing suggest that users

that, at lower levels of concurrency, might have been shrouded by other application or system characteristics. Furthermore, perpetual improvements in single node performance will continue revealing

"Rather than capture a verbatim trace of MPI activity, (mpiP) summarizes statistics at a per process level at run time and merges the statistics at the completion of the job."

P. Carns et al. "24/7 Characterization of Petascale I/O Workloads." IASDS 2009.

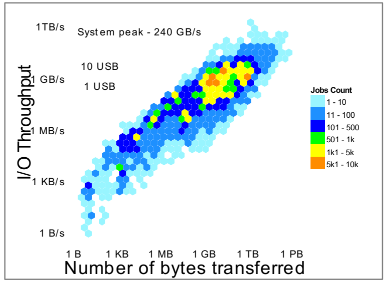
DARSHAN I/O CHARACTERIZATION TOOLKIT

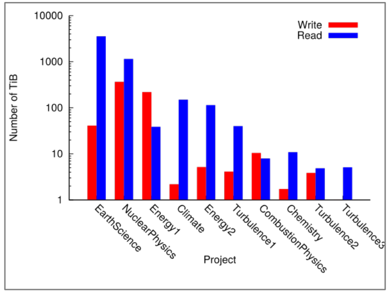
Designed to capture an accurate picture of application I/O behavior, including properties such as patterns of access within files, with minimum overhead

Not primarily a tracing library
although it can do that

Useful for:

- Identifying trends in how applications are using the storage system
- Identifying applications with problem behaviors
- Understanding I/O behavior at multiple software layers
- Capturing data inputs and outputs of workflow tasks





H. Luu et al. “A Multiplatform Study of I/O Behavior on Petascale Supercomputers”, HPDC 2015, 2015. (top)
Philip Carns et al. “Understanding and improving computational science storage access through continuous characterization”, ACM ToS, 7:8:1-8:26, October 2011. (bottom)

DARSHAN IMPLEMENTATION

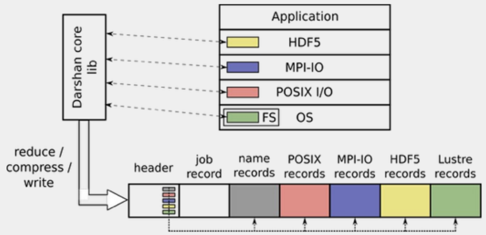
Darshan records file access statistics for each process as app executes

At app shutdown, collect, aggregate, compress, and write log data

Darshan can insert I/O instrumentation at link-time (for static/ dynamic executables) or at runtime using LD_PRELOAD (for dynamic executables)

After job completes, analyze Darshan log data

- PyDarshan: Python analysis module for Darshan logs
- `darshan-parser:` provides complete text-format dump of all counters in a log file



Darshan provides a variety of modules capturing specific classes of information, typically related to a specific API or file system.

Thanks to S. Snyder for this slide (and all his hard work on Darshan!).

WHAT HAVE WE GOTTEN RIGHT?

Low overhead, “in production” focus

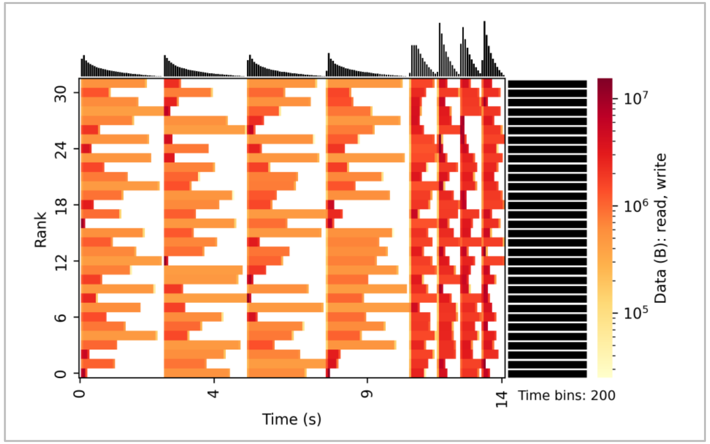
Modularity/ extensibility (eventually)

Collaboration with facilities

Compression

Data releases

Simple, structured output for processing



Temporal view of I/O (in bytes), broken down by MPI rank. Bins are populated based on number of bytes read/written in the given time interval (temporal binning). Bins are combined as needed to maintain fixed memory footprint (time series coalescing).

The vertical bar graph sums each time slice across all ranks to show the total I/O time, while the horizontal bar graph sums all the I/O events for each rank to illustrate how I/O is distributed across ranks.

