OctopusFS: A Distributed File System with Tiered Storage Management

Elena Kakoulli and Herodotos Herodotou

Cyprus University of Technology

Cyprus University of Technology

Current Trends in Data-Processing Systems

- Run on large clusters of **commodity hardware**
  - Significant improvements over the years
- Execute diverse workloads ⇒ **diverse I/O patterns**
  - Batch, iterative, interactive
- Process data residing on **distributed file systems**
  - HDFS, GFS, MapR
- Exploit recent **improvements in storage media**
  - Store data in memory & SSDs, memory caching




Proposition

Run on clusters of commodity hardware

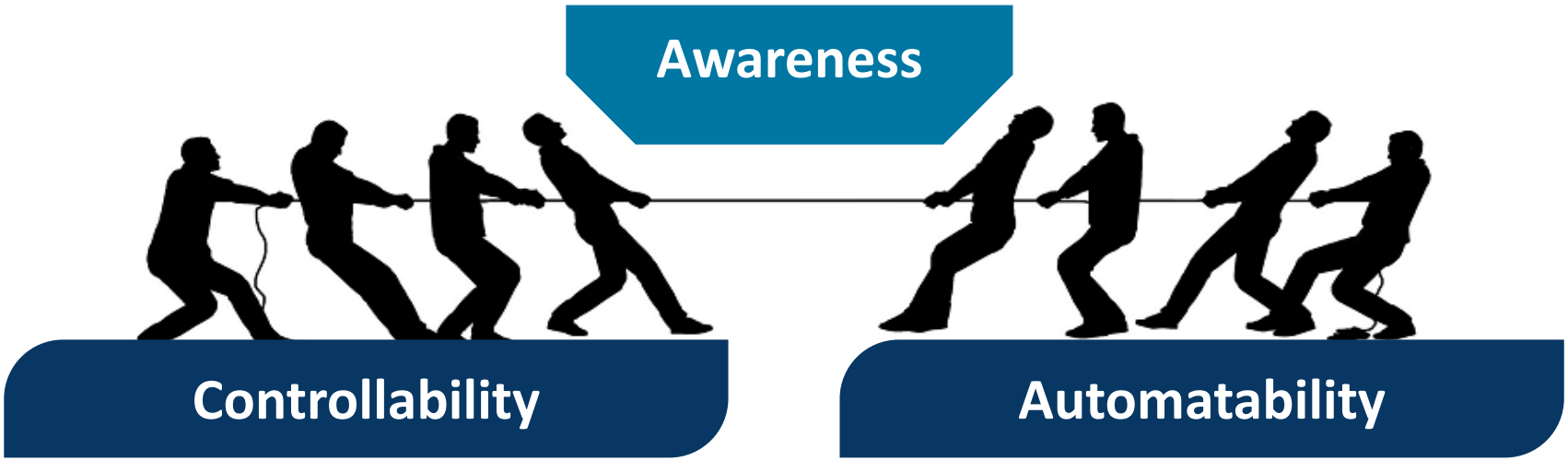
Exploit heterogeneous storage media

Maintain prevalent file abstraction

Design a **distributed**, **multi-tiered** **file system** (**OctopusFS**) that is ***aware*** of storage media with different capacities and **performance characteristics**

