SCALABLE COMPUTING SOFTWARE LABORATORY

ILLINOIS INSTITUTE OF TECHNOLOGY

PhD Thesis

Department of Computer Science

Illinois Institute of Technology

Optimizing complex scientific workflows using a re-configurable heterogeneous-aware storage system for extreme scale computing

Hariharan Devarajan, PhD Candidate

hdevarajan@hawk.iit.edu

Advisor: Dr. Xian-He Sun

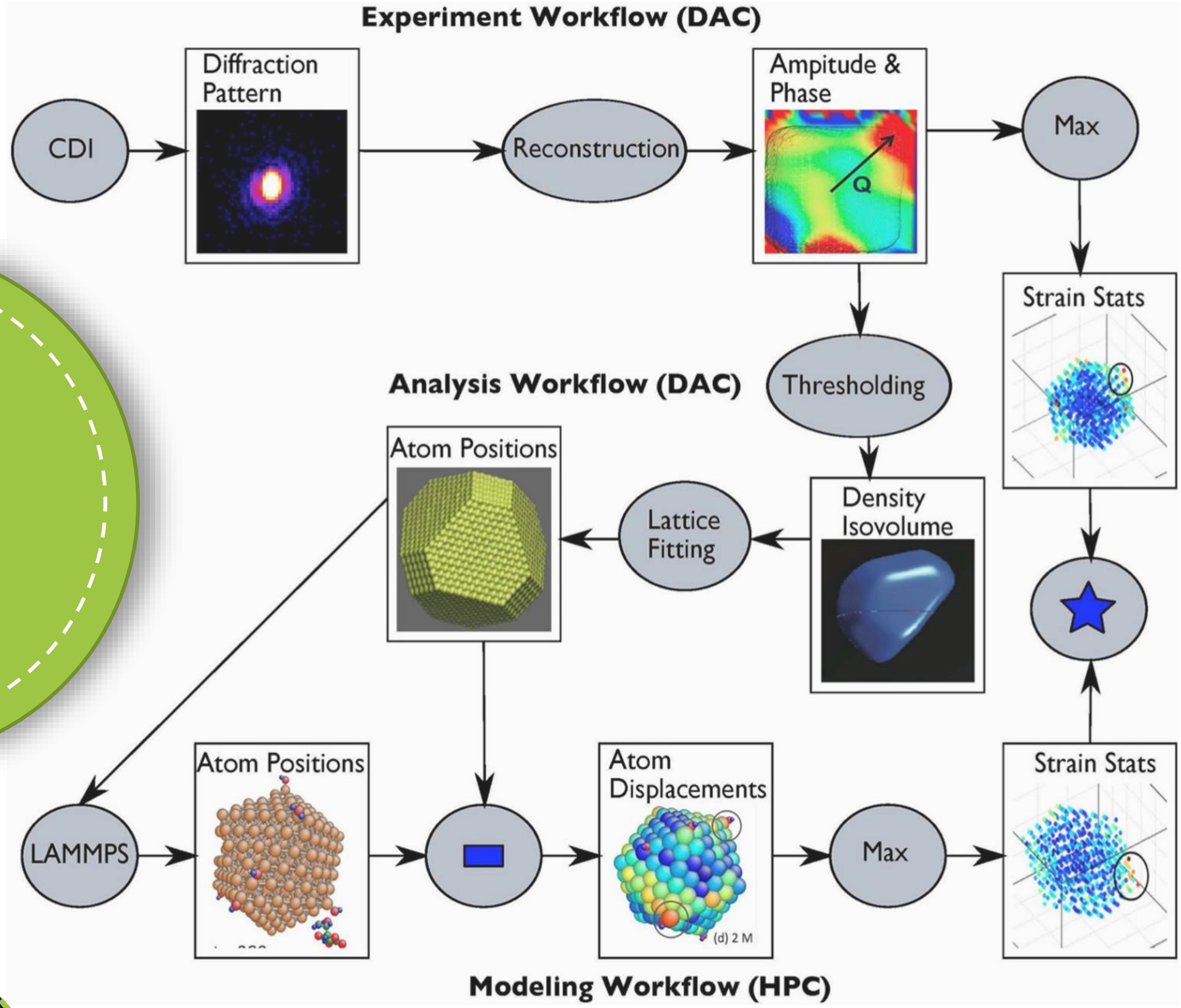
Premise of the work

Optimizing complex scientific workflows using a re-configurable heterogeneous-aware storage system for extreme scale computing

HPC Applications

- **Highly data-intensive**
  - multi-stage
  - E.g., three sub stages of simulation, analysis and modeling.
- **Data Dependent**
  - Many stages interchange data or compare results to reach to a convergence
- **Iterative**
  - The cycle of simulation, analysis and modeling is repeating for gaining higher resolution of data.
- Managed manually by application developers.

Complex Scientific Workflows



Source: The International Journal of High Performance Computing Applications 32, no. 1 (2018): 159-175.

Diverse Storage

- A variety of storage and memory hardware
  - Different characteristics
    - Sensitivity to Random accesses
    - Concurrency of operations
    - Device layouts
    - Power requirements
    - Performance requirements
  - Different Vendors
    - Optimizations
    - Device drivers
    - Interfaces

Heterogeneous storage devices & solutions

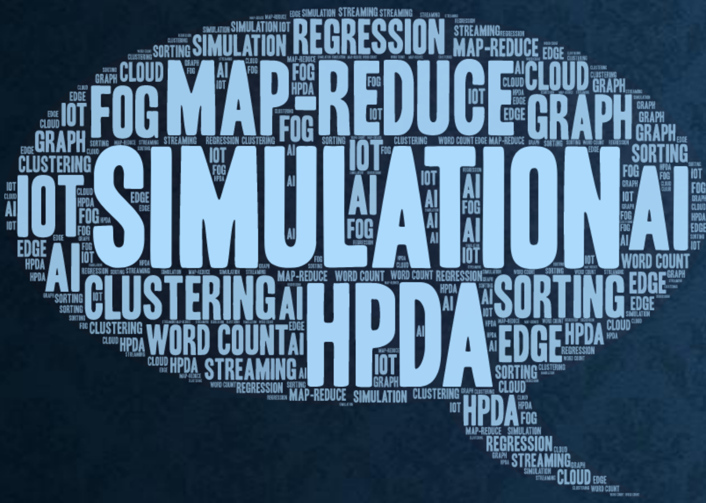
NVRAM:
- Single 5V Supply.
- Infinite EEPROM to RAM Recall.
- Latency 3µs

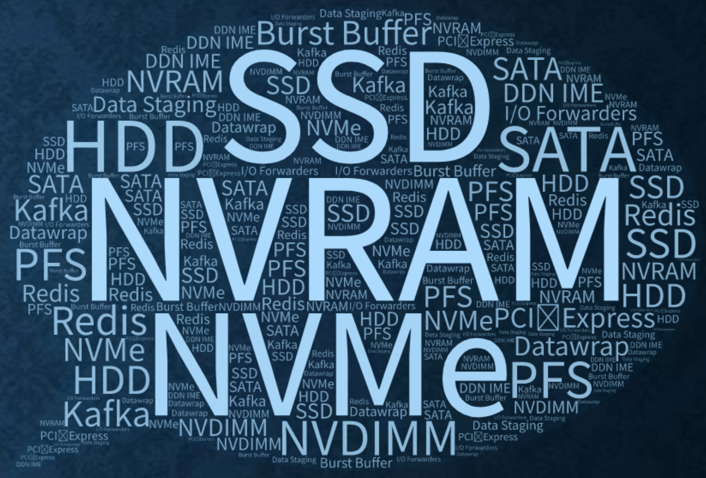
NVMe SSD:
- I/O Multipath.
- Multi-stream Writes.
- Latency: 12µs

SATA SSD:
- TLC flash memory.
- NAND flash memory ce
- Latency: 500µs

SATA HDD:
- Mass device storage.
- mechanical complexity makes it fragile.
- Latency: 7000µs

Current Situation





?

Diverse Applications

Diverse Storage Devices

Problem Statement

How can we support multiple diverse applications under a single platform that abstracts the complexity of efficiently utilizing heterogeneous storage technologies and maximizes I/O performance?

Profile I/O calls with low overhead. (1)

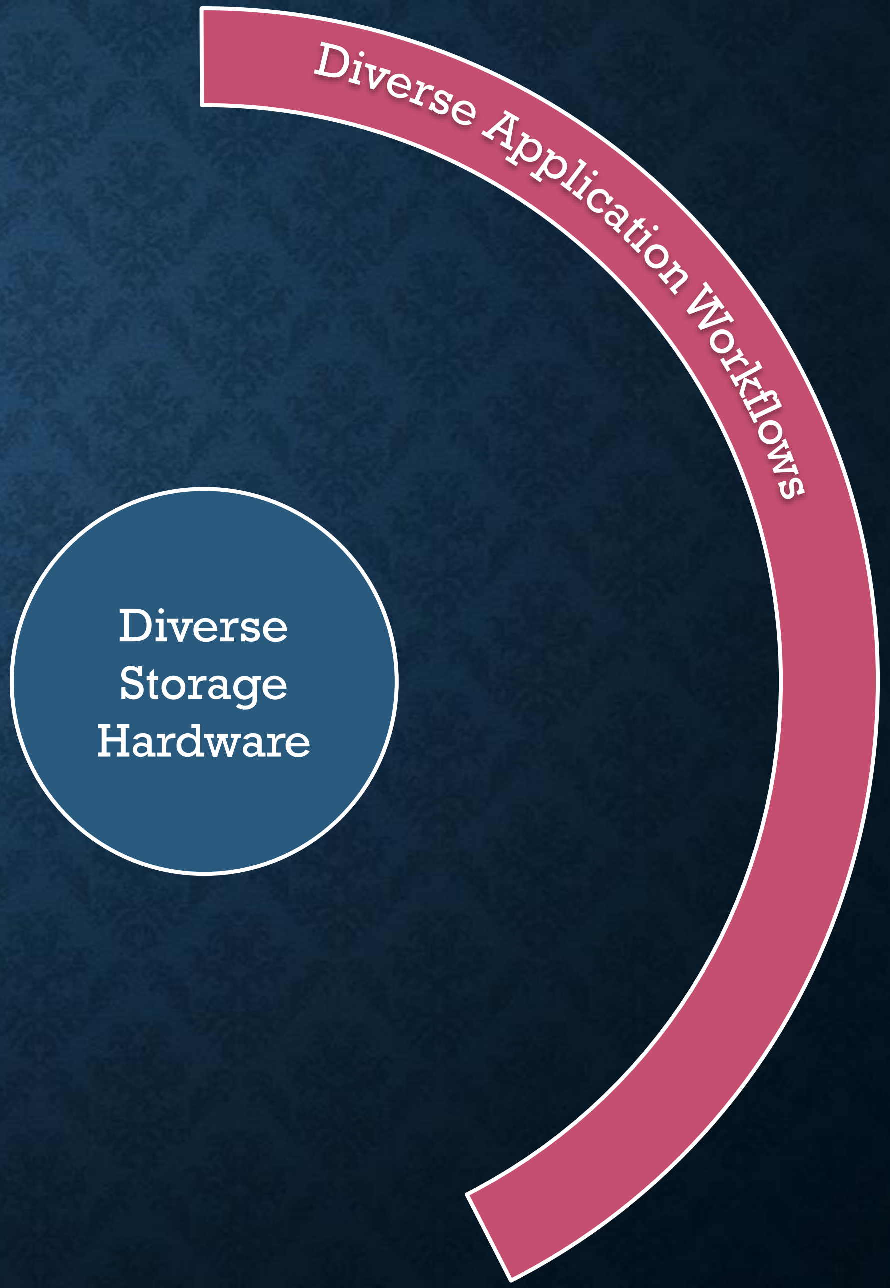
Automatically map I/O calls to app's characteristics. (2)

Map different app characteristics to storage configurations. (3)

Perform I/O access optimization on diverse storage. (4)

Adapt storage software to changing configurations. (5)

Unify diverse storage devices and software. (6)



