TidyFS: A Simple and Small Distributed Filesystem

Dennis Fetterly[1], Maya Haridasan[1], Michael Isard[1], and Swaminathan Sundararaman[2]

[1]Microsoft Research, Silicon Valley

[2]University of Wisconsin, Madison

Introduction

- Increased use of shared nothing clusters for Data Intensive Scalable Computing (DISC)
- Programs use a data-parallel framework
  - MapReduce, Hadoop, Dryad
  - PIG, HIVE, or DryadLINQ

DISC Storage Workload Properties

- Data stored in streams striped across cluster machines
- Computations parallelized so each part of a stream is read sequentially by a single process
- Each stream is written in parallel
  - each part is written sequentially by a single writer
- Frameworks re-execute sub-computations when machines or disks are unavailable

TidyFS Design Goals

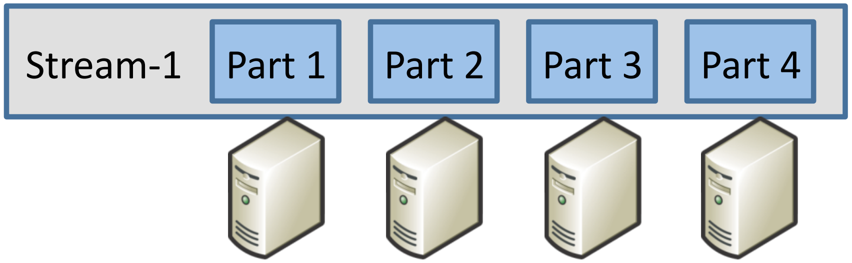
- Targeted only to DISC storage workload
  - Exploit this for simplicity
  - Data invisible until committed, then immutable
- Rely on fault-tolerance of the framework
  - Enables lazy replication

TidyFS Data Model

- Data
  - Stored as blobs on compute nodes
  - Immutable once written
- Metadata
  - Stored in centralized, reliable component
  - Describe how datasets are formed from data blobs
  - Mutable

Client Visible Objects

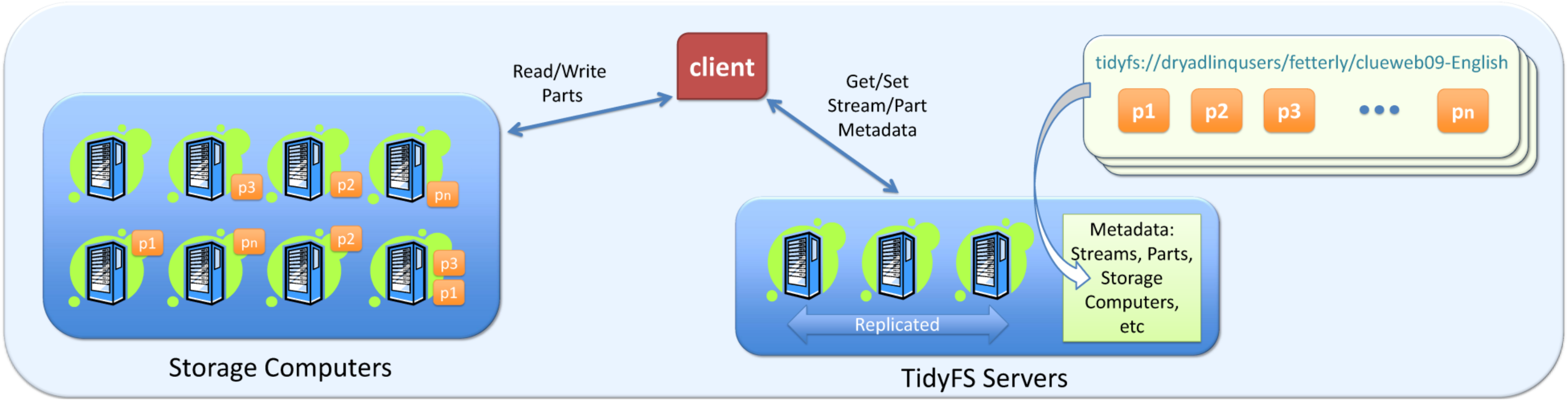
- Stream: a sequence of parts
  - i.e. tidyfs://dryadlinqusers/fetterly/clueweb09-English
  - Names imply directory structure
- Part:
  - Immutable
  - 64 bit unique identifier
  - Can be a member of multiple streams
  - Stored on cluster machines
  - Multiple replicas of each part can be stored



Part and Stream Metadata

- System defined
  - Part: length, type, and fingerprint
  - Stream: name, total length, replication factor, and fingerprint
- User defined
  - Key-value store for arbitrary named blobs
  - Can describe stream compression or partitioning scheme