Make decisions based on them

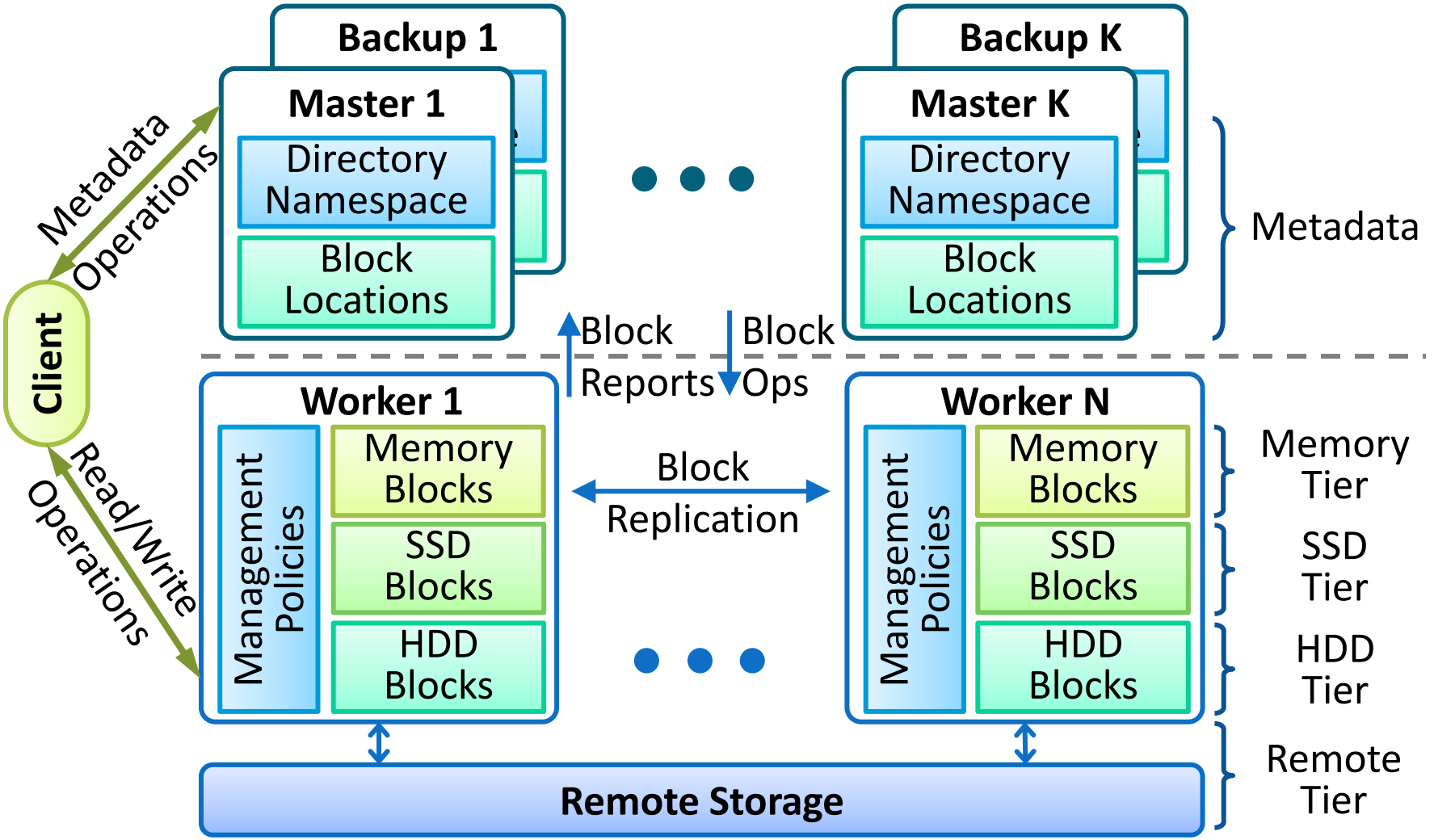
OctopusFS Design Focus



- Storage media are explicitly exposed to applications
- Applications can control the distribution and placement of replicas

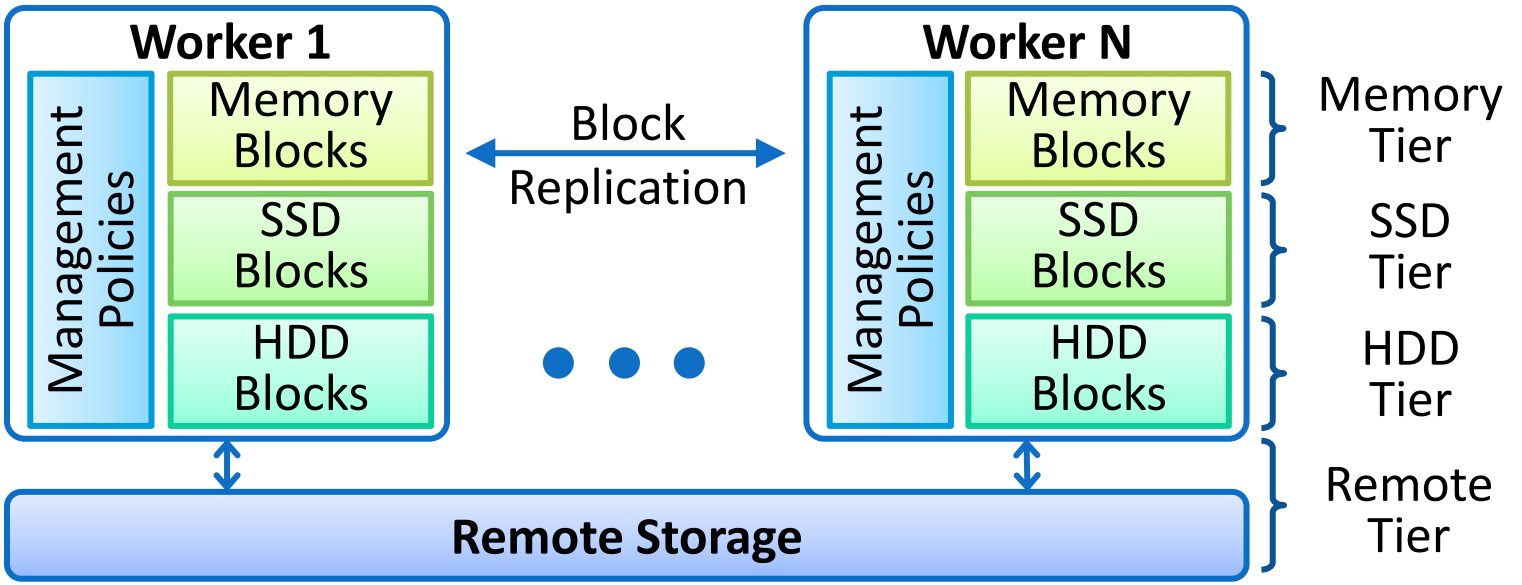
- Automate data management decisions based on network topology and storage tiers
- Offer pluggable decision policies

