SCALABLE COMPUTING SOFTWARE LABORATORY

ILLINOIS INSTITUTE OF TECHNOLOGY

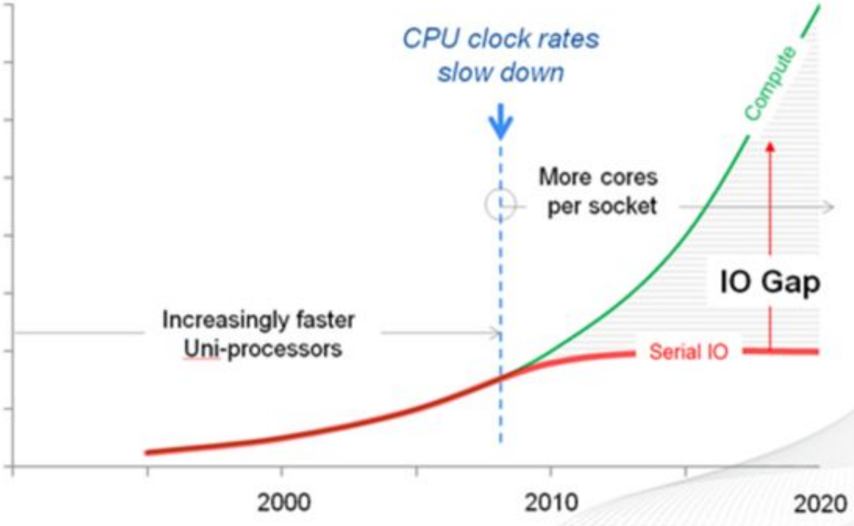
IEEE International Parallel and Distributed Processing Symposium
(IPDPS)

Hierarchical Data Prefetching for Scientific Workflows in Tiered Storage Environments

Hariharan Devarajan, Anthony Kougkas, and Xian-He Sun
hdevarajan@hawk.iit.edu

I/O Bottleneck

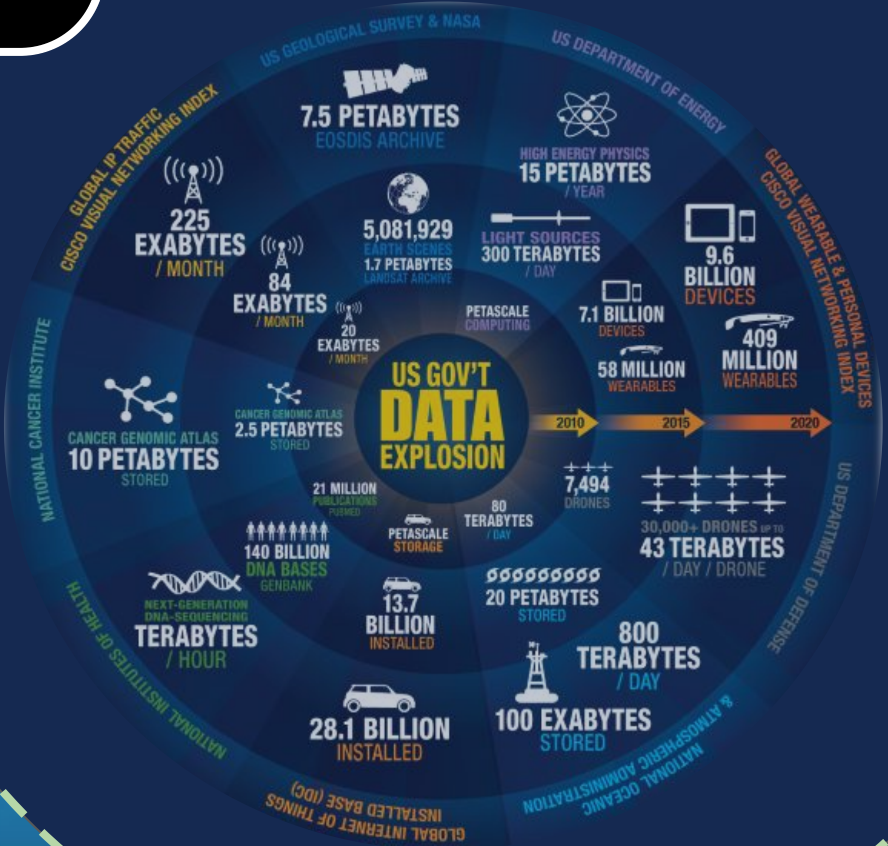
- In the data-intensive era, producing and consuming data is critical for scientific discovery.
- I/O subsystems struggle to match growing compute parallelism.
- System performance is bound by its slowest component. (Amdahl's "well-balanced" law)
- I/O performance is a concern in petascale, and would exaggerate even more as we ascend towards exascale.



