Design Document – Assignment 3 (DIRT)

Overview

This project implements a **DIRT-style unsupervised relation similarity system** using Hadoop MapReduce. The goal is to extract syntactic paths from a large dependency-parsed corpus, represent each path as a distribution over argument words, compute Mutual Information (MI)–weighted feature vectors, and finally compute similarity scores between pairs of relational paths. The system is evaluated against a provided gold standard of relation pairs.

The pipeline consists of **seven MapReduce steps**, orchestrated by a single driver and executed on an Amazon EMR cluster.

Input Data

1. **Biarcs corpus** Dependency-parsed syntactic fragments of the form:

   ```
   head<TAB>token1 token2 ... tokenN<TAB>count
   ```

   where each token is encoded as:

   ```
   word/POS/dep/headIndex
   ```

2. **Test pairs file**

   ```
   pathA<TAB>pathB
   ```

3. **Gold labels file**

   ```
   pathA<TAB>pathB<TAB>label
   ```

Step 1 – Path and Argument Extraction

Goal: Extract binary relations of the form `X <path> Y` and collect argument words.

Preprocessing (within Mapper)

- **Stemming:** All words (verbs in paths, noun slot fillers) are stemmed using a Porter Stemmer to unify suffix-based inflections (e.g., "involves" → "involve", "solutions" → "solution").

- **Auxiliary Verb Filtering:** Paths with auxiliary verb heads (e.g., "is", "are", "was", "were", "be", "been", "being", "have", "has", "had") are filtered out.
- **Preposition Inclusion:** Unlike MiniPar, Stanford parser retains prepositions in the dependency tree. Paths include dependency relations connecting prepositions (IN, TO) alongside content words (nouns, verbs, adjectives, adverbs), as required by the assignment.
- **Path Constraints:** Only paths where the head is a verb and both X/Y slots are filled by nouns are retained.

Mapper

Input key/value:

- Key: byte offset in input file (LongWritable)
- Value: one biarc record (Text, ~100–200 bytes)

Output key/value:

- Key: `(path, slot, word)` encoded as Text (~40–60 bytes)
- Value: count (IntWritable, 4 bytes)

Each valid relation produces **two output pairs** (one for `X`, one for `Y`).

Volume:

- Small run (10 files): ~120M map input records, ~40M map outputs
- Large run (100 files): ~1.78B map input records, ~607M map outputs

Combiner / Reducer

- Aggregates counts for identical `(path, slot, word)` keys
- Output key/value identical to mapper output

Effectiveness (large run):

- Combine input records: ~623M
- Combine output records: ~101M (≈6× reduction)

Memory:

- Combiner operates on streaming hash maps, peak <200MB per task

---

Step 2 – Marginal Count Construction

Goal: Compute joint and marginal counts for MI.

Mapper

From each `(path, slot, word, count)` emits three records:

Type	Key structure	Value

Type	Key structure	Value		
PSW	path	slot	word	count
PS*	path	slot	*	count
*SW	*	slot	word	count

Output size:

- Key: ~30–50 bytes
- Value: 4 bytes

Volume:

- Small: ~3× Step 1 reducer output
- Large: ~237M map outputs (~3× Step 1 output)

Combiner / Reducer

- Sums counts per key
- Shuffle size (large): ~937MB

Output format:

- Key: as above
- Value: aggregated count (IntWritable)

Memory:

- Reducers stream aggregation; peak memory ~2.0GB

Step 3 – Mutual Information Computation

Goal: Compute `MI(path, slot, word)` for all path-slot-word triples in the corpus.

Mapper

Re-keys records to ensure required marginals meet in the same reducer.

Key:

- Prefix + `(slot)` or `(slot, word)`

Value:

- Tagged count record (~16–24 bytes)

Reducer

- Single reducer (by design)
- Holds:
  - `c(slot)` totals

    - `c(path,slot)` totals

Computation:

```
MI = log( (c_psw * c_s) / (c_ps * c_sw) )
```

Only positive MI values are emitted.