OctopusFS Architecture

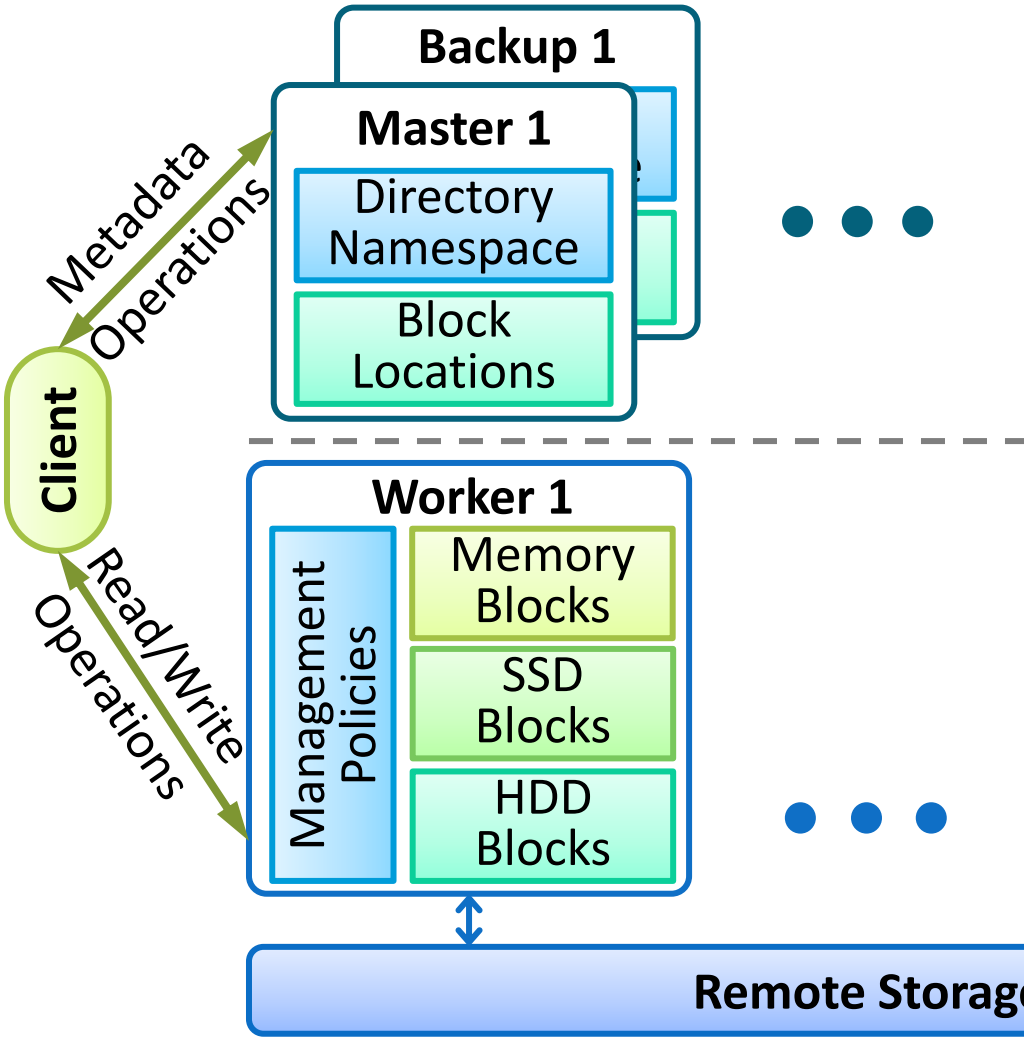


Storage Tiers

- Storage media are **logically** grouped into **storage tiers**
  - "logically" ⇒ based on **I/O performance**
- Files are striped and replicated across Workers and storage tiers based on **client requests** or **automated policies**.



Client



- **Client**
  - Exposes APIs for all typical file system operations (extensions based on Apache Commons FileSystem API v.2.7.0)
  - Exposes locations and storage tiers of block replicas
  - APIs based on the notion of the **replication vector**

Replication Vector

- Specifies number of file replicas for each storage tier
  - V = <"Mem","SSD","HDD","Rem","Unspec"> = <1, 0, 2, 0, 0>
- Specify the replication vector upon **file creation**:

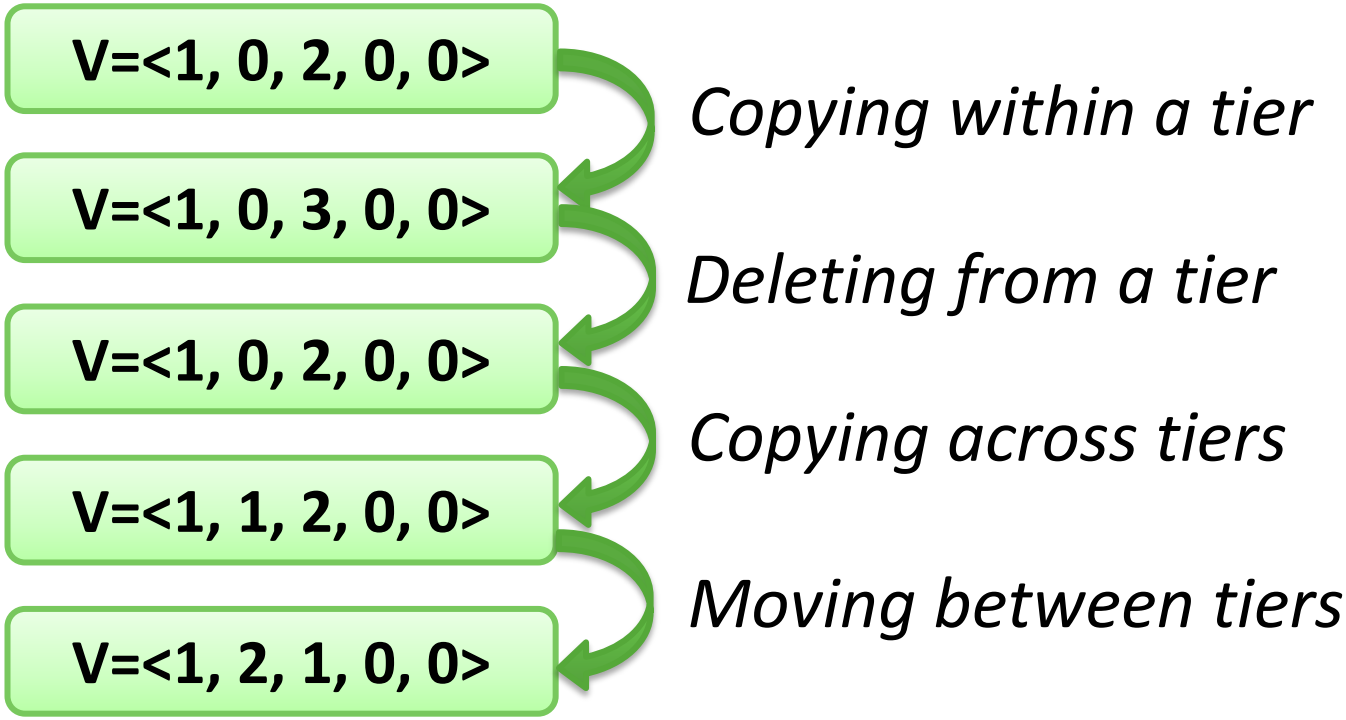
V=<1, 0, 2, 0, 0> ← *Create 1 replica in memory and 2 on the HDD tier*

V=<1, 0, 0, 0, 2> ← *Create 1 replica in memory and 2 on any tier (system decides)*

