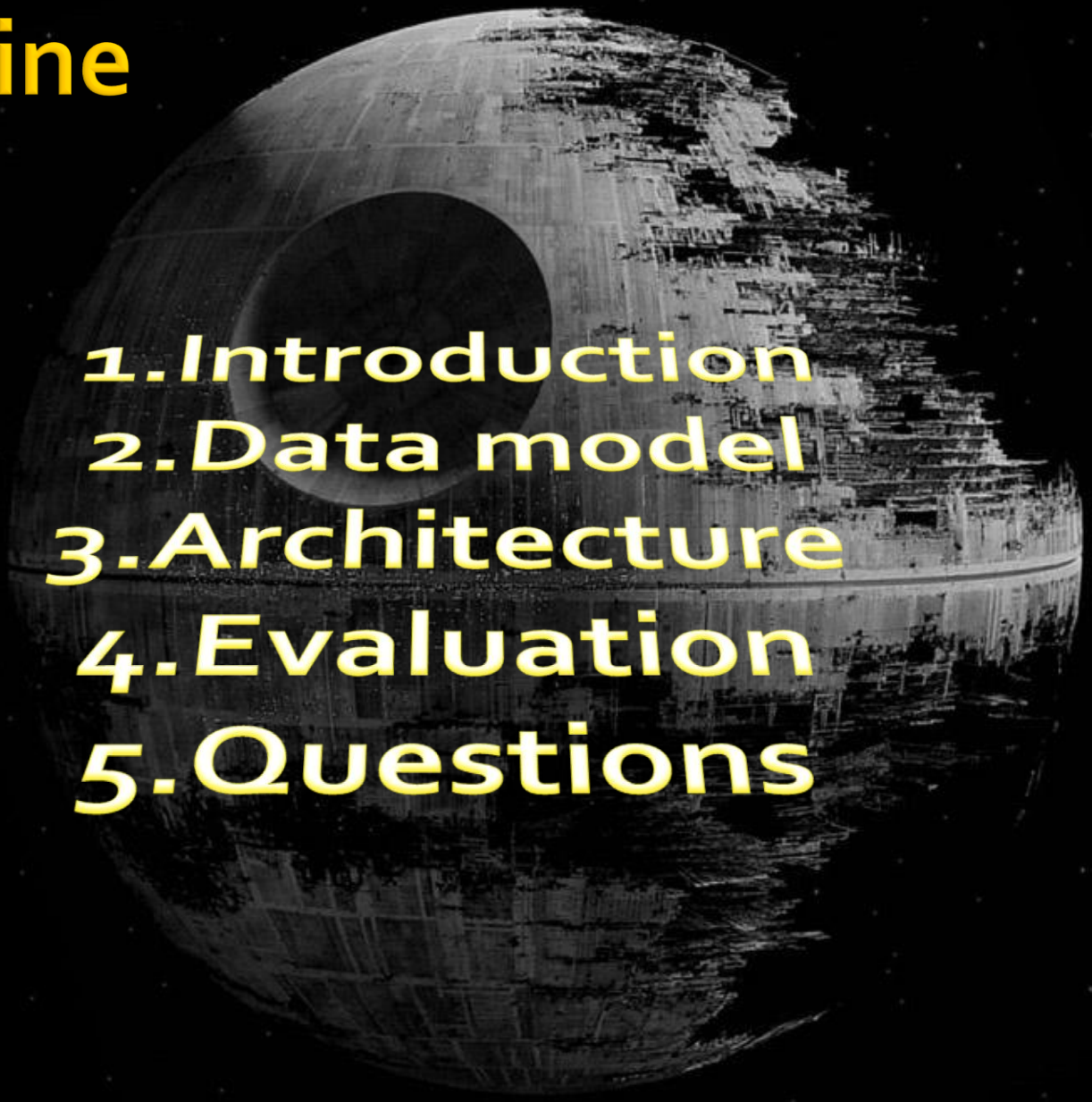
A Simple and Small Distributed File System

TidyFS

Based on article '*TidyFS: A Simple and Small Distriburted File System*' by Dennis Fetterly, Maya Haridasan, Michael Isard, Swaminathan Sundararaman.