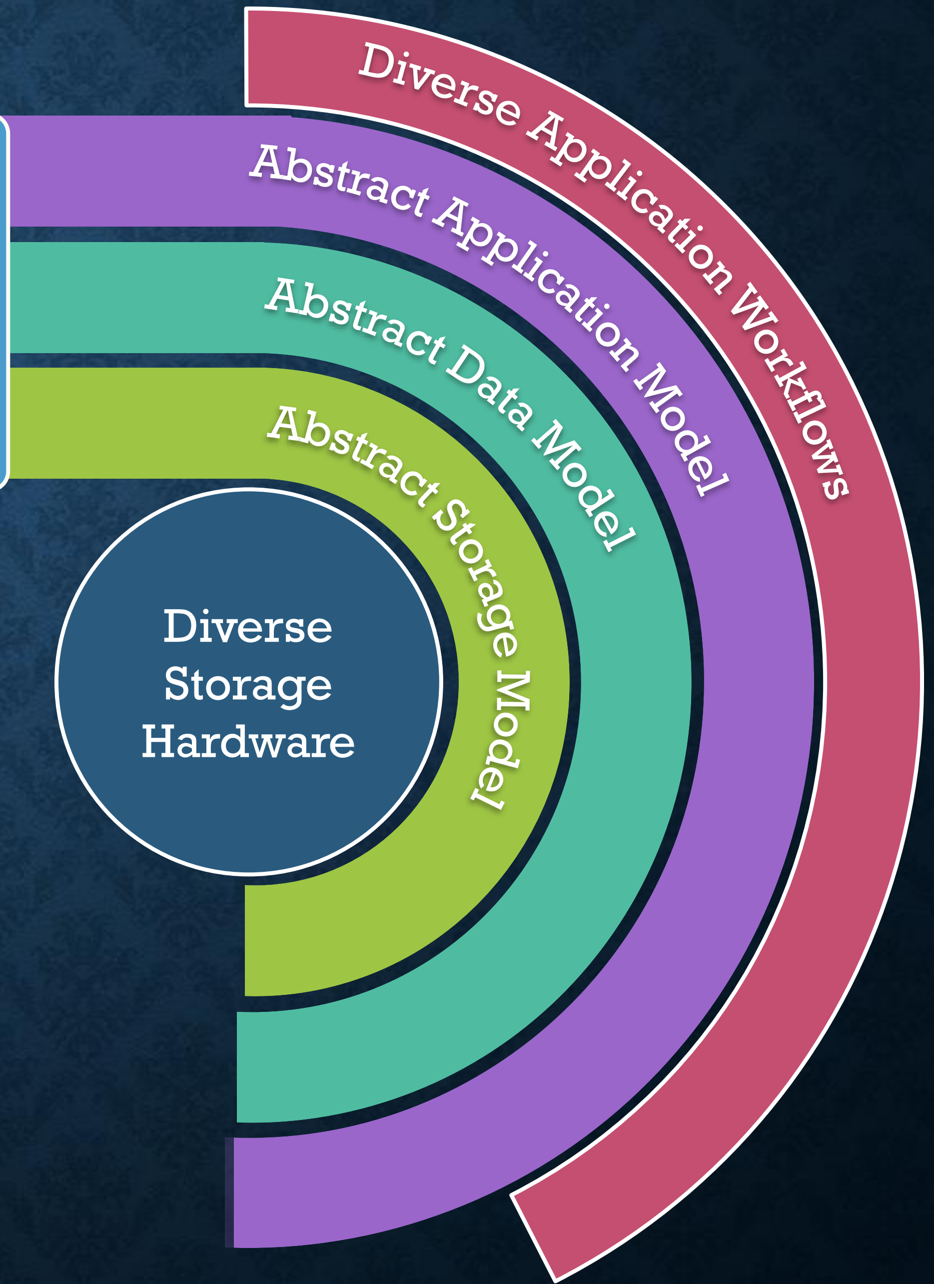
Identifying Challenges

Jal storage system

Perform I/O access optimization on diverse storage. (4)

Adapt storage software to changing configurations. (5)

Unify diverse storage devices and software. (6)



Our Proposal

Scope of this research

Application Model	• Source Code based Application Profiler	Vidya

Data Model	• Use-cases
	  • Compression
	  • Prefetching
	  • Replication
		Ares
		HCompress
		HFetch
		HReplica

Storage Model	• Hierarchical Shared log Store.	HCL ChronoLog

Outline

Jal

1 **Profiler**
Code-block level application profiling.

2 **Data Compression**
Multi-tiered data compression engine.

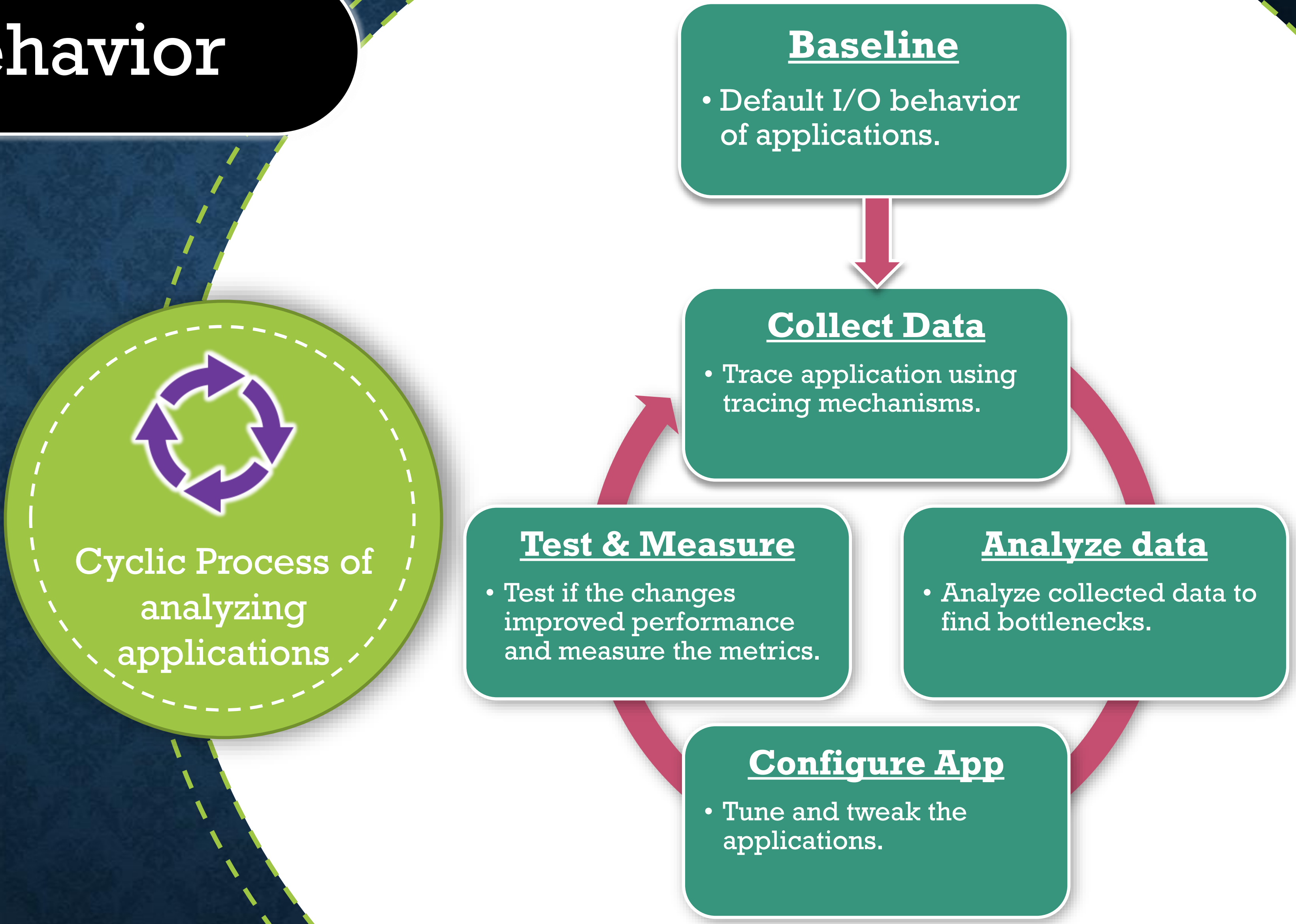
3 **Data Prefetching**
Multi-tiered data prefetching technology.

5 **ChronoLog**
A Shared log store.

6 **Conclusion**

Application's I/O behavior

- **Tracing Applications**
  - Observing what application is doing.
- **Analyze data**
  - Using data mining to extract patterns and co-relate back to application behavior.
- **Configure**
  - Trail and error on various configurations to tune application behavior.
- **Test and Measure**
  - Rerun application with new changes.



Current Methodology of Profiling

Offline Profiling

- Profiling is done before the actual execution of program.
- + High profiling accuracy.
- − High profiling cost

Online Profiling

- Profiling is done during the execution of program.
- + Low profiling cost
- − Low profiling accuracy

Observation

Behavior of an application stems from its source-code.

Predicting I/O behavior from source-code can enable us to understand cause of an I/O behavior.

Hypothesis

Vidya

Performing Code-Block I/O Characterization for Data Access Optimization

Publications

1) Hariharan Devarajan, Anthony Kougkas, Prajwal Challa, and Xian-He Sun, 2018, December. Vidya: Performing Code-Block I/O Characterization for Data Access Optimization. In 2018 IEEE 25th International Conference on High Performance Computing **(HiPC)** (pp. 255-264).
2) Hariharan Devarajan, Anthony Kougkas, Prajwal Challa, and Xian-He Sun, 2018, April. Poster: Performing Code-Block I/O Characterization for Data Access Optimization. In 2018 IEEE 6th Greater Chicago Area Systems Research Workshop **(GCASR).**

Approach

1 Profile

- Use montage as a case-study.
- Profile using existing tools for CPU, memory, & I/O.
- Correlate with code structures.

2 Classify

- Hypothesize several code-structures.
- Classify them to increase/decrease I/O intensity.

3 Model

- Collect several source-code
- Extract identifies code-structures
- Measure I/O intensity through profiling
- Train a ML model, **code-block I/O Characterization (CIOC)**, to predict I/O intensity.

4 Build

- Design an automated tool
  - Extracts features
  - Predicts I/O intensity
  - Performs code-optimizations

Building the ML model

- Collect source code from different domains (graph, scientific, AI, benchmarks)
- Extract features and build dataset
- code-block unit (function/class/branch/loop/line)
- 4200 records dataset

01 Collecting Data

Training model 02

Linear Regression Model

$$Y_m(v) = \beta_0 + \sum_{i=1}^{v} \beta_i * X_{im}$$

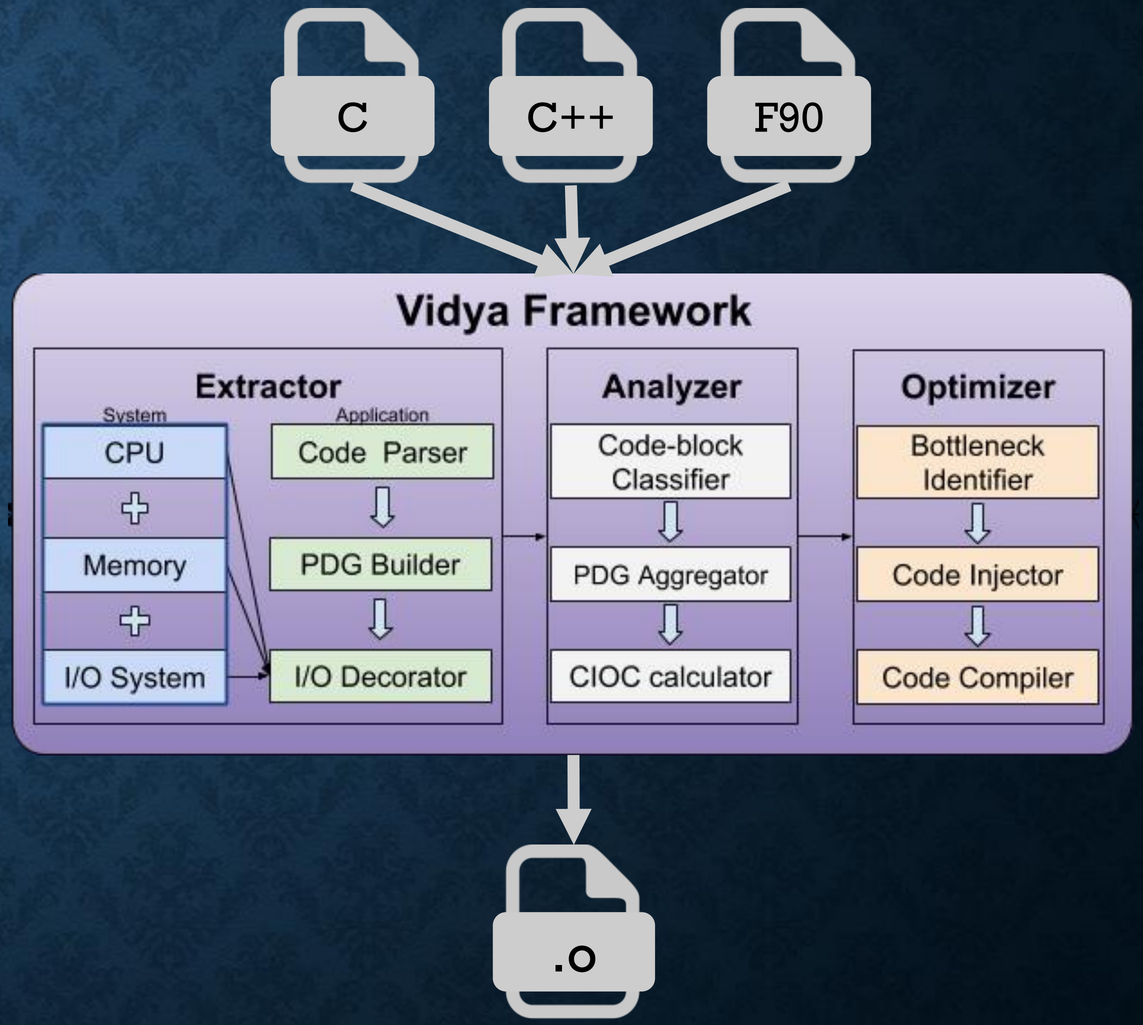
- Good model fit
  - $R^2 = 0.92$, f-statistics = 785
- Top two significant variables
  - Amount of I/O
  - Number of files opened

03 Validating Model

CIOC:
Code-block I/O characterization

Vidya: Design

- **Extractor**
  - Uses LLVM to parse the source code and build a Program Dependency Graph (PDG).
  - PDG is enhanced with I/O features on various pieces of code.
- **Analyzer**
  - Analyzes the PDG and extracts code features.
  - The aggregator combines code features to the root of the PDG and calculates the I/O intensity using CIOC.
- **Optimizer**
  - Identifies which code-feature can decrease I/O intensity.
  - Injects the changes and recompiles the code.