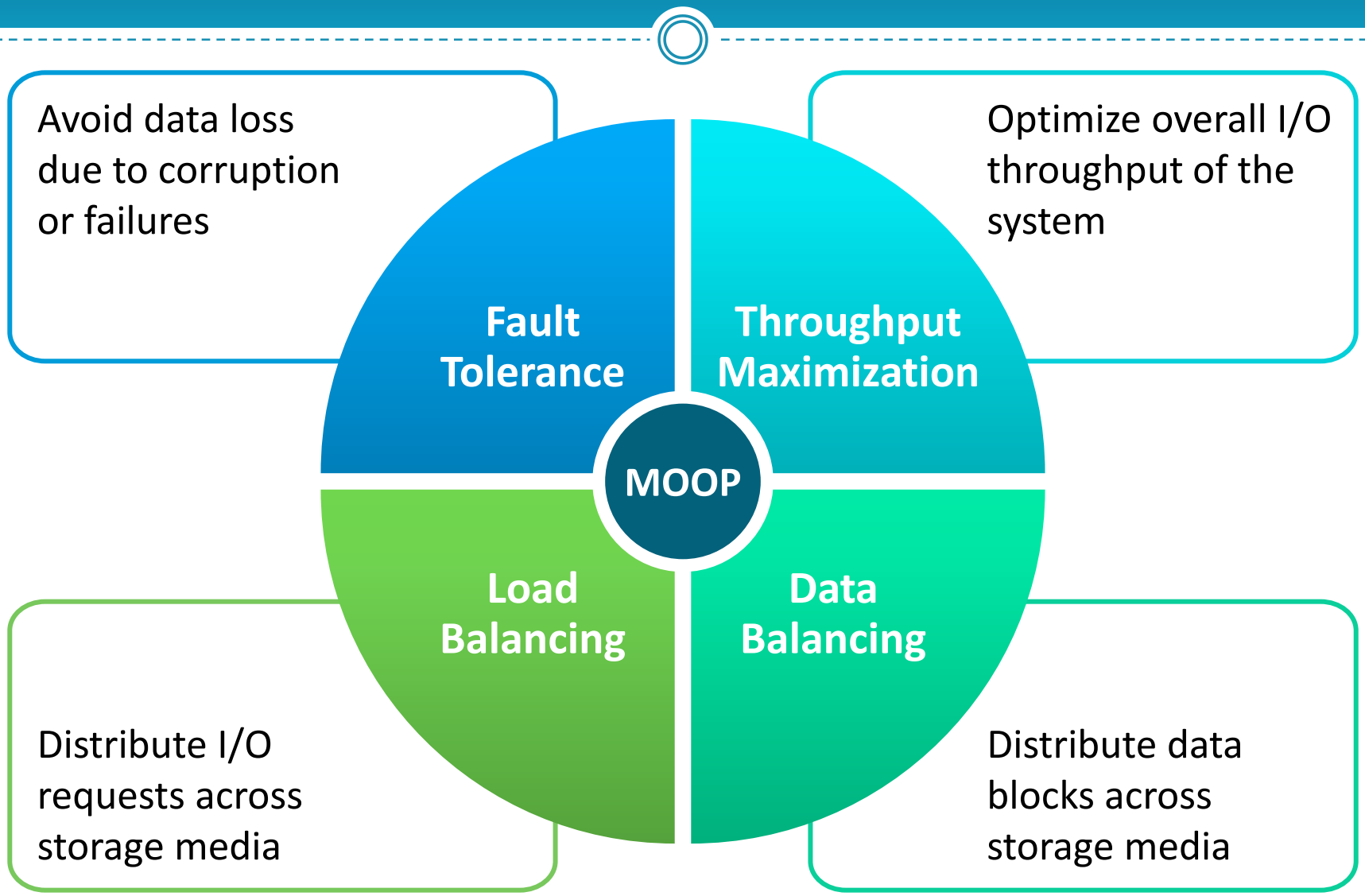
V=<0, 0, 0, 0, 3> ← *Create 3 replicas on any tier(s) (system decides)*