TidyFS System Architecture



- Metadata server
- Node Service
- TidyFS Explorer

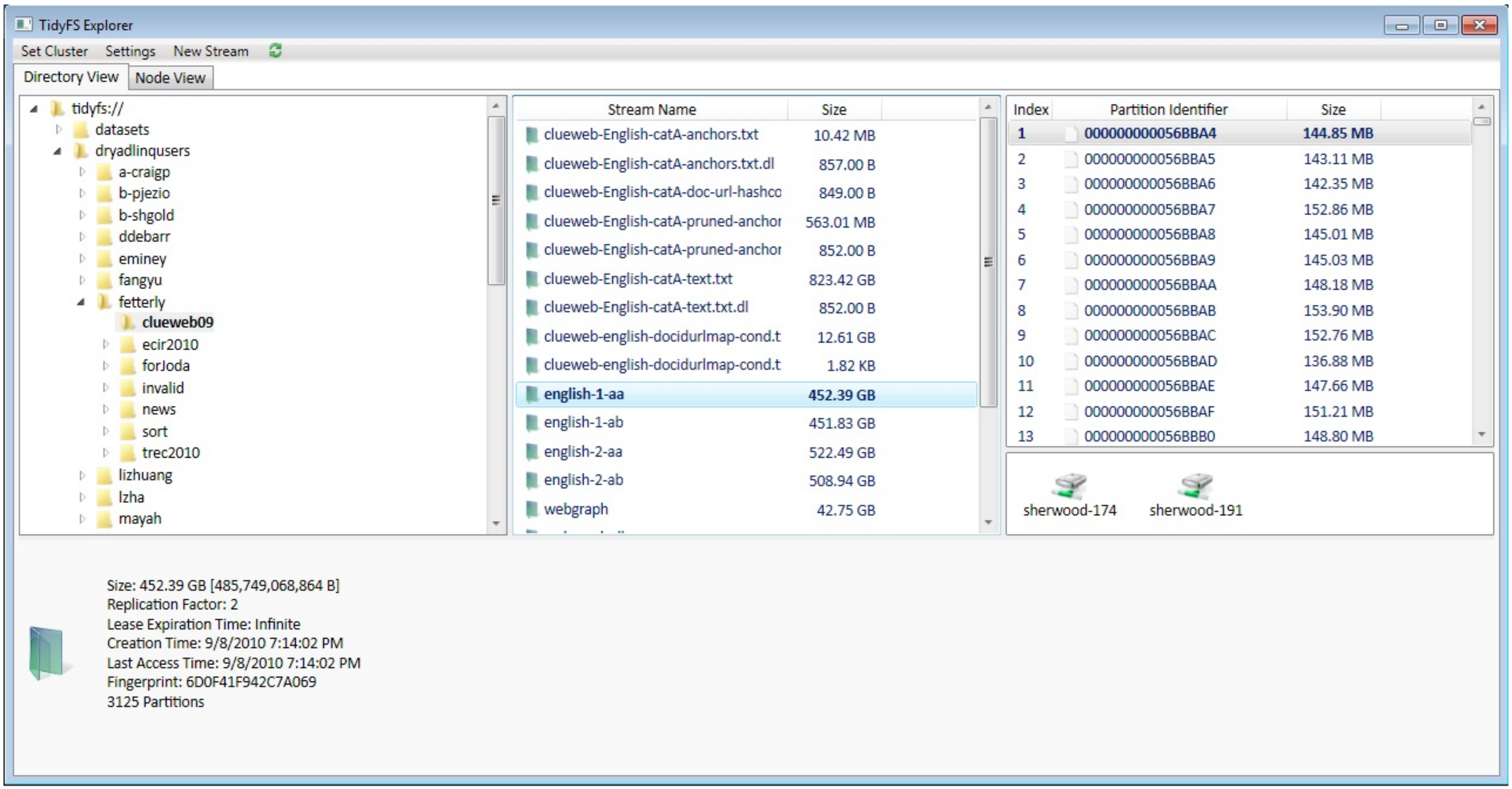
Metadata Server

- Maintains metadata for the system
  - Maps streams to parts
  - Maps parts to storage machine and data path
    - NTFS file, SQL table
  - Contains stream and part metadata
  - Maintains machine state
  - Schedules part replication and load balancing
- Replicated for scalability and fault tolerance

Node Service

- Runs on each storage machine
- Garbage Collection
  - Delete parts that have been removed from TidyFS server (i.e. parts from deleted streams)
  - Verify machine has all parts expected by TidyFS server to ensure correct replica count
- Replication
  - TidyFS server provides list of part ids to replicate
  - Machine replicates partition to local filesystem
- Validation
  - Validate checksum of stored parts

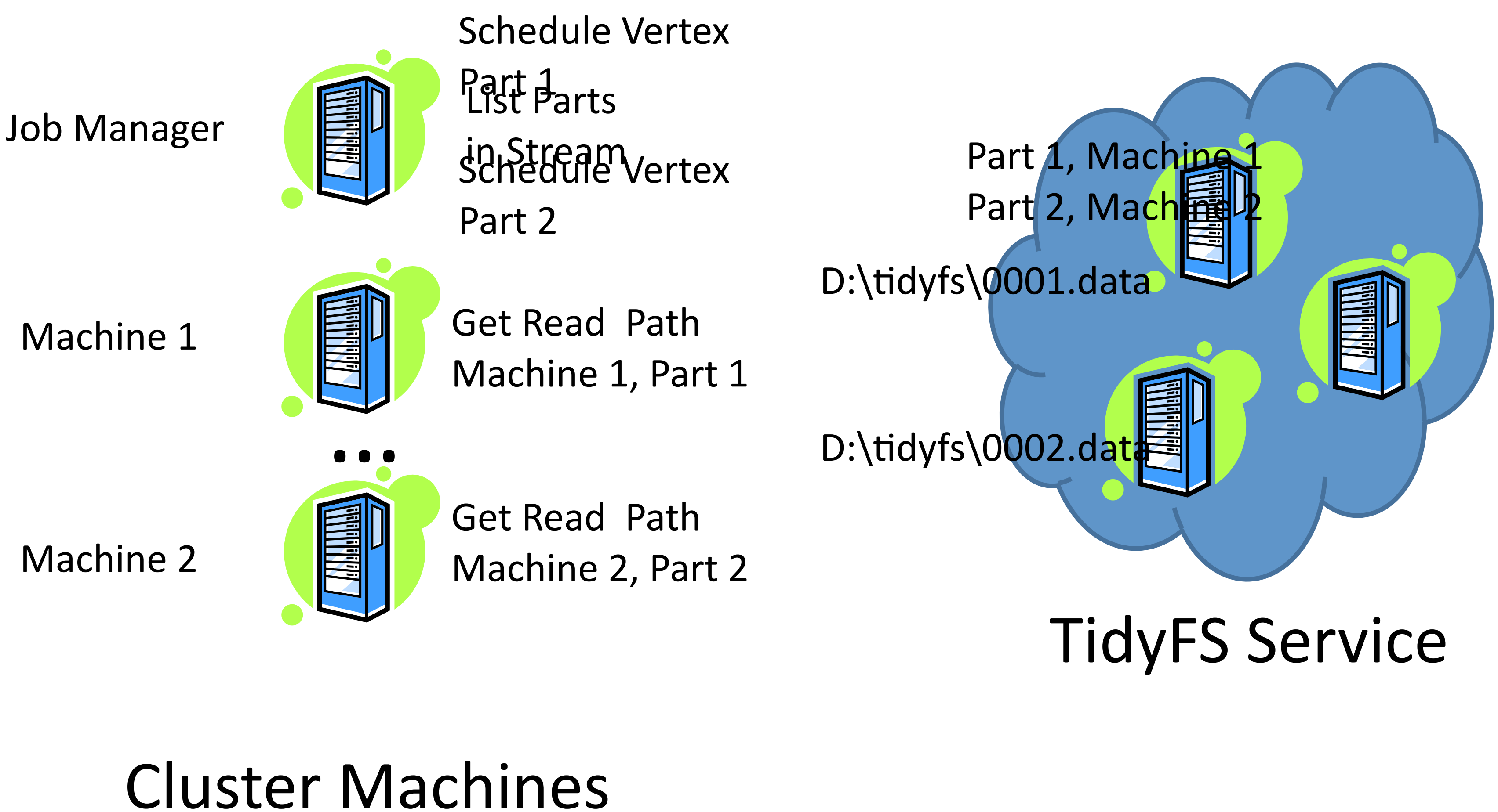
TidyFS Explorer



Client Read Access Patterns

- To read data in stream, a client will:
  - Obtain sequence of part ids that comprise stream
  - Request path to directly access part data
    - Read only file in local file system
    - CIFS path if remote file
    - Paths to DB and log file for DB part
    - Metadata server uses network topology to return the part replica closest to reader