Outline



1. Introduction
2. Data model
3. Architecture
4. Evaluation
5. Questions

Development Guidelines

1. Parallel computations on clusters
2. Shared nothing commodity computers
3. High-throughput
4. Sequential access
5. Read-mostly
6. Fault-tolerance
7. Simplicity

Main competitors:
GFS and HDFS

***Source**: http://pl.wikipedia.org/w/index.php?title=Plik:Us-nasa-columbia.jpg&filetimestamp=20050116090033*

Distinguishing features

1. Writes are invisible to readers until commited.
2. Data are immutable.
3. Replication is lazy.
4. Relying on the end-to-end fault tolerance of the computing platform.
5. Using native IO.
6. Strongly connected with DryadLINQ system (parallelizing compiler for .NET) and Quincy (cluster-wide scheduler).

Data and metadata

Data

- Stored on the compute nodes (distribution)
- Immutable
- FS does replication.

Metadata

- Stored on dedicated machines (centralisation)
- Mutable
- Servers should be replicated.

Source: *http://niels85.wordpress.com/2011/03/24/review-1982-blade-runner-top-250-at-imdb/*

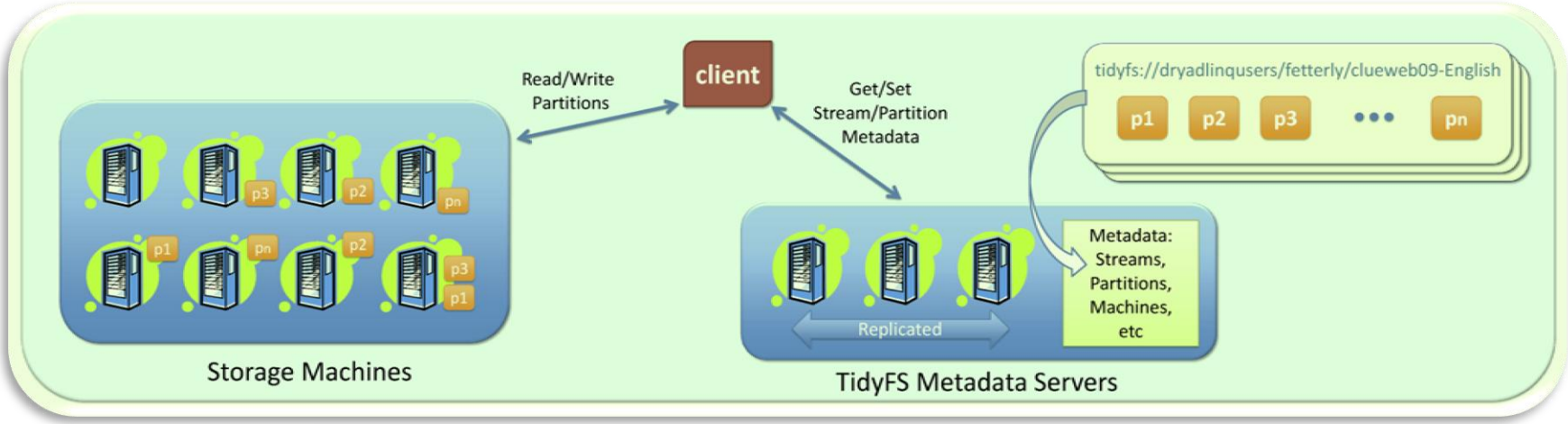
Data abstraction

Streams and parts

- Data are stored in abstract *streams*.
- A stream is a sequence of *parts*.
- Part is atomic unit of data.

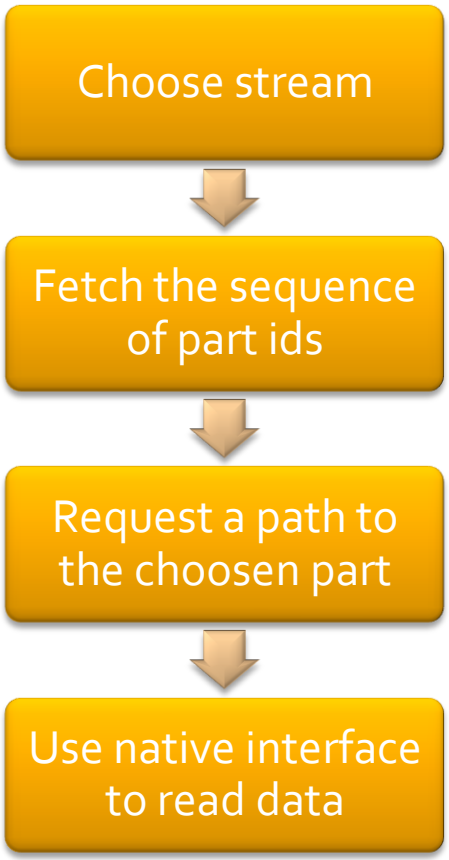
- Each part is replicated on multiple cluster computers.
- Part can be a member of multiple streams.
  - Streams can be modificated, parts are immutable.
- Part may be:
  - Single file.
  - Colection of files of more complex type (SQL databases).
- Streams has (possibly infinite) lease time.
- Streams are decorated with extensible metadata.
- Streams and parts are fingerprinted.

