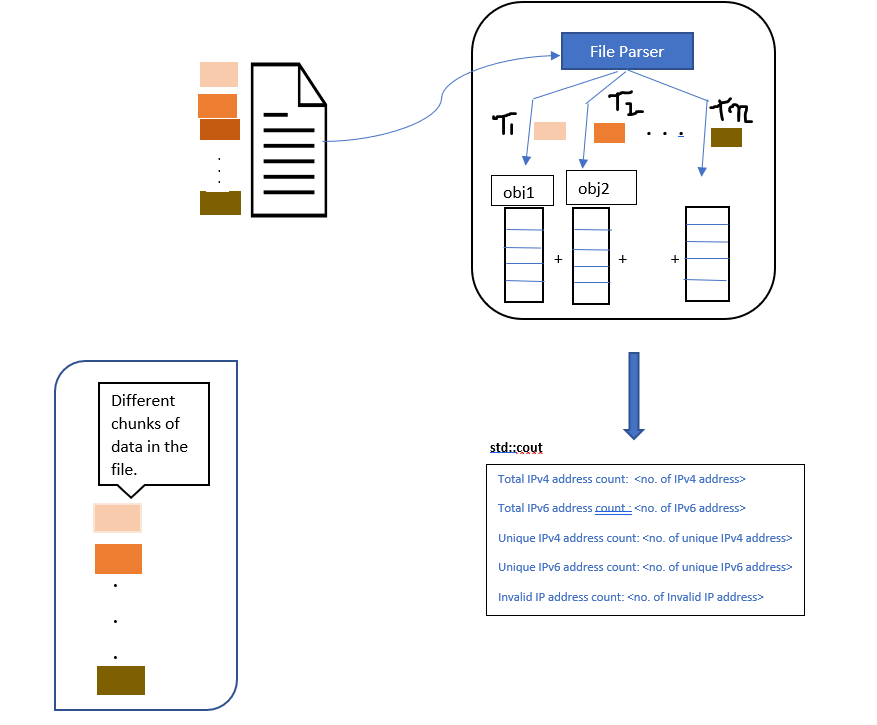
**Problem statement:**

Input:

•File consists of millions of records separated by new line character.

•Each line can be a valid IPv4/IPv6 or invalid string.

**HLD:**



**Functional Requirements:**

1.The file parser must be able to read the file.

2.The file parser must be able to differentiate between IPv4, IPv6 and Invalid ips.

3.The respective count variables of IPv4, IPv6, unique IPv4, unique IPv6 and Invalid ips should return the accurate number of occurrences in the file at the end of execution.

**Non- Functional Requirements:**

1.In order to handle huge amount of data, concurrent processing should be opted.

2.Threads must be synchronized in order to prevent data corruption.

3.Main thread must wait at the end for all other worker threads to join.

4. Performnamce requirements:

- time complexity: O(n), where n represents number of lines in the input file

- space complexity: O(n), where n represents number of lines in the input file

- benchmarking: On Macbook Pro with M1 chip and 8GB RAM it takes around 2450 ms to process 1 million lines. considering max length of every line is 39 chars(max length of IPv6 IP)

5. Robust requirements:

- On a multi core system with hyper threading enabled, the system can process millions of entries in seconds or less.

- For a large in put the number of worker threads can be increased.

6. Redundancy Requirement:

- Redundant inputs: In case of redundant strings in the input, the counters are incremented but the strings are not stored and hence ignoring it.

- Redundent code: Proper structuring of the code is done so that specific parts/codes of different classes can be reused avoiding redundunt codes.

**Error Case :**

File read failure

**Cpp file functionality :**

main.cpp :creates instance of fileparser and calls createWorkerThread.

fileParser.cpp : Creates worker threads and assigns them a chunk of input.

ipValidator.cpp: iterates over the chunk of file allocated and validates each ip and increments the respective counters.

utils.cpp: Definitions of counters are defined here.

**Use of data structures:**

1. vectos are used to store the thread details. The reasons for using are as follows:

- insersion at the begining and deletion at the ned is needed. No such operation is needed in between.

- no key based random access is needed.

- just iterative behavior is needed

2. static unordered\_sets are used to keep a track of unique IPs. The reasons for using are as follows

- unordered\_sets only keep record of unique entries only

- unordered\_sets are basically hash sets so quering can be done in O(1) time complexity

- it is made static in order to have single point of reference for all the threads