NUMEROUS COMMUNITY CONTRIBUTIONS

Cong Xu

and Intel's High Performance Data Division contributed DxT tracing capability

Wei-keng Liao

contributed a Parallel netCDF module

Glenn Lockwood

worked out non-MPI support

Jakob Luettgau

provided the first complete implementation of PyDarshan, which has become the de facto tool for analyzing logs

Jean Luca Bez and Suren Byna

have provided visual analysis and advisory tools

Enabling Agile Analysis of I/O Performance Data with PyDarshan

Jakob Luettgau, luettgauj@acm.org, Inria, Rennes, France
Shane Snyder, ssnyder@mcs.anl.gov, Argonne National Laboratory, Lemont, IL, USA
Tyler Reddy, treddy@lanl.gov, Los Alamos National Laboratory, Los Alamos, NM, USA
Nikolaus Awtrey, Los Alamos National Laboratory, Los Alamos, NM, USA
Kevin Harms, harms@alcf.anl.gov, Argonne National Laboratory, Lemont, IL, USA
Jean Luca Bez, jlbez@lbl.gov, Lawrence Berkeley National Laboratory, Berkeley, CA, USA
Rui Wang, rwang@anl.gov, Argonne National Laboratory, Lemont, IL, USA
Rob Latham, robl@mcs.anl.gov, Argonne National Laboratory, Lemont, IL, USA
Philip Carns, carns@mcs.anl.gov, Argonne National Laboratory, Lemont, IL, USA

ABSTRACT

Modern scientific applications utilize numerous software and hardware layers to efficiently access data. This approach poses a challenge for I/O optimization because of the need to instrument and correlate information across those layers. The Darshan characterization tool seeks to address this challenge by providing efficient, transparent, and compact runtime instrumentation of many common I/O interfaces. It also includes command-line tools to generate actionable insights and summary reports. However, the extreme diversity of today's scientific applications means that not all applications are well served by one-size-fits-all analysis tools.

In this work we present PyDarshan, a Python-based library that enables agile analysis of I/O performance data. PyDarshan caters to both novice and advanced users by offering ready-to-use HTML reports as well as a rich collection of APIs to facilitate custom analyses. We present the design of PyDarshan and demonstrate its effectiveness in four diverse real-world analysis use cases.

KEYWORDS

High-Performance Computing, Storage, Performance Analysis, Input/Output

ACM Reference Format:

Jakob Luettgau, Shane Snyder, Tyler Reddy, Nikolaus Awtrey, Kevin Harms, Jean Luca Bez, Rui Wang, Rob Latham, and Philip Carns. 2023. Enabling Agile Analysis of I/O Performance Data with PyDarshan. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3624062.3624207

1 INTRODUCTION

Understanding how applications and workflows access data is an incredibly important aspect of computational workloads on HPC platforms. This importance is amplified by the increasing fidelity and rate of data produced by scientific instruments and simulations [9, 20, 40]. At the same time, new workloads are emerging that stress I/O systems differently from traditional applications [12, 22]. The variety of workloads present on modern platforms include simulations, experimental data analysis, inference and training of machine learning, and hybrid workloads incorporating elements of each. Many applications run into I/O bottlenecks that require understanding I/O behavior, especially as they scale. As a result, better tools for I/O analysis are vital. In the short term, such tools can directly improve application and workflow performance. In the long term, such tools can provide insight into compute, network, and storage utilization to inform the construction of efficient future systems.

Scientific applications use many different programming interfaces for I/O, from high-level interfaces for conveniently describing data to low-level interfaces that interact more directly with storage hardware. In many cases, data is translated through multiple layers on its way to a distributed storage system. This complexity makes analyzing and optimizing application and workflow I/O a daunting task requiring multilevel instrumentation. A common solution for the instrumentation of HPC applications is Darshan [18], which allows efficient instrumentation for profiling and tracing across many I/O layers such as POSIX, STDIO, MPI-IO, HDF5, and Lustre.

Up to now, the infrastructure for analyzing Darshan data has been limited to ad hoc scripts and C-coded tools. In this paper we present PyDarshan, a library of tools designed to enable agile analysis of I/O performance data captured using Darshan. Our contributions with PyDarshan are the following:

- Connect Darshan performance data to a rich Python ecosystem of data science, machine learning, and visualization libraries
- Promote interoperability of performance analysis tools by facilitating access to reusable analysis routines
- Enable more efficient access to Darshan data for the analysis of large collections of Darshan logs and longitudinal studies
- Present multiple use cases to illustrate how PyDarshan capabilities enable agile development of insightful I/O analysis tools

PyDarshan opens up the analysis of I/O behavior to a much broader audience, allowing a deeper understanding of the importance and characteristics of I/O in scientific computing today and

SC-W 2023, November 12–17, 2023, Denver, CO, USA

ACM ISBN 979-8-4007-0785-8/23/11…$15.00
https://doi.org/10.1145/3624062.3624207

Luettgau, Jakob, et al. "Enabling agile analysis of I/O performance data with PyDarshan." *Proceedings of the SC'23 PDSW Workshop*. 2023.

WHAT NEEDS MORE WORK?

I'm sure this isn't a complete list

Coverage

Coverage is still often very low (right) due to unclean job terminations, opting out, etc.