Volume:

- Small: ~2–3M MI records
- Large: ~60.6M MI records (with ~18.3M non-positive filtered out)

Memory (large):

- Peak reducer physical memory: ~2.3GB
- Heap committed: ~29GB (cluster-wide)

Output Deliverable:

The MI values for all path-slot-word triples are persisted to S3 as a system output, enabling future similarity calculations for path pairs not included in the test set.

Step 4 – Path Feature Vector Construction

Goal: Build MI-weighted feature vectors per path.

Mapper

Key: `path`

Value: `word,slot,MI` (~20–30 bytes)

Reducer

- Concatenates all features for a path
- Output value size varies widely

Volume:

- Distinct paths (large): ~1.22M
- Typical vector: 20–200 features
- Heavy tail: a few thousand features (11,010 paths with >1000 features, 1,183 with >5000)

Memory:

- Reducer buffers features per path; peak ~1.9GB

Step 5 – Path Pair Alignment

Goal: Join path vectors with test and gold pairs.

Preprocessing

- **Predicate Stemming:** Predicates from the test and gold pair files are stemmed using the same Porter Stemmer to ensure alignment with the stemmed paths extracted in Step 1.

Path Normalization

Paths are normalized to a canonical form before joining to handle equivalent representations (e.g., directionality normalization).

Mappers

1. **Vector Mapper**
   - Key: normalized path
   - Value: vector blob (~1–10KB)
2. **Test Pair Mapper**
   - Key: normalized path (stemmed)
   - Value: paired path ID
3. **Gold Pair Mapper**
   - Same structure as test mapper

Reducer

- Joins vectors with all requested pairs
- Emits `(pathA, pathB, side, vector)`

Volume:

- large: ~2.4K pairs emitted
- small: ~500 pairs emitted
- Paths with vectors: ~1.22M

Memory:

- Negligible (<100MB)

Step 6 – DIRT Similarity Computation

Goal: Compute similarity between path pairs.

Reducer

- Parses two vectors
- Splits features by slot (X/Y)
- Computes (following DIRT paper Section 4.3, Equation 3):

```
sim = sqrt( sim(SlotX1, SlotX2) * sim(SlotY1, SlotY2) )
```

Where slot similarity follows Equation 2:

```
sim(slot1, slot2) = sum of shared MI / (sum of MI in slot1 + sum of MI in slot2)
```

Expanded:

```
sim = sqrt( (sharedX / (sumX1 + sumX2)) *
            (sharedY / (sumY1 + sumY2)) )
```

Volume:

- Output pairs (large): 500 (with 123 zero-similarity pairs filtered)
- Output pairs (small): 18 (all non-zero)

Memory:

- Feature maps per reducer: <50MB

Step 7 – Evaluation Join

Goal: Join similarities with gold labels.

Reducer

Key: `(pathA, pathB)`

Value: `similarity, label`

Output:

- Large: final evaluation table with 500 pairs
- Small: final evaluation table with 18 pairs

System Outputs

The system produces two primary outputs as required by the assignment:

1. **Similarity Measures:** The similarity score between each pair in the given test set (output of Step 7).
2. **MI Values:** The `mi(p, Slot, w)` value for each path-slot-word triple found in the corpus (output of Step 3, persisted to S3), supporting further similarity calculations for path pairs not in the test set.

Experimental Resource Usage Summary

Small Experiment (10 files)

- Map input records: ~120M
- Map output records: ~35–40M
- Shuffle size: ~80–100MB
- Peak map memory: ~600–800MB
- Peak reduce memory: ~900MB

Large Experiment (100 files)

- Map input records: ~1.78B
- Map output records: ~607M
- Shuffle bytes: ~845MB (Step 1), ~937MB (Step 2), ~958MB (Step 3)
- Reduce input groups: ~79M (Step 1)
- Peak map memory: ~1.0GB
- Peak reduce memory: ~2.3GB
- Total S3 read: ~55GB
- Total S3 write: ~1.9GB (Step 1)

Summary

This revised design explicitly characterizes each MapReduce component in terms of key–value structure, data volume, and memory footprint. Empirical Hadoop counters confirm that:

- Combiners are essential and highly effective (~6× reduction in Step 1)
- Similarity computation is inexpensive due to aggressive filtering

Overall, the system scales linearly with corpus size and conforms well to the MapReduce execution model, while faithfully implementing the DIRT algorithm.