How a Dryad job reads from TidyFS



Client Write Access Patterns

- To write data to a stream, a client will:
  – Determine a destination stream
  – Request Metadata server to allocate part ids assigned to destination stream
- To write to a given part id, the
  – Request write path for that part id
  – Write data, using native interface
  – Close file, supply size and fingerprint to server
    - Data becomes visible to readers

Write Replication

- Default is lazy replication
- Client can request multiple write paths
  - Write data to each path provides fault tolerance
- Client library also provides byte-oriented interface
  - Used for data ingress/egress
  - Will optionally perform eager replication

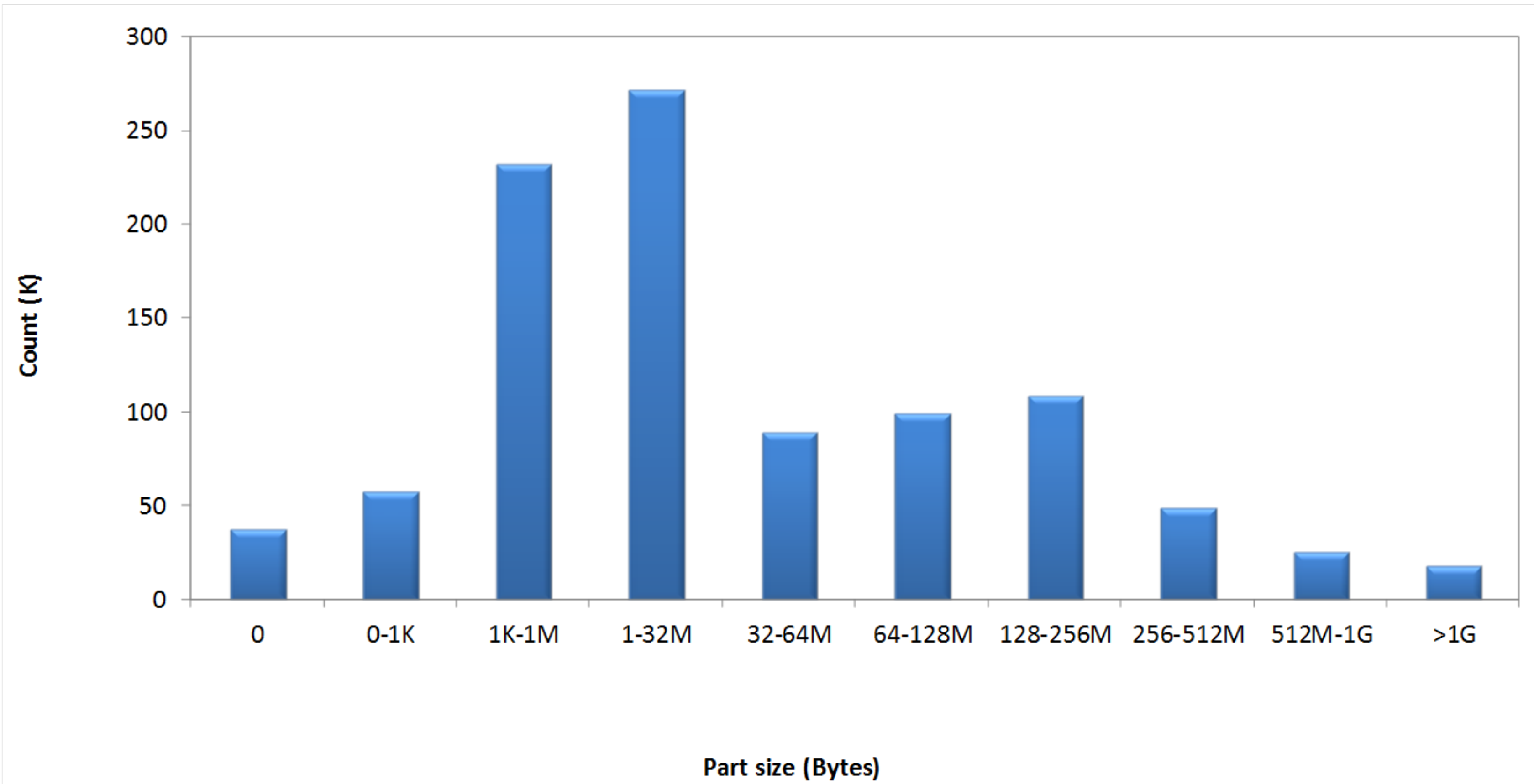
Design Points – Direct Access to Parts

- Enables application choice of I/O pattern
- Avoids extra layer of indirection
- Simplifies legacy applications
- Enables use of native ACL mechanisms
- Fine grained control over part sizes