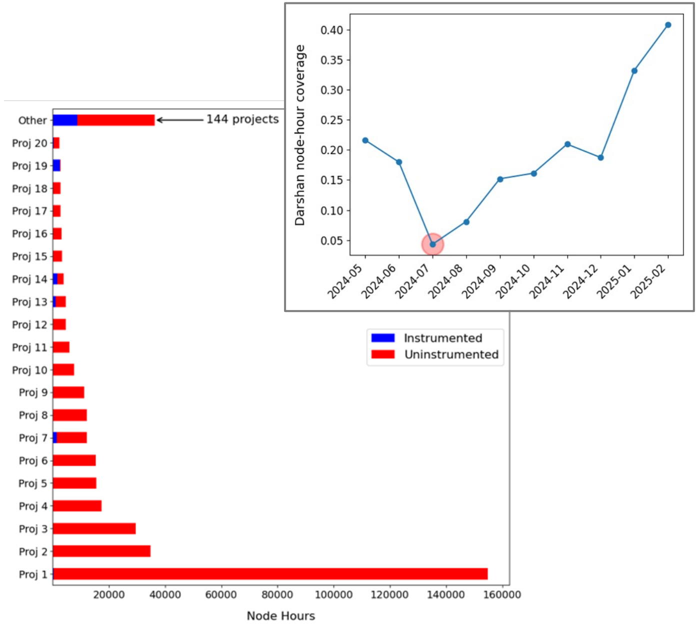
Data Fusion

Data isn't especially easy to fuse with other sources (e.g., time series data).

https://doi.org/10.1109/CLUSTER51413.2022.00082

Python Workflows

So many files! Have mitigated with filters. Frameworks that execute many tasks under the same process are a separate challenge.



Darshan coverage on Polaris system at Argonne over 10-month period: ranges from 5% to over 40%.

S. Snyder et al. " Expanding Community Access to Real-world HPC Application I/O Characterization Data Using Darshan." Cray Users Group Meeting. May 2025.

SERVICE DEVELOPMENT AND SPECIALIZATION

U.S. DEPARTMENT of ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.
Argonne NATIONAL LABORATORY

HPC DATA SERVICES

In the mid-1990s, HPC data services were synonymous with NFS.

Some projects were emerging that looked to specialize for HPC

- PIOUS – user space, data in local files, transactions
- Vesta – 2D view of data, hashing filenames for fast lookup, algorithmic data placement

Commodity clusters, MPI, and Linux were relatively new things.

PIOUS: A Scalable Parallel I/O System for Distributed Computing Environments*

Design and Implementation of the Vesta Parallel File System

Peter F. Corbett   Dror G. Feitelson
IBM T. J. Watson Research Center
P. O. Box 218, Yorktown Heights, NY 10598

Abstract

The Vesta parallel file system is designed to provide parallel file access to application programs running on multicomputers with parallel I/O subsystems. Vesta uses a new abstraction of files: a file is not a sequence of bytes, but rather it can be partitioned into multiple disjoint sequences that are accessed in parallel. The partitioning — which can also be changed dynamically — reduces the need for synchronization and coordination during the access. Some control over the layout of data is also provided, so the layout can be matched with the anticipated access patterns.

The system is fully implemented, and is beginning to be used by application programmers. The implementation does not compromise scalability or parallelism. In fact, all data accesses are done directly to the I/O node that contains the requested data, without any indirection or access to shared metadata. There are no centralized control points in the system.

1 Introduction

With the recent introduction of scalable parallel multicomputers by Cray Research and IBM, it is clear that the next generation of supercomputers will be parallel machines [4]. These supercomputers will achieve their high performance by employing hundreds of workstation-class processors in tandem. Likewise, parallel I/O subsystems will be used in order to balance the I/O capabilities with the processing capabilities. This is already being done by most vendors of parallel machines. Examples include the Scalable Disk Array of the CM-5 from Thinking Machines, and the I/O partitions of the Intel Paragon.

An application's interface to a system's I/O facilities is most often through a file system. Multicomputer file systems make use of the parallel I/O subsystem by declustering files, meaning that the blocks of each file are distributed across distinct I/O nodes. For example, this is done in Intel's Concurrent File System (CFS) [7] and Thinking Machines' Scalable File System (sfs) [6]. However, this feature is hidden from the users. The user interface employs the traditional notion of a file being a linear sequence of records (or bytes), and the mapping to multiple disks is done beneath the covers. Thus users are prevented from tailoring their I/O patterns to match the available disks. Users do not even know where block boundaries are, so a small access might require data residing on two different I/O nodes.

In contrast, the Vesta file system exposes its parallel structure at the user interface [1, 2]. While users do not have full control over the mapping of data to disks, they are able to create files that are distributed so as to match their applications. For example, in a matrix-multiply application each compute node only needs to access a band of rows or columns of each matrix. Vesta allows the files containing the matrices to be partitioned into such bands. Furthermore, it is possible to have each band stored on a distinct I/O node. Then each processor only accesses one I/O node, reducing interference among processors and message passing overhead. Finally, Vesta also allows parallel file access using other decompositions of the file data: for example, a file that was stored as a set of rows can also be accessed by column. This does not require any movement of the data.

The next section outlines the basic guidelines underlying the Vesta design. Subsequent sections show how these guidelines were applied to various issues, including the design of the user interface, the handling of metadata, the implementation of data access, and support for sharing files among multiple processes. For each issue, we discuss the current implementation, its consequences, and possible alternatives.

2 Design Guidelines

The overriding goal of the Vesta file system is to provide high performance for I/O intensive scientific applications on massively parallel multicomputers. This workload is characterized by very large files which are mostly read. In many cases, the file data is distributed among the application processes, such that each reads a certain part, and all together read the whole file. This workload has much in common with other I/O intensive workloads, such as parallel database analysis, or video services. This is different enough from I/O on traditional supercomputers that storing the whole file sequentially on one device, even a very fast device, is not a good solution. The Vesta design was guided by the following principles:

63

0-8186-5680-8/94 $03.00 © 1994 IEEE

Steven Moyer and V. S. Sunderam. "PIOUS: a scalable parallel I/O system for distributed computing environments." Proceedings of SHPCC-92. IEEE, 1992.
Peter F. Corbett and D. G. Feitelson. "Design and implementation of the Vesta parallel file system." Proceedings of SHPCC-94. IEEE, 1994.