Growing gap of CPU and I/O performance

Explosion of data

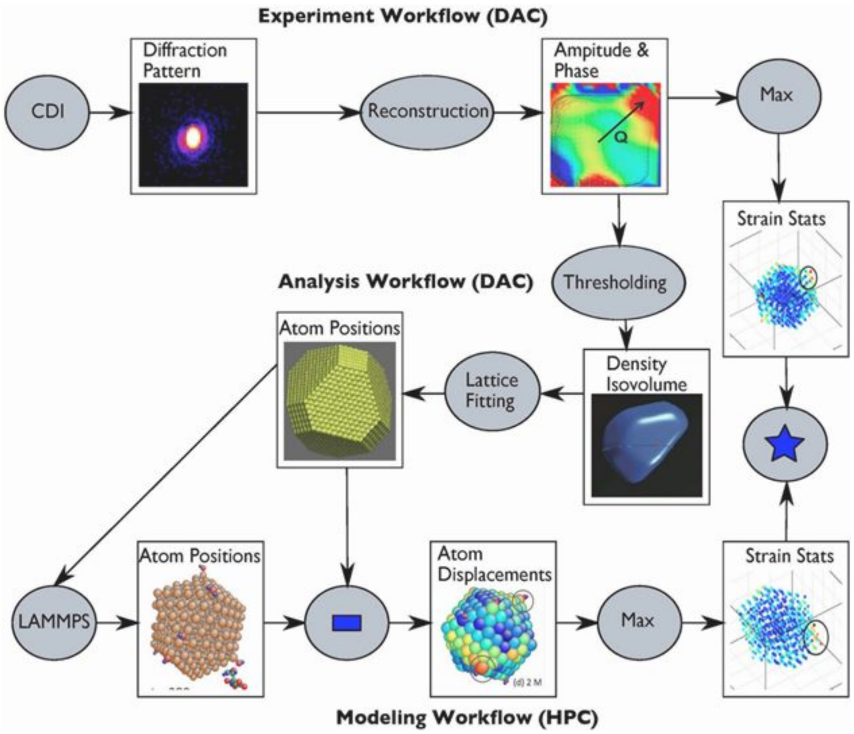
- Data is crucial to enable discovery.
- IDC reports predict that by 2025:
    - global data volume will grow to 163 ZB
    - 10x the data produced in 2016



Scientific Workflow

- **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.

Complex Scientific Workflows



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 data access.

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.

SCAN ME

Code: https://bitbucket.org/scs-io/hfetch

HCompress Goals

Server-Push

Lightweight and asynchronous data push.

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

Data Centric

Utilize how data is accessed in a workflow.

