TidyFS: A Simple and Small Distributed Filesystem

Dennis Fetterly¹, Maya Haridasan¹, Michael Isard¹, and Swaminathan Sundararaman²
¹Microsoft Research, Silicon Valley
²University 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
Primary mechanism for users and administrators to interact with TidyFS

- Users can operate on streams – Rename, delete, re-order parts, ...
- Administrators can monitor system state – Healthy nodes, free storage space, ...

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

Job Manager
- Schedule Vertex Part 1
- List Parts in Stream
- Schedule Vertex Part 2

Machine 1
- Get Read Path Machine 1, Part 1

Machine 2
- Get Read Path Machine 2, Part 2

Cluster Machines

TidyFS Service

- Part 1, Machine 1
- Part 2, Machine 2
- D:\tidyfs\0001.data
- D:\tidyfs\0002.data
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

![Data Volume Chart]

- **TB Read**
- **Local**
- **Within Rack**
- **Cross Rack**
- **Remote**

![Data Volume Chart (TB Written/Deleted)]

- **TB Written/Deleted**
- **Written**
- **Deleted**

**Dates:**
- 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
Access Patterns

![Graph showing cumulative ratio of read data over time since part creation. The x-axis represents time since part creation in increments of less than 1 minute, 5 minutes, 10 minutes, 1 hour, 6 hours, 1 day, 1 week, 1 month, and greater than 1 month. The y-axis represents the cumulative ratio of read data, ranging from 0 to 1.]
Data Locality

The graph shows the ratio of data read over time since part creation for different timescales. The data is divided into three categories: Cross Rack, Within Rack, and Local. The time intervals are as follows:

- <1 min
- <5 min
- <10 min
- <1 hour
- <6 hours
- <1 day
- <1 week
- <1 month
- >1 month

The graph indicates that a significant portion of data reads are within the same rack, especially close to the time of part creation.
Part Read Histogram
Distribution of Last Read Time
Evaluation of Lazy replication

- Part replication times over a 3 month window

<table>
<thead>
<tr>
<th>Mean time to replication (s)</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 30</td>
<td>6.7 %</td>
</tr>
<tr>
<td>30 – 60</td>
<td>62.9 %</td>
</tr>
<tr>
<td>60 – 120</td>
<td>14.6 %</td>
</tr>
<tr>
<td>120 – 300</td>
<td>1.1 %</td>
</tr>
<tr>
<td>300 – 600</td>
<td>2.2 %</td>
</tr>
<tr>
<td>600 – 1200</td>
<td>4.5 %</td>
</tr>
<tr>
<td>1200 – 3600</td>
<td>3.4 %</td>
</tr>
<tr>
<td>3600 -</td>
<td>4.5%</td>
</tr>
</tbody>
</table>
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

![Graph showing average load balancing coefficient over days for Space-Based Policy and Randomized Policy. The Space-Based Policy maintains a higher average load balancing coefficient compared to the Randomized Policy throughout the days.](image)
Cluster Configuration

Head Node

TidyFS Servers

Cluster machines running tasks and TidyFS storage service
How a Dryad job writes to TidyFS

Cluster Machines

Job Manager

Machine 1

Machine 2

Schedule Vertex 1

Schedule Vertex 2

create Str1_v1

create Str1_v2

TidyFS Service

Str1_v1

Str1_v2

Part 1

Part 2

Part 1

Part 2
How a Dryad job writes to TidyFS

1. Job Manager
   - Create Str1
   - Create Str1_v1
   - Create Str1_v2
   - Concatenate Streams (str1, str1_v1, str1_v2)

2. Machine 1
   - GetWritePath (Part 1, Machine 1, Completed)
   - AddPartitionInfo (Part 1, Machine 1, Size, Fingerprint, ...)

3. Machine 2
   - GetWritePath (Part 2, Machine 2, Completed)
   - AddPartitionInfo (Part 2, Machine 2, Size, Fingerprint, ...)

4. Cluster Machines

5. TidyFS Service
   - Str1
   - Str1_v1
   - Str1_v2
   - Part 1
   - Part 2

- D:\tidyfs\0001.data
- D:\tidyfs\0002.data