LEARNING FROM A SERIES OF SERVICES

1994–2009

PVFS and PVFS2

- Structured data access
- Concurrency control – sockets, custom state machines
- Breaking implementation into simple modules – BMI and Trove
- linux-fsdevel hell

2008–2011

IOFSL

- I/O forwarding service
- We got Mercury out of this!
- We got to know the SCR (now Unify) team

2009–2015

Triton

- Versioned I/Os in object stores
- Source-to-source translation to aid programmability
- We convinced ourselves there's no one solution
- DOD funding is "different"

2015–Present

Mochi

- Still learning…

A PROJECT ABOUT BUILDING HPC SERVICES

Will someone fund that?

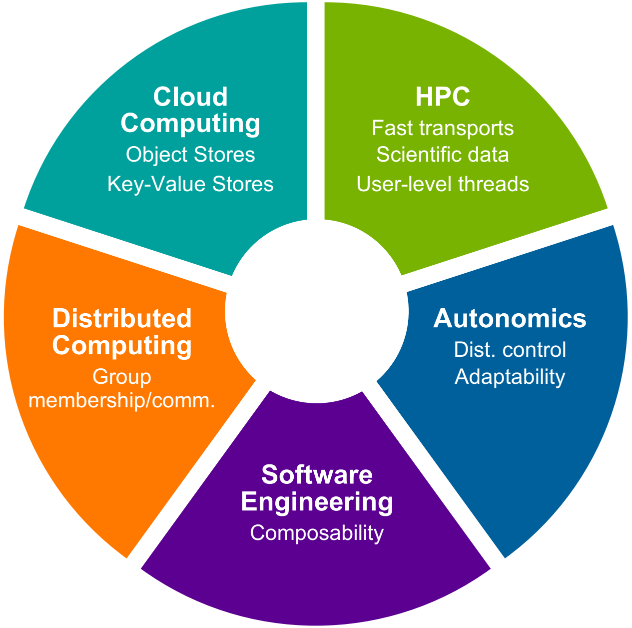
Started in 2015 with Garth Gibson and team as an effort to explore how composition and componentization could be exploited in HPC storage services

Grew into Mochi, an effort to define a methodology and develop a set of components for building HPC (data) services

- Faster development and porting time
- Exploit HW capabilities
- Lots of code reuse

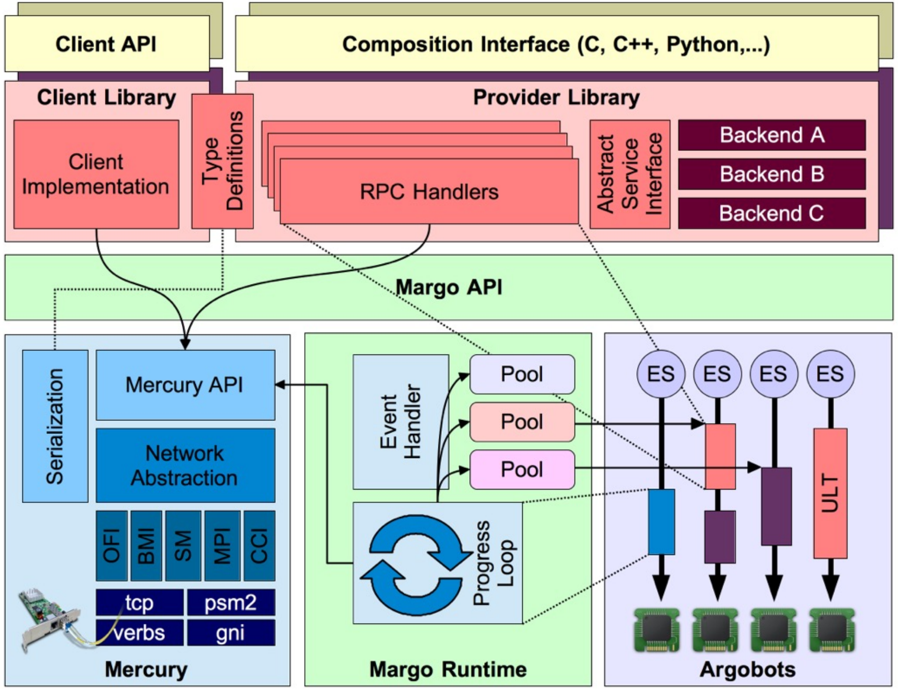
Current focus is more on ease of configuration and improved adaptivity

(e.g., dynamic data placement, adding/removing resources)



The Mochi project draws inspiration, algorithms, and code from a variety of related computer science fields.

MOCHI IMPLEMENTATION



Thanks to M. Dorier for this slide.

