भारतीय प्रौद्योगिकी संस्थान जोधपुर
Indian Institute of Technology Jodhpur

ASSIGNMENT 2

FUNDAMENTALS OF DISTRIBUTED SYSTEMS

Submitted by

Sqn Ldr Shreyas B Gaikwad

Roll No. G24AI1060

Post Graduate Diploma in Data Engineering

PREREQUISITES AND ENVIRONMENT SETUP

System Overview

The system is structured into the following five key components:

1. **Data Cleansing and Parsing**
   Validates and extracts structured fields from raw input data.
2. **Action Aggregation and Sorting**
   Aggregates user activity and ranks users by post count.
3. **Trending Content Identification**
   Detects high-engagement content using statistical thresholds.
4. **Dataset Joining**
   Merges user activity logs with corresponding profile metadata.
5. **Data Visualization**
   Offers both static and interactive insights from processed data.

System Architecture

The architecture follows a modular pipeline:

- Input is passed through a series of MapReduce jobs
- Intermediate outputs flow into subsequent components
- Final outputs drive visualization and analysis

Prerequisites

For running on Windows with PyCharm:

- Windows OS
- Python 3.6 or higher
- PyCharm (Community/Professional)
- Required Python packages:
  `numpy`, `psutil`, `matplotlib`, `pandas`, `dash`, `plotly`, `scipy`

Project Structure (https://github.com/geek1060/fds_assignment2)

```
fds_assignment2/
├── config.json                    # Configuration file
├── README.md                      # Project documentation
│
├── data/                          # Input datasets
│   ├── social_media_logs.txt       (File ommitted due to size restrictions)
│   └── user_profiles.txt
│
├── output/                        # Generated outputs
│   ├── cleansed_data.txt          (File ommitted due to size restrictions)
│   ├── user_activity.txt
│   ├── trending_content.txt
│   ├── skew_analysis.json
│   ├── joined_data.txt
│   └── workflow_summary.txt
│
├── src/                           # Source code
│   ├── cleansing_mapper.py
│   ├── cleansing_reducer.py
│   ├── action_aggregation_mapper.py
│   ├── action_aggregation_reducer.py
│   ├── trending_content_mapper.py
│   ├── trending_content_combiner.py
│   ├── trending_content_reducer.py
│   ├── join_activity_mapper.py
│   ├── join_profile_mapper.py
│   ├── join_reducer.py
│   ├── visualize_analytics.py
│   ├── skew_detection.py
│   └── memory_monitor.py
│
└── visualizations/                # Output visualizations
```

Quick Start

- **Clone the repository**

- **Install required libraries**

```
pip install numpy psutil matplotlib pandas dash plotly scipy
```

Unzip large files manually

(GitHub restricts uploads >100MB)

```
social_media_logs.zip
cleansed_data.zip
```

Execute the workflow

```
python social_media_analytics_driver.py --config config.json
```

Create static visualizations

```
python visualize_analytics.py --input-dir output --output-dir
visualizations
```

Question 1: MapReduce Workflow for Social Media Analytics

Method for cleansing data and parsing:

1. **Parsing Records:**
   - The `cleansing_mapper.py` script processes each line of the input file (`social_media_logs.txt`).
   - Each record is expected to be tab-separated, containing fields: Timestamp, UserID, ActionType, ContentID, and Metadata (JSON string).

2. **Validation of Fields:**
   - **Timestamp Validation:** A regex pattern checks if the timestamp matches the expected format (`YYYY-MM-DDTHH:MM:SSZ`). If it doesn't match or fails parsing, the record is discarded.
   - **JSON Validation:** The metadata field is validated using Python's `json.loads()`. If it raises a `JSONDecodeError`, the record is discarded.

3. **Filtering Malformed Records:**
   - Records with missing fields (less than 5) are discarded.
   - Records with invalid timestamps or malformed JSON are also discarded.

4. **Tracking Discarded Records:**
   - Hadoop counters (`sys.stderr.write`) are used to track the number of discarded records due to specific issues (e.g., invalid timestamp, malformed JSON, missing fields).

5. **Output Valid Records:**
   - Valid records are passed to the reducer in as-is state.

Implementation of Code:

1. cleansing_mapper.py – Handles validation and filtering

```
#!/usr/bin/envpython3
import sys
importjson
import re
from datetime import datetime

discarded_records = {
    'invalid_timestamp':0,
    'malformed_json':0,
    'missing_fields':0,
    'total_discarded':0
}

timestamp_pattern=re.compile(r'^\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}Z?$')

def validate_timestamp(timestamp):
    if not timestamp_pattern.match(timestamp):
        return False

    try:
        if timestamp.endswith('Z'):
            timestamp = timestamp[:-1]
        datetime.strptime(timestamp,'%Y-%m-%dT%H:%M:%S')
        return True
    except ValueError:
        return False

def validate_json(json_str):
    try:
        json.loads(json_str)
        return True
    except json.JSONDecodeError:
        return False

for line in sys.stdin:
    try:
        fields = line.strip().split('\t')

        if len(fields) <5:
            discarded_records['missing_fields']+=1
            discarded_records['total_discarded']+=1
            continue
```

```
        timestamp,user_id,action_type,content_id,metadata_json= fields[:5]

        if not validate_timestamp(timestamp):
            discarded_records['invalid_timestamp'] +=1
            discarded_records['total_discarded'] +=1
            continue

        if not validate_json(metadata_json):
            discarded_records['malformed_json'] +=1
            discarded_records['total_discarded'] +=1
            continue

print(f"{user_id}\t{timestamp}\t{action_type}\t{content_id}\t{metadata_json
}")


    except Exception as e:
        discarded_records['total_discarded'] += 1
        sys.stderr.write(f"Errorprocessingline:{line.strip()},Error:
{str(e)}\n")

for counter_name,counter_valueindiscarded_records.items():

sys.stderr.write(f"reporter:counter:DataQuality,{counter_name},{counter_val
ue}\n")
```

2. cleansing_reducer.py- Passes through valid records

```
#!/usr/bin/envpython3
import sys

for line in sys.stdin:
    print(line.strip())
```

Aggregation and Sorting

1. **Aggregating Actions:**
   - The `action_aggregation_mapper.py` aggregates counts of actions (posts, likes, comments, shares) per user.
   - A dictionary (`user_action_counts`) stores the counts for each user.

2. **Sorting Output:**
   - The mapper outputs composite keys in the format: `<user_id>,<sort_key>`, where `sort_key` is derived from the number of posts (descending order).
   - The reducer processes these sorted keys and formats the output.

3. **Custom Partitioner and Secondary Sort:**
   - Composite keys ensure that sorting happens by post count first, followed by user ID.
   - No custom partitioner is needed since the default partitioner works well with composite keys.