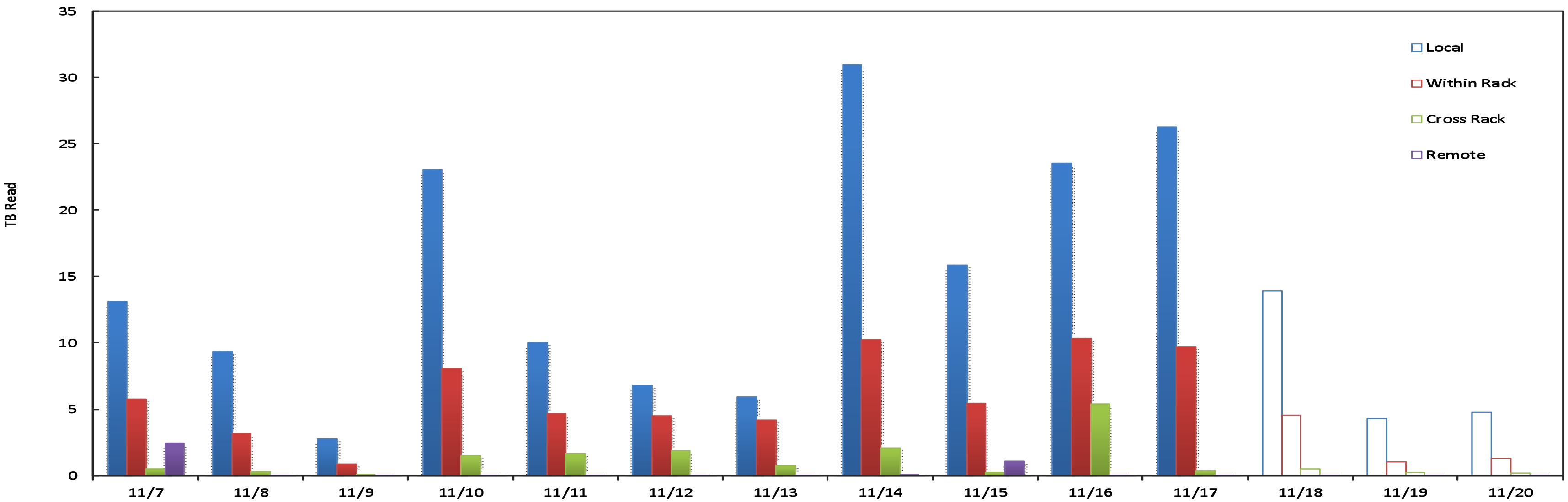
Operational Experiences

- 18 month deployment and active use
- 256 node research cluster
  - Exclusively for programs run using DryadLINQ
  - DryadLINQ programs are executed by Dryad
  - Dryad is a dataflow execution engine
    - Dryad uses TidyFS for input and output
  - Dryad processes are scheduled by Quincy
    - Attempts to maintain data-locality and fair sharing