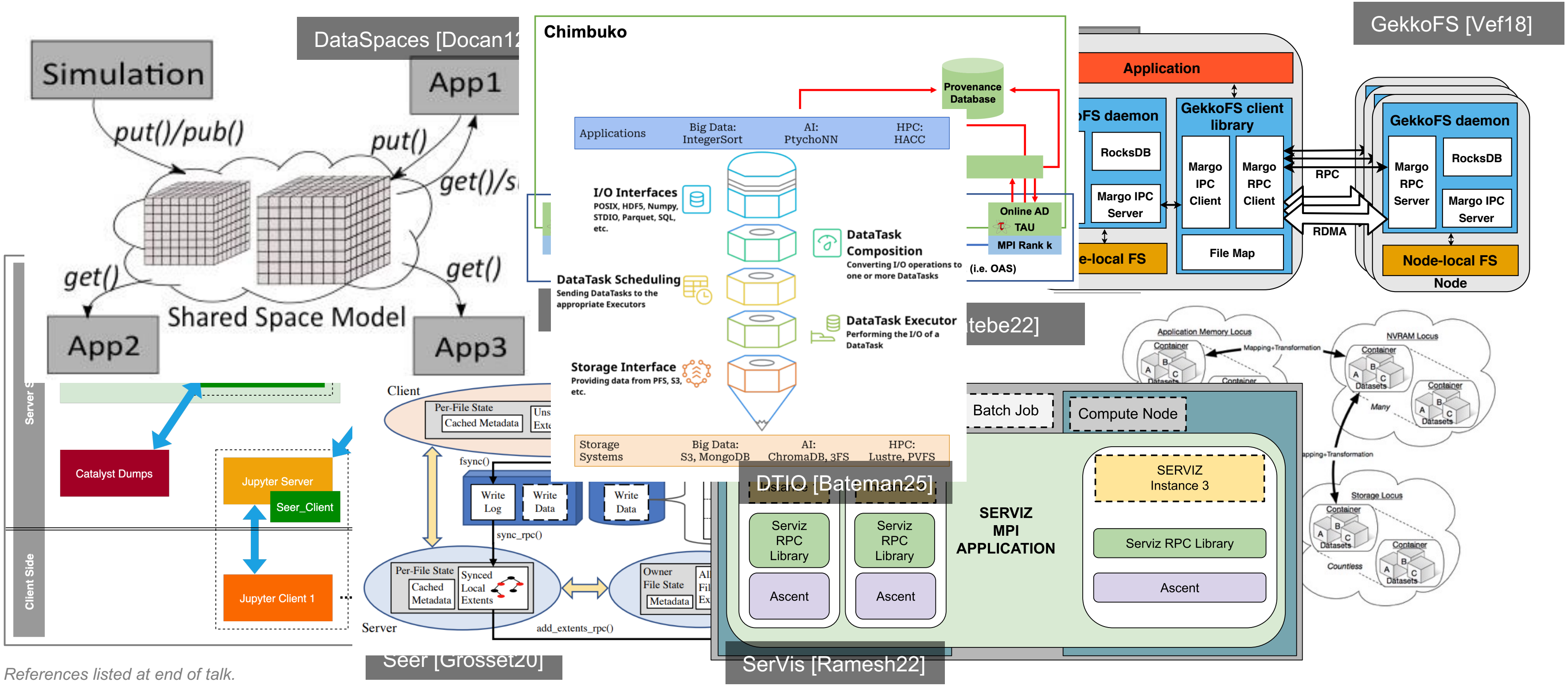
MARGO (C) / THALLIUM (C++)

- Very easy to understand and program with
- Hides the Mercury progress loop
- No more callbacks! Everything is a ULT
- RPC (Remote Procedure Calls) turned into ULT
- Argobots takes care of scheduling to resources

METHODOLOGY

- Components provide a client and a server library
- Functionalities implemented in different ways
- Everything can look like an RPC (even if everything executes in the same process or node)

COMMUNITY UPTAKE



References listed at end of talk.

WHAT HAVE WE GOTTEN RIGHT?

RPC *and* RDMA

Mercury is a good level of abstraction for services and exposes what is needed for performance.

Thanks Jerome Soumagne!

C++

It is saving competent developers loads of time as compared to C development. Maybe more so than componentization in general?

Thanks Matthieu Dorier!

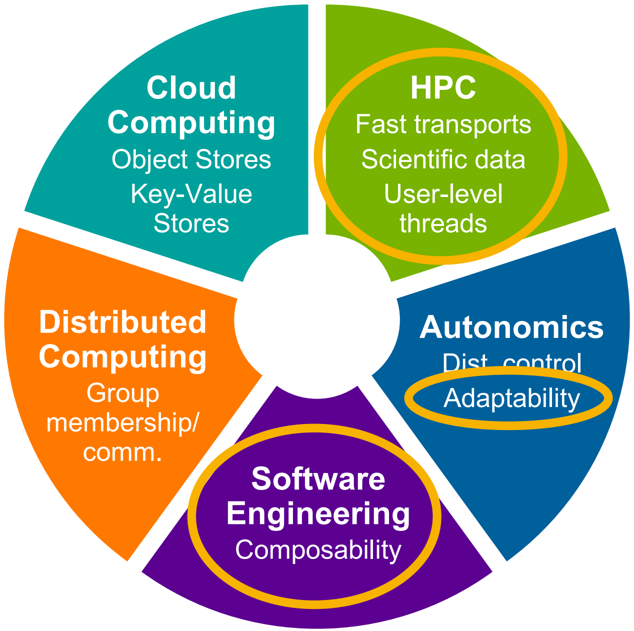
User-level threads

Argobots as a tool to help manage concurrency and balance work has been invaluable.

