System Overview
HyDFS (Hybrid Distributed File System) is a distributed file system implementing consistent hashing with 3-way replication across a 10-VM cluster. The system provides block-based append-only storage with automatic failure recovery and consistency guarantees.

Key Properties:
- Replication Factor: 3 (tolerates 2 simultaneous failures)
- Cluster Size: 10 VMs (fa25-cs425-a701 through a710)
- Communication: TCP on port 7000 (HyDFS), 9091 (controller)
- Storage: Block-based, append-only architecture

Architecture Components
1. Ring Manager (ring.py)

Key Operations:
- hash_to_ring(key): SHA-1 hash → 160-bit ring position
- get_successors(key, n): Returns n successor nodes for a key
- add_node(node_id) / remove_node(node_id): Dynamic membership

2. Storage Layer (storage.py)

Key Features:
- Each append creates a new immutable block
- Metadata stored in metadata.json per file
- Block data in separate .block files
- File identification via SHA-1 hash (16-char hex)

3. Consistency Manager (consistency.py)

ClientSequence: Assigns monotonically increasing sequence numbers per client
- Ensures per-client append ordering
- Thread-safe sequence generation

ConsistencyManager:
- Pending Writes Tracking: Monitors in-flight writes for read-my-writes
- Block Merging: Combines replicas by sorting on (client_id, sequence_num, timestamp)
- Conflict Detection: Identifies duplicate blocks and sequence gaps
- Ordering Validation: Verifies monotonic sequences per client

MergeCoordinator: Prevents concurrent merges on same file

4. Replication Manager (replication.py)

Operations:

- check_and_rereplicate(): Periodic scan (every 5s) for under-replicated files
- fetch_file_from_replica(): Retrieve missing files from other replicas
- replicate_file(): Push file to target nodes
- handle_node_failure(): Trigger re-replication on failure detection
- handle_node_join(): Rebalance files when nodes join

Re-replication Logic:

```
For each file:
  current_replicas = ring.get_successors(filename, 3)
  if this_node in current_replicas and !has_file:
    fetch_from_other_replicas()
  if this_node not in current_replicas and has_file:
    delete_file()
  if actual_replica_count < 3:
    replicate_to_successors()
```

5. Network Layer (network.py)

Message Format:

- Size-prefixed: 10-byte header with message length
- JSON serialization for metadata
- Handler pattern: message_type → handler_function

Key Methods:

- send_message(): Synchronous request-response
- send_message_async(): Non-blocking with callback
- broadcast_message(): Parallel multi-target send

Message Types:

- CREATE_FILE, APPEND_BLOCK, GET_FILE
- CHECK_FILE, GET_FILE_BLOCKS, REPLICATE_FILE
- MERGE_FILE, LIST_FILES

6. Controller (controller.py)

Features:

- Multi-VM targeting: vm 1,2,5,8 or vm all
- Command routing to selected VMs

Available Commands:

- File ops: create, get, append, merge, ls
- System ops: liststore, list_mem_ids, getfromreplica
- Testing: multiappend for concurrent append demos

Failure Handling

Node Failure Detection

- Integrated with MP2 membership protocol
- Failure notifications trigger replication checks

Automatic Recovery

1. Under-replication Detection: Monitor loop identifies files with < 3 replicas
2. Re-replication: Fetch from surviving replicas, push to new successors
3. File Deletion: Remove files no longer assigned to this node
4. Measurement: Track re-replication time and bandwidth