Evaluation

- Node Configuration
  - 128 GB RAM,
  - 10Gbit Ethernet, and
  - 200GB HDD
- Cluster Configuration
  - 32 client nodes
  - 8 storage nodes

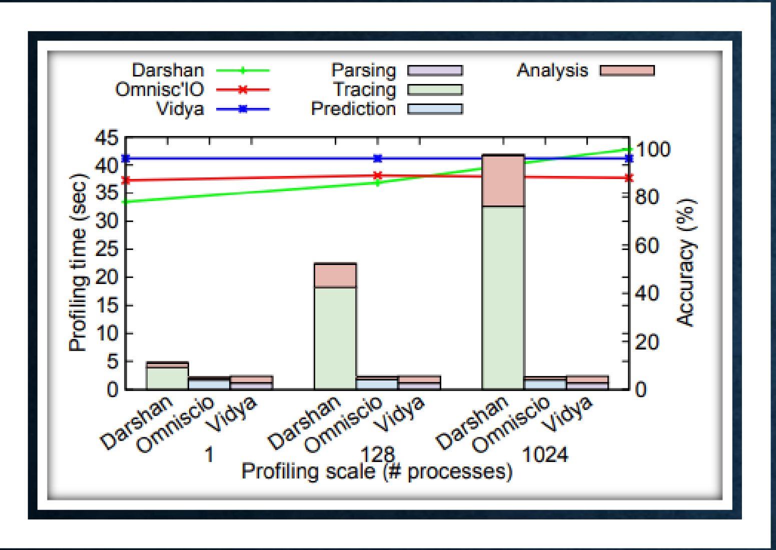
Testbed

Configuration

- Applications tested
  - Synthetic Benchmarks,
  - CM1,
  - WRF, and
  - Graph500's BFS and GMC
- Compared solutions
  - Darshan
  - Omnisc'IO

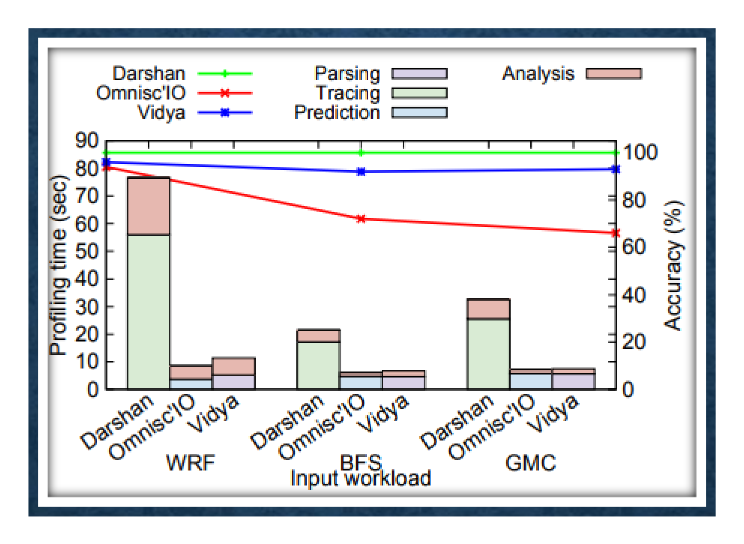
Profiling Performance

Profiling Scale



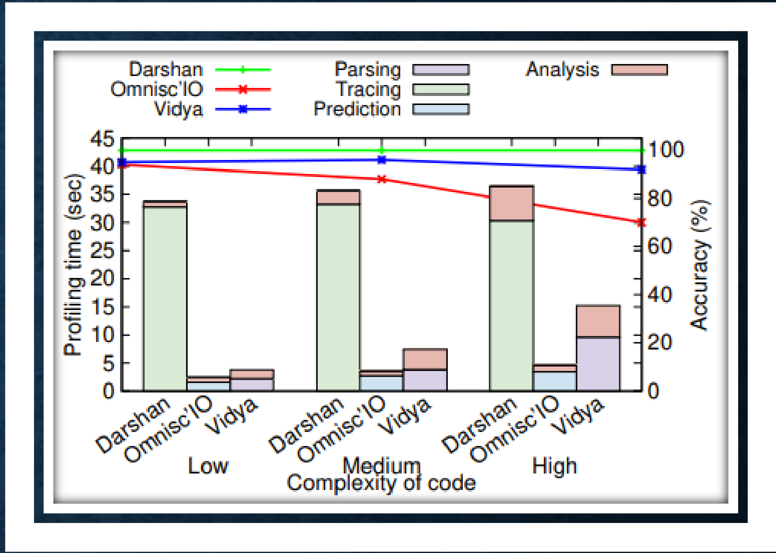
- Darshan
  - profiling cost increases as scale increases
  - On lower scales the profiling accuracy decreases
- Vidya and Omnisc'IO is unaffected.

Workload Irregularity



- Omnisc'IO's profiling accuracy decreases as irregularity increases.
- Vidya and Darshan is unaffected.

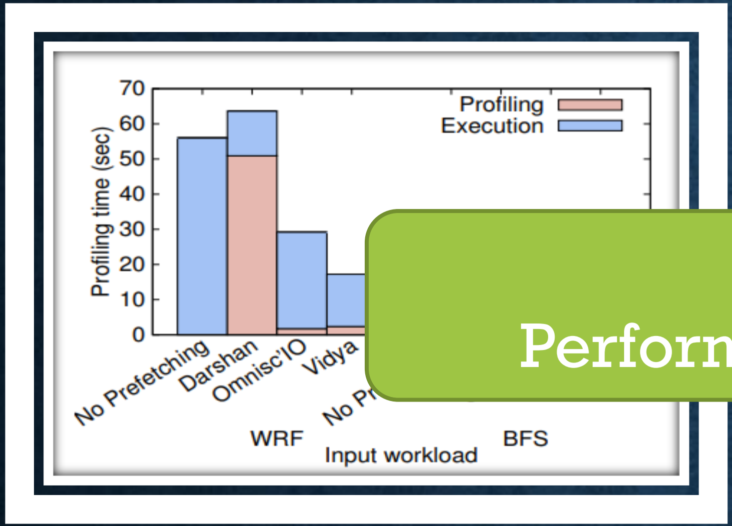
Complexity of Code



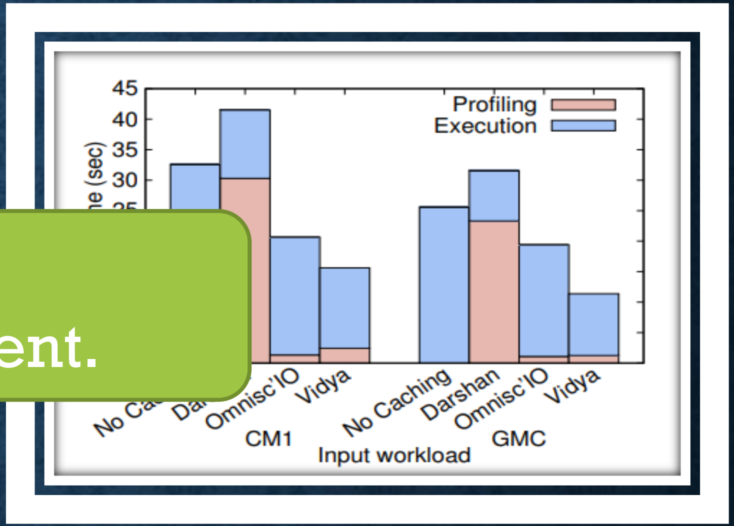
- Complexity: loops, functions, classes, and files
- Vidya
  - parsing time increases as complexity increases.
  - 3x faster than Darshan
  - 2x slower than Omnisc'IO

Optimization Benefits

Prefetching On/Off



Caching On/Off



2x - 3.7x
Performance improvement.

- Characteristics: Irregular workloads with simple code.

- Characteristics: repetitive with complex code structures.

- Overall observation:
  - Darshan has the highest accuracy and, hence, potentially be manually optimized.
  - Omnisc'IO has less cost but inaccurate.
  - Vidya bridges this gap with overall best result (profiling + execution time).

Summary

A list of all observations

01 Vidya proposes a trade-off between accuracy & cost of profiling.

02 Vidya proposes a methodology to calculate I/O intensity using source-code structures.

03 Vidya can reduce the cost of application profiling 9x while maintaining a high accuracy of 98%.

04 Vidya can be used to automatically optimize applications source-codes up to 3.7x.

Outline

Jal

1 **Profiler**
Code-block level application profiling.

2 **Data Compression**
Multi-tiered data compression engine.

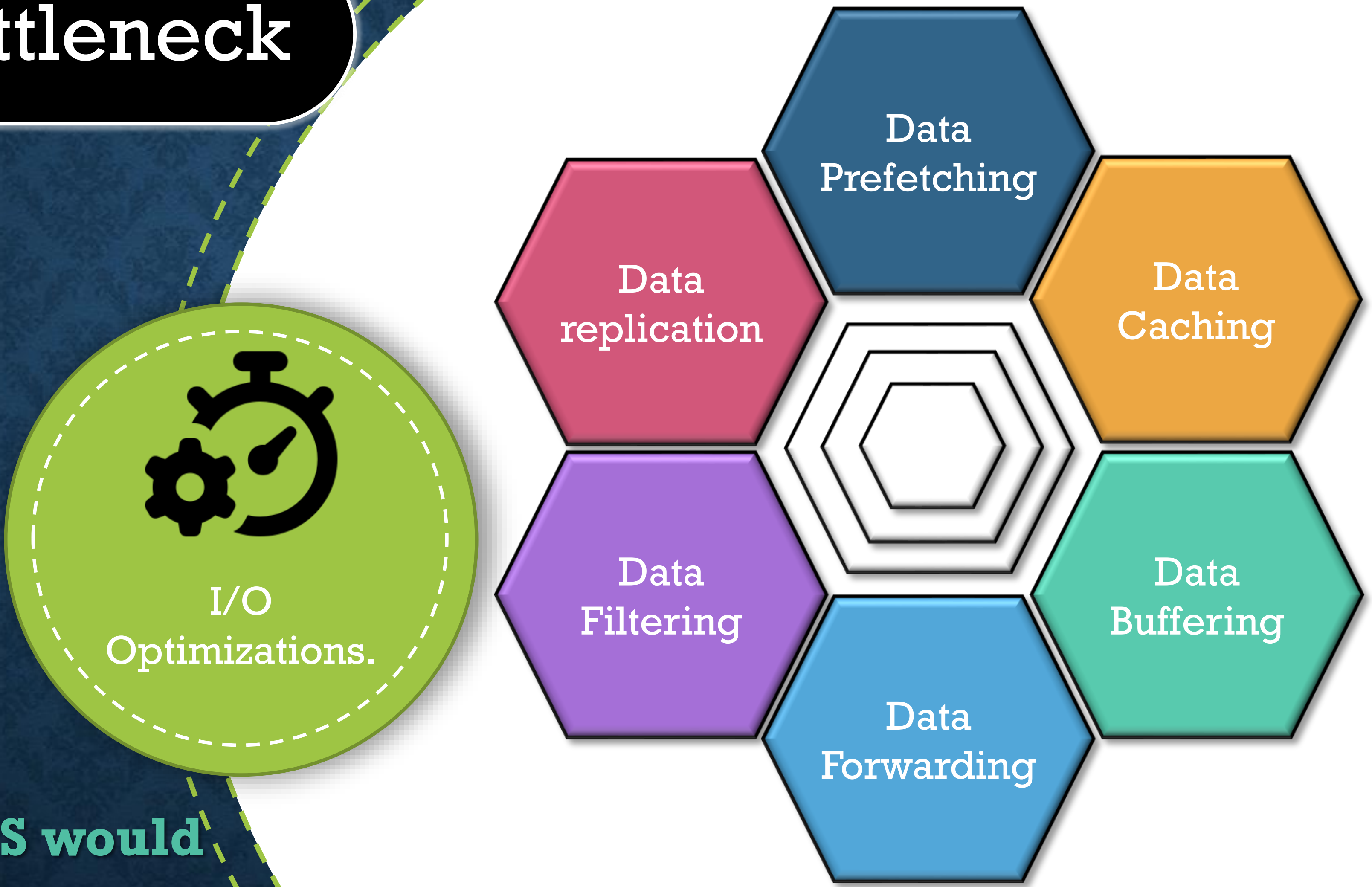
3 **Data Prefetching**
Multi-tiered data prefetching technology.