Replication Vector

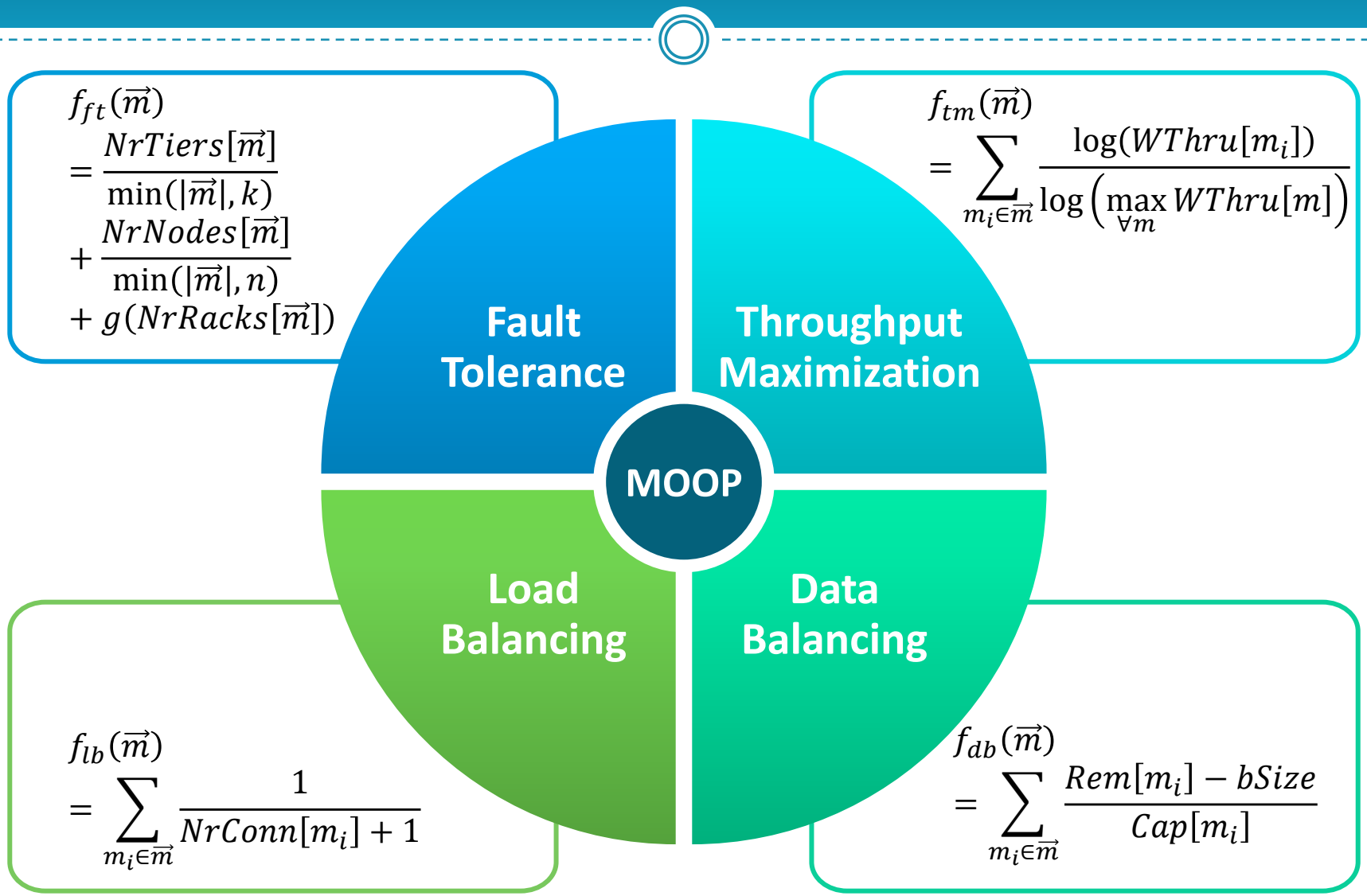
- Specifies number of file replicas for each storage tier
  - V = <"Mem","SSD","HDD","Rem","Unspec"> = <1, 0, 2, 0, 0>
- Single API to **modify** a replication vector for:



Data Placement Objectives



Data Placement Objectives



Data Placement Policy

- **Multi-Objective Optimization Problem (MOOP):** Select the best r **replica locations** (**<Worker, Tier> pairs**) for storing a block that optimize the four objectives simultaneously

$$\min \| f(\vec{m}) - z^*(\vec{m}) \| , \text{s.t.} \vec{m} \in \vec{M}$$

- **Issue:** Decision space is combinatorial => exponential running time
- **Solution:** A **greedy algorithm** for finding a near-optimal solution to the MOOP (details in the paper)