Implementation:

1. action_aggregation_mapper.py – Counts per user are aggregated

```
#!/usr/bin/env python3

import sys
from collections import  defaultdict

user_action_counts = defaultdict(lambda: {'post':0,'like':0,'comment': 0,
'share': 0})


for line in sys.stdin:
    try:
        fields = line.strip().split('\t')

        if len(fields) >= 3:
            user_id = fields[0]
            action_type = fields[2].lower()

            if action_type in ['post','like','comment','share']:
                user_action_counts[user_id][action_type] += 1

    except Exception as e:
        sys.stderr.write(f"Errorprocessingline: {line.strip()}, Error:
{str(e)}\n")

for user_id, counts in user_action_counts.items():
    sort_key = f"{10000-counts['post']:05d}"

    key = f"{user_id},{sort_key}"

    value =
f"{counts['post']},{counts['like']},{counts['comment']},{counts['share']}"

    print(f"{key}\t{value}")
```

2. action_aggregation_reducer.py – Formats output with proper sorting

```
#!/usr/bin/env python3

import sys
current_user = None

for line in sys.stdin:
    try:
        key,value=line.strip().split('\t')

        user_id=key.split(',')[0]

        if user_id != current_user:
            current_user=user_id

            post_count,like_count,comment_count,share_count=map(int,
value.split(','))

            output =
f"{user_id}\tposts:{post_count},likes:{like_count},comments:{comment_count}
,shares:{share_count}"
            print(output)

    except Exception as e:
        sys.stderr.write(f"Errorprocessingline:{line.strip()},Error:
{str(e)}\n")
```

Identification of Trending Content

1. **Calculating Engagement Scores:**
    - The `trending_content_mapper.py` calculates the sum of likes and shares for each content item.
2. **Determining Threshold:**
    - The 90th percentile of engagement scores is calculated using NumPy to determine the threshold for trending content.
3. **Handling Data Skew:**
    - Combiners (`trending_content_combiner.py`) perform local aggregation to reduce network traffic during the shuffle phase.
    - The reducer filters out content items below the threshold.

Implementation:

1. trending_content_mapper.py – to calculate counts of likes and shares per content value.

```
#!/usr/bin/env python3

import sys
From collections import defaultdict

content_engagement = defaultdict(int)

for line in sys.stdin:
    try:
        fields=line.strip().split('\t')

        if len(fields) >= 4:
            content_id = fields[3]
            action_type = fields[2].lower()

            if action_type in ['like', 'share']:
                content_engagement[content_id] += 1

    except Exception as e:
           sys.stderr.write(f"Errorprocessingline:{line.strip()},Error:
{str(e)}\n")

    for content_id, engagement in content_engagement.items():
    print(f"{content_id}\t{engagement}")
```

2. trending_content_combiner.py – To carry out local aggregation.

```
#!/usr/bin/env python3

importsys
from collections import defaultdict

content_engagement = defaultdict(int)

for line in sys.stdin:
    try:
        content_id,engagement=line.strip().split('\t')
        content_engagement[content_id] += int(engagement)

    except Exception as e:
        sys.stderr.write(f"Errorprocessingline:{line.strip()},Error:
{str(e)}\n")

for content_id, engagement in content_engagement.items():
    print(f"{content_id}\t{engagement}")
```

3. trending_content_reducer.py – To identify the trending content based on the threshold.

```
#!/usr/bin/env python3

import sys
import os
import numpy as np

threshold = int(os.environ.get('TRENDING_THRESHOLD', -1))

all_engagements = []
content_data = []

for line in sys.stdin:
    try:
        content_id,engagement=line.strip().split('\t')
        engagement = int(engagement)

        all_engagements.append(engagement)
        content_data.append((content_id, engagement))

    except Exception as e:
        sys.stderr.write(f"Errorprocessingline:{line.strip()},Error:
{str(e)}\n")

if threshold <0:
    if all_engagements:
        threshold = np.percentile(all_engagements,90)
    else:
        threshold = 0

sys.stderr.write(f"reporter:counter:TrendingStats,ThresholdUsed,{int(thresh
old)}\n")

for content_id, engagement in content_data:
    if engagement >= threshold:
        print(f"{content_id}\t{engagement}")
```

Dataset Joining

1. **Preparing Data for Join:**
    - The `join_activity_mapper.py` and `join_profile_mapper.py` prepare the activity and profile data for joining.
    - Salted keys are used for power users to handle data skew.
2. **Performing the Join:**
    - The `join_reducer.py` performs the join operation by matching user IDs from both datasets.
    - It ensures that valid joins are produced, combining activity counts with profile information.

Implementation:

1. join_activity_mapper.py – Creates join of activity data

```
#!/usr/bin/envpython3

import sys
import os

skewed_keys_str = os.environ.get('skewed.keys','')
skewed_keys=set(skewed_keys_str.split(','))ifskewed_keys_strelseset()

NUM_SALTS = 10

for line in sys.stdin:
    try:
        fields=line.strip().split('\t',1)

        if len(fields) >= 2:
            user_id=fields[0]
            activity_data=fields[1]

            if user_id in skewed_keys:
                for i in range(NUM_SALTS):
                    salted_key =
                    f"{user_id}_{i}"print(f"{salted_key}\tA:{activity_data}")
            else:
                print(f"{user_id}\tA:{activity_data}")

    except Exception as e:
        sys.stderr.write(f"Errorprocessingline:{line.strip()},Error:
{str(e)}\n")
```

2. join_profile_mapper.py – The profile data is prepared for joining

```
#!/usr/bin/env python3

import sys
import os

skewed_keys_str=os.environ.get('skewed.keys','')
skewed_keys=set(skewed_keys_str.split(','))ifskewed_keys_strelseset()

NUM_SALTS = 10

for line in sys.stdin:
    try:
        fields=line.strip().split('\t',1)

        if len(fields) >= 1:
            user_id_parts=fields[0].split(',')
            user_id = user_id_parts[0]

            profile_data = fields[0]
            if len(fields) > 1:
                profile_data += "\t" + fields[1]

            if user_id in skewed_keys:
                for i in range(NUM_SALTS):
                    salted_key = f"{user_id}_{i}"
                    print(f"{salted_key}\tP:{profile_data}")
            else:
                print(f"{user_id}\tP:{profile_data}")

    except Exception as e:

        sys.stderr.write(f"Errorprocessingline:{line.strip()},Error:
{str(e)}\n")
```