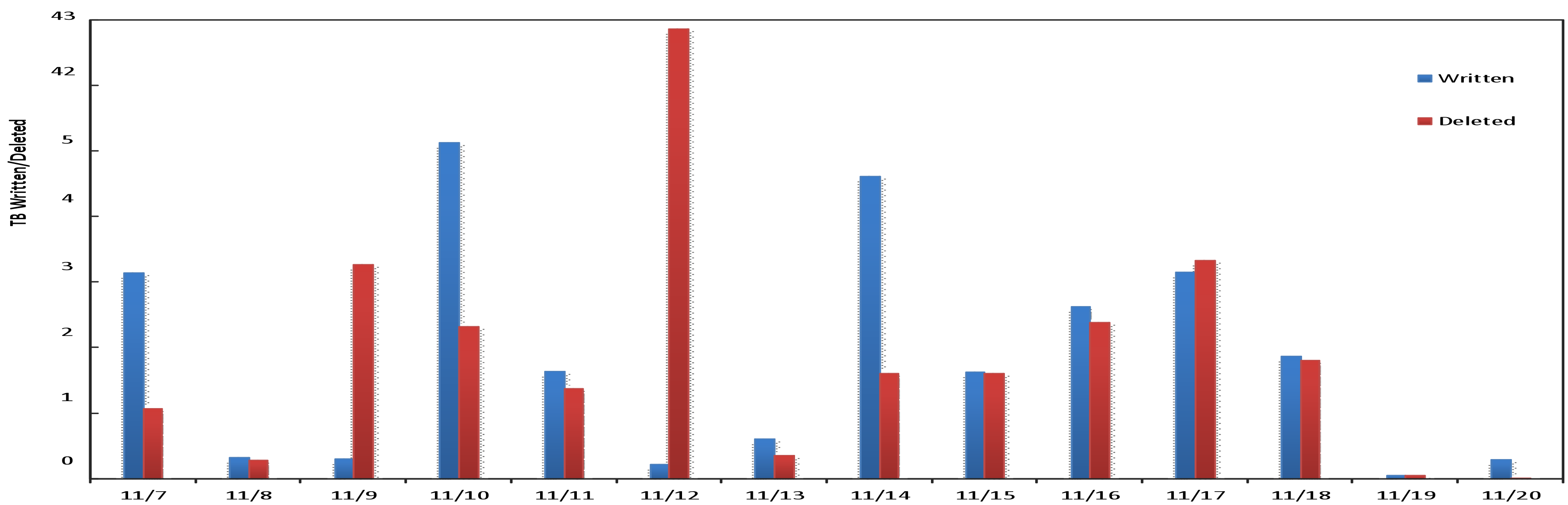
Distribution of Part Sizes



Data Volume

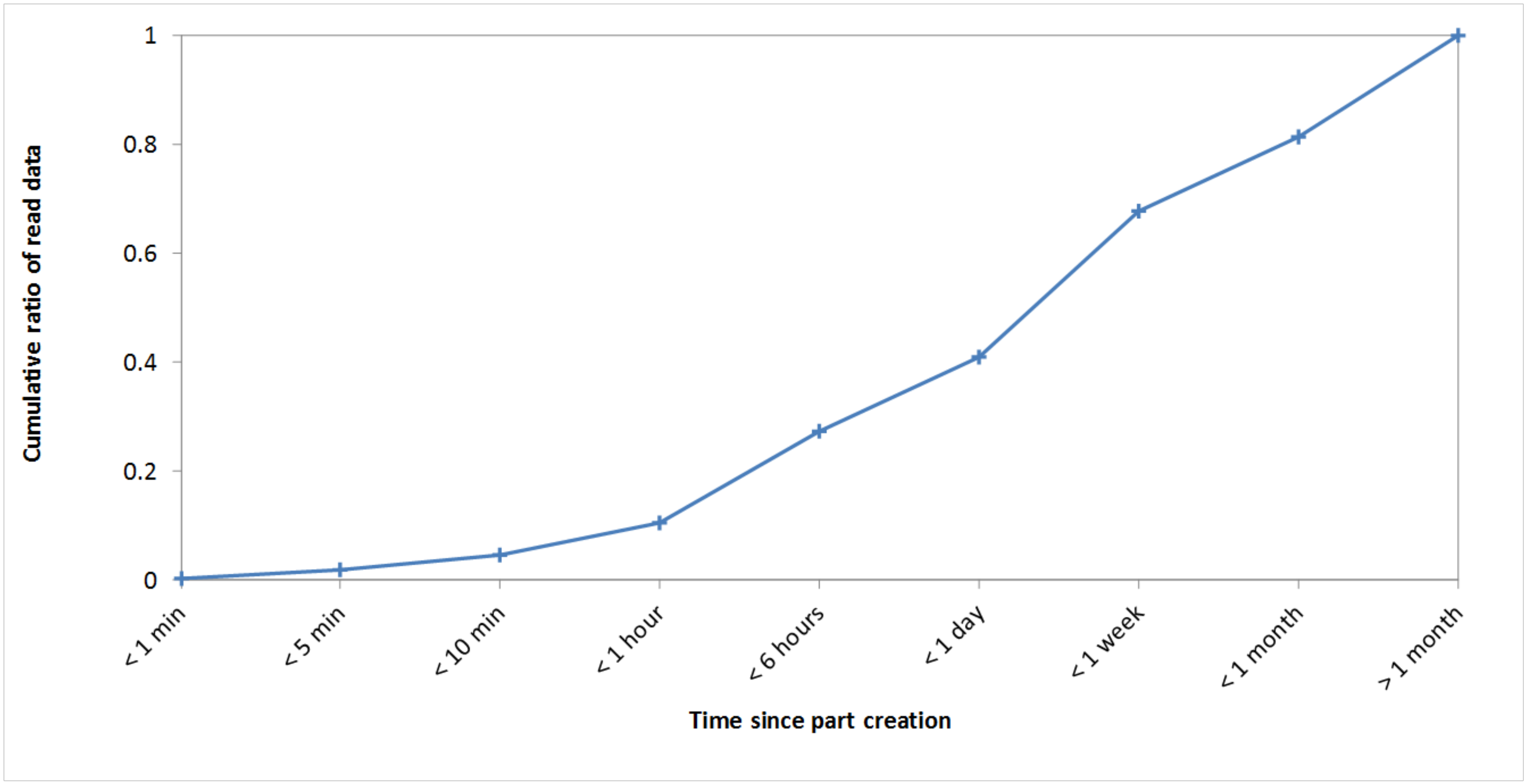


TB Read
35
30
25
20
15
10
5
0
11/7
11/8
11/9
11/10
11/11
11/12
11/13
11/14
11/15
11/16
11/17
11/18
11/19
11/20
Local
Within Rack
Cross Rack
Remote

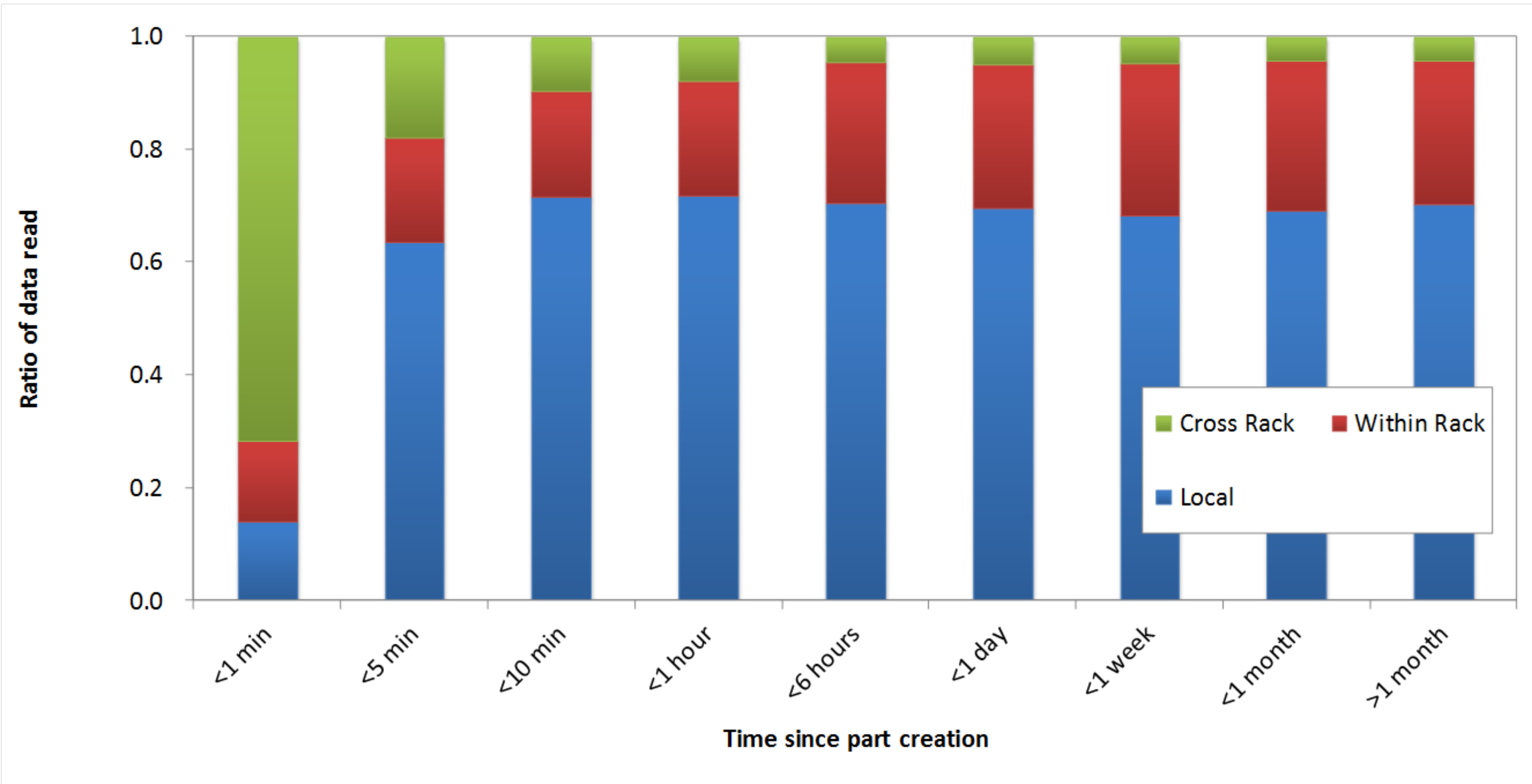


TB Written/Deleted
43
42
5
4
3
2
1
0
11/7
11/8
11/9
11/10
11/11
11/12
11/13
11/14
11/15
11/16
11/17
11/18
11/19
11/20
Written
Deleted