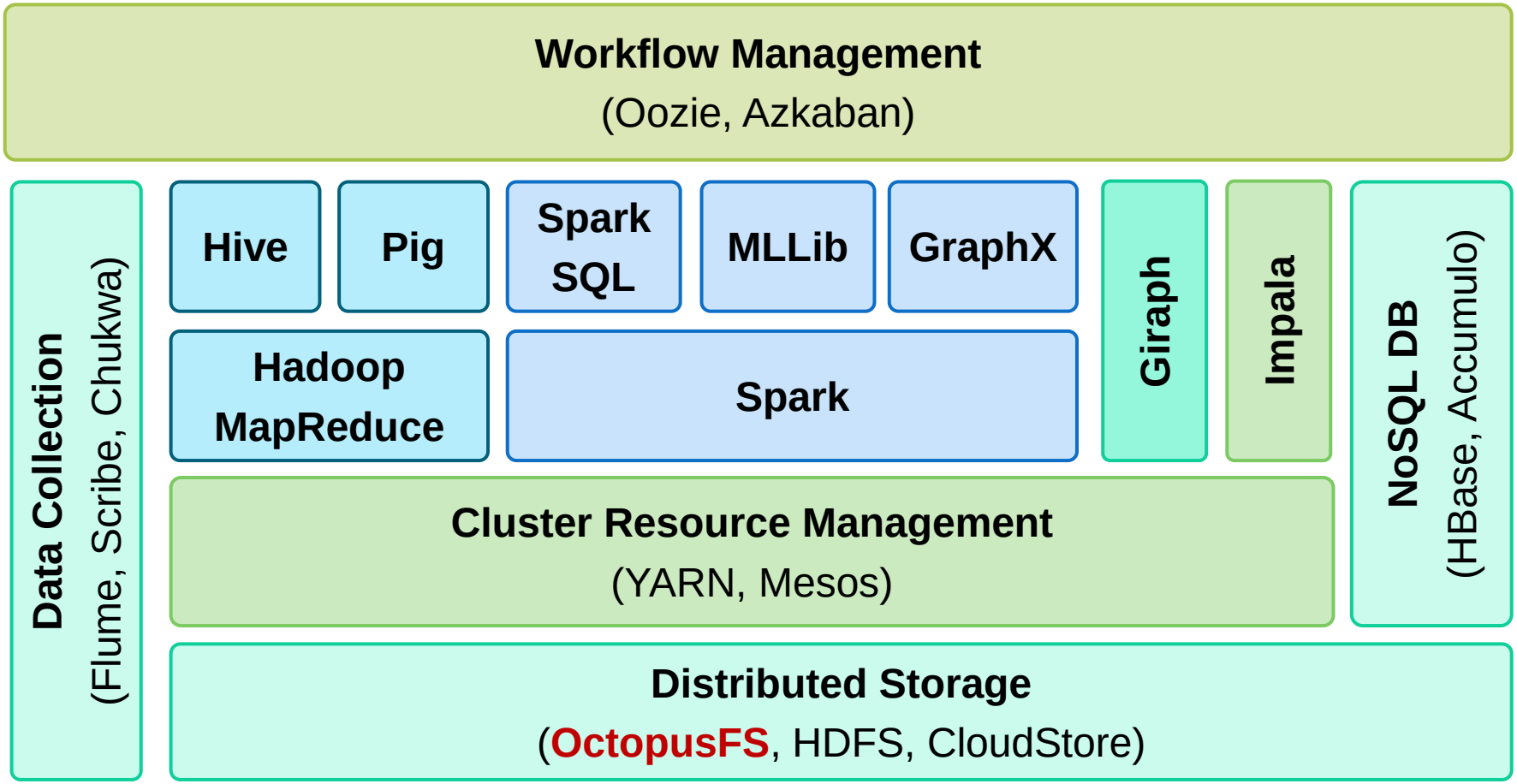
Data Retrieval Policy

- Find the best **replica location** (**<Worker, Tier> pair**) to read from based on:
  - client location
  - replica locations
  - network topology
  - throughput rates of storage media & network
  - active connections per Worker & storage media

$$\min\left(\frac{NetThru[W_j]}{NrConn[W_j]}, \frac{RThru[m_i]}{NrConn[m_i]}\right)$$

- **Data retrieval policy**
  - Calculates data transfer rates from each replica location to the Client and returns a sorted list of locations based on the calculation

OctopusFS in the Big Data Ecosystem



(Selected) Enabling Use Cases

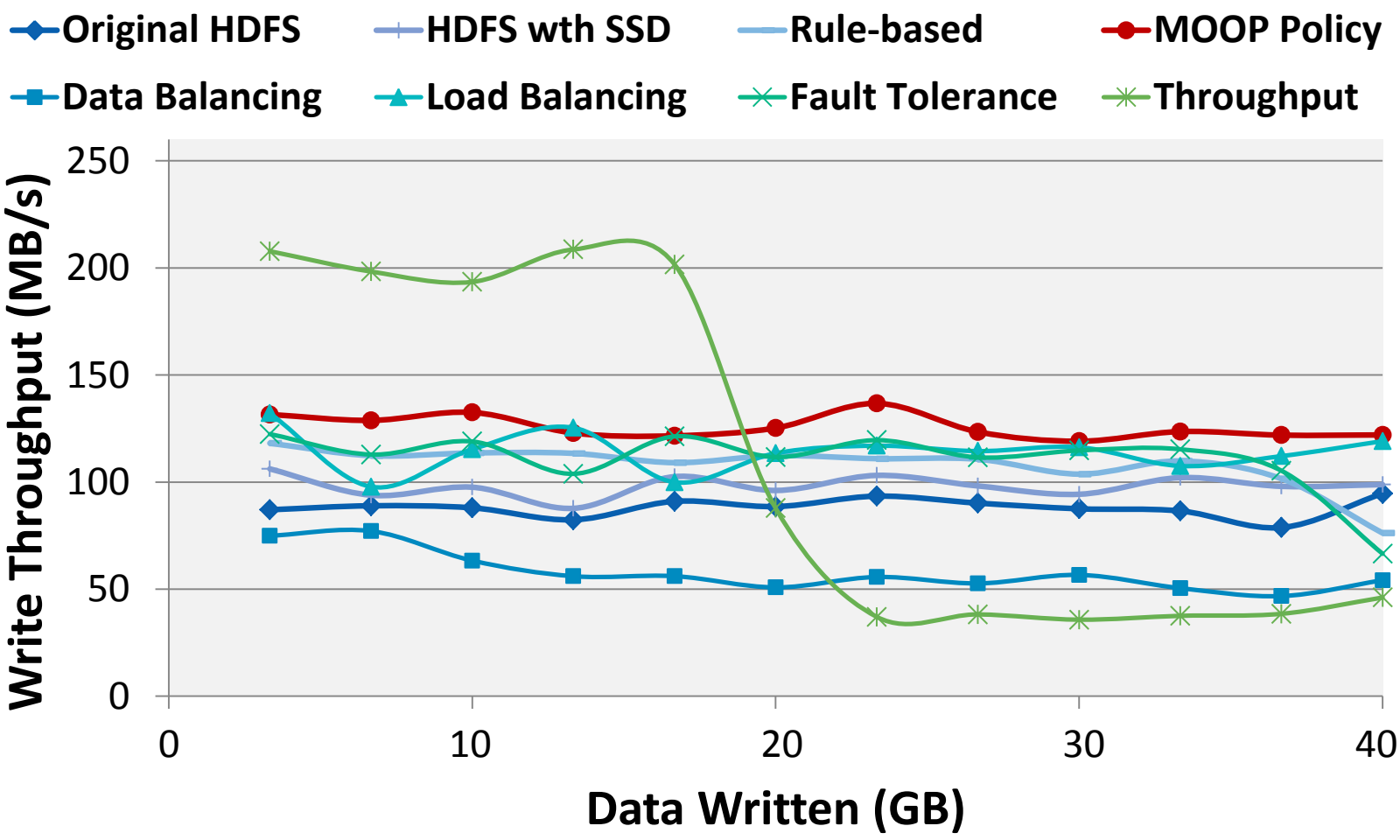
- **MapReduce Task Scheduling**
  - *Schedule tasks* based on network location and storage tier of block replicas
  - Implement *pre-fetching algorithms* for moving block replicas to higher storage tiers before the scheduling of tasks
- **Workload Scheduling**
  - Place *intermediate data* between jobs on higher tiers
  - *Cache hot data* based on workload characteristics
- **Interactive/Iterative Applications**
  - Use explicit memory management to *pin working sets* of interactive/iterative application in cluster memory