Thanks Sangmin Seo, we miss you!

Toolkit, not a service

This has made it easy (we think) for other teams to use Mochi while having clear intellectual ownership of their service(s).



WHAT DID WE MISS?

Configuration and Tuning

It’s still very difficult. Modern nodes are complex, and every system is different.

Interoperability

We don’t have a good story for interacting with, for example, things using Arrow.

Group Membership

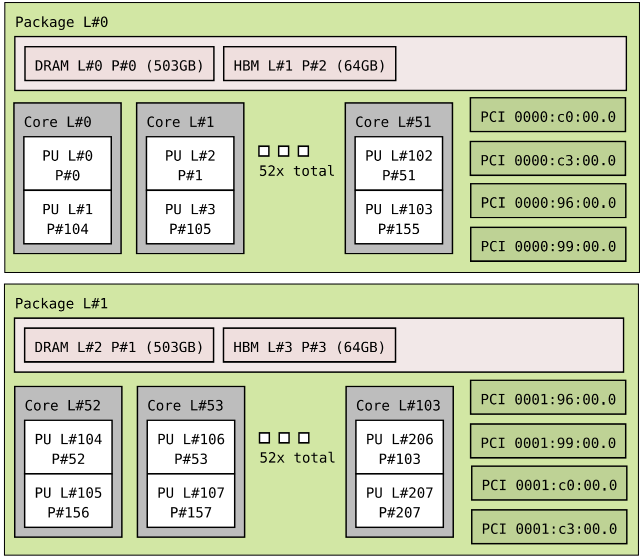
Nobody seems to need it?

Flow Control

People aren’t clamoring for it, yet. Not clear in what component it should be implemented?

Our Services

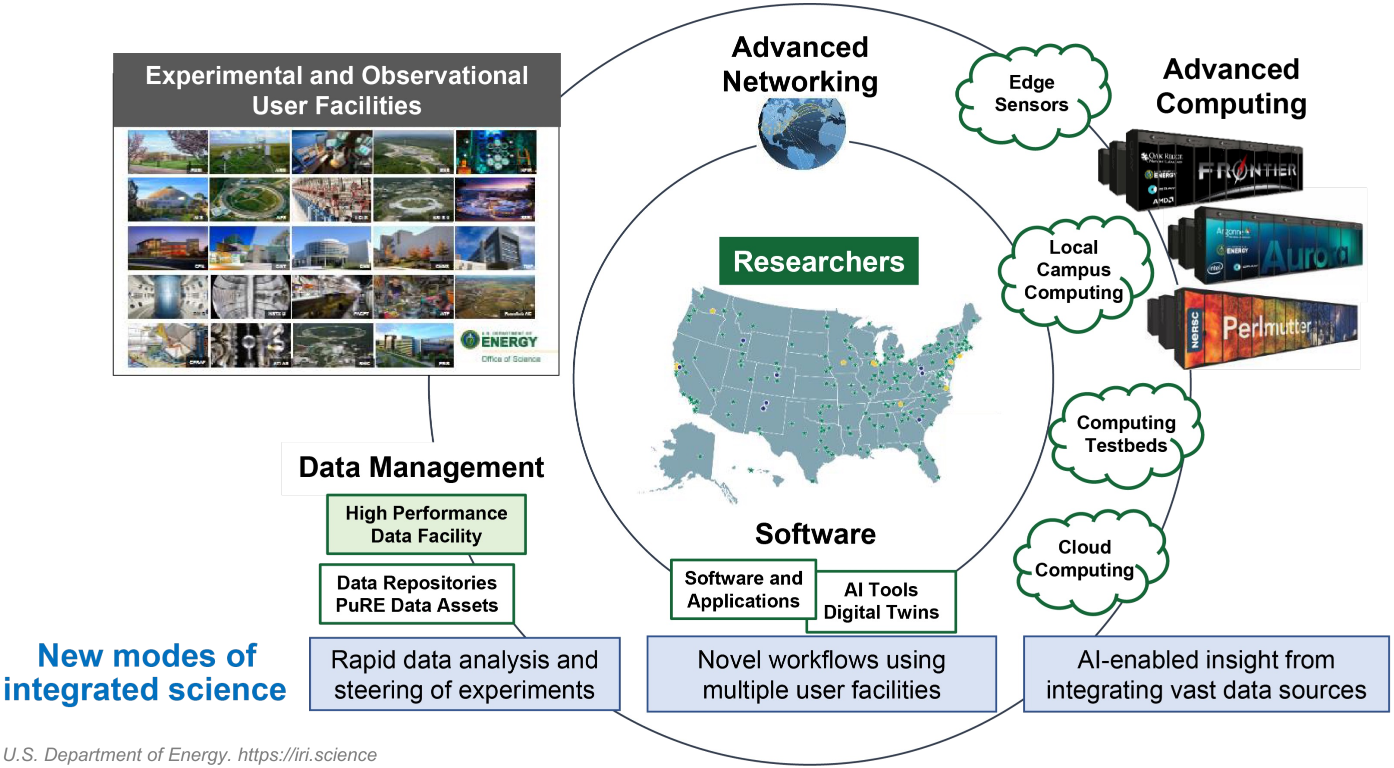
Ironically (?) nobody uses the actual services we have built, preferring to build their own.