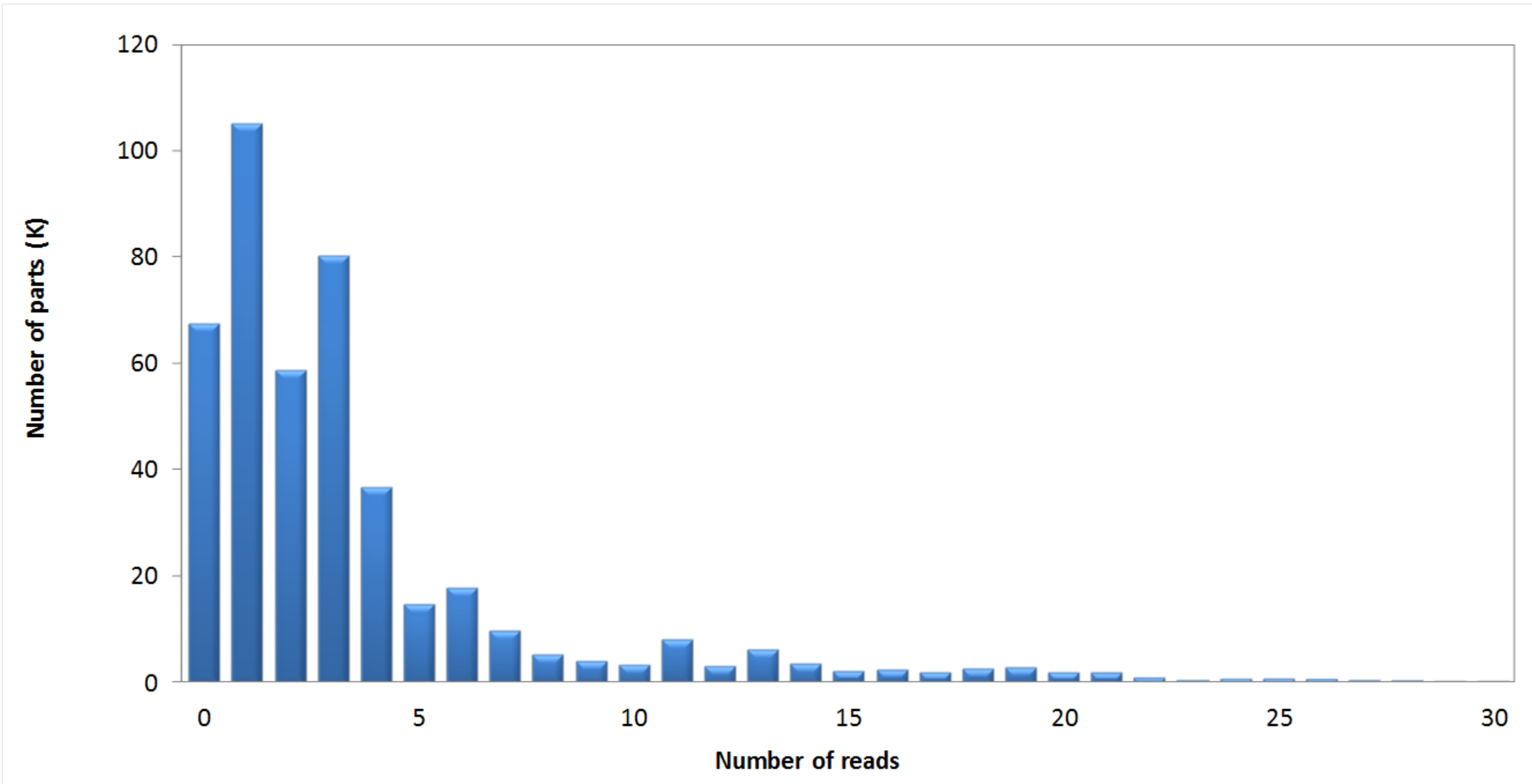
Access Patterns



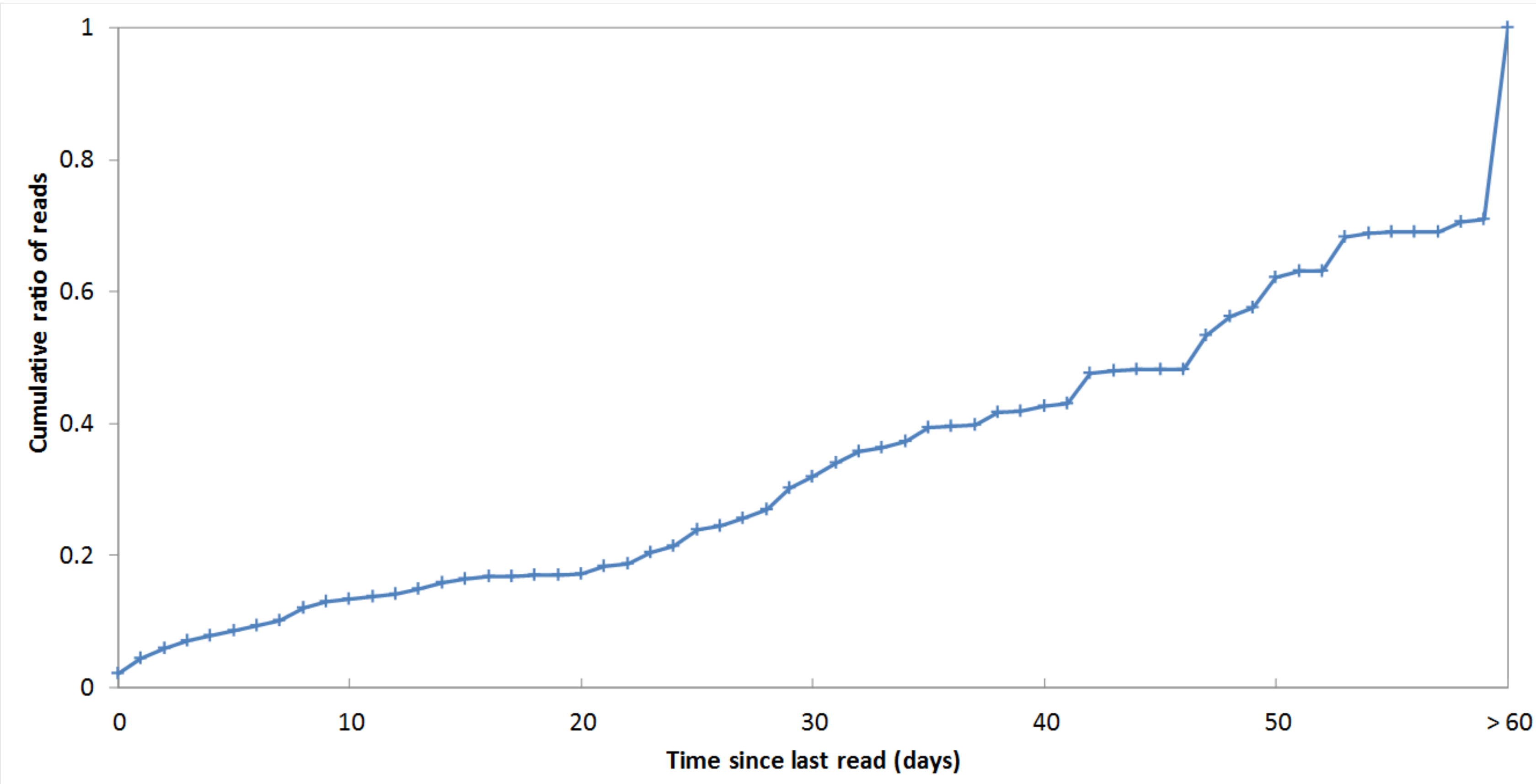
Data Locality



Part Read Histogram



Distribution of Last Read Time



Evaluation of Lazy replication

- Part replication times over a 3 month window

Mean time to replication (s)	Percent
0 – 30	6.7 %
30 – 60	62.9 %
60 – 120	14.6 %
120 – 300	1.1 %
300 – 600	2.2 %
600 – 1200	4.5 %
1200 – 3600	3.4 %
3600 -	4.5%

