SPoP Project

Dr. Rahul Jain, rENIAC Inc

Problem

- Build an In-Memory Key-Value Storage Software in C++
- Supported APIs
  - get(key): returns value for the key
  - put(key, value): add key-value, overwrite existing value
  - delete(key)
  - get(int N): returns Nth key-value pair
  - delete(int N): delete Nth key-value pair
- Spec
  - Max key size: 64 bytes
  - Each key char can be (a-z) or (A-Z): Total 52 possible chars
  - Max Value Size: 256 bytes, any ASCII value
  - No DS/boost/STL etc Libraries to be used

Other Spec

- Only pthread to be used for multithreading
- Keys to be in ~~Lexicographical~~ strcmp (cstring) Order
- API calls would be blocking

```
#include <iostream>
#include <cstring>
using namespace std;
int main(int argc, char** argv) {
  if(argc !=3) cout << "Usage lex str1 str2" << endl;
  else cout << "strcmp(" << argv[1] << ", " << argv[2] << ") = " << strcmp(argv[1], argv[2]) << endl;
}
```

Class Definition

```
struct Slice{
     uint8_t size;
     char*   data;
};

class kvStore {
public:
     kvStore(uint64_t max_entries);
     bool get(Slice &key, Slice &value): //returns false if key didn't exist
     bool put(Slice &key, Slice &value): //returns true if value overwritten
     bool del(Slice &key);
     bool get(int N, Slice &key, Slice &value): //returns Nth key-value pair
     bool del(int N): //delete Nth key-value pair
};
```

Evaluation Benchmark

- Multithreaded benchmark application
- Runs:
  - Benchmark would first load data via put calls (10 million entries, 2 min time limit)
  - Perform Single Threaded Transactions to verify kvStore functionality
  - **Multiple Transaction Threads with each thread calling one of the APIs**
- Evaluation Metrics:
  - TPS (Transactions Per Second)
  - **Average** CPU Usage
  - **Peak** Memory Usage

Ranking Methodology (Might Change)

- Normalize all 3 metrics with Avg
- Score = (TPS*TPS)/(CPU*Mem)

Relevant Concepts

- Ordered Data Structures
  - B-Tree, BST, etc
  - Tries, FST (Advanced)
  - Hybrid: Combination of Hash Table + Tree
  - https://www.coursera.org/learn/cs-fundamentals-2
- Bit Hacks
- Cache Optimizations
- Memory Allocation Optimizations
  - Memory Object Reuse
  - No Dynamic Memory Allocation
- Multithreading (pthreads)
- These are only starting pointers, feel free to explore

Submission Deadline

- Form Groups of 2, can be different from assignment group
  - No Groups of 3 allowed. Group of 1 is OK
- Read papers or other relevant material around better data structures
  - must provide reference
- Implementation Spec: 3rd Feb
- Code+Report: 18th Feb **(Hard Deadline)**
- If a working code has been submitted on/before 18th Feb, a more optimized version can be submitted upto **29th Feb**