Experimental Setup

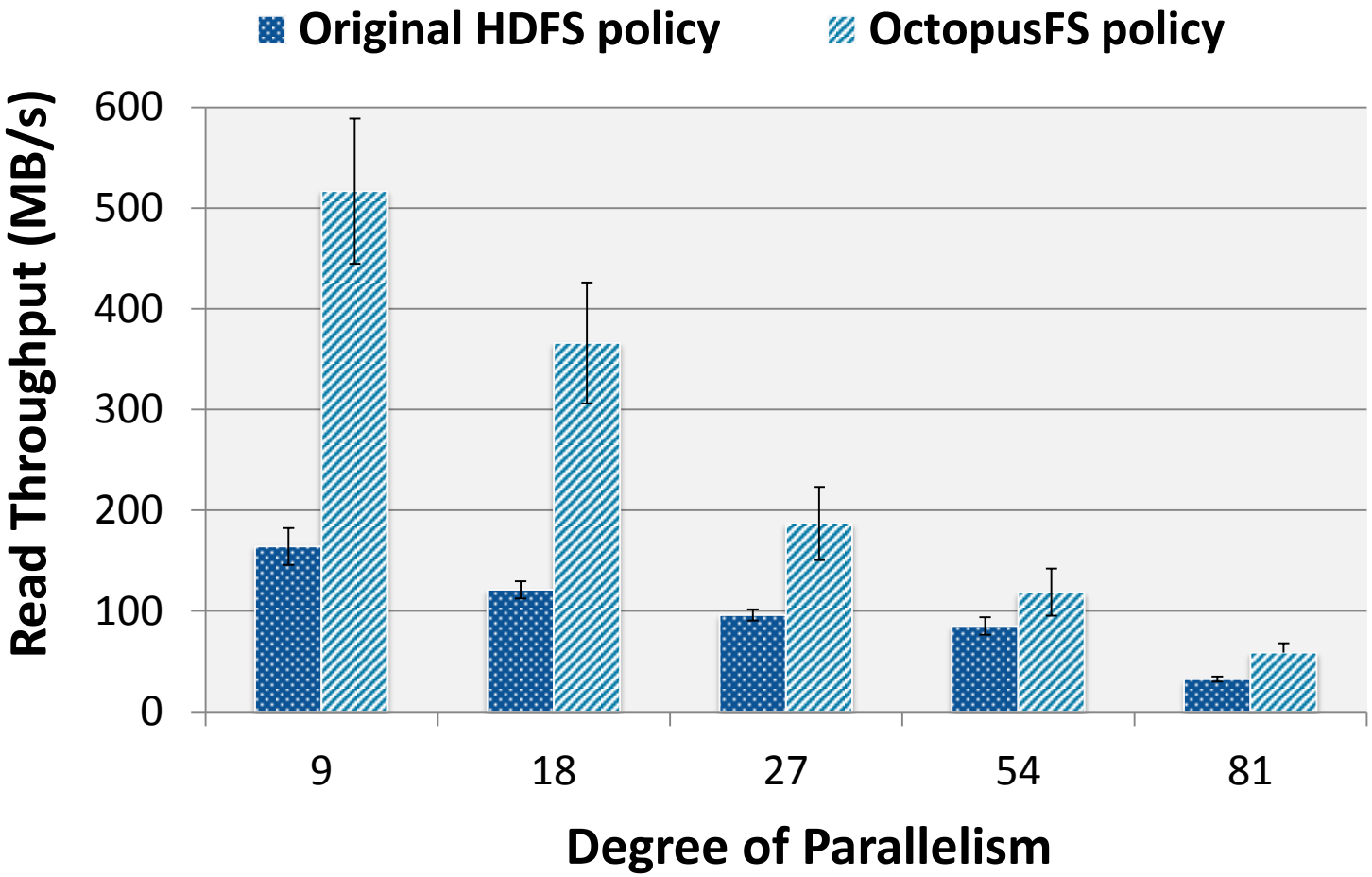
- **1 Master & 9 Workers**
  - 8 core, 2.4 GHz CPU
  - 24 GB RAM
  - One 120GB SATA SSD
  - Three 500GB SAS HDDs
- **Benchmarks**
  - DFSIO
  - S-Live
  - HiBench

Storage Media	Write Throughput (MB/s)	Read Throughput (MB/s)
Memory	1897.4	3224.8
SSD	340.6	419.5
HDD	126.3	177.1