Conclusion

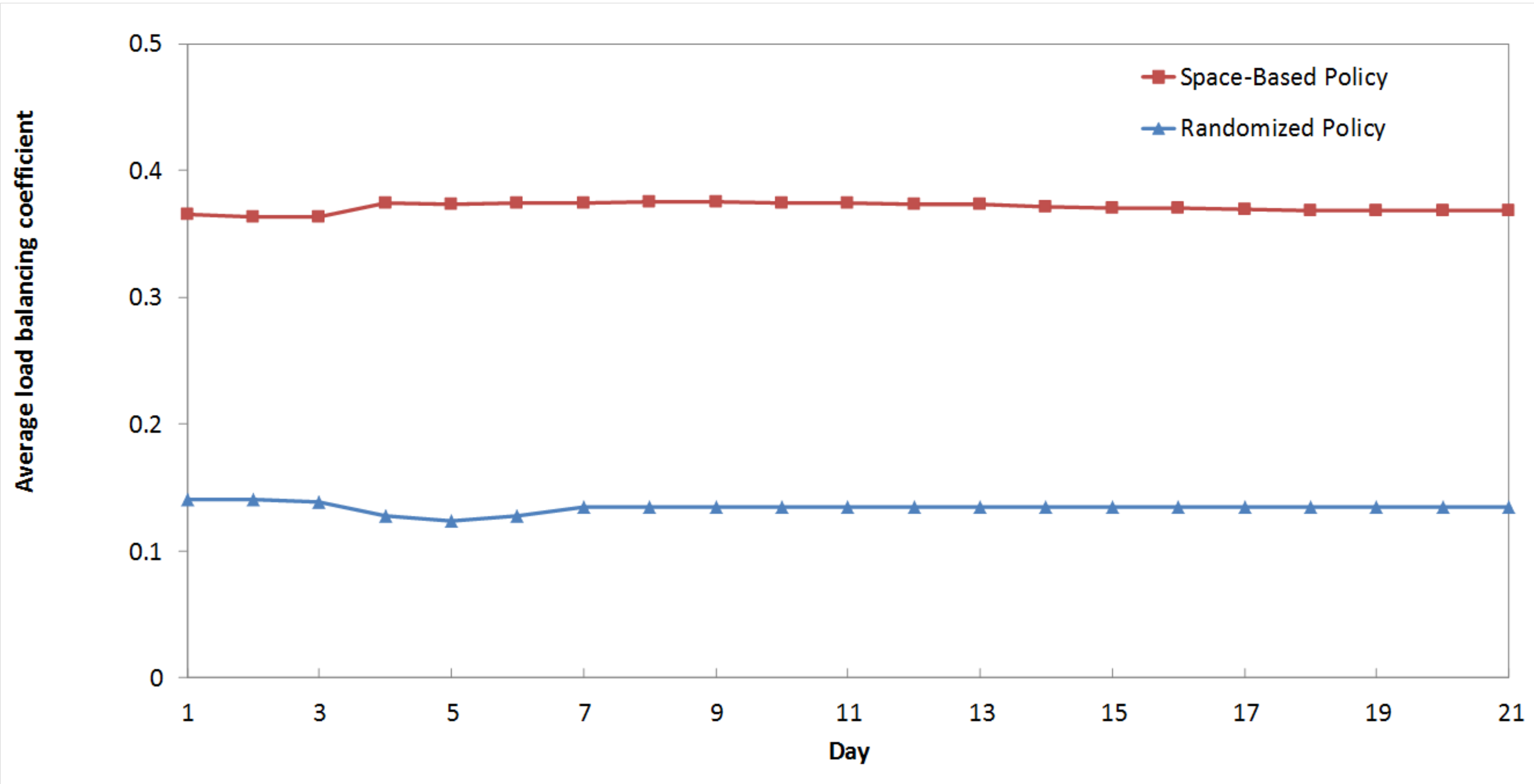
- Design tradeoffs have worked well
  - Pleased with simplicity and performance
- TidyFS gives clients direct access to part data
  - Performance
  - Easy to add support for additional part types such as SQL databases
  - Prevents providing automatic eager replication
  - Lack of control over part sizes
- Considering tighter integration with other cluster services