Simplified view of Aurora’s node topology. Each PCI device corresponds to a distinct Slingshot network card (8!) attached to the same system fabric. Fully exploiting networking resources is challenging.

TRENDS AND IMPLICATIONS

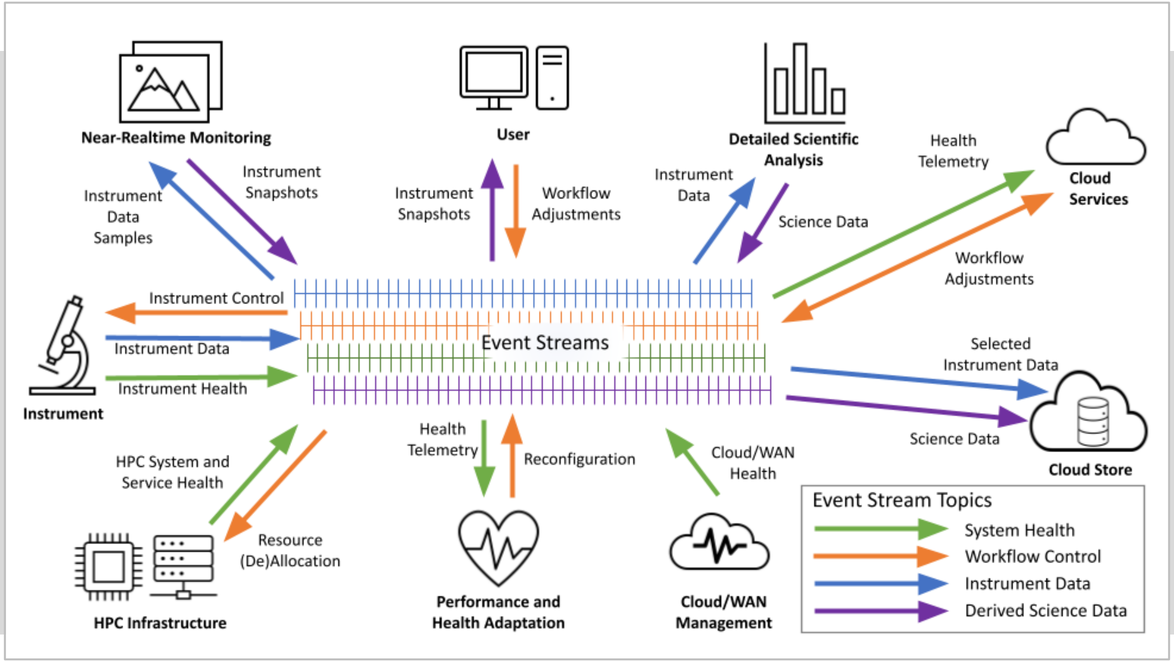
DISTRIBUTED SCIENCE ON FEDERATED RESOURCES



Slide source: *U.S. Department of Energy. https://iri.science*

EVENT STREAMS IN DISTRIBUTED SCIENCE

Ongoing work in Ian Foster's Diaspora project



Slide source:
https://diaspora-project.github.io

AUTOMATING THE SCIENTIFIC METHOD

STUDY

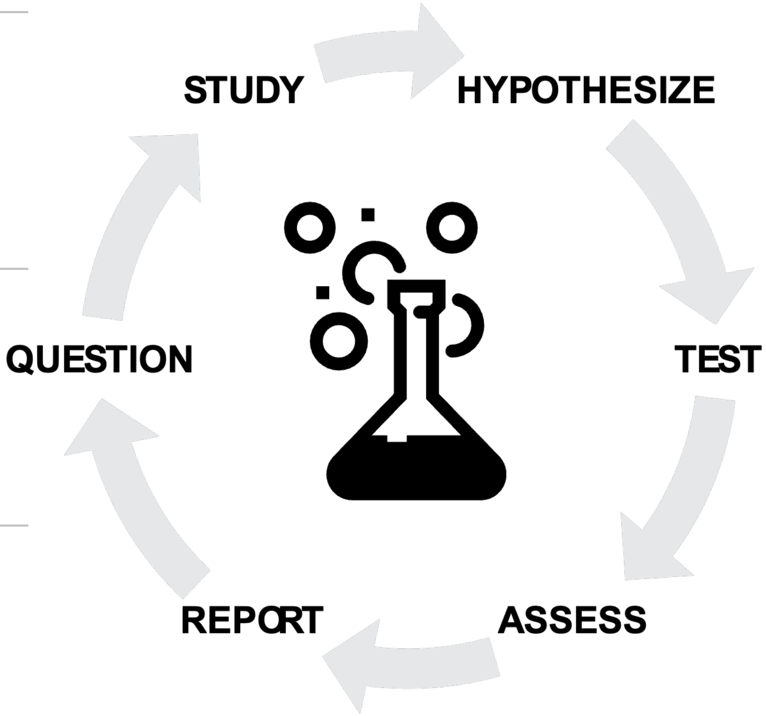
Extraction, integration, and reasoning with knowledge at scale

QUESTION

Tools help **identify new questions** based on needs and gaps in knowledge

REPORT

Machine representation of knowledge leads to new hypotheses and questions



HYPOTHESIZE

Generative models automatically propose new hypotheses that expand discovery space

