MangoDB Technical Report

A detailed technical analysis of our document-based database system implementation.

Table of Contents

System Architecture Overview

MangoDB is built as a layered architecture:

```
┌─────────────────────┐
│   Web Interface     │  Flask-based UI
├─────────────────────┤
│   Query Parser      │  MongoDB-like syntax
├─────────────────────┤
│ Transaction Mgr     │  ACID & Concurrency
├─────────────────────┤
│   Storage Layer     │  JSON files & Indexes
└─────────────────────┘
```

Core Components

1. Document Storage

Concept: Document-based storage allows flexible schema and nested data structures.

Implementation:

```
class DocumentDB:
    def __init__(self):
        self.databases_dir = "databases/"
        self._ensure_databases_dir()
```

Each document is stored as a JSON object with a unique UUID:

```
doc_id = str(uuid.uuid4())
document = {
    "_id": doc_id,
    "data": {...}
}
```

2. Transaction Management

ACID Properties Implementation

Atomicity:

- Each transaction is wrapped in a try-except block
- Operations are logged before execution
- Two-phase commit protocol:
    1. Prepare phase: Log operations
    2. Commit phase: Apply changes

```
def execute_transaction(self, transaction_id):
    try:
        # Log operation
        self.log_operation(transaction_id, operation)
        # Execute operation
        result = self.execute_operation(operation)
        # Commit if successful
        self.commit_transaction(transaction_id)
        return result
    except Exception:
        self.abort_transaction(transaction_id)
        raise
```

Consistency:

- Document validation before writes
- Schema enforcement through JSON structure
- Referential integrity checks

Isolation Levels:

```
class IsolationLevel(Enum):
    READ_UNCOMMITTED = 1
    READ_COMMITTED = 2
    REPEATABLE_READ = 3
    SERIALIZABLE = 4
```

Implementation in `transaction_manager.py`:

```
def begin_transaction(self, isolation_level):
    transaction_id = str(uuid.uuid4())
    self.transactions[transaction_id] = {
        "state": TransactionState.ACTIVE,
        "isolation_level": isolation_level,
        "locks": set()
    }
    return transaction_id
```

Durability:

- Transaction logging
- Periodic checkpointing
- Crash recovery mechanism

Concurrency Control

1. Locking Mechanism

Concept: Document-level locking prevents concurrent modifications to the same document.

Implementation Flow:

```
Transaction Start
       ↓
Request Lock
       ↓
Check Lock Table
       ↓
Grant/Block Lock
       ↓
Operation Execution
       ↓
Release Lock
```

Lock Types:

```
class LockType(Enum):
    READ = 1
    WRITE = 2
```

Lock Manager Implementation:

```
class LockManager:
    def __init__(self):
        self.lock_table = {}  # doc_id -> {lock_type, transaction_id}

    def acquire_lock(self, doc_id, lock_type, transaction_id):
        if self._is_lock_compatible(doc_id, lock_type):
            self.lock_table[doc_id] = {
                "type": lock_type,
                "transaction_id": transaction_id
            }
            return True
        return False
```

2. Deadlock Detection

Implementation:

- Timeout-based approach
- Lock wait graph construction
- Cycle detection

```
def detect_deadlock(self, transaction_id):
    wait_graph = self._build_wait_graph()
    if self._has_cycle(wait_graph):
        self._resolve_deadlock(transaction_id)
```

Data Storage and Persistence

1. File Structure

```
databases/
├── db_name/
│   ├── collection.json
│   └── indexes/
│       └── field_name.idx
├── transaction_logs/
└── checkpoints/
```

2. Indexing System

B+ Tree Implementation:

```
class BPlusTree:
    def __init__(self):
        self.root = None
        self.order = 4  # Maximum children per node

    def insert(self, key, value):
        if not self.root:
            self.root = LeafNode()
        return self._insert_recursive(self.root, key, value)
```

Query Processing

1. Query Pipeline

```
Query String
    ↓
Parser (query_parser.py)
    ↓
Optimizer (queryOptimizer.py)
    ↓
Executor (queryProcessor.py)
    ↓
Result
```

2. Query Execution

```
def execute_query(self, query):
    # Parse query
    operation, collection, params = parse_raw_query(query)

    # Start transaction
    transaction_id = self.begin_transaction()

    try:
        # Execute operation
        result = self._execute_operation(operation, collection, params)
        # Commit
```

```
            self.commit_transaction(transaction_id)
            return result
        except Exception:
            self.abort_transaction(transaction_id)
            raise
```

Recovery System

1. Checkpointing

- Every 60 seconds
- Saves system state
- Maintains last 5 checkpoints

2. Log-Based Recovery

```
def recover_from_crash(self):
    # Load latest checkpoint
    checkpoint = self._load_latest_checkpoint()

    # Replay logs
    logs = self._get_logs_after_checkpoint(checkpoint)
    for log in logs:
        if log["status"] == "committed":
            self._redo_operation(log)
        else:
            self._undo_operation(log)
```

Limitations and Trade-offs

Current Limitations

1. **Scalability**:
    - File-based storage limits concurrent access
    - No distributed architecture
    - Memory constraints for large datasets

2. **Performance**:
    - JSON parsing overhead
    - File I/O bottlenecks
    - Limited index types
3. **Features**:
    - No complex queries
    - Limited aggregation
    - Basic indexing only

Design Decisions and Trade-offs

1. **Document-Level vs Collection-Level Locking**
    - Chose document-level for better concurrency
    - Trade-off: More complex lock management
2. **File-Based vs In-Memory Storage**
    - Chose file-based for durability
    - Trade-off: Slower performance
3. **B+ Tree Indexing**
    - Chose B+ tree for range queries
    - Trade-off: More complex implementation

Future Improvements

1. **Performance**:
    - Implement caching
    - Add more index types
    - Optimize file I/O
2. **Features**:
    - Add complex queries

- Implement aggregation
- Support more data types

3. **Scalability**:

   - Add sharding
   - Implement replication
   - Support distributed architecture