Fine picture

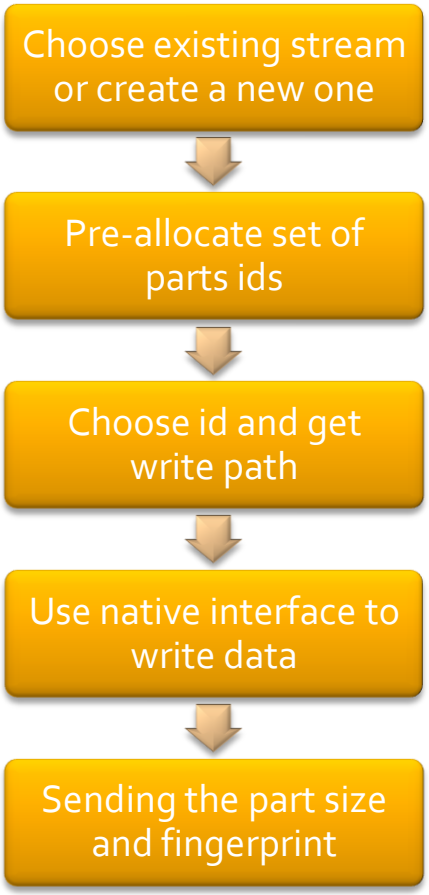


Access patterns

Read



Write



Remarks

- Typically we write on the local hard drive.
- Optionally we can simultaneously write multiple replicas.

Available native interfaces: NTFS, SQL Server, (CIFS).