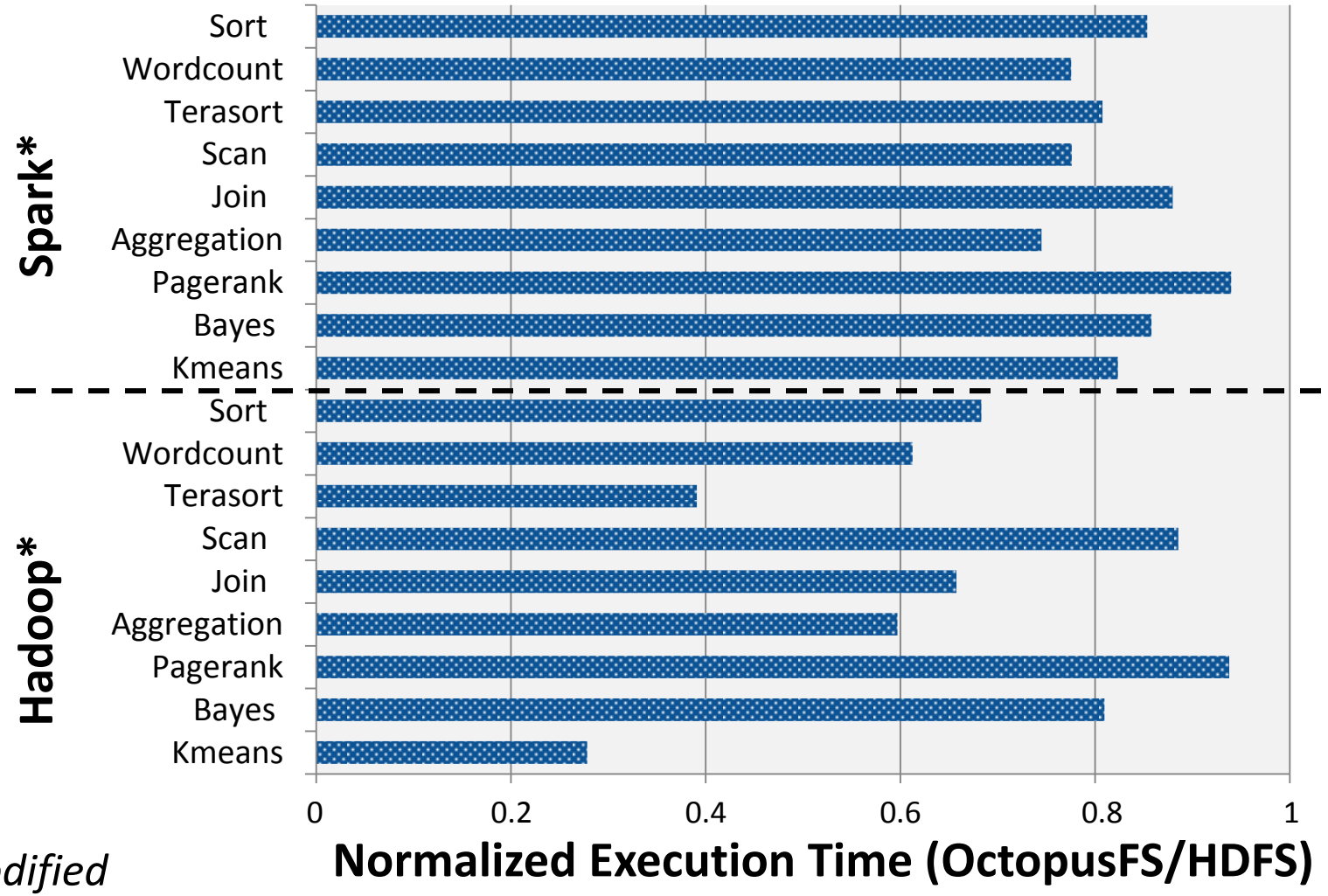
Data Placement Evaluation



Data Retrieval Evaluation



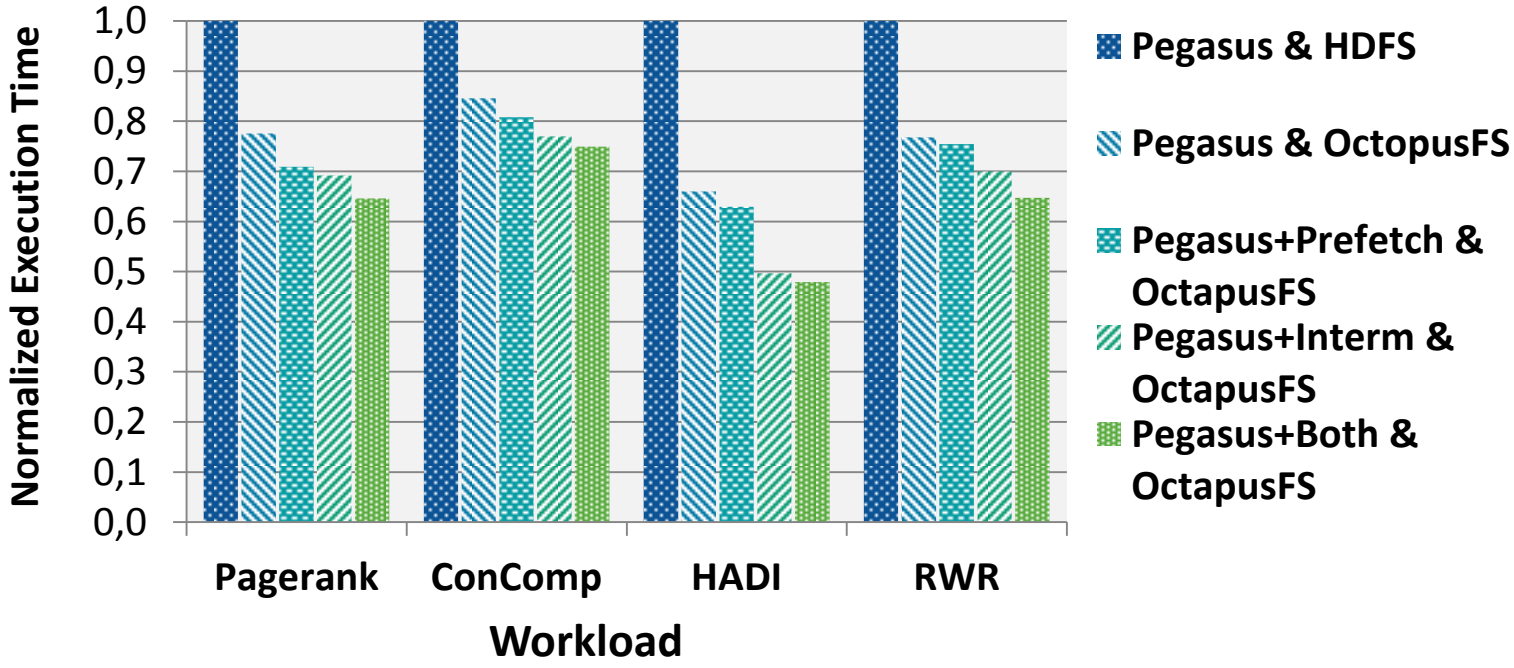
Evaluation over Hadoop and Spark



**Unmodified*

Evaluation of Pegasus Use Case

- **Pegasus:** Graph mining system over Hadoop
- Implemented optimizations:
  - Move data reused between iterations to higher tiers (“prefetch”)
  - Place short-lived intermediate data to higher tiers (“interm”)



Conclusions & Future Work

- **Goal:** Build a *distributed*, *multi-tiered file system* that is *aware* of heterogeneous storage media and offers both *controllability* & *automatability*
- **Current status:**
  - Working system with replication vector APIs
  - Auto data placement & retrieval policies
- **Future work:**
  - Finalize data replication & caching policies
  - Investigate enabling use cases across the stack

Thank You

Cyprus University of Technology

http://dicl.cut.ac.cy