Assignment #2 Report

Timofey Brayko
t.brayko@innopolis.university

April 15, 2025

Methodology

Architecture Design

The search engine indexing system implements:

- **2-Stage MapReduce Pipeline**:
  - Pipeline 1: Term Frequency (TF) calculation with Hadoop
  - Pipeline 2: Document Frequency (DF) aggregation
- **Cassandra Data Model**:
  - `inv_index`: Term-document mappings
  - `doc_stats`: Document lengths
  - `vocab_stats`: Global term frequencies
- **Batch Processing**:
  - Batched Cassandra writes (100 operations/batch)
  - Hadoop-Cassandra integration via Python driver

Implementation Details

Listing 1: Mapper 1 (TF Calculation)

```
def count_tf():
    for content in sys.stdin:
        input_file = os.getenv('mapreduce_map_input_file')
        id = extract_id(input_file)  # Custom logic
        tokens = re.findall(r'\w+', content.lower())
        print(f"DOC_{id}\tLEN\t{len(tokens)}")
        for term, count in Counter(tokens).items():
            print(f"{term}\t{id}\t{count}")
```

Implementation Details (Reducer 2)

Listing 2: Cassandra Batch Reducer (reducer2.py)

```
# Batch flushing logic
def flush_batches(force=False):
    #Executing collected commands and clear batches...

def process_key_data(key, data):
    global batch_counts
    if key.startswith("DOC_"):
        # Handle document metadata
        doc_id = int(key.split("_")[-1])
        for value_type, value in data:
            if value_type == "LEN":
                batch_doc_stats.add(doc_stats_stmt,
                    (doc_id, int(value)))
                batch_counts['doc'] += 1
    else:
        # Handle term statistics
        term = key
        doc_frequency = 0
        term_doc_pairs = []
        # Process DF and TF values
        for value_type, value in data:
            if value_type == "DF":
                doc_frequency += int(value)
            else:
                term_doc_pairs.append(
                    (int(value_type), int(value)))
        # Update vocabulary and inverted index
        if doc_frequency > 0:
            batch_vocab.add(vocab_stats_stmt,
                (term, doc_frequency))
            batch_counts['vocab'] += 1
        for doc_id, tf in term_doc_pairs:
            batch_inv_idx.add(inverted_idx_stmt,
                (term, doc_id, tf))
            batch_counts['inv'] += 1
    flush_batches()

# Main execution flow
if __name__ == "__main__":
    # Cassandra connection setup
    #Here connecting to Cassandra and creating batch processors....

    # Data processing loop
    current_key = None
    key_data = defaultdict(list)
    for line in sys.stdin:
        key, value_type, value = line.strip().split('\t')
        # Key change detection
        if current_key and key != current_key:
            process_key_data(current_key, key_data[current_key])
            del key_data[current_key]
        current_key = key
        key_data[current_key].append((value_type, value))

    # Final flush
    if current_key:
        process_key_data(current_key, key_data[current_key])
    flush_batches(force=True)
```

Demonstration

Execution Guide

1. Start infrastructure:

```
docker-compose up -d
```

2. Enter docker entry:

```
sudo docker exec -it cluster-master bash
```

3. Run indexing pipeline:

```
bash demo_indexing.sh  # Processes 100 docs
```