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

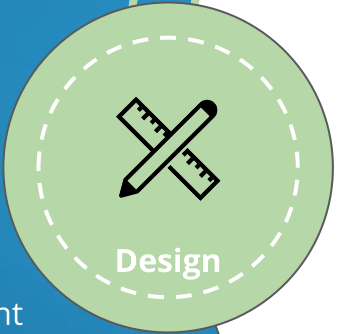
Hierarchical

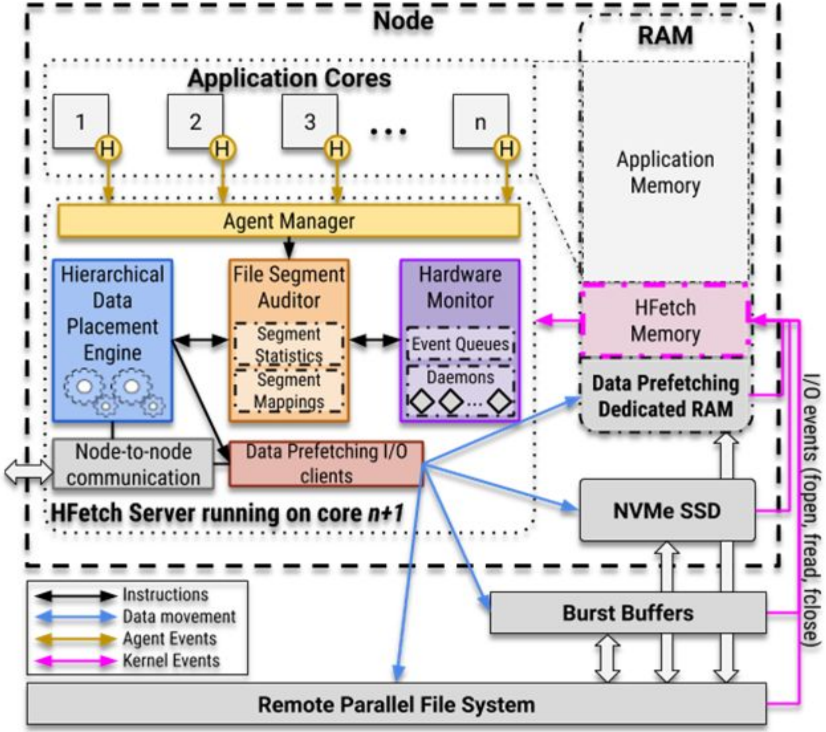
Unify the diverse hardware tiers.

The engine matches data hotness to the device characteristics.

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





Example

Time	Client space				Kernel space	HFetch Server space					
	Applications		HFetch Agents		inotify_handle_event	Hardware Monitor	Auditor				Data Placement Engine Tiers: T1<T2<T3<T4
							Update Segment Statistics			Calculate Segment Score	
	#1	#2	Agent#1	Agent#2	(push to event queue)		Frequency	Recency	Sequence		
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)					

- Specific Client I/O interception of open/close
- Monitoring through VFS layer
- Collect event through Hardware Monitor.
- Each layer has a different daemon

- Update Auditor
  - Calculate scores
  - Rearranges scores in descending order
- Run DPE
- 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