3. join_reducer.py – The join operation is carried out.

```
#!/usr/bin/env python3

import sys

current_user  =  None
profile_data  =  None
activity_data=None

for line in sys.stdin:
    try:
        user_id,tagged_data=line.strip().split('\t',1)

        if '_'in user_id:
            user_id = user_id.split('_')[0]

        if user_id != current_user:
            if current_user is not None and profile_data is not None and
activity_data is not None:
                print(f"{current_user}\t{profile_data}\t{activity_data}")


            current_user = user_id
            profile_data = None
            activity_data = None

        tag = tagged_data[0]
        data = tagged_data[2:]

        if tag == 'P':
            profile_data = data
        elif tag == 'A':
            activity_data = data

    except Exception as e:
        sys.stderr.write(f"Errorprocessingline:{line.strip()},Error:
                {str(e)}\n")
if current_user is not None and profile_data is not None and activity_data
is not None:
    print(f"{current_user}\t{profile_data}\t{activity_data}")
```

4. skew_detection.py – The skewed keys are to detected and handled.

```
#!/usr/bin/envpython3

import sys
import json
from collections import Counter
import numpy as np

def analyze_key_distribution(lines, skew_threshold_factor=0.01):
    key_counter = Counter()
    total_records = 0

    for line in lines:
        total_records += 1
        try:
            key = line.strip().split('\t')[0]

           if ',' in key:
               key = key.split(',')[0]

           key_counter[key] += 1
     except Exception as e:
           sys.stderr.write(f"Errorprocessingline:{line.strip()},
Error: {str(e)}\n")

    if not key_counter:
        return{
            "total_records":0,
            "unique_keys":0,
            "skew_threshold":0,
            "skewed_keys":0,
            "top_keys":0,
           }
     unique_keys = len(key_counter)
     avg_records_per_key = total_records/unique_keys if unique_keys > 0
else 0
```

```
    skew_threshold=max(
        skew_threshold_factor * total_records,
        avg_records_per_key * 5
    )

    skewed_keys = [Key for key, count in key_counter.items() if count >
skew_threshold]

    top_keys = key_counter.most_common(10)

    counts=np.array(list(key_counter.values()))

    return{
        "total_records": total_records,
        "unique_keys": unique_keys,
        "average_records_per_key":float(avg_records_per_key),
        "skew_threshold": float(skew_threshold),
        "skewed_keys": skewed_keys,
        "skewed_keys_count":len(skewed_keys),
        "top_keys": top_keys,
        "distribution_stats": {
            "min":int(np.min(counts)),
            "max": int(np.max(counts)),
            "median":float(np.median(counts)),
            "mean": float(np.mean(counts)),
            "std_dev": float(np.std(counts)),
            "p90":float(np.percentile(counts,90)),
            "p95":float(np.percentile(counts,95)),
            "p99":float(np.percentile(counts,99))
        }
    }
def main():
    try:
        lines = sys.stdin.readlines()
        analysis = analyze_key_distribution(lines)
        print(json.dumps(analysis, indent=2))

    except Exception as e:
        error_output = {
            "error":str(e),
            "skewed_keys":[]
        }
```

```
        print(json.dumps(error_output,indent=2))
        return 1

    return 0

if name____== "main":
    sys.exit(main())
```

Performance Optimization and Monitoring

Optimization Strategies:

1. **Combiners:**
    - Use combiners to perform local aggregation, reducing network traffic during the shuffle phase.
2. **Efficient Data Structures:**
    - Use dictionaries (`defaultdict`) for efficient aggregation and counting.
3. **Cluster Configuration:**
    - Tune the number of reducers based on dataset size and complexity.
    - Allocate sufficient memory to avoid spilling to disk.

Monitoring Tools:

1. **Hadoop Counters:**
    - Track discarded records and other metrics using counters.
2. **Logs:**
    - Analyze logs to identify bottlenecks and optimize performance.
3. **System Fault Tolerance:**
    - Ensure fault tolerance by configuring replication factors and checkpointing mechanisms.

DATA VISUALIZATION

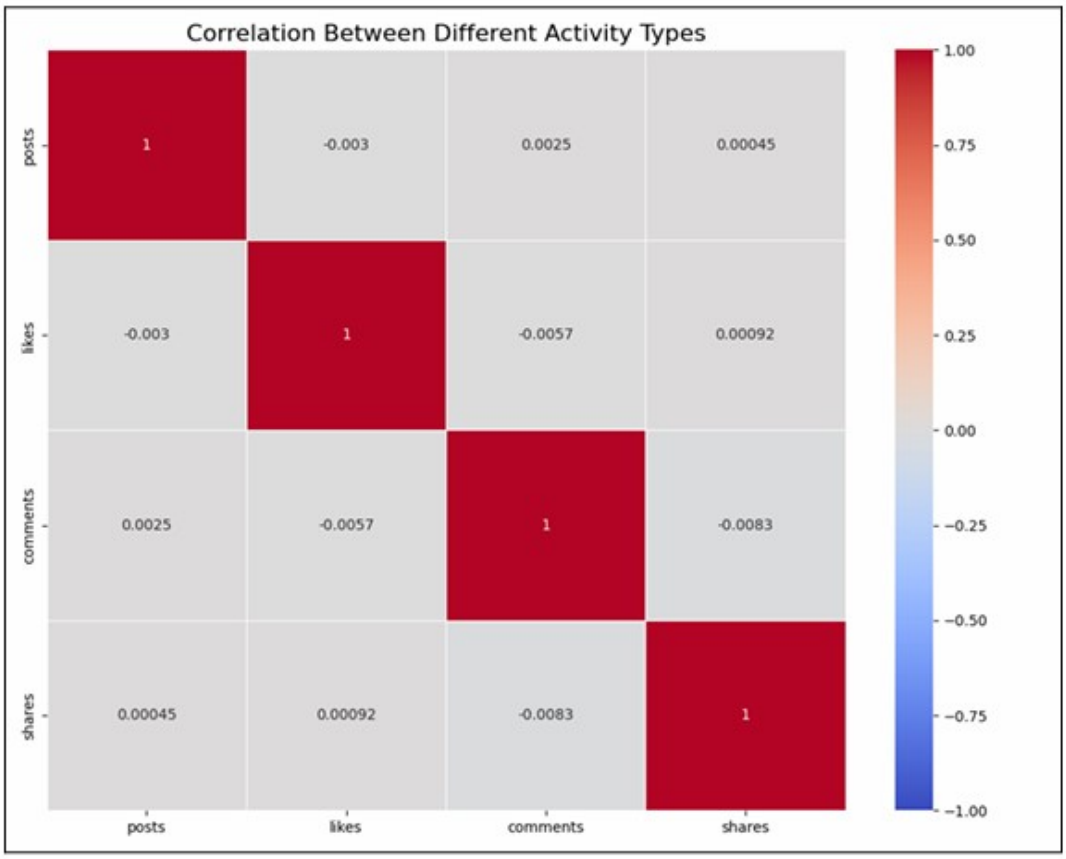
1. To create visualisation of the metrics given in the files and analysed above.
2. The dashboard is interactive and can be explored for data exploration.

