More than you ever wanted to know about CSV

Digging into the CSV script

Script Outline

- Load file
- Split by line into records
- Split by delimiter into fields
- Test for a condition (field 2 != "Ensign")
- Print out another column (field 1 i.e., "Name")

Survey: Common Bottlenecks

- File IO: open(…) and for line in f
- String splitting: split(“,”, line)
- String-parsing: int(field[2])

Accessing Data: Streams and Paged Access

- Access data on the HDD/SSD/Network
  - API: Read Page, Write Page
  - Access Cost: Latency vs Throughput (Review of Memory Hierarchy)
    - Network: ? Latency, Good throughput, Ginormous size
    - HDDs: Bad latency, Good throughput, Huge size
      - Why is paged access a good fit for HDDs?
    - SSDs: Good Latency, Good throughput, Large size
    - Memory: Great Latency, Great throughput, Small size
    - Cache: Amazing Latency, Amazing throughput, Tiny size
- Python File API: Stream of Bytes
  - How is the translation implemented?
    - Read a page at a time, scan through it, then read the next page.
  - Optimization idea: Pre-buffer (parallelize IO and compute tasks)
- For x in Stream API: Stream of Record strings
  - How is the translation implemented?
    - readline = buffer data until you hit a newline, return the buffer
  - Similar to record parsing… buffer until you hit a comma
- String parsing
  -

Optimization Ideas… i.e., Let’s reinvent CSV (and the script)

Idea 1: Normalize Column Widths

- Instead of delimiters, have each “field” located in a well-known range of bytes
  - Bytes 0-1 == ID
  - Bytes 2-9 == Name
  - Bytes 10-15 == Rank
  - Bytes 16-18 == Age
- Benefits
  - Don’t need split()
  - Don’t need field delimiters (save ~4 bytes/line)
  - Don’t need to parse irrelevant fields (e.g., bytes 1-2 of each line)
- Drawbacks
  - Need to know how big each column is… need a “Schema” to track this information.
  - Doesn’t quite work with variable-length fields (e.g., name, rank)
    - Need to allocate space for max record size
    - Need to include space to signal string size (e.g., ‘\0’ character)
    - What if max record size changes?
- Variant idea: Directory
  - Store field offsets in a fixed-size “header” for each row.

Idea 2: Pre-parsed fields

  - Store direct byte representation on disk
    - e.g., 41 == 0x00000029 == “\0\0\0A”
  - Benefits
    - Can be Faster (int(…) is slow)
    - Typically ints/floats are more compact
  - Drawbacks
    - Be careful: Int = 8 (or 4 on older machines) bytes
      - vs 2-3 bytes per number in the CSV file
      - More bytes = more IOs = more slower…
        - Tradeoff with performance improvement from removing int().
        - Usually not worth it, but depends on where the data lives (HDD vs Memory).
      - Idea: byte / short instead of int
        - … but need to know max number size.
  - Drawbacks
- **Idea 3: Rewrite the script**
  - ‘!= “Ensign”’ is more expensive than ‘> 25’ so put > 25 first.
  - Why is this allowed?
    - AND is commutative
  - Benefits
    - Faster
  - Drawbacks
    - … not really any (as long as you pre-parse)

What are some (other) things that we might want to do with a CSV file

- **Filter it**
  - How do we specify a filtering condition?
    - By Expression
    - Nth - Kth records
  - What do we need to know about the dataset?
    - Can we expect the structure to be regular?
    - Do fields follow common type patterns (e.g., dates, ints, etc...)?
    - Maybe we'd like to have names to address different columns by?
- **Transform it**
  - Pick out certain columns?
  - Compute new columns (e.g., Birth Year)
  - Again... what do we need to know about the dataset?
- **Summarize it**
  - For discussion later on
- **Repeatedly ask (different) questions**
  - Parse once, leave it in memory (if you can)
- **Modify it**
  - Add/Delete new columns?
  - Alter existing fields?
  - Add new rows?

Making the format write-friendly

- **Challenges**
  - Field sizes might change after updates
  - Field size statistics might change (e.g., max size)
  - Where do you insert new records?
    - Append to end?
      - But what if you need them in a specific order
    - Idea: Adapt record layout techniques to pages (i.e.,
      - Challenge: Need to leave open space in the file
      - Need a way to link pages together out of order

            - Hierarchy
            - Linked List
        - How do you delete records?
            - “Mark” records as deleted

Recap

The choice of storage format impacts performance

- **Store data in its native byte encoding**
- **Layout fields in predictable locations**
    - Standardize layout for all fields (if possible)
    - Use a directory header (if not)
- **Layout records in predictable locations in a page**
- **… but you need to store a record of how the data is organized… a “schema”**
    - How are pages organized?
    - How are records organized?
    - How are fields organized?
    - What is each field’s type (string, int, date, float, etc…)
        - Additional type information: How “big” is the field: see varchar / char
- **Tradeoff Questions**
    - Do you have variable length fields?
    - Do you need to modify data?
    - Do you need to insert data?
    - Do you expect random access or scans?
    - Does the data need to be kept sorted?

Know your Data Access Patterns:

- **Stream (aka iterator): a sequence of records that you can scan through once**
- **Buffer (aka array): a randomly addressable sequence of records**
- **Paged Access: Hierarchical access: “randomly” addressable blocks are expensive, once loaded accesses within a block are cheap**
    - Parallels: HDD->Mem (disk pages/blocks), SSD->Mem (disk pages/blocks), Mem->Cache (cache lines), HDFS (pages)

Know your Memory Hierarchy

- **Registers -> Cache (L1->L2->L3) -> Memory -> SSD -> HDD -> Network (Same Switch, Same Rack, Same LAN, WAN)**
- **Going left-to right:**
    - Data Volumes increase (good)
    - Latency/Throughput increase (bad)
        - They increase at different rates, which affects algorithm tradeoffs
    - Moving data between levels is EXPENSIVE
        - ***90% of databases is figuring out ways to avoid moving data between levels***