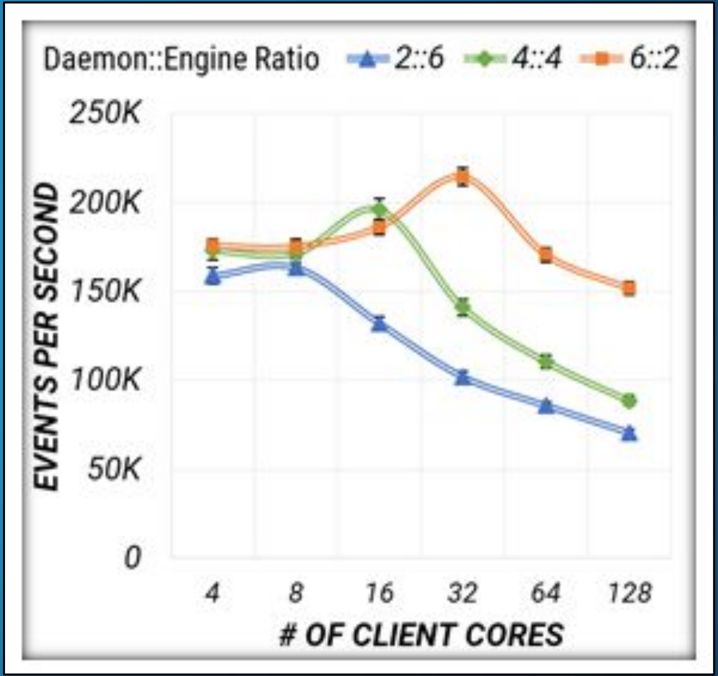
Tedbed

Config

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

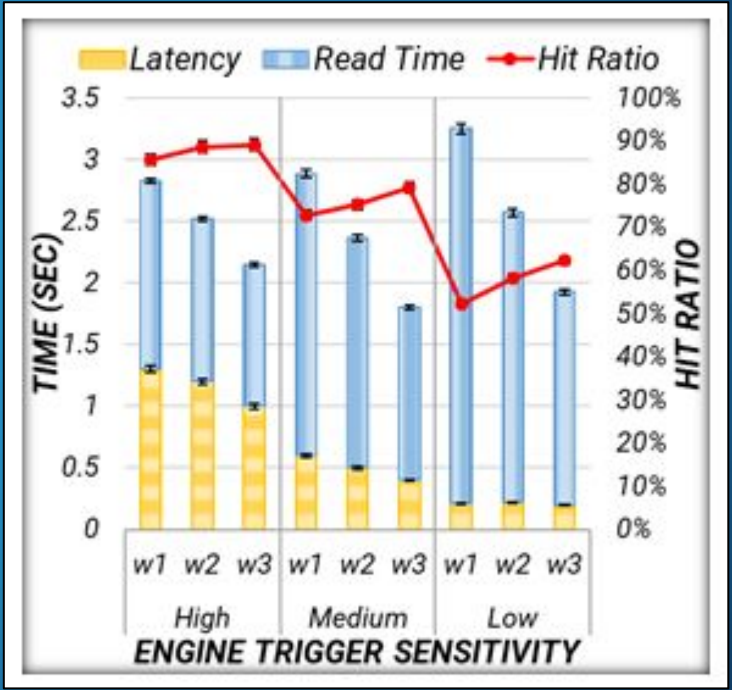
Server-Client Ratio

- Single Node test
- Test the Server with different thread counts for client, daemon and engine
  - As clients increase more threads on daemon to match production rate.
- **<u>Observations</u>**
  - Match production rate with consumption rate.
  - Max throughput is **213K** ops/sec.
  - 1 HFetch server 32 clients.



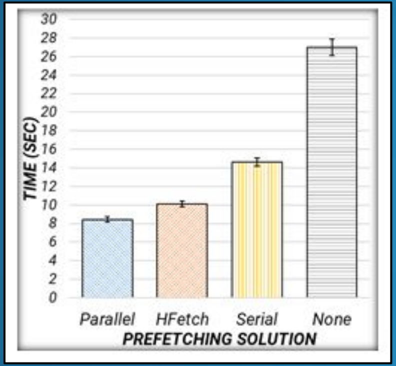
Placement Engine Reactiveness

- Single Node test
- Test the how sensistive engine should be with different updates
  - The engine should match update rate to be optimal.
- <u>**Observations**</u>
  - Trade-off “optimal placement” and “engine cost”.
  - We provide a **auto tuning** of engine based on rate of updates.

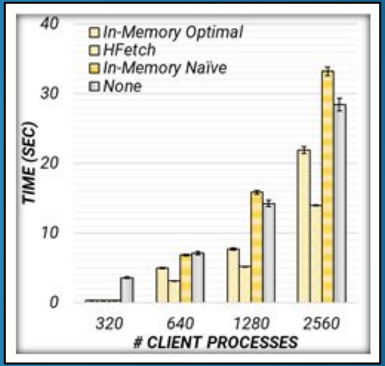


Benefit of Hierarchical Prefetching

Lower-RAM footprint



Extending Prefetching cache.



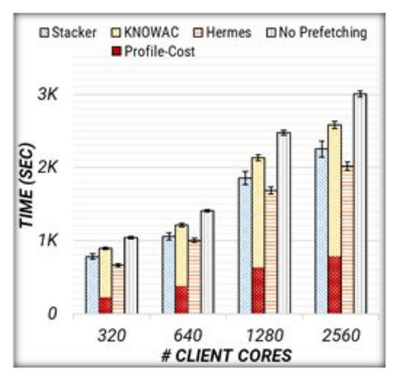
Observations:

- 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.

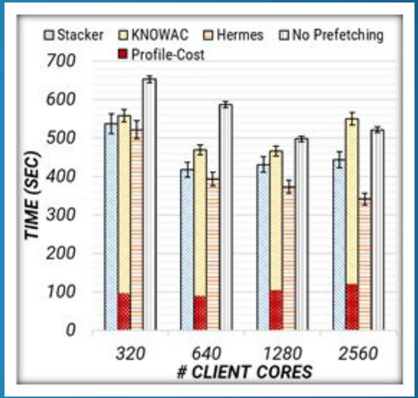
- Adding more layers reduces the cost of miss penalty
  - Additional cache space on lower tiers
  - Devices slower than RAM but faster than PFS.
- 35% to 50% faster.

Scientific Workflows

Montage



WRF



Observations:

- Offline Profiler is accurate with high profiling cost.
- 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%.

Conclusions

A list of all observations

1. HFetch introduces a data-centric hierarchical prefetching methodology.
2. HFetch proposes a novel data centric scoring mechanism to measure the hotness of data.
3. Quantified the benefit of utilizing hierarchical hardware and data prefetching cohesively.
4. HFetch can optimize scientific workflows up to 35% compared to competitive solutions.

Q&A

Thank you

hdevarajan@hawk.iit.edu

Video

SCAN ME

Code

SCAN ME