Implementation:

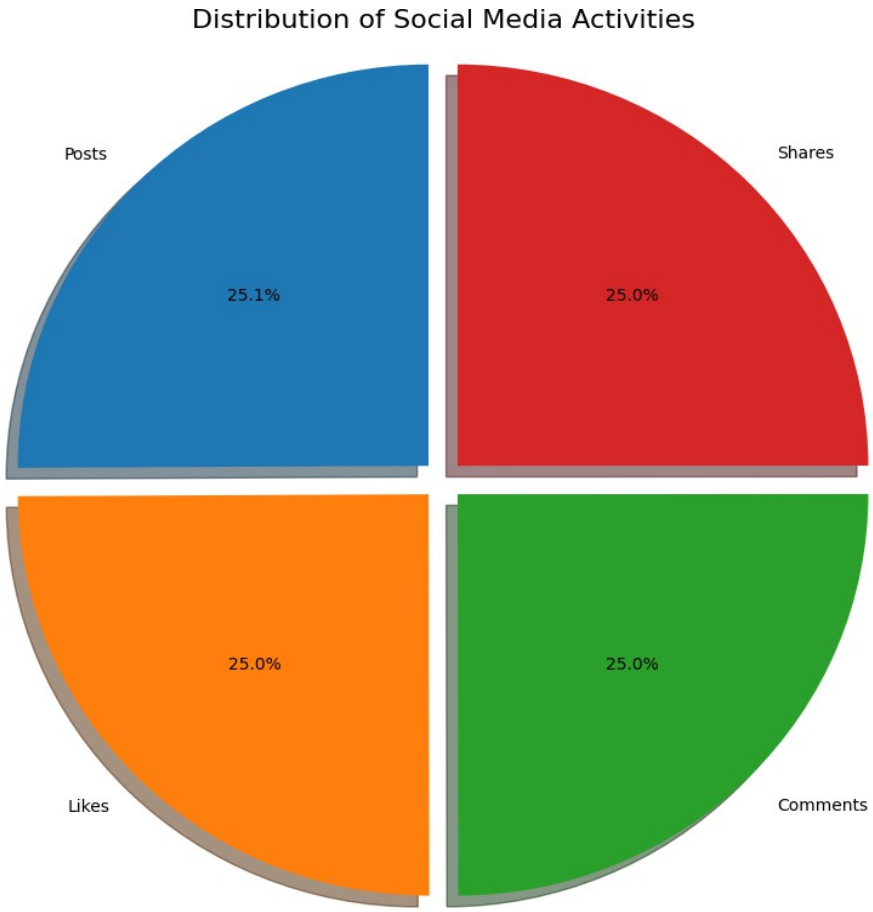
1. visualize_analytics.py - Visualisation is reated in PNG format and displayed.
2. analytics_dashboard.py - Provides web-based interactive dashboard

Visualizations:

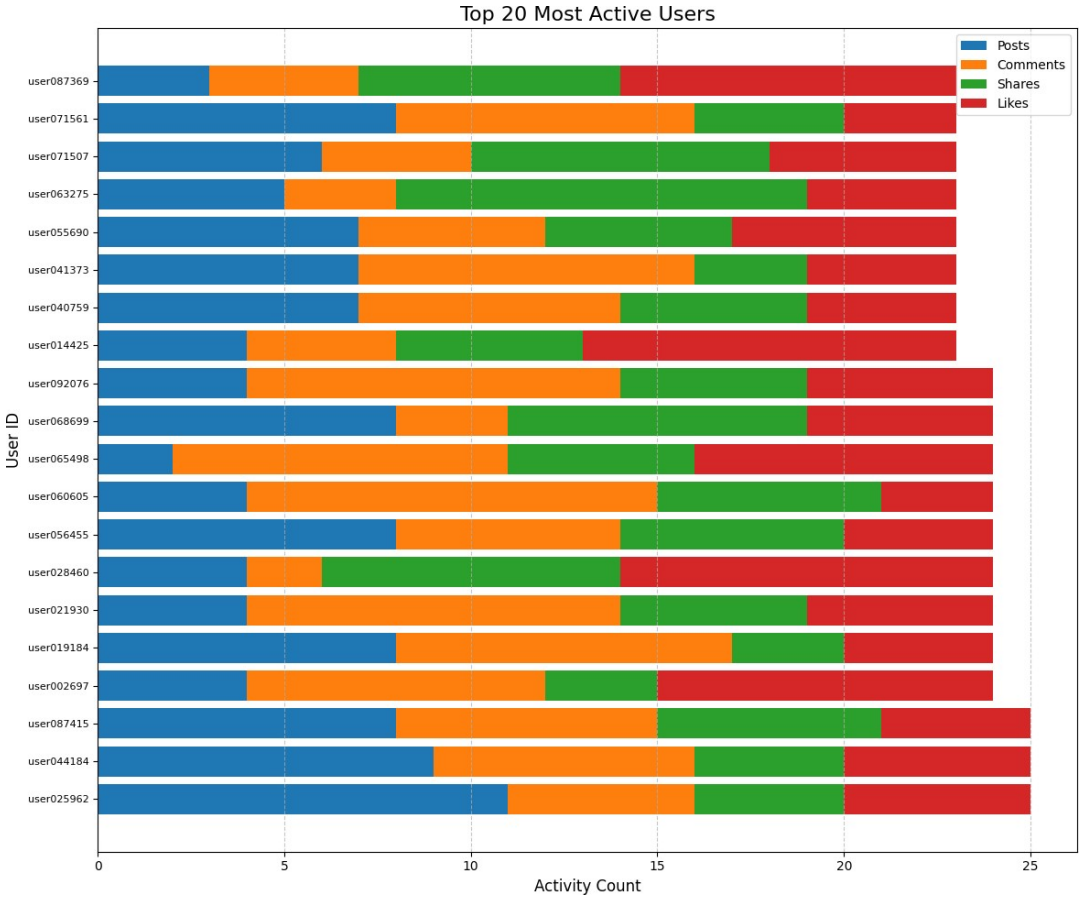
1. Activity Correlation:



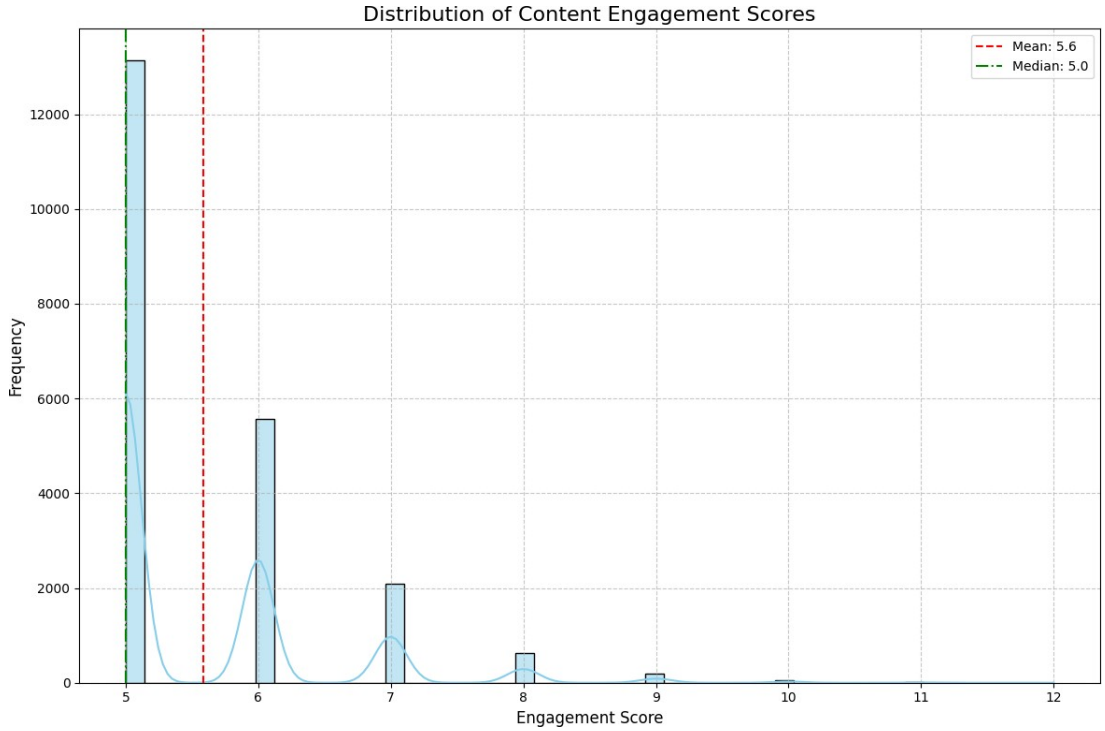
2. Activity Distribution



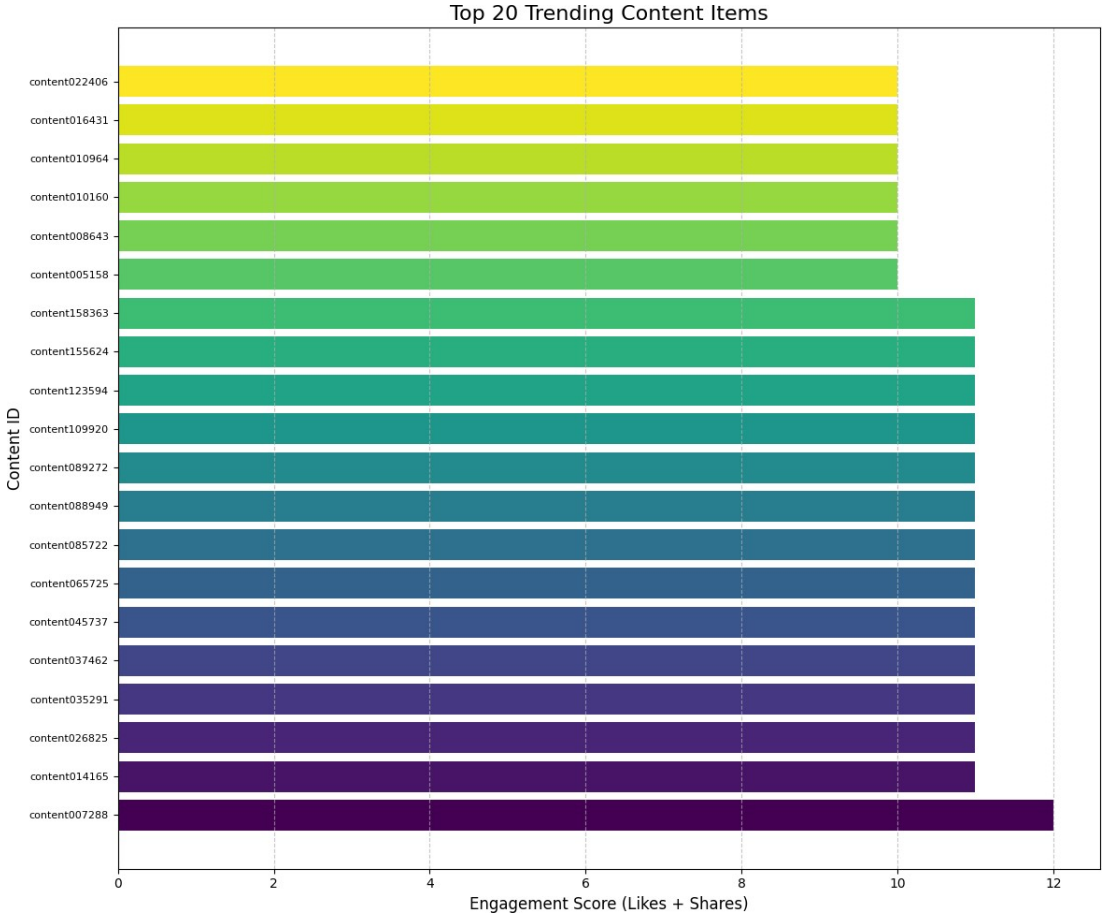
3. Top Users:


Top 20 Most Active Users
Posts
Comments
Shares
Likes
user087369
user071561
user071507
user063275
user055690
user041373
user040759
user014425
user092076
user068699
user065498
user060605
user056455
user028460
user021930
user019184
user002697
user087415
user044184
user025962
User ID
0
5
10
15
20
25
Activity Count

