Design of Interview Question: A Log Simulator and Query Tool

Yuanda Xu

2015-05-07

History

1. 2015-05-07 Yuanda Xu Requirements and Clarification.
2. 2015-05-08 Yuanda Xu Architecture Design.
3. 2015-05-10 Yuanda Xu Detailed Design and Synchronization after coding.
4. 2015-05-16 Yuanda Xu Detailed Design update and Verification.

Abstract

The main task is to develop a simulator to generate logs in specific format, and develop a command line tool to implement query according to input parameters. The simulator will generate logs for 1000 servers and save all logs into a file under an assigned directory, and the command tool will print out results filtered by input parameters from all saved logs.

Requirements

1. Simulator:
   a) The simulator shall take directory as a parameter and generate log data into file(s) in that directory.
   b) To generate the logs for one day, for example 2014-10-31.
   c) The log data should follow following format:

timestamp	*IP*	*cpu_id*	*usage*
1414689783	*192.168.1.10*	*0*	*87*
1414689783	*192.168.1.10*	*1*	*90*
1414689783	*192.168.1.11*	*1*	*93*

   d) The timestamp should be Unix time stamp.
   e) The CPU usage should be a random numbers between 0% and 100%.
   f) The run the generator should be executed by running command "./generate.sh DATA_PATH"
2. Command line tool:
   a) The command tool shall also take a directory of