5 **ChronoLog**
A Shared log store.

6 **Conclusion**

Reduction of I/O bottleneck

- Several middleware solutions are proposed to reduce the I/O latency and increase application performance.
- In all approaches, the solutions utilize an Intermediate Temporary Scratch (ITS) space (e.g., Main Memory) to optimize I/O access.



Increasing the space of ITS would greatly enhance the effectiveness of these solutions.

Current approach: Increase ITS space.

Tiered Hardware

- Add new intermediate resources to increase layers.
- E.g., HBM, NVRAM, NVMe SSD, etc.
- Increases space availability.

Software

- Reduce I/O footprint through data reduction techniques.
- E.g., Data Compression.
- Reduces Data Footprint.

Observation

Benefit of compression comes from trading CPU cycles to reduce I/O cost.

The new hardware reduces this I/O cost.

A combination of these two approaches can compound the increase of available ITS for I/O optimizations.

Hypothesis

HCompress

Hierarchical & Intelligent Data Compression for Multi-Tiered Storage Environments

Publications

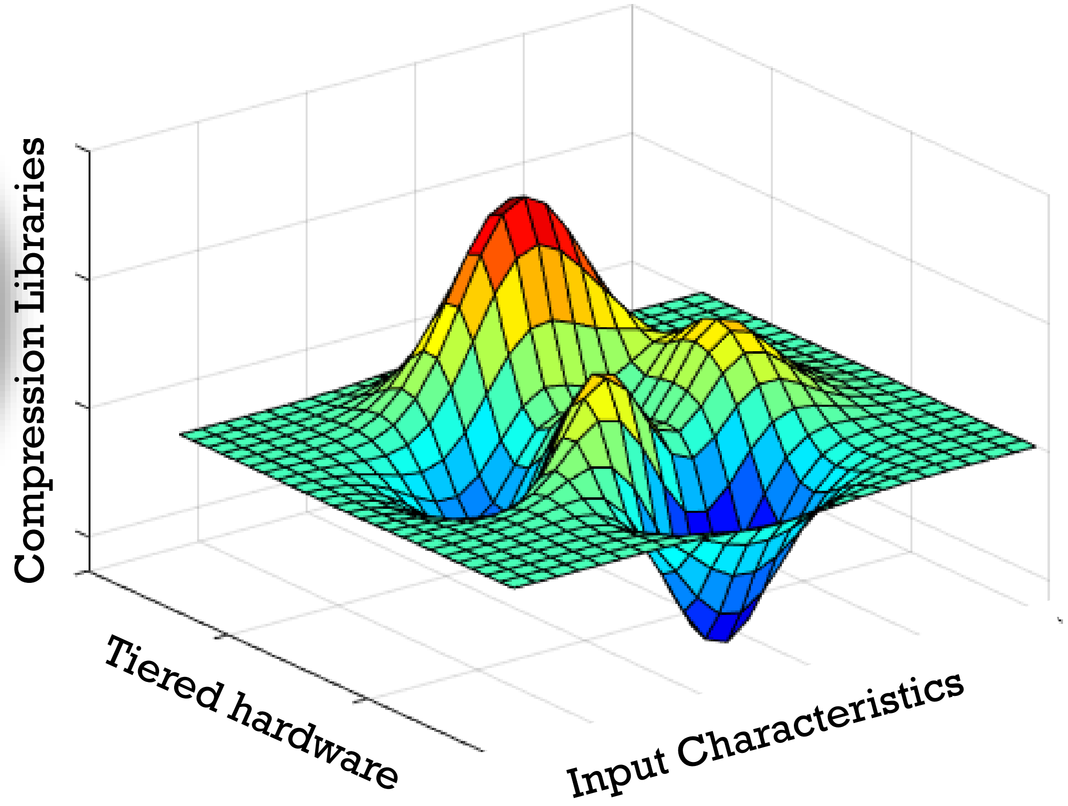
1) Hariharan Devarajan, Anthony Kougkas, and Xian-He Sun. "HCompress: Hierarchical Data Compression for Multi-Tiered Storage Environments" IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2020. **(to appear)**
2) Hariharan Devarajan, Anthony Kougkas, and Xian-He Sun. "Ares: An Intelligent, Adaptive, and Flexible Data Compression Framework." In 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 82-91. 2019.
3) Hariharan Devarajan, Anthony Kougkas, and Xian-He Sun. "An Intelligent, Adaptive, and Flexible Data Compression Framework. (Poster)" In 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), 2019.

Problem Formulation

- Match three dimensions
  - Application Characteristics
  - Compression Characteristics
  - Hierarchical Tier Characteristics
- We can formulate it as a minimization of total time for executing an I/O task
- The constraints required
  - # sub-problems should be small.
  - Data compression is useful.
  - Compressed data fits in a tier.

Multi-dimensional Optimization

Visual representation of 3D space.



HCompress Goals

1 Hierarchical

Utilize all storage hardware efficiently.

Match hardware speed of devices to ideal compression libraries.

2 Dynamic

Dynamically switch compression libraries.

Cost of reconfiguration of compression engine should be low.

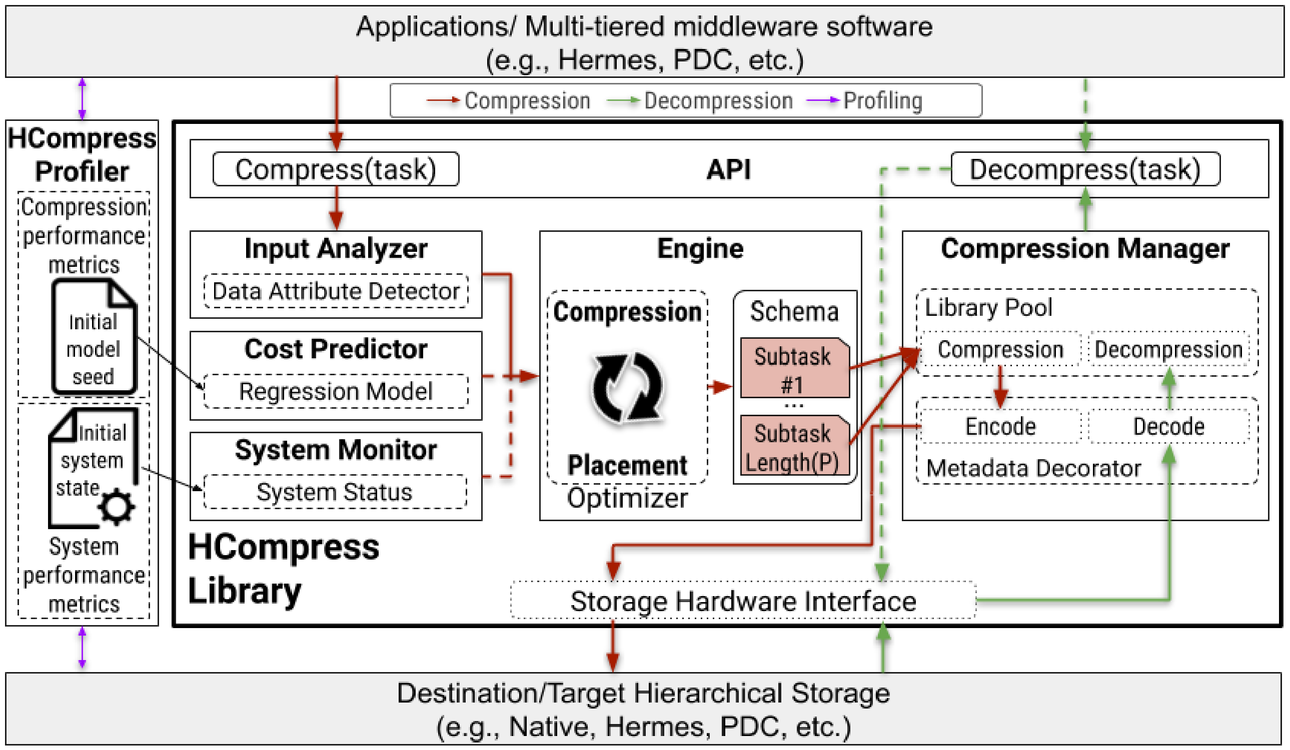
3 Flexible

Unify the interface to compression libraries.

Configure, add, and apply compression using a simplified interface.

HCompress Design

- **HCompress Profiler**
  - Runs a exhaustive benchmark to capture system and compression characteristics.
- **Compression Cost Predictor**
  - Uses linear regression model
  - Uses reinforcement learning to improve accuracy.
- **Engine**
  - Employs a dynamic programming (DP)
    - Data characteristics, Compression libraries, and Storage tiers
- **Compression Manager**
  - Manages library pool
  - Performs metadata encoding/decoding



Evaluation

- **Cluster Configuration**
  - 64 compute nodes
  - 4 shared burst buffer nodes
  - 24 storage nodes
- **Node Configurations**
  - compute node
    - 64GB RAM and 512GB NVMe
  - Burst Buffer node
    - 64GB RAM and 2x512GB SSD
  - Storage node
    - 64GB RAM and 2TB HDD

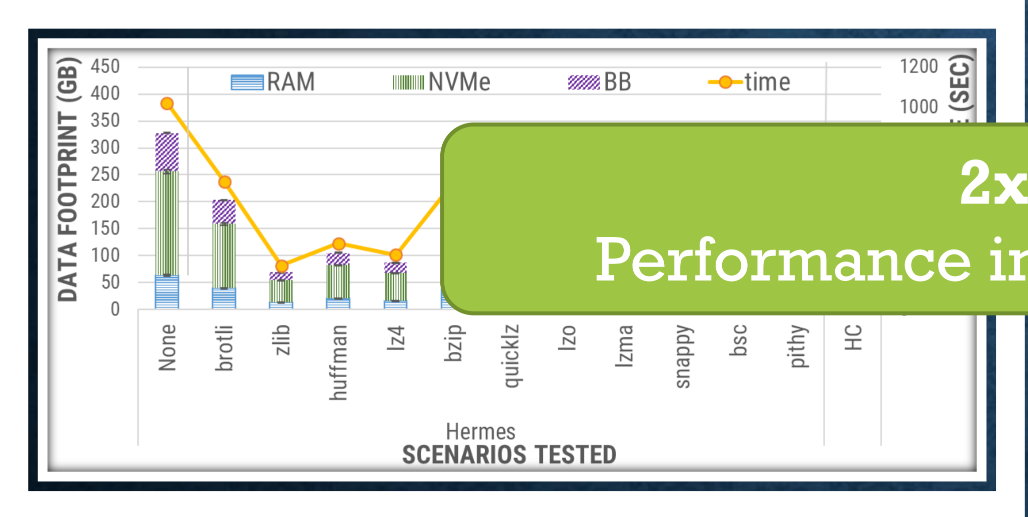
Testbed

Configuration

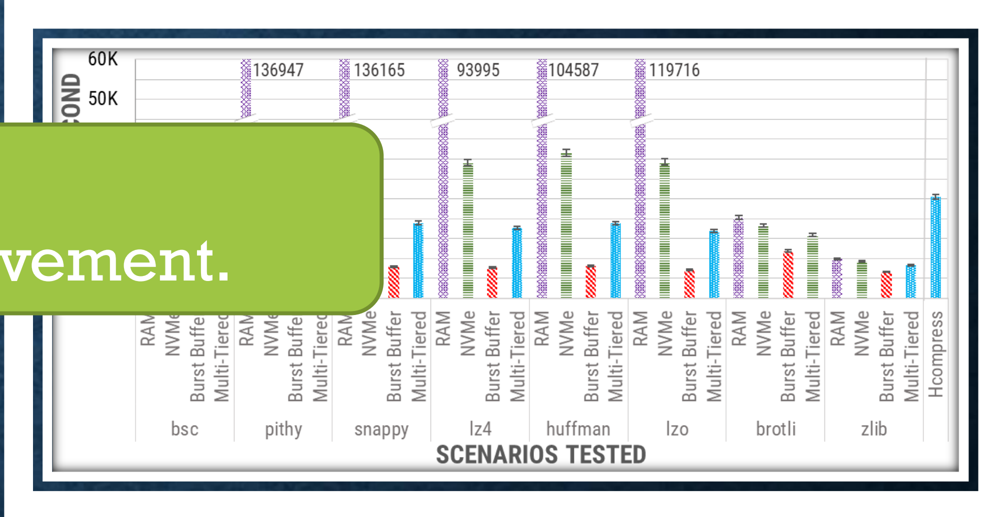
- Applications tested
  - Synthetic Benchmarks,
  - VPIC, and
  - BD-CATS
- Compared solutions
  - Baseline vanilla PFS
  - Single-tier with compression
  - Multi-tiered without compression