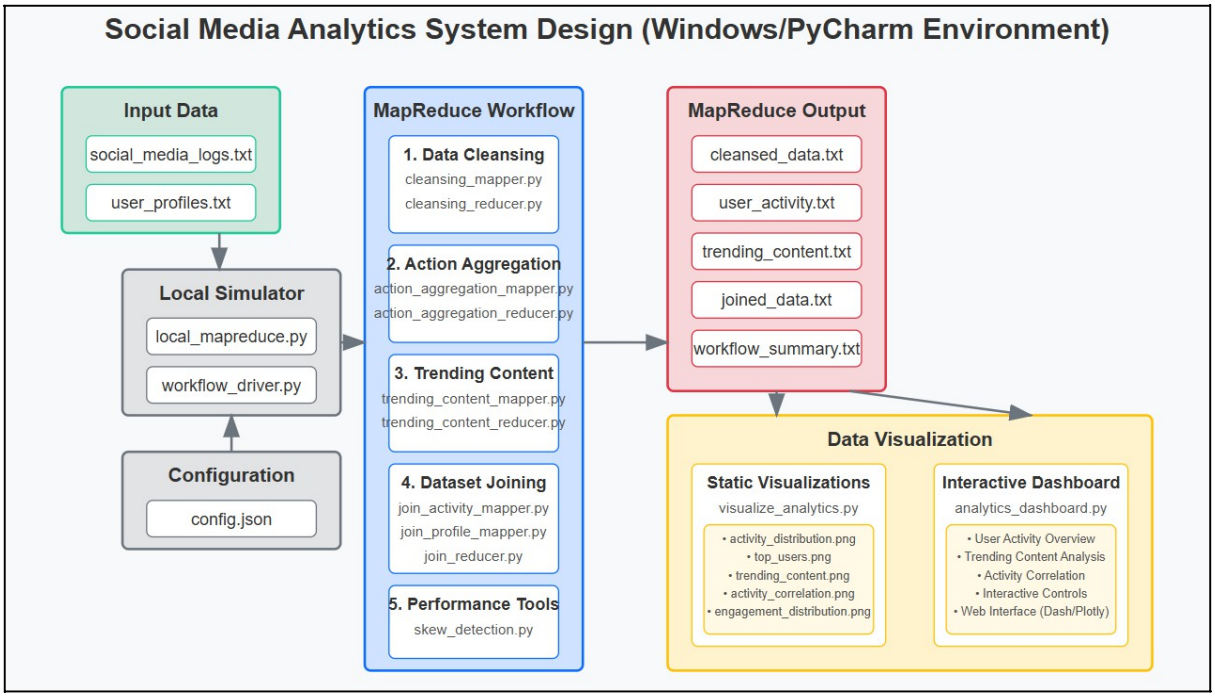
4. Distribution of Content Engagement Scores



5. Top trending Content Items



6. Workflow Block Diagram of the System Design



G24AI1060
Shreyas Gaikwad

ANSWERS

Ans 2 :

To design and analyze B-tree indexing solution for real world transaction system.

Given that

50, 20, 70, 10, 30, 60, 80, 5, 15, 25, 35, 55, 65, 75, 85

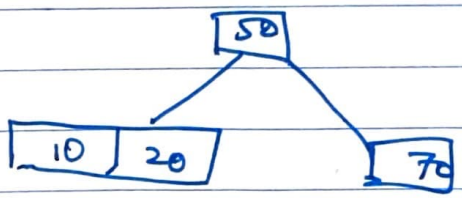
$m = 4$, max 4 children & 3 keys. ($m-1 \approx 4-1 \simeq 3$)

Create a block, upto max of 3 keys

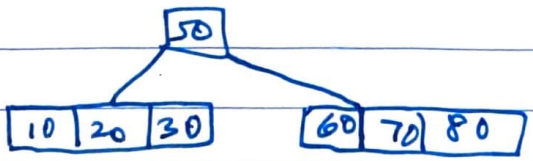
Insert the first four number & answer

i.e. 

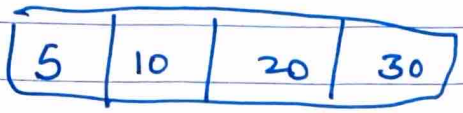
But there has to be only 3 keys. So split the tree into right heavy



Insert next element i.e. '30', '60', '80'

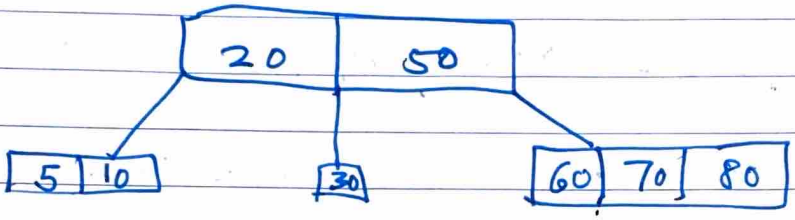


Once we input '5' the node is overflowed.
ie. 4 elements.

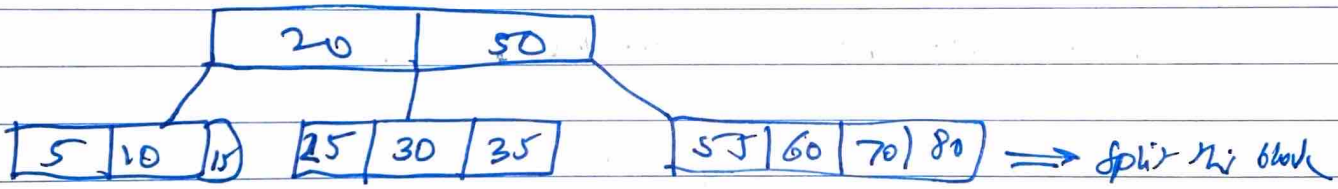


→ Some split with 20 as node.

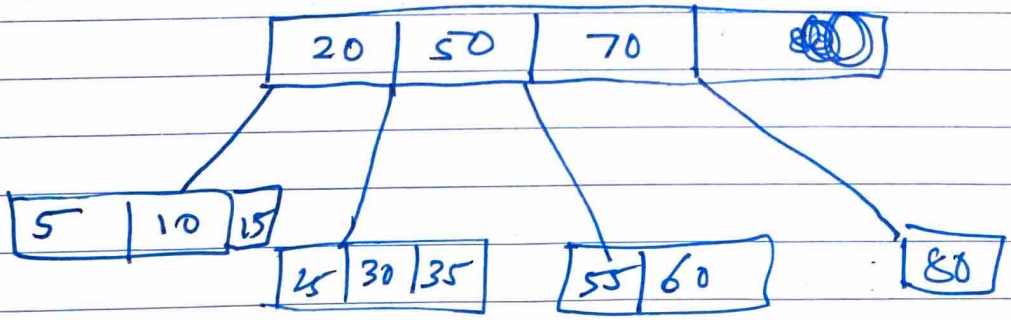
we get.



