Concurrent File Transfers in Storage Virtual Nodes

Implementing Threading for Improved Performance

Engr. Daniel Moune

August 6, 2025

Original Implementation

Sequential Chunk Processing

```python
def process_chunk_transfer(self, file_id: str,
                           chunk_id: int,
                           source_node: str) -> bool:
    if file_id not in self.active_transfers:
        return False

    transfer = self.active_transfers[file_id]
    try:
        chunk = next(c for c in transfer.chunks
                     if c.chunk_id == chunk_id)
    except StopIteration:
        return False

    # Sequential processing
    chunk_size_bits = chunk.size * 8
    available_bandwidth = min(
        self.bandwidth - self.network_utilization,
        self.connections.get(source_node, 0)

    transfer_time = chunk_size_bits / available_bandwidth
    time.sleep(transfer_time)  # Blocks here

    chunk.status = TransferStatus.COMPLETED
    # ... (rest of sequential updates)
```

Threading Modifications (1/2)

New Thread Class and Lock

```python
class StorageVirtualNode:
    def __init__(self, node_id: str, ...):
        # Existing initialization
        self.lock = threading.Lock()  # Add thread lock
        self.transfer_threads = {}    # Track active threads
```

Thread Worker Function

```python
def _process_chunk_thread(self, file_id: str,
                          chunk_id: int,
                          source_node: str):
    with self.lock:
        if file_id not in self.active_transfers:
            return
        transfer = self.active_transfers[file_id]
        chunk = next((c for c in transfer.chunks
                      if c.chunk_id == chunk_id), None)

    if not chunk:
        return

    # Simulate transfer (outside lock)
    time.sleep(calculate_transfer_time(chunk))

    with self.lock:
        self._update_chunk_status(chunk, file_id)
```

Threading Modifications (2/2)

New Process Chunk Transfer

```python
def process_chunk_transfer(self, file_id: str,
                           chunk_id: int,
                           source_node: str) -> bool:
    if file_id not in self.active_transfers:
        return False

    # Create and start thread
    thread = threading.Thread(
        target=self._process_chunk_thread,
        args=(file_id, chunk_id, source_node),
        daemon=True
    )
    thread.start()

    # Track thread (protected by lock)
    with self.lock:
        thread_key = f"{file_id}_{chunk_id}"
        self.transfer_threads[thread_key] = thread

    return True
```

Status Update Helper

```python
def _update_chunk_status(self, chunk, file_id):
    chunk.status = TransferStatus.COMPLETED
    self.total_data_transferred += chunk.size
```

Thread Safety Implementation

Critical Sections

```
with self.lock:
    # Access shared resources
    self.network_utilization += bw
    self.used_storage += size
```

Metrics Collection

```
def get_performance_metrics(self):
    with self.lock:
        return {
            'transfers': len(self.
     transfer_threads),
            'data': self.
     total_data_transferred,
            'failed': self.failed_transfers
        }
```

Connection Management

```
def add_connection(self, node_id, bw):
    with self.lock:
        self.connections[node_id] = bw
```

Cleanup

```
def _complete_transfer(self, file_id):
    with self.lock:
        transfer = self.active_transfers.pop
     (file_id)
        self.stored_files[file_id] =
     transfer
        # Cleanup completed threads
```

Code Changes Summary

- **Added threading infrastructure**:
  - Thread creation and management
  - Worker function for chunk processing
- **Implemented thread safety**:
  - Lock protection for shared resources
  - Atomic operations design
- **Modified transfer workflow**:
  - Non-blocking chunk processing
  - Parallel transfer execution
  - Clean completion handling

`git diff` shows +142 -23 lines changed