Backup Slides

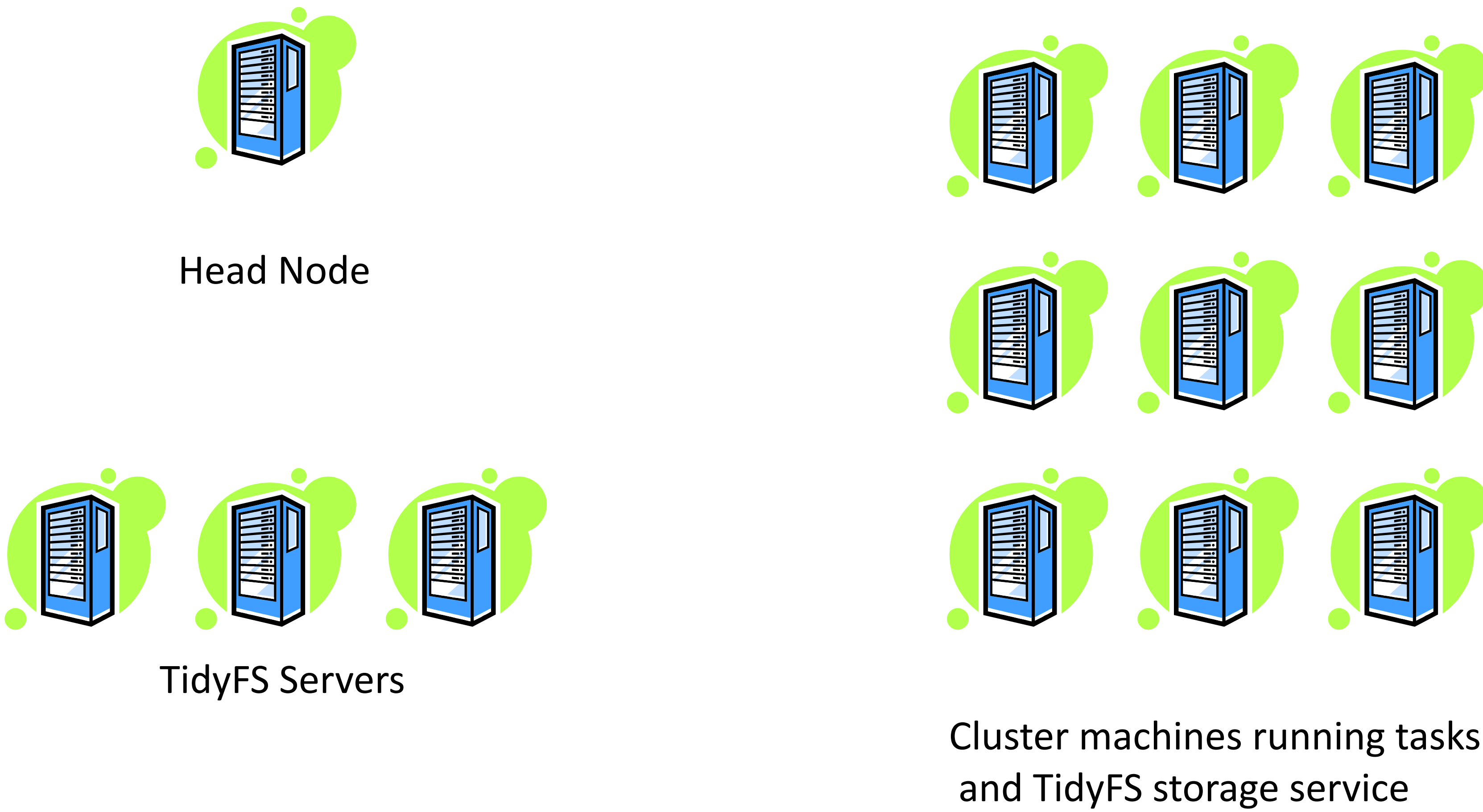
Replica Placement

- Initially had a space based assignment policy
- Stream balance affects performance
- Moved to best of 3 random choices
- Evaluate balance using L^2 norm

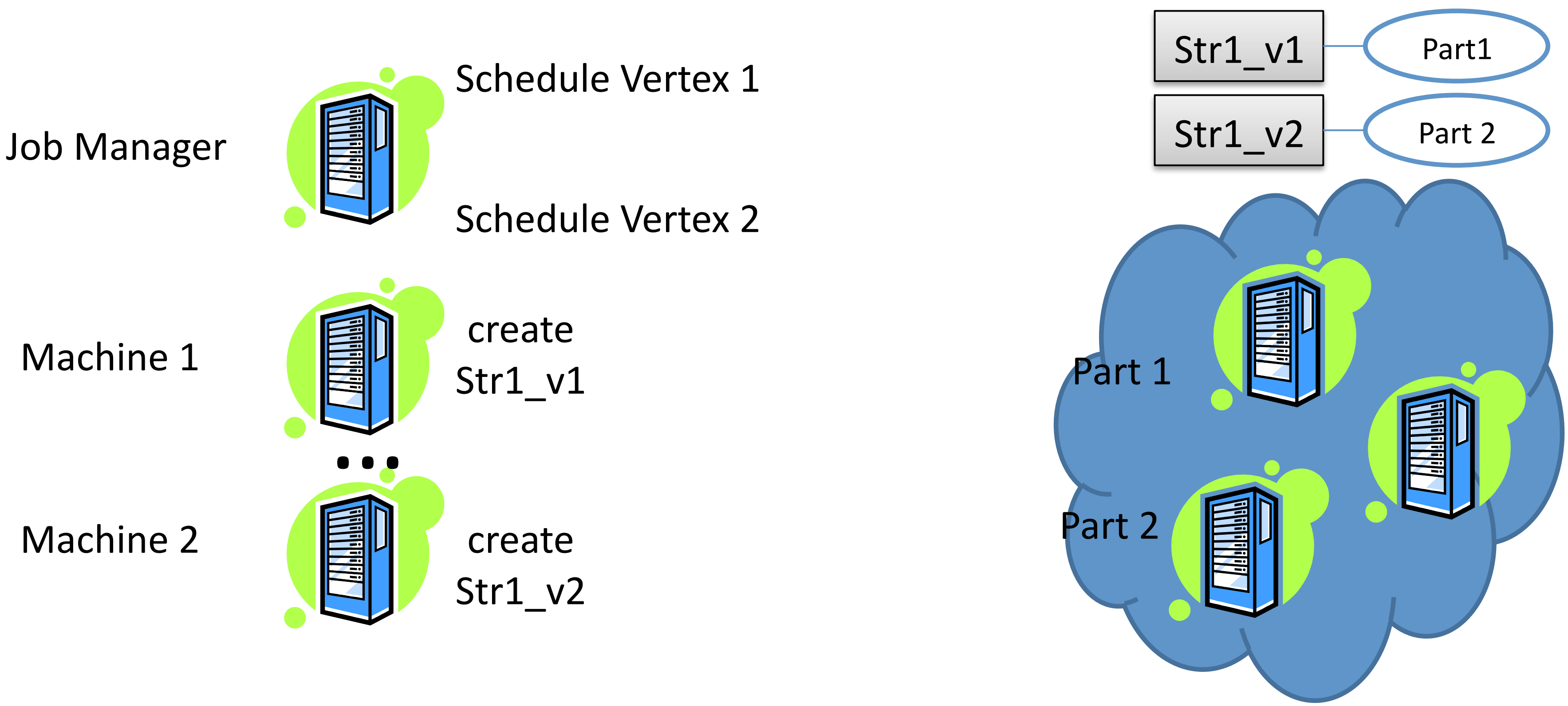
Replica Placement Evaluation



Cluster Configuration



How a Dryad job writes to TidyFS



Cluster Machines

TidyFS Service

How a Dryad job writes to TidyFS