Input '15', '25', '35' & '55'

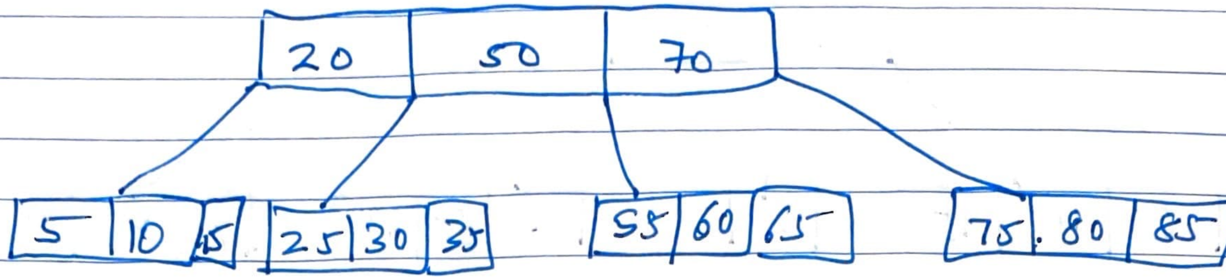


⟹ split this block



Input
65, 75, 85

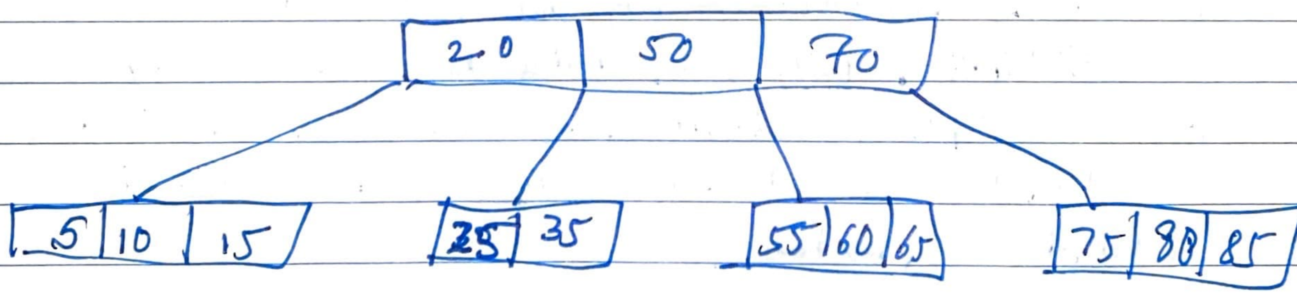
Final B-tree we get. (Right split).



In case of deletion of any key from child/root, all will be >2, hence deleting 1 key will keep the structure same.

[illegible] redistribution required.

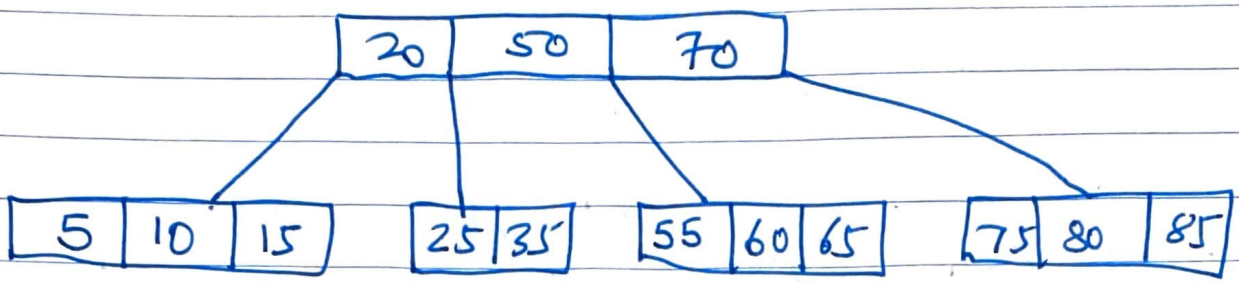
Delete 30, new tree



Now delete 30 from new tree then distribution will be, same as min 1 key is present.

Hence B-tree structure remains without change even if two keys from same child are deleted

∴ Even after removing 30, the final structure is



2) Real world Query Optimization

(a) How B tree improves performance of range queries.

Ans (i) The storage of keys is in ascending order (sorted)
(ii) Logarithmic key search is possible due to sorting.
(iii) The keys are not random, hence all keys in range will be found together sequentially.
(iv) The structure is even and balanced, thus the distribution with large datasets also is easy.
(v) The depth of distribution is also even, ie. all leaf nodes are at same level, thus search time remains constant.

(b) -Challenges and limitations:
(i) Concurrent inserting elements can add to delay as multiple nodes can be involved at different levels to maintain the storage structure.
(ii) The lock/unlock mechanism will cause delay due to concurrent changes at different nodes & leaves.

(iii) There might be cases of deadlock due to multiple transactions happening together.

(iv) During split, the other nodes might read inconsistency hence avoid further transactions.

(v) Locked subtrees can lead to deadlocks in high concurrent trees.

Solutions:

→ Node level locking to be implemented so that lock only node is updated.

→ follow sequence of lock parent then child and then release parent lock to avoid inconsistency.

→ Provide mechanisms to read data anytime without lock restrictions.

→ Use lock for individual nodes instead of trees.

→ Use logs to reacquire consistency states

→ Create partitions of B-trees based on certain criteria if data set is too large. (improves time).

→ Ensure back up state after periodic intervals.

→ Plan prelock image of every transaction which is overwritten with every successful transaction.

Q.3. Advance fault tolerance in Cloud based Financial trading system.

1. Reliability Analysis

(a) Derive expression for P_{avail}

The probability that single critical data element is available depends on Replication factor 'R' and probability of node failure 'p'.

Assuming that,
Each node failure is independent with probability 'p'

and critical data element is available if atleast one replica 'R' is online.

Hence
considering all replica's fail simultaneously is
$P_{all\_fail} = p^R$ where 'p' is single node fail probability

The probability that atleast one replica is operational, the probability is

$$P_{avail} = 1 - P_{all\_fail}$$

$$= 1 - p^R$$

∴ Availability of single node is

$$P_{avail} = 1 - p^R$$

(b) for $R = 3$, $p = 0.05$,

we have $P_{avail} = 1 - (0.05)^3$

$= 1 - 0.000125$

$= 0.999875 \approx 99.9875\%$

∴ If $R = 3$ for $p = 0.05$, the availability of single data client is 99.9875%

2] Consensus Protocol Impact

(a) Choice of consensus protocal affects fault tolerance in terms of performance & recovery.

Ans: The choice of consensus protocal significantly affects financial trading system in a numbers of ways.

(a) Performance consideration:

(i) Latency: paxos requires multiple rounds of communication, Prepare, promise, acknowledge and accept phase, to reach consensus. This incurs latency in distributed systems over a large network

— In a financial trading system, especially with high rates of transaction there can be performance bottlenecks. if not optimized correctly.

(ii) Throughput : Multiphase network Paxos can reduce throughput as it processes fewer transactions per second.

Optimized implementation like multi-Paxos can reduce overhead as it can process multiple batches together. thus overall efficiency will increase.

(b) Recovery time:

(i) Leader Election: In Paxos, there is no explicit leader election phase, Nodes act as proposers, acceptors & learners. A stable leader is chosen to optimize performance

If current leader fails, other nodes are to detect failures, propose new values with increased recovery time.