Native interface usage

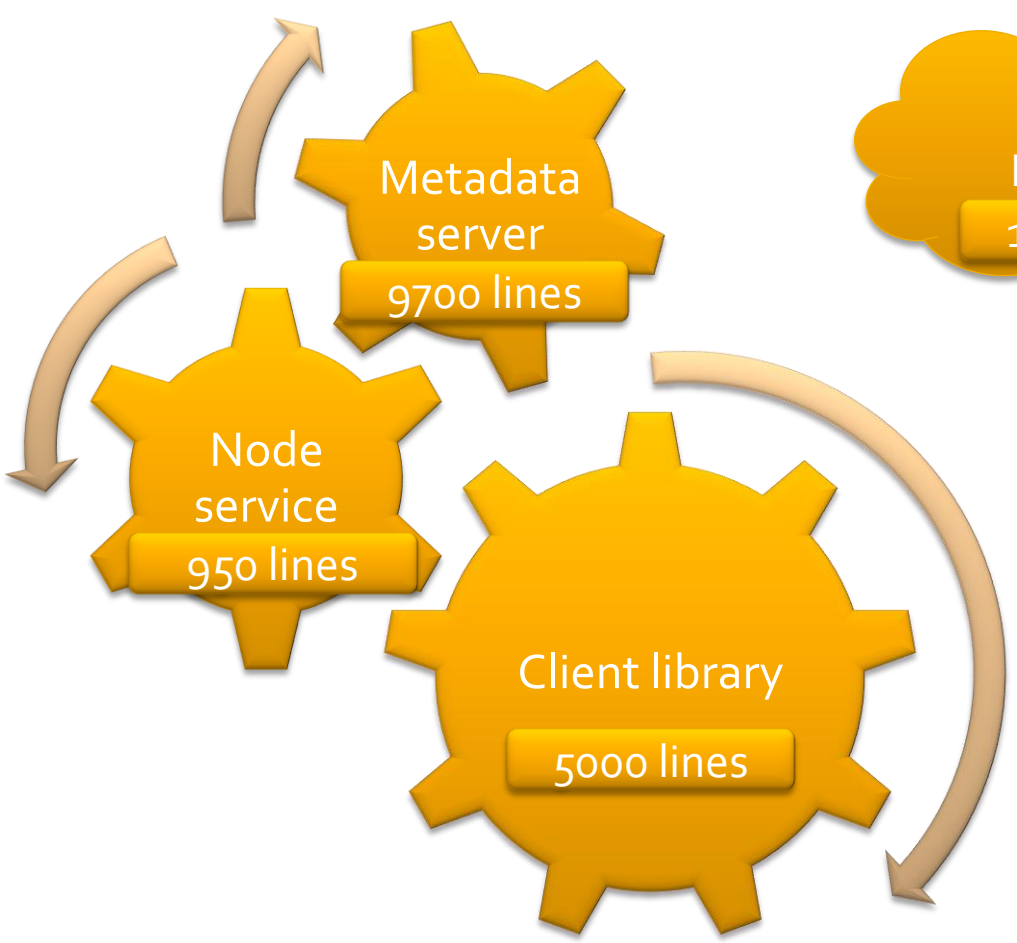
PROS

- Allows applications to choose the most suitable parts access patterns.
- Avoids extra indirection layer.
- Allows to use native access-control mechanisms (ACLs).
- Simplicity and performance.
- Gives clients precise control over the size and contents.

CONS

- Loss of control over parts access patterns.
- Loss of generality.
- Lack of automatic eager replication.
- Some parts can be much bigger than other ones.
  - Problems with replication and rebalancing.
  - Sometimes a defragmentation is needed.

Architecture



Source: *http://the-moviebuff.blogspot.com/2011/07/winnie-pooh-updating-classic.html*

Metadata server

- Stores and tracks:
  - Parts, streams, names and id's mappings.
  - Per-stream replication factor.
  - Locations of each replica.
  - State of the each computer:
    - ReadWrite
    - ReadOnly
    - Distress
    - Unavailable
- Replicated component.
  - Uses Paxos algorithm for synchronization.

Source: *http://moviesandsongs365.blogspot.com/2011/05/movie-of-week-2001-space-odyssey-1968.html*

Node service

- Periodically performs maintanance actions:
  - Reporting the amount of free space.
  - Garbage collection.
  - Part replication.
  - Part validation.
    - Checking againts latent sector errors.
- Runs periodically (each 60 seconds).
- Gets from metadata server two list:
  A. List of parts that the server believes should be stored on the computer.
  B. The list of parts that should be replicated onto the computer but have not yet been copied.

List A (existing parts) handling.

- The list contains the parts that should be already stored.
- Two kinds of inconsistency:
  A. We do not have expected part -> error
     1. Create new replicas.
  B. We have unexpcted parts -> prepare for deletion
     1. Send the list of parts to be deleted.
     2. Delete confirmed parts.
        - Metadata server is aware of parts currently written but not yet commited.

List B (part replication) handling.

- List consists of the parts that should be replicated on the computer.

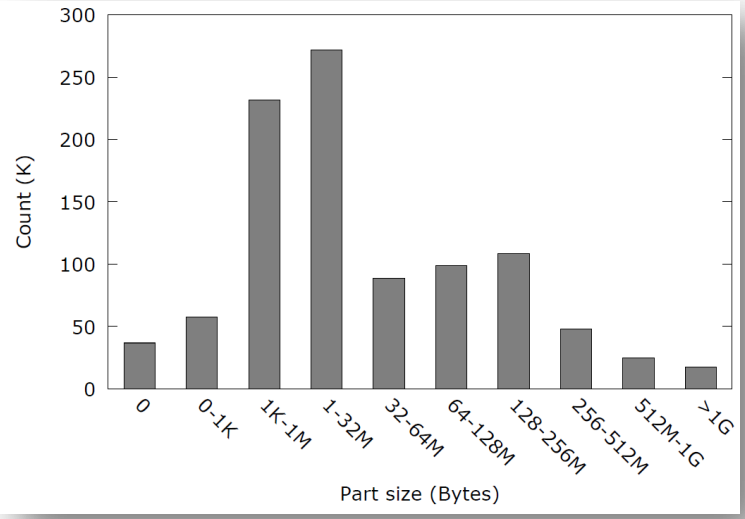
1. Obtain paths to the parts.
2. Download parts.
3. Validate fingerprint.
4. Acknowledge the parts existence.