Impact of Data Compression & Tiered Storage

Compression on Tiered Storage



Tiered Storage on Compression



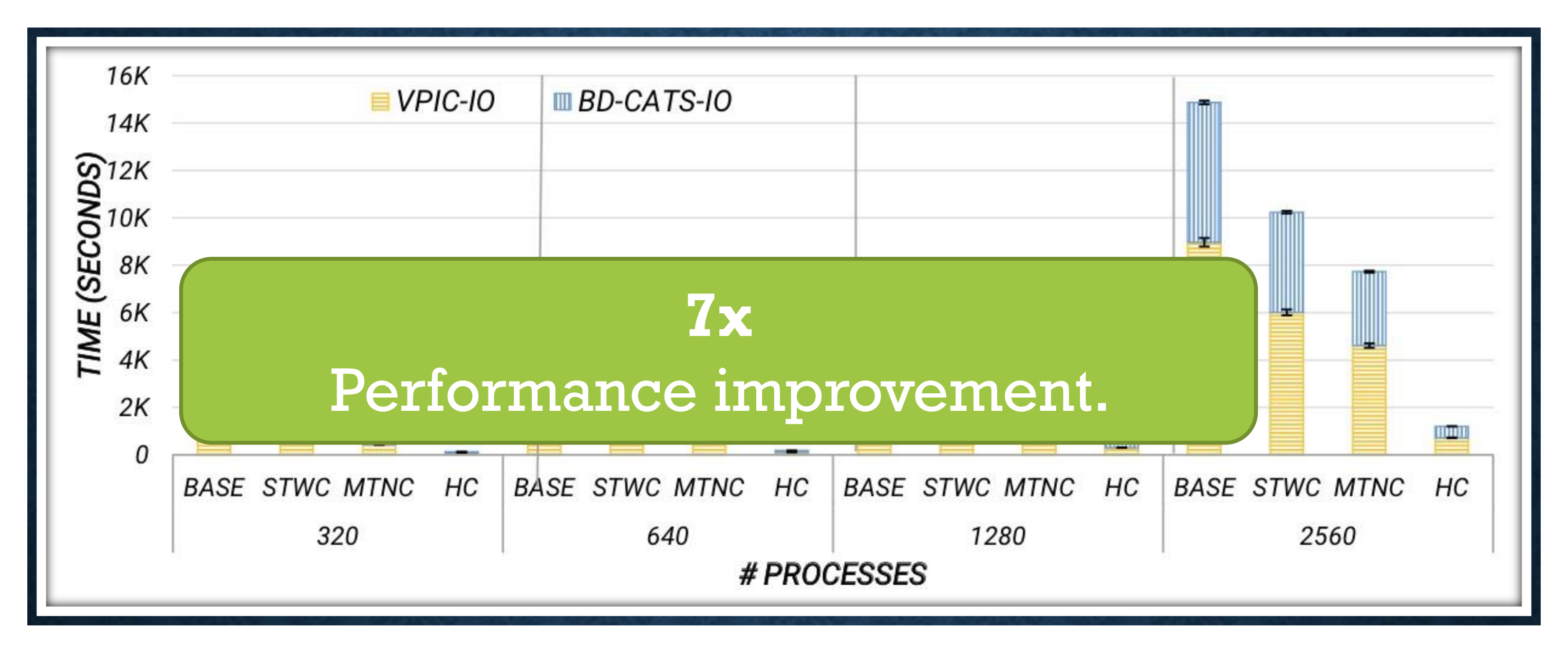
2x
Performance improvement.

Observations:

- Performing multi-tiered buffering with single compression doesn't maximize the benefit.
  - data placement is not aware of compression.
- HCompress achieves a benefit of 2x.

- Different tier effect differently for each compression
- HCompress balances trade-off dynamically and achieves the best multi-tiered throughput.

Scientific workflow



Observations:

- Optimizes both write and read performance significantly
  - Optimizes all three parameters: compression time, decompression time and compression ratio equally
  - Achieves a performance boost of 7x.

Summary

A list of all observations

01 HCompress showcased how data characteristics and system characteristics affect data compression.

02 HCompress proposes a hierarchical compression engine for multi-tiered storage environments

03 Quantified the benefit of utilizing hierarchical hardware and data compression cohesively.

04 HCompress can optimize scientific workflows up to 7x compared to competitive solutions.

Outline

Jal

1 **Profiler**
Code-block level application profiling.

2 **Data Compression**
Multi-tiered data compression engine.

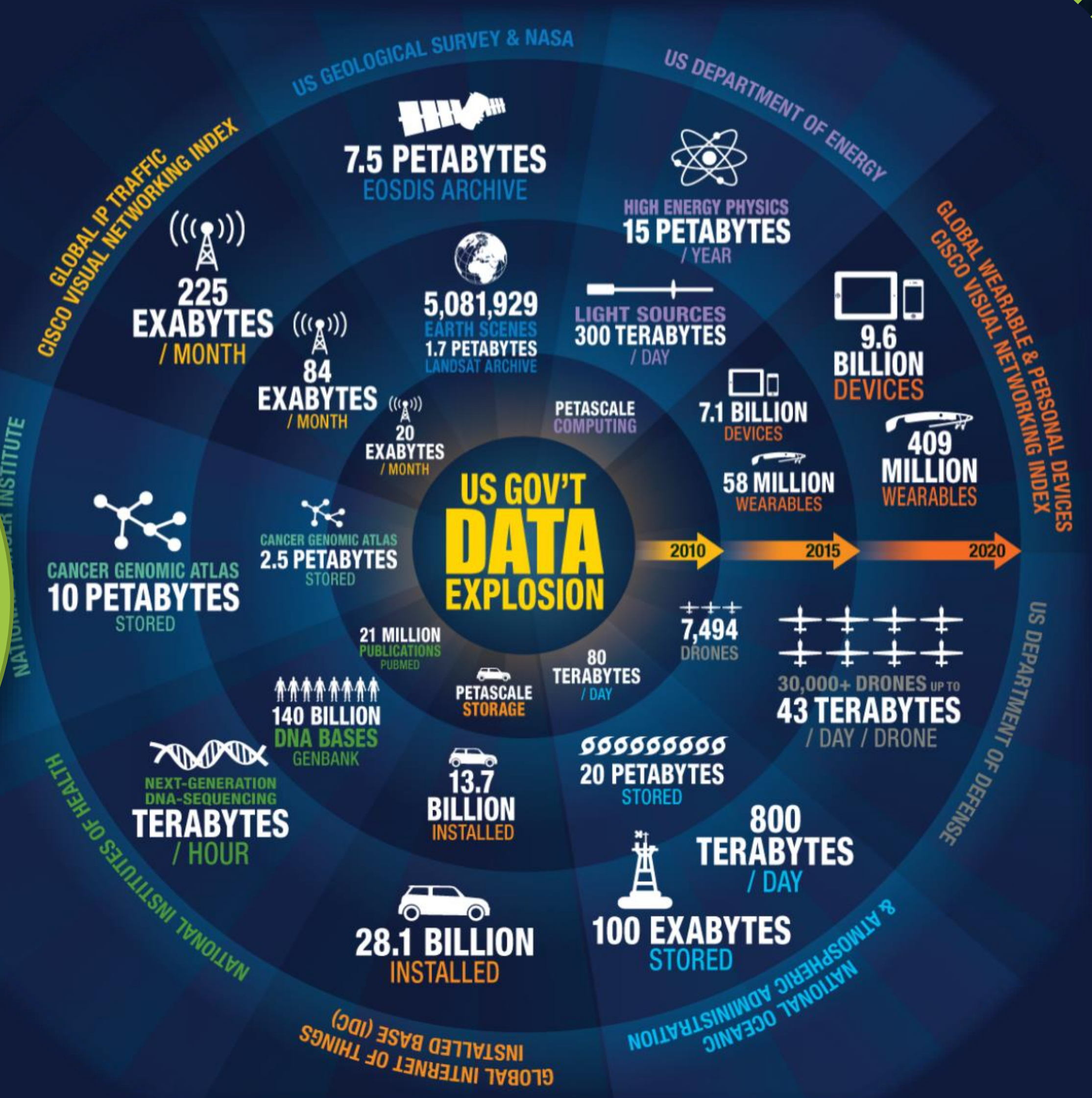
3 **Data Prefetching**
Multi-tiered data prefetching technology.

5 **ChronoLog**
A Shared log store.

6 **Conclusion**

Explosion of data

- Data is crucial to enable discovery.
- IDC reports predict that by 2025:
  - global data volume will grow to 163 ZB
  - 10x the data produced in 2016



Applications spend majority of their time on data retrieval.

Current approach: Optimize data access.

Tiered Hardware

- New intermediate resources with higher bandwidth.
- E.g., HBM, NVRAM, NVMe SSD, etc.
- Increases performance and reduces access latency.

Software

- Reduce I/O cost using several data access optimizations.
- E.g., Data prefetching, data staging, data replication, etc.
- Reduces access cost by preloading data to compute.

Observation

Both tiered storage and data prefetching optimize the same problem.

A combination of these two approaches can compound the benefit to improve data access.

Hypothesis

HFetch

Hierarchical Data Prefetching for Scientific Workflows in Multi-Tiered Storage Environments

Publications

1) Hariharan Devarajan, Anthony Kougkas, and Xian-He Sun. "HFetch: Hierarchical Data Prefetching in Multi-Tiered Storage Environments" IEEE International Parallel and Distributed Processing Symposium (IPDPS'20), 2020. **(to appear)**
2) Hariharan Devarajan, Anthony Kougkas, and Xian-He Sun. "HFetch: Hierarchical Data Prefetching in Multi-Tiered Storage Environments (Poster)" Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'19), 2019.

HFetch Goals

1
Server-Push

Lightweight and asynchronous data push.

Server pushes appropriate data to the app in place of it pulling.

2
Data Centric

Utilize how data is accessed in a workflow.

scheme looks at how data is accessed instead of apps accessing it.

3
Hierarchical

Unify the diverse hardware tiers.

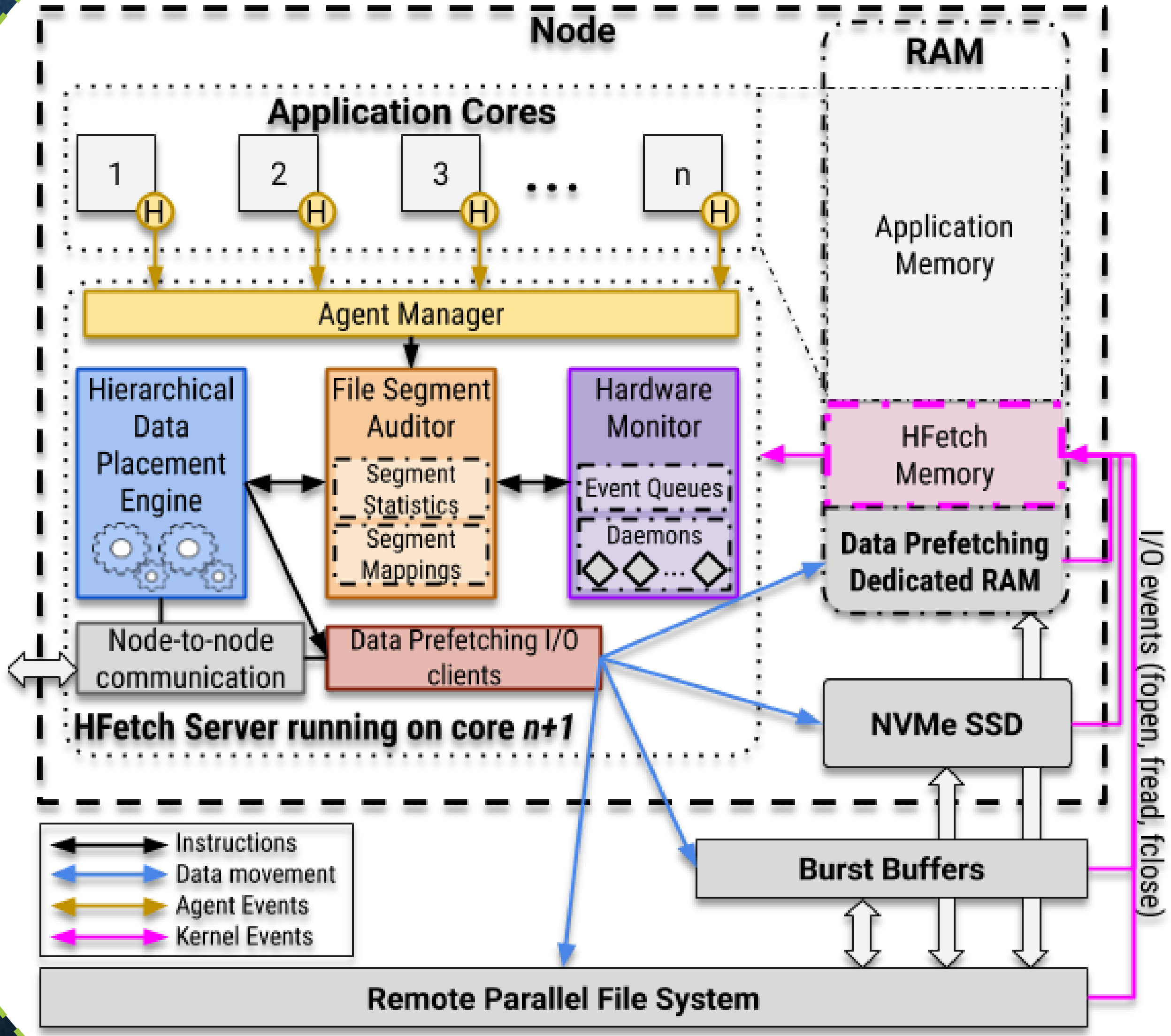
The engine matches data hotness to the spectrum of device characteristics.