TEST

Robotic labs automate experimentation and bridge digital models and physical testing

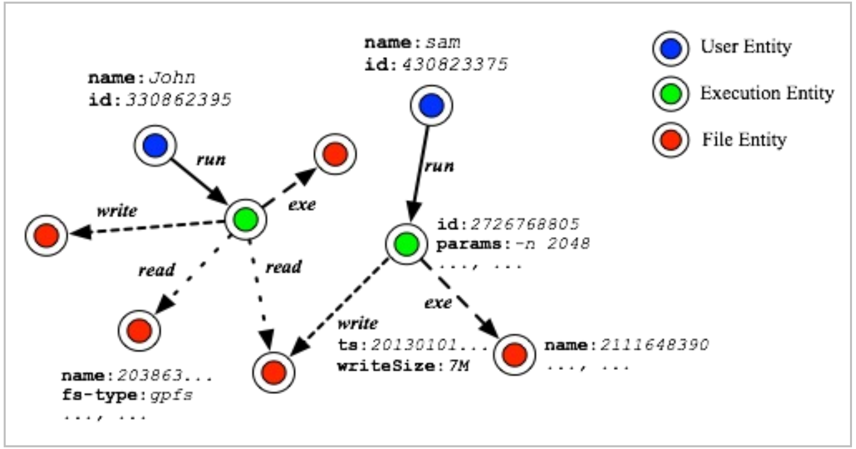
ASSESS

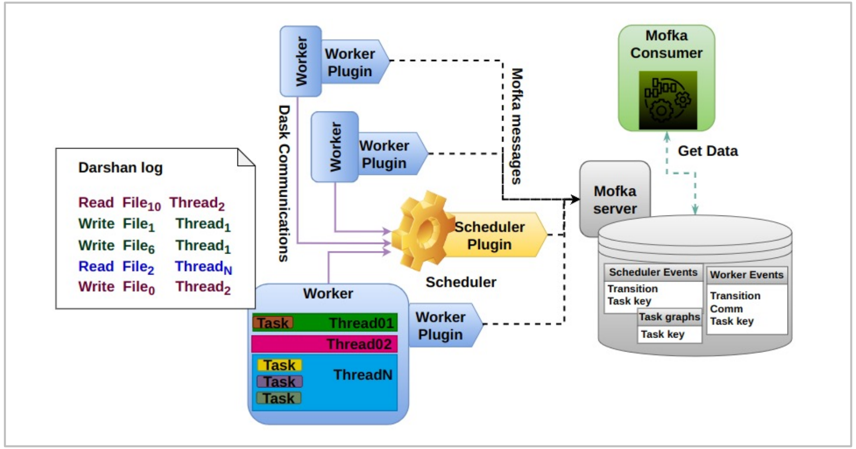
Pattern and anomaly detection integrated with simulation and experiment extract new insights

E.O. Pyzer-Knapp et al. Accelerating materials discovery using artificial intelligence, high performance computing and robotics. *npj Comput Mater* **8**, 84 (2022).

OBSERVATION OF COMPLEX WORKFLOWS

Recent work under Line Pouchard's RECUP project





Observations of data interactions are a natural part of capturing workflow behavior. Property graphs can provide insights into key relationships.

Dong Dai et al. "Using property graphs for rich metadata management in HPC systems." *PDSW*, 2014.

The event streaming model is one way to approach this. "Kafka style" with persistent data allows for both in situ analysis for decision-making plus deeper post hoc analysis as appropriate.

Amal Gueroudji et al. "Performance Characterization and Provenance of Distributed Task-based Workflows on HPC Platforms." *WORKS* 2024.

SCIENCE IS INCREASINGLY MULTI-MODAL

Domain Science

Domain science data, including literature, data from experiments and observations, and simulation output must be stored and made available to distributed agents and associated teams.

Provenance

Provenance information aids in understanding, trusting, and reproducing results.

Indices/Search

Search capabilities and associated indices are required to rapidly identify relevant data.

Resilience

Ensuring resilience of data and computation will place additional requirements on data management systems.

Performance

Performance telemetry is required to adapt resource use in order to achieve science, performance, and energy utilization goals.

DATA ACCESS BY AGENTS

Humans are no longer the primary "user" for many data management services

- Directories, files, and even many scientific data analysis approaches may need to be rethought

---

What's the right way to reference things?

What are agents allowed to do (to our data)? Override mechanisms?

Knowledge graphs, clearly defined data management objectives, schemas as tools?

It's not a mid-2020s talk without an AI generated image. Image from DALL-E, prompt *"Please draw a picture evoking the idea of agents accessing data of many modalities."*

CONCLUDING THOUGHTS

OBSERVATION AND SERVICES IN SCIENTIFIC COMPUTING

There are numerous opportunities for this community to contribute to the success of science teams in a world of increased federation, automation, and multi-modality.

Observation continues to be the vehicle through which we can:

- learn how our systems are behaving and being used,
- provide the data needed to apply runtime optimizations, and
- procure more effective future platforms.

New service capabilities and interfaces will help:

- apply AI technologies to complex scientific questions, and
- bridge between HPC/AI platforms, distributed experimental facilities, and science teams.

SPECIAL THANKS!