Replica placement problem

- Aims:
  1. Spread replicas across the available computers.
     - It enables more local reads.
       - TidyFS is aware of network topology.
       - First write if a part is always on the local hard drive.
       - Depending on the computional framework's fault-tolerance.
  2. Storage space usage should be balanced across the computers.

Replica placement algorithms

A. Always choose the computer with most free space.
   - Can result in poor balance.
B. Choose three random computers, and then selects the one with most free space.
   - Acceptable balance (more than 2 times better than for A).



Histogram of part sizes (in MB).

Evaluation environment

- Research cluster with 256 servers.
- Real large-scale data-intensive computations.
- DryadLINQ and Quincy.
  - Processes are being scheduled close to at least one replica of their input parts.
- Operating for a one year.

Lazy replication evaluation

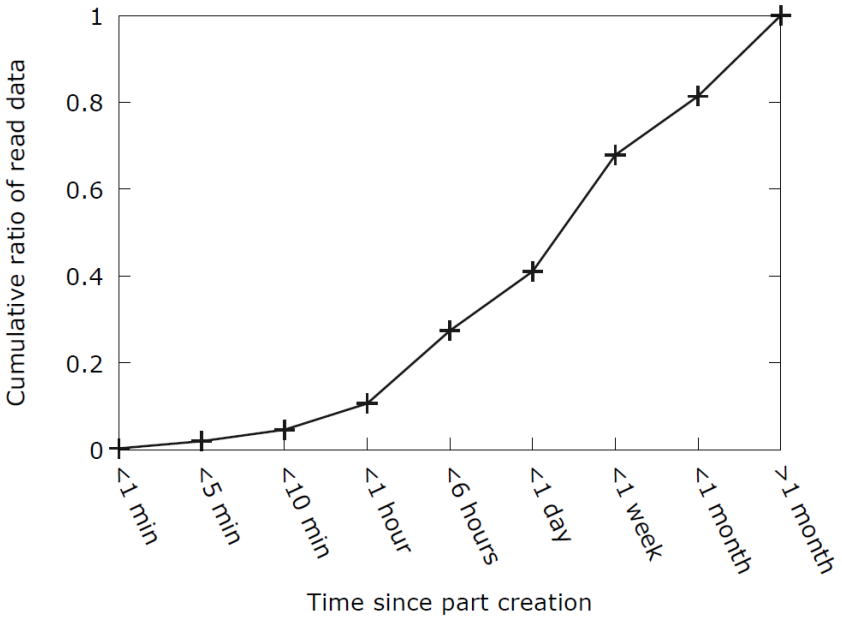
- „We find, that lazy replication provides acceptable performance for clusters of a few hundred computers.”
- One unrecoverable computer failure per month, no data loss.

Mean time to replication

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%

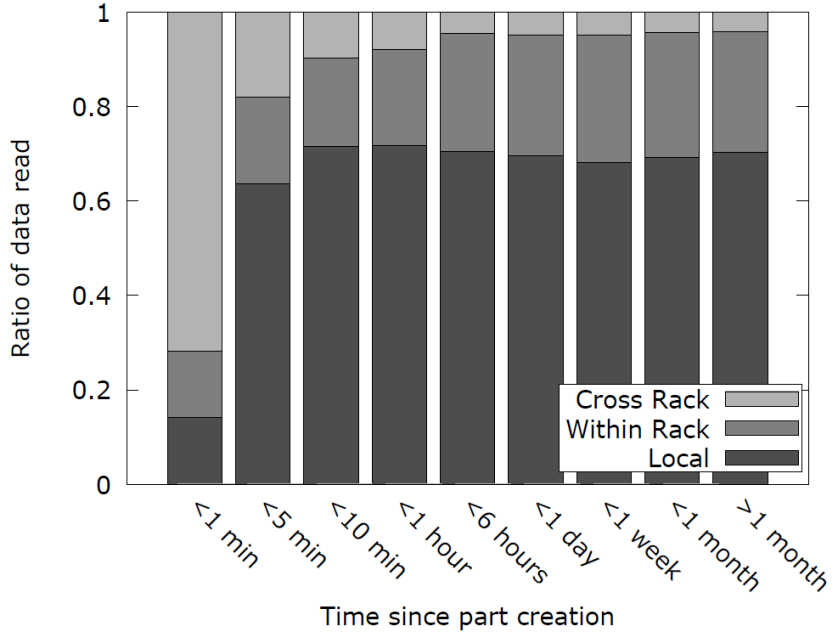
Performace

READ AGE



Cumulative distribution of read ages.