HFetch Design

- **Server-Push**
  - Event are captured through kernel's inotify utility
  - Prefetched data is push to the hierarchy
- **Data Centric**
  - Score Incorporates
    - recency, frequency, and sequencing

$$Score_s = \sum_{i=1}^{k} \left(\frac{1}{p}\right)^{\frac{1}{n}*(t-t_i)}$$

- **Hierarchical Placement**
  - The engine calculates placement of prefetch data based on multi-tiered storage and data characteristics.



Example

Client space					Kernel space	HFetch Server space					
Time	Applications #1	Applications #2	HFetch Agents Agent#1	HFetch Agents Agent#2	inotify_handle_event (push to event queue)	Hardware Monitor	Auditor: Update Segment Statistics: Frequency	Auditor: Update Segment Statistics: Recency	Auditor: Update Segment Statistics: Sequence	Auditor: Calculate Segment Score	Data Placement Engine Tiers: T1<T2<T3<T4
t0	fopen(f1, READ)	-	start_epoch(f1)	-		inotify_add_watch(f1)	[0,0,0,0]	[0,0,0,0]	null	[0.0,0.0,0.0,0.0]	[T4,T4,T4,T4]
t1	fopen(f2, WRITE)	-	IGNORE	-							
	-	fopen(f1, READ)	-	start_epoch(f1)		IGNORE					
t2	fread(f1,0,1)	-	-	-	f1,offset:0,size:1,t2	collect_event()	[+1,0,0,0]	[+t2,0,0,0]	prev->s0	[1.0,0.0,0.0,0.0]	[T1,T4,T4,T4]
t3	fread(f1,1,1)	fread(f1,0,1)	-	-	[{f1,offset:1,size:1,t3}, {f1,offset:0,size:1,t3}]	collect_event()	[+1,+1,0,0]	[+t3,+t3,0,0]	prev->[s0,s1]	[1.5,1.0,0.0,0.0]	[T1,T2,T4,T4]
t4	fread(f1,0,1)	fread(f1,1,2)	-	-	[{f1,offset:2,size:1,t4}, {f1,offset:1,size:2,t4}]	collect_event()	[+1,+1,+1,0]	[+t4,+t4,+t4,0]	prev->[s0,s1,s2]	[1.5,1.5,1.0,0.0]	[T1,T2,T2,T4]
t5	fread(f1,0,1)	-	-	-	f1,offset:0,size:1,t5	collect_event()	[+1,0,0,0]	[+t5,0,0,0]	prev->s0	[1.2,0.5,0.3,0.0]	[T1,T2,T3,T4]
t6	fclose(f1)	-	end_epoch(f1)	-		IGNORE					
t7	fclose(f2)	fclose(f1)	IGNORE	end_epoch(f1)		inotify_rm_watch(f1)					

1. Specific Client I/O interception of open/close
2. Monitoring through VFS layer
3. Collect event through Hardware Monitor.
   1. Each layer has a different daemon
4. Update Auditor
   1. Calculate scores
   2. Rearranges scores in descending order
5. Run DPE
6. Perform I/O on different layers.

Evaluation

- **Cluster Configuration**
  - 64 compute nodes
  - 4 shared burst buffer nodes
  - 24 storage nodes
- **Node Configurations**
  - compute node
    - 64GB RAM and 512GB NVMe
  - Burst Buffer node
    - 64GB RAM and 2x512GB SSD
  - Storage node
    - 64GB RAM and 2TB HDD

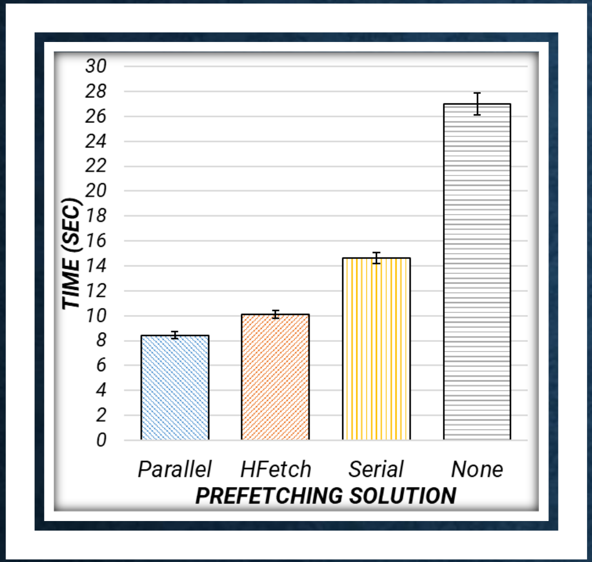
Testbed

Configuration

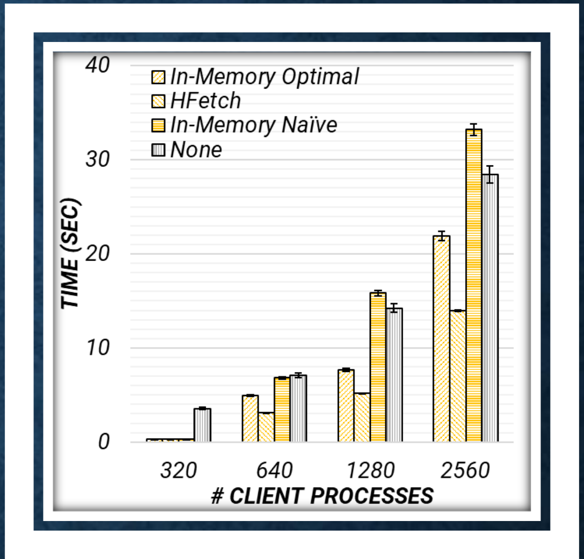
- Applications tested
  - Synthetic Benchmarks,
  - Montage, and
  - WRF
- Compared solutions
  - Stacker: ML-based online prefetching
  - KnowAc: offline prefetching

Benefit of Hierarchical Prefetching

Lower-RAM footprint



Extending Prefetching cache.



Observations:

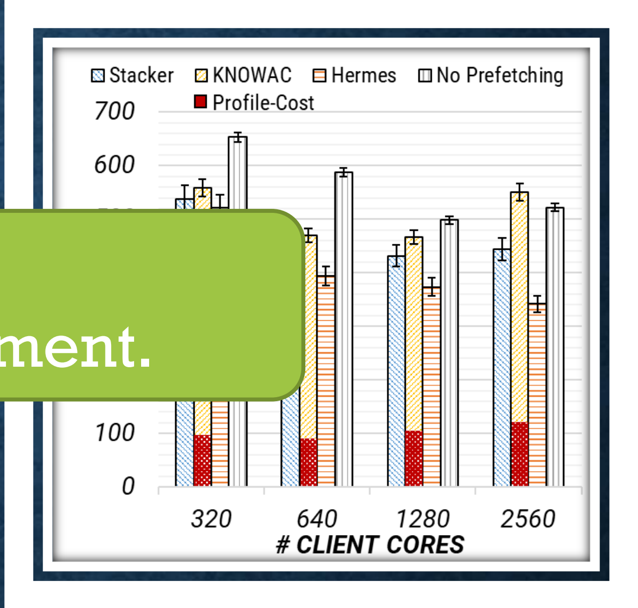
- A perfect parallel prefetching has 89% hit ratio.
- Most common serial prefetching cannot overlap the data perfectly and has more misses.
- HFetch uses ⅛ of ram and is 17% slower.

- Adding more layers reduces the cost of miss penalty
  - Additional cache space on lower tiers
  - Devices slower than RAM but faster than PFS.
- 35% to 50% faster.

Scientific Workflows

Montage

WRF



20-40%
Performance improvement.

Observations:

- Offline Profiler is accurate but has an initial cost through profiling.
- Stacker doesn't have that cost, but application-level prefetching hurts due to cache evictions and pollution.
- HFetch optimized this using a global data-centric score which helps the overall workflow.
- HFetch boosts read performance by **20-40%.**

Summary

A list of all observations

01 HFetch introduces a data-centric hierarchical prefetching methodology.

02 HFetch proposes a novel data centric scoring mechanism to measure the hotness of data.

03 Quantified the benefit of utilizing hierarchical hardware and data prefetching cohesively.

04 HFetch can optimize scientific workflows up to 35% compared to competitive solutions.

Outline

Jal

1 **Profiler**
Code-block level application profiling.

2 **Data Compression**
Multi-tiered data compression engine.

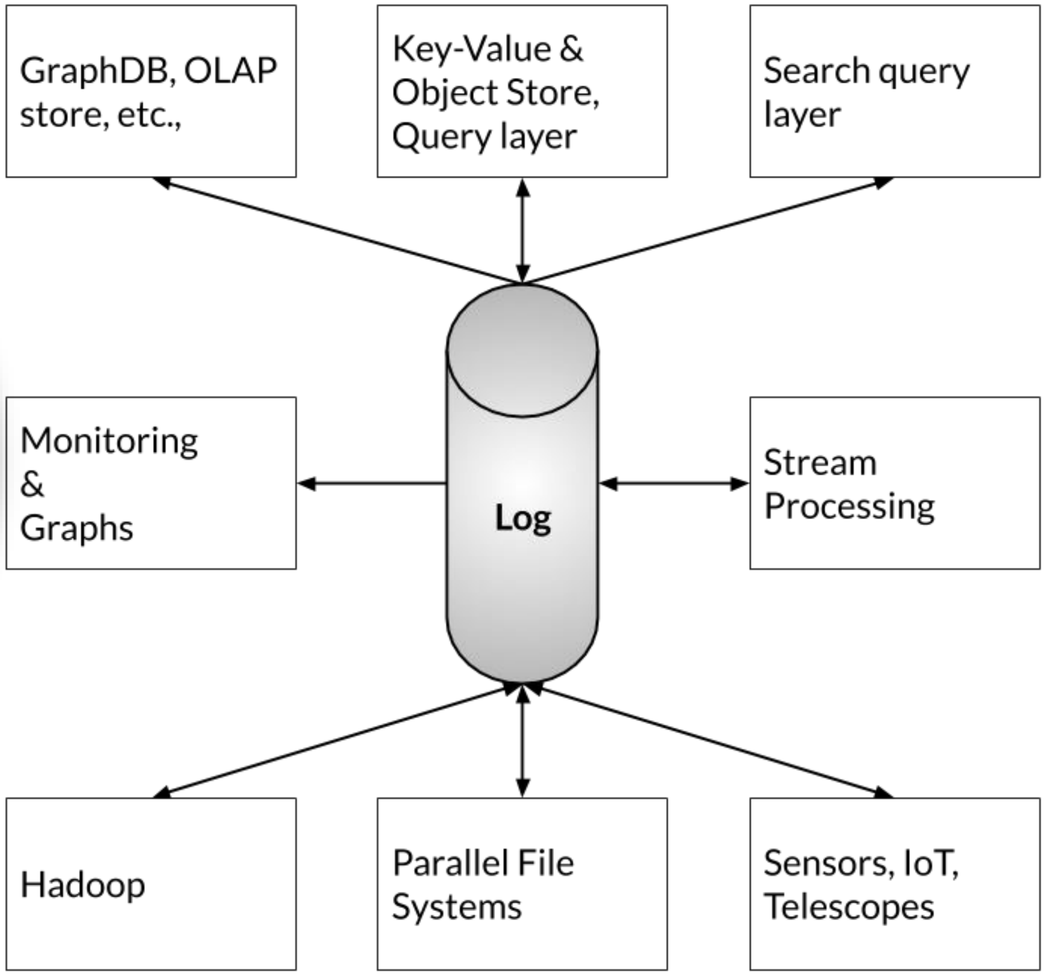
3 **Data Prefetching**
Multi-tiered data prefetching technology.

5 **ChronoLog**
A Shared log store.

6 **Conclusion**

Shared Log as storage model

- Storage is cheap and hence maintain what happened and when instead of mutation of data.
  - Inherent versioning semantics
- Enables high performance with append only semantics.
  - Deletes are through invalidations and background compactions of log.
- Enable decoupled consumer producer semantics.
- Achieves tunable consistency semantics.



A Shared log is an ideal backbone for any storage requirement.

Observation

Shared log is a good data abstractions for many storage systems.

A hierarchical storage and time-based data ordering to build an efficient shared log store

Hypothesis

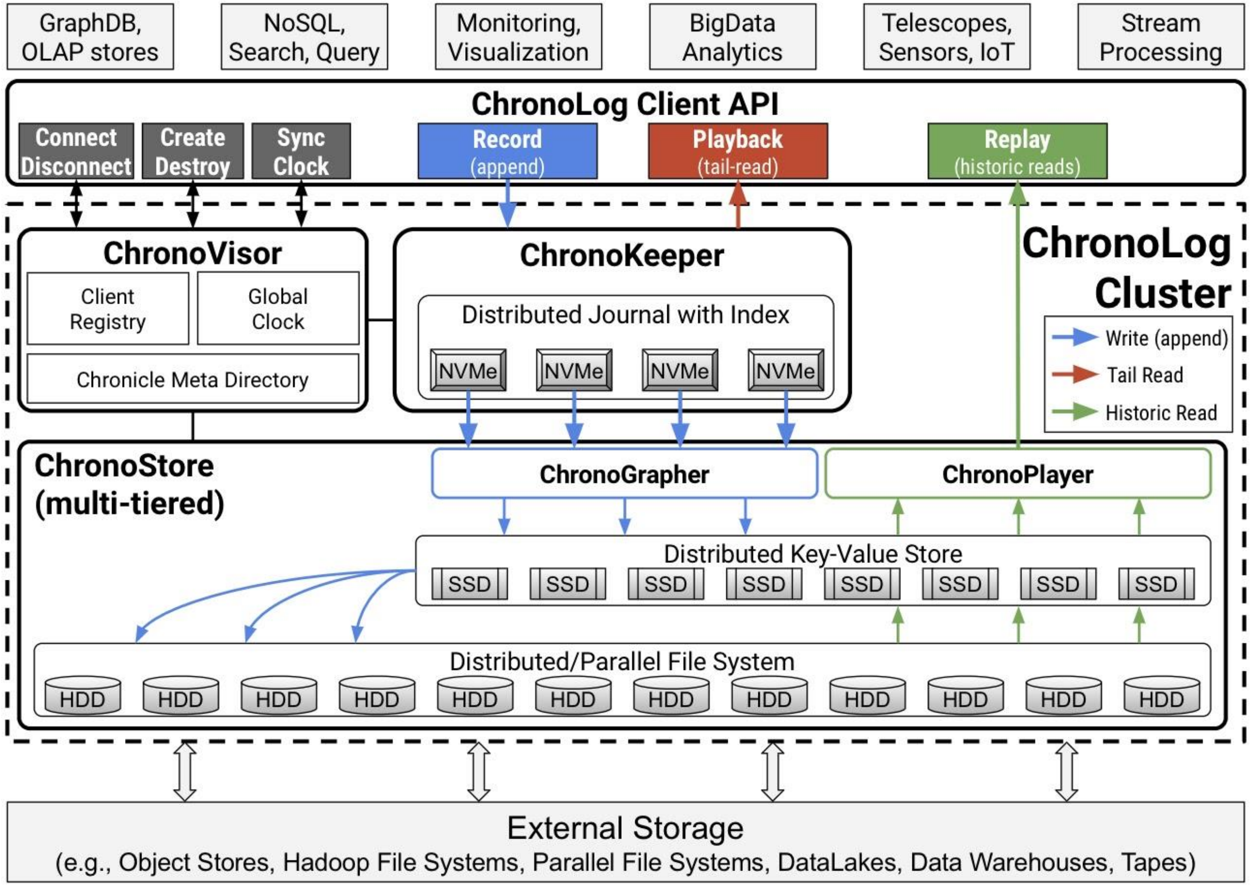
A Distributed Shared Tiered Log Store with Time-based Data Ordering

Publications

1) Anthony Kougkas, Hariharan Devarajan, Keith Bateman, Jaime Cernuda, Neeraj Rajesh and Xian-He Sun. ChronoLog: A Distributed Shared Tiered Log Store with Time-based Data Ordering" Proceedings of the 36th International Conference on Massive Storage Systems and Technology (MSST 2020). (to appear)

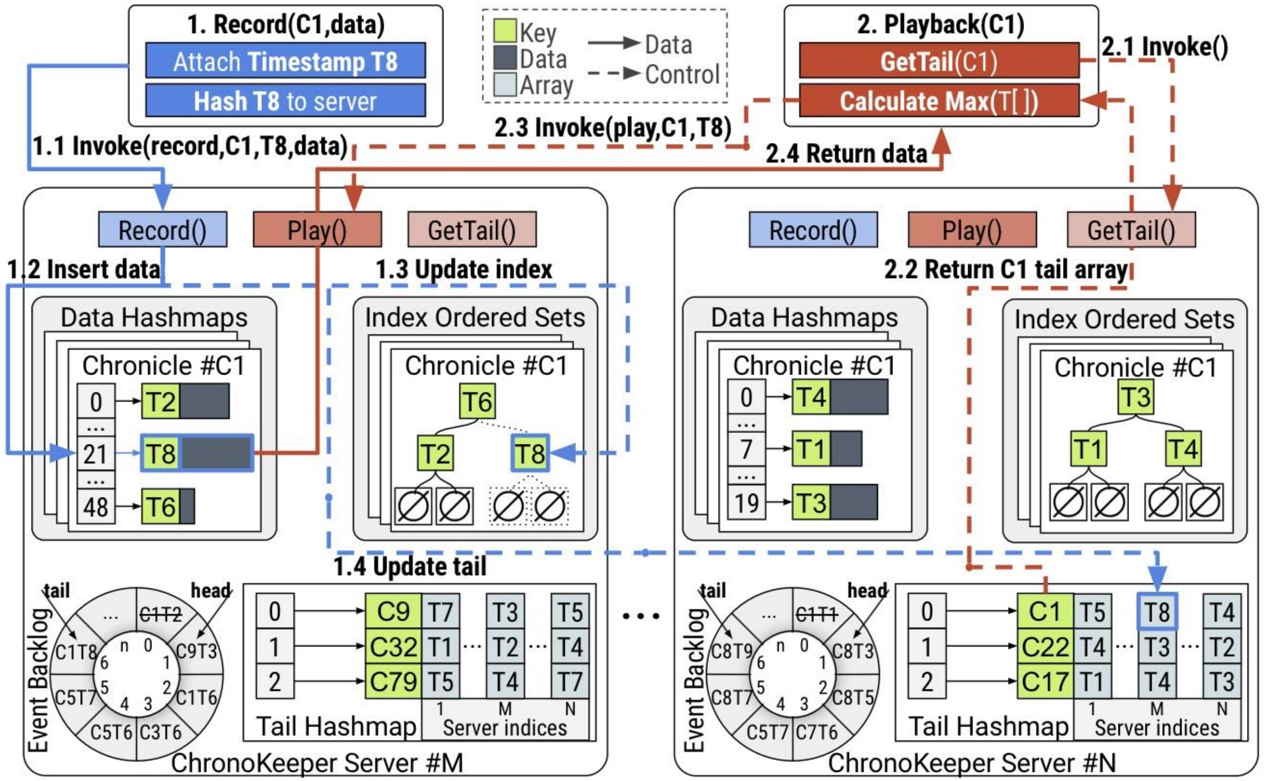
ChronoLog: High Level Design

- **Objectives**
  - Log distribution
    - Parallel 3D data distributions
  - Log ordering
    - Complete ordering with indexing
  - Log access
    - Concurrent data access based on I/O size.
  - Log scaling
    - Capacity and auto-tiering
  - Log storage
    - Tunable parallel I/O



ChronoKeeper

- Distributed Journal
  - Fast Data Ingestion
  - Fast Tail Operation
    - Lock-free tail updates
  - Uniform Data Distribution
    - Through distributed Hash Map
  - Time Data Ordering
    - Through Partitioned Ordered Map
  - Caching of Latest Events
    - Using backlogs



ChronoStore

Stream Paradigm

- Enables Explicit Parallelism based on Operation Size (Not Clients)
- Growing and shrinking of resources to enable efficient resource utilization

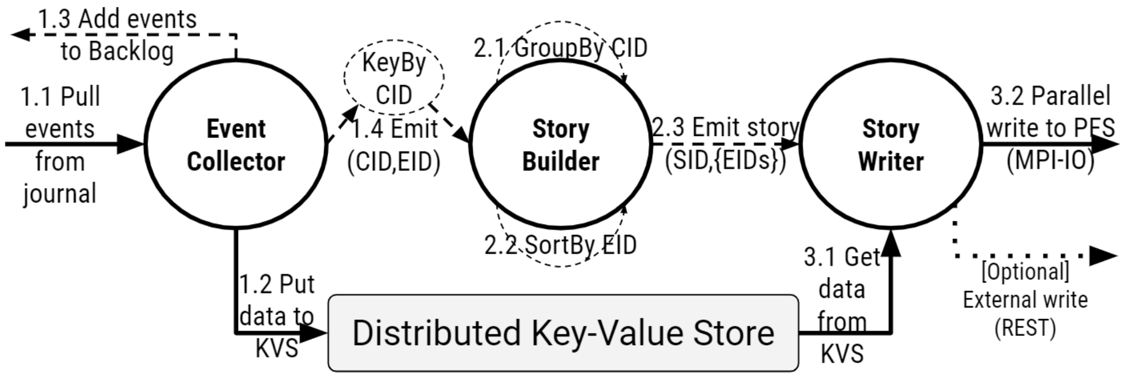
ChronoGrapher

- Continuously moves data from ChronoKeeper to PFS
- Aggregates I/O

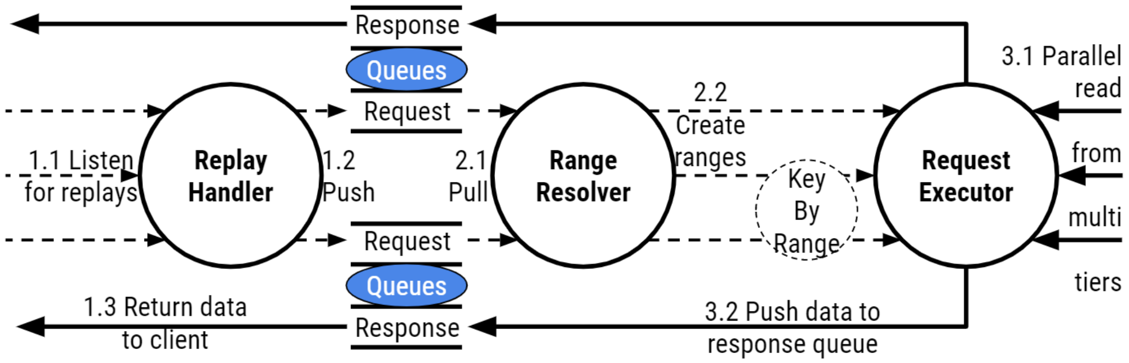
ChronoPlayer

- Retrieves data from PFS, SSD KV and ChronoKeeper
- Resolves range and perform I/O once for duplicate ranges.

ChronoGrapher



ChronoPlayer



Evaluation

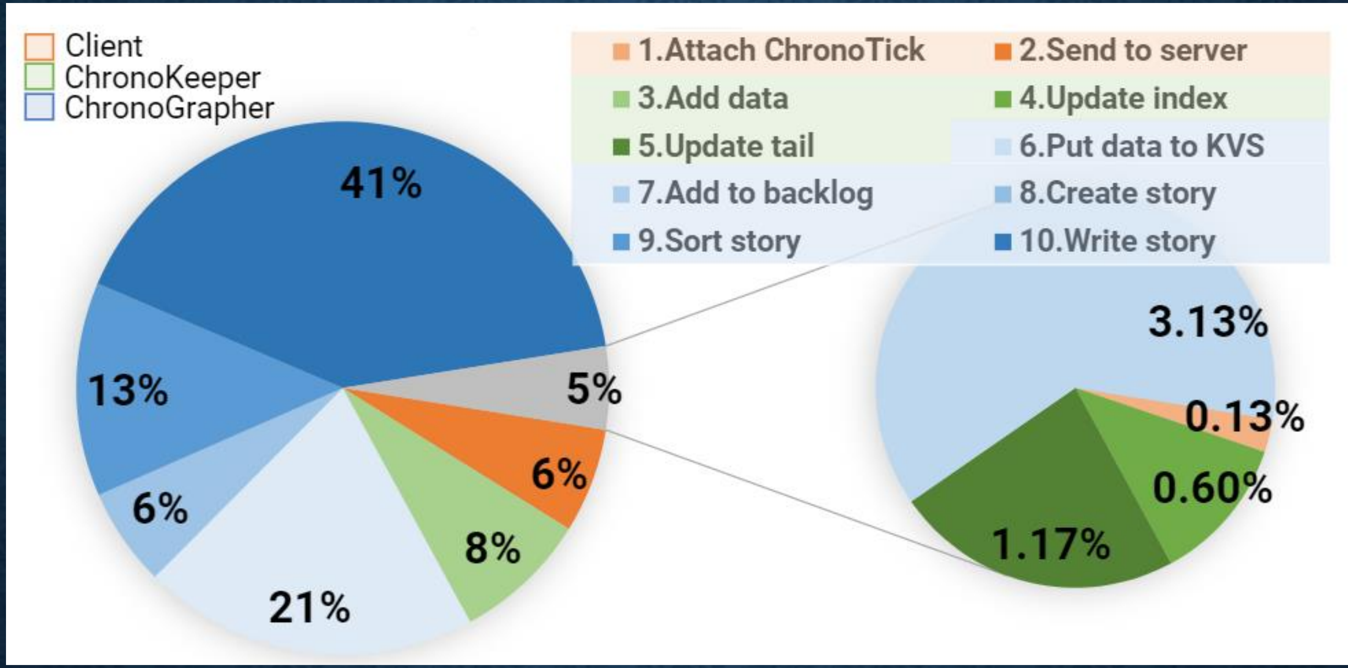
- **Cluster Configuration**
  - 64 compute nodes
  - 4 Key-Value Store Nodes
  - 24 storage nodes
- **Node Configurations**
  - compute node
    - 64GB RAM and 512GB NVMe
  - Key-Value Store Node
    - 64GB RAM and 2x512GB SSD
  - Storage node
    - 64GB RAM and 2TB HDD

Testbed

Configuration

- Applications tested
  - Synthetic Benchmarks
- Compared solutions
  - BookKeeper
  - Corfu

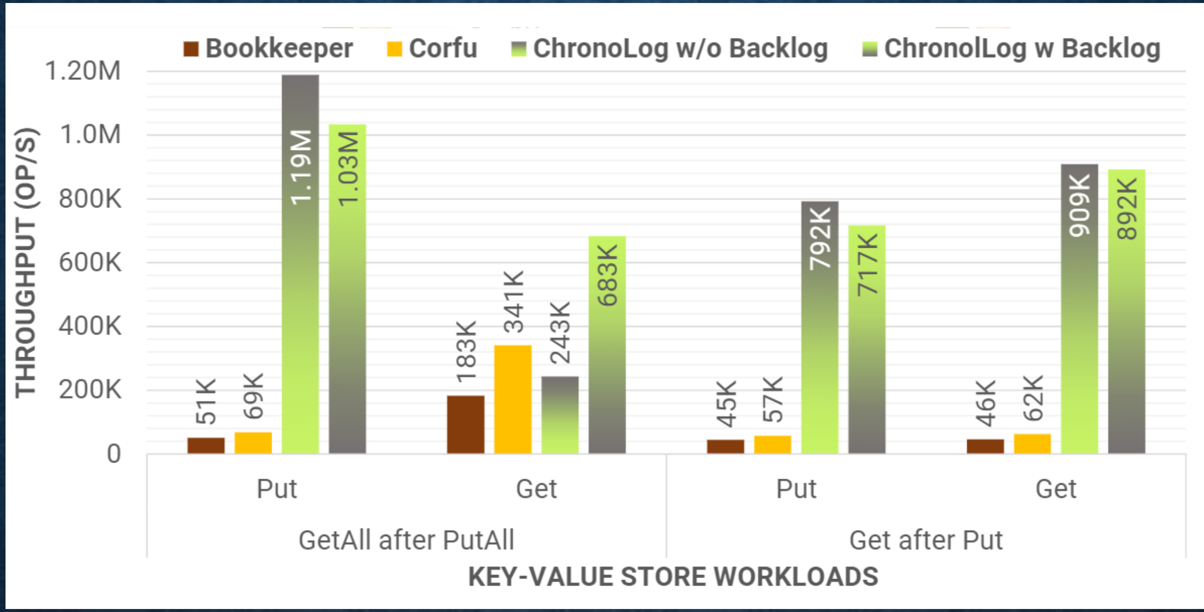
Write Operation breakdown.



Observations:

- The observed Write Operation cost is 14% of the whole journey.
- Asynchronously, data is flushed in the background where writing to KV store and writing to PFS takes 62% of the time.
- Building of Story (aggregation) is 13% of time.

Key-Value Store Performance



Observations:

- BookKeeper is the slowest as operations are served by one server always.
- Corfu uses better data distribution.
- ChronoLog, uses hierarchical storage which increases the throughput of operations
  - For get all after put all, as data is already flushed to slower mediums, hence, reads are slower.
  - It has better locality in Get after Put.

Summary

A list of all observations

01 A distributed log store which utilizes hierarchical storage and time-based data ordering

02 We showcased the design of real-time data movement paradigm to enable MWMR semantics.

03 Quantified the benefit of utilizing hierarchical hardware and time-based ordering.

04 ChronoLog can optimize applications by almost 12x.

Outline

Jal

1 **Profiler**
Code-block level application profiling.

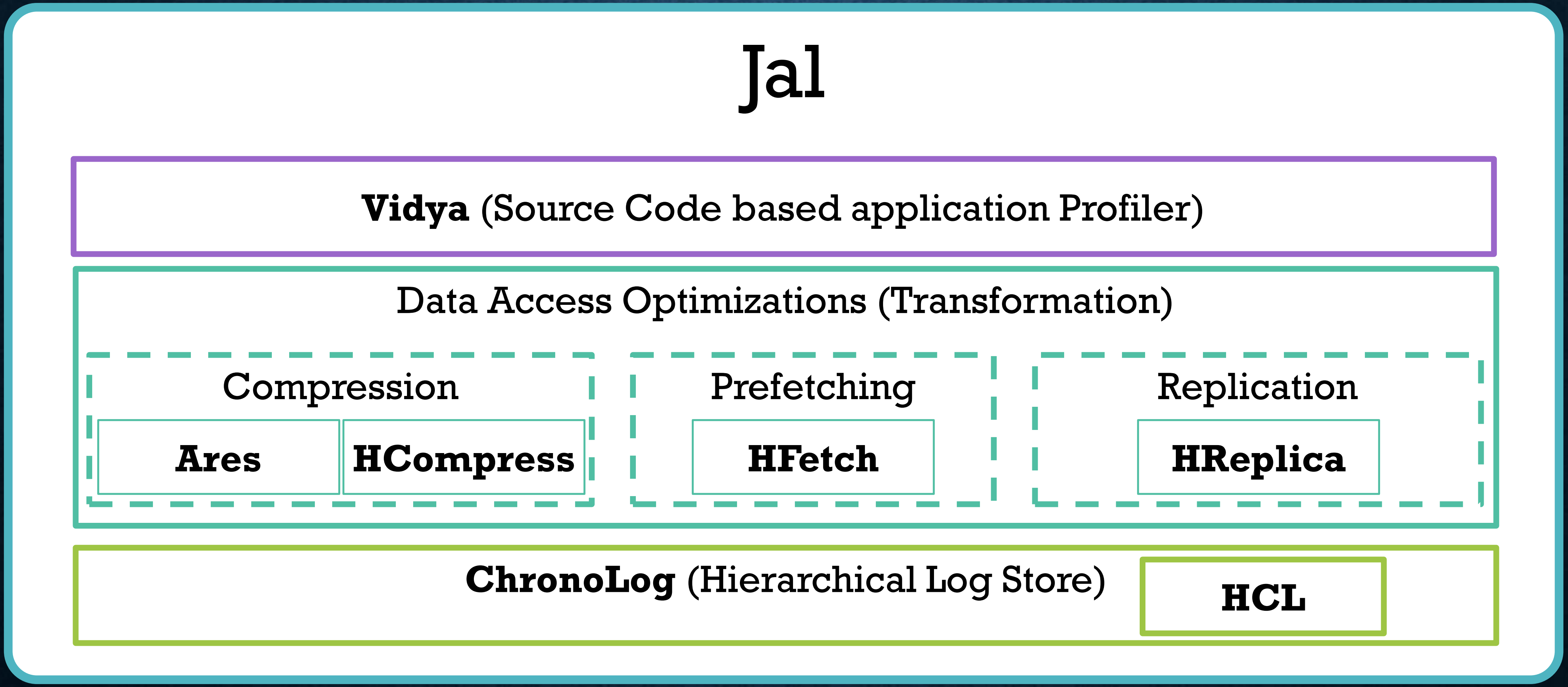
2 **Data Compression**
Multi-tiered data compression engine.

3 **Data Prefetching**
Multi-tiered data prefetching technology.

5 **ChronoLog**
A Shared log store.

6 **Conclusion**

Jal Storage System



Accomplishments

- **Conference Papers**
  - Anthony Kougkas, Hariharan Devarajan, Keith Bateman, Jaime Cernuda, Neeraj Rajesh and Xian-He Sun. ChronoLog: A Distributed Shared Tiered Log Store with Time-based Data Ordering" Proceedings of the 36th International Conference on Massive Storage Systems and Technology (MSST 2020). (to appear)
  - Hariharan Devarajan, Anthony Kougkas, and Xian-He Sun. "*HFetch: Hierarchical Data Prefetching for Scientific Workflows in Multi-Tiered Storage Environments*," 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), New Orleans, Louisiana, USA, 2020.
  - Hariharan Devarajan, Anthony Kougkas, Luke Logan, and Xian-He Sun. "*HCompress: Hierarchical Data Compression for Multi-Tiered Storage Environments*," 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), New Orleans, Louisiana, USA, 2020.
  - Hariharan Devarajan, Anthony Kougkas, and Xian-He Sun. "*An Intelligent, Adaptive, and Flexible Data Compression Framework*", In Proceedings of the IEEE/ACM International Symposium in Cluster, Cloud, and Grid Computing (CCGrid'19)
  - Hariharan Devarajan, Anthony Kougkas, Prajwal Challa, and Xian-He Sun. "*Vidya: Performing Code-Block I/O Characterization for Data Access Optimization*", In Proceedings of the IEEE International Conference on High Performance Computing, Data, and Analytics 2018 (HiPC'18)
- **Journal Papers**
  - Hariharan Devarajan, Anthony Kougkas, and Xian-He Sun, "*I/O Acceleration via Multi-Tiered Data Buffering and Prefetching*", Journal of Computer Science and Technology, 2019, (pre-print and scheduled to appear in 1st quarter of 2020)
- **Workshop Papers**
  - Hariharan Devarajan, Anthony Kougkas, Hsing-Bung Chen, and Xian-He Sun. "*Open Ethernet Drive: Evolution of Energy-Efficient Storage Technology*", In Proceedings of the ACM SIGHPC Datacloud'17, 8th International Workshop on Data-Intensive Computing in the Clouds in conjunction with SC'17.

Related Work

Data Prefetching and Compression

04

- Hardware prefetchers move data from memory into CPU caches to increase the hit ratio.
- Offline data prefetchers involves a pre-processing step which identifies application's access pattern and device a prefetching plan.
- Smart compression asymmetric compression schemes to reduce energy consumption.

I/O characterization in HPC

01

- Static Tools
  - Captures application-level access pattern information per-process and per-file granularity
- Dynamic Tools
  - Uses models the behavior of I/O in any HPC application and predicts future accesses

Shared Log Store

03

- Corfu:
  - Distributed Log store for SSD
  - Uses sequencer for data ordering
- BookKeeper:
  - Uses implicit parallelism for reading.
  - Writing to a jounral goes to one server.
  - Tail is maintained using metadata service.

Tiered storage management

02

- transparent management of this hierarchy for buffering purposes
  - Hermes
  - Proactive Data Container
  - Univistor
- significant boost to I/O performance through data buffering in faster devices.

Q&A

Thank you

hdevarajan@hawk.iit.edu