(ii) Replication Overhead: High replication factor (R) improves recovery & fault tolerance but also is complex to maintain consistency across large distributed networks.

Simple implementation recovers faster as compared to complex networks.

(b) Trade-off between R & consensus Overhead.

(1) Mathematical Justification:

- Availability: Increasing R improves availability

$$P_{avail} = 1 - p^R$$

as R grows, p^R decreases exponentially,

making P_{avail} approach 1.

(2) Latency: Large R is required to involve the number of nodes involved in consensus which raises latency and network traffic.

(2) Trade-off:

- High R:- Improves availability by reducing the risk of data loss.
  - Increases consensus latency due to more nodes participating in Paxos protocol.

Lower R:- Reduces latency and resource usage but makes the system more vulnerable to failures.

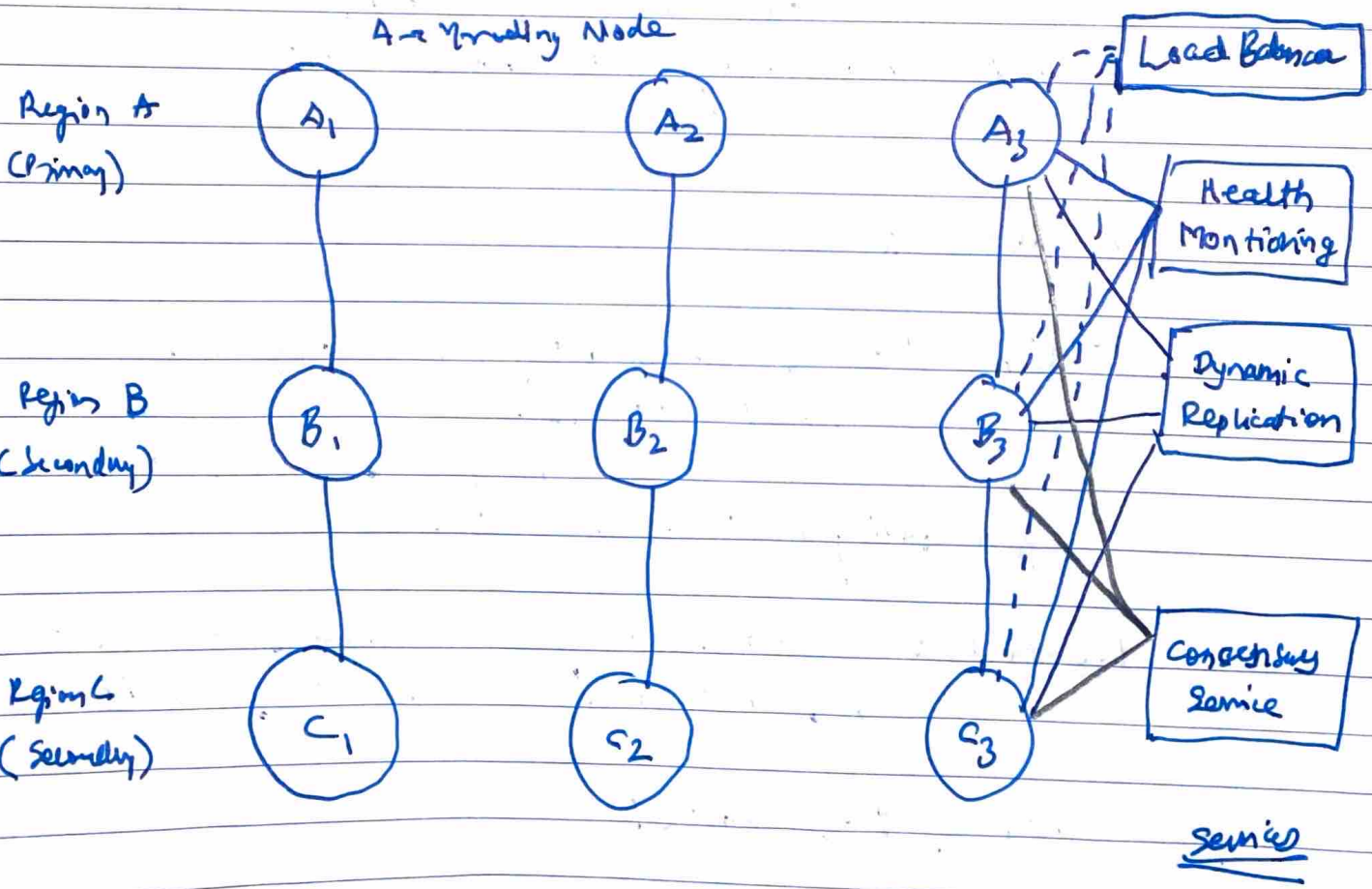
3.) Impact of Paxos on R:

→ With Paxos, increasing R means more acceptors are involved in each phase of the protocol. This increases message overhead and delay agreement. especially in wide-area networks.

- for example, if R=5, Paxos requires atleast 3 acceptors to agree whereas if R=3 only 2 acceptors are needed. The additional nodes adds complexity & delay.

3)

High Level Architecture :



Service

Fault tolerance Strategy:

This fault tolerance strategy combines dynamic application management with rapid failure detection, designed for high stakes environment.

⇒ Core components and Interactions

→ Layered distribution: - The system is distributed geographically
- Each region has a complete set of nodes that operate independently
- cross-region replication is done for data validation and availability even if entire region fails.

→ Health Monitoring System: - Continuous health monitoring node health.
- Hardware metrics CPUs, memory, Input/output
- Network latency and packet loss
- Application level metrics.
- Errors & Exception rate & patterns.
- Use failure prediction methods. based on statistical anomaly detection.
- Use methods to determine optimal 'R' values.

→ Consensus Service :- Use effective methods for high-value trades.
- categorically distinct data categories.
- Transaction batching can be done to reduce overheads during peak volume periods.

Optimization Techniques:

① — Adaptive failure detection Thresholds
- System dynamically adjusts to required failure detection:
- more conservative during calm periods (avoid false positive)
- more aggressive during instability.

- Mathematical model: $T_{detection} = T_{base} \times (1 - \alpha \times current\_rate)$

② Intelligent Data placement
- Replicate data to maximize diversity
- Use network topology awareness to minize cross server latency
- Implement follow the sun replication strategy that shifts primary replicas to region with active trading sessions.

③. Predictive Scaling:
- Anticipates increased trading volumes based on events, time of day and historical patterns.
- pre-scale replication factors before peak loads occur
- Implements gradual R reduction during periods of reduced activity.

④ Self healing mechanisms.
- Automatic construct lost replicas with priority based recovery
- Background verification of replica consistency with hash based integrity checks.
- Facility window checkpoint for rapid recovery.