READ TYPE

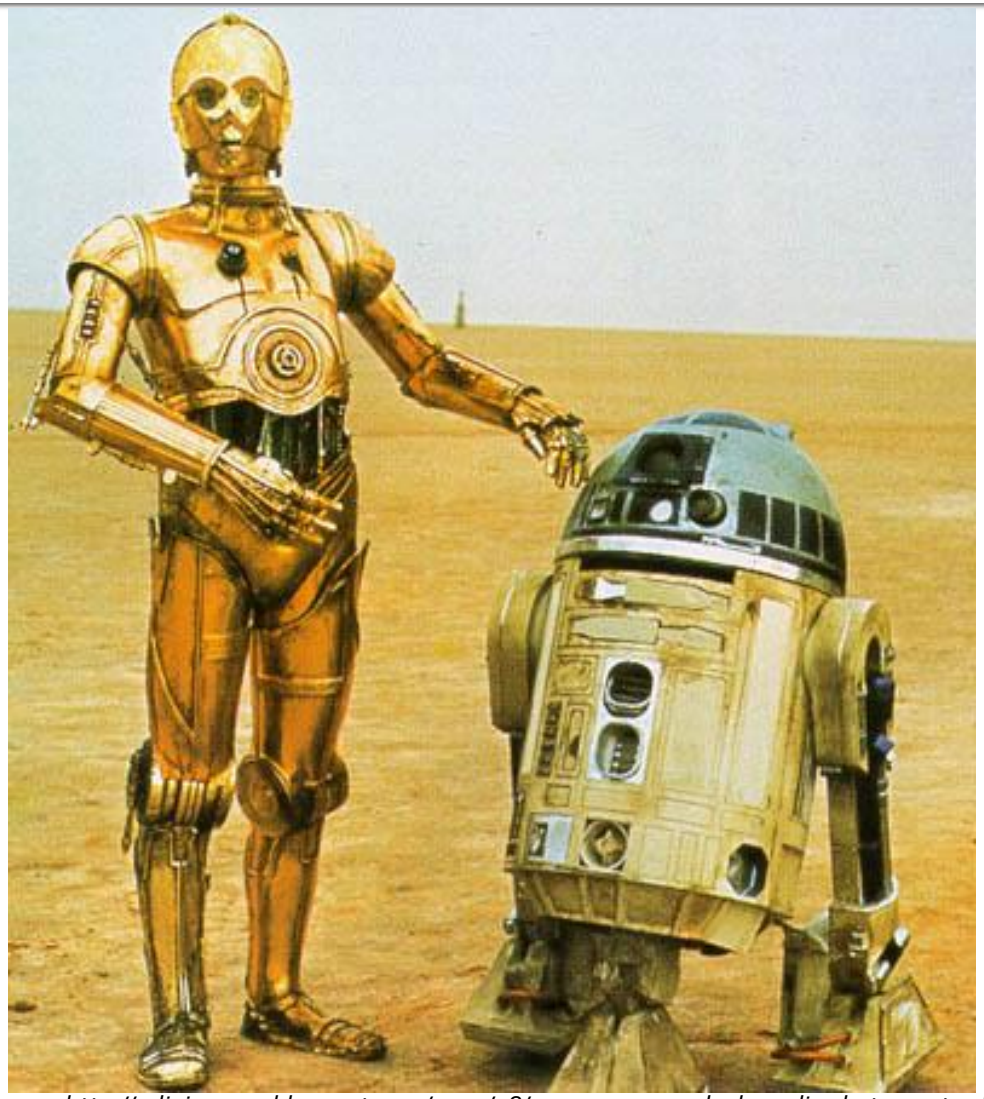


Proportion of local, within rack and cross rack data read grouped by age of reads.

Summary

1. Direct access to part data using native interfaces.
2. Support for multiple part types.
3. Not general – tightly integrated with Microsoft's cluster engine.
4. Leveraging the client's existing fault-tolerance.
5. Clients has precise knowledge about parts sizes.
6. Sometimes defragmentation is needed.
7. Simplification.
8. Good performance in the target workload.

Questions



Source: *http://religiamocy.blogspot.com/2010/08/moc-w-przewodach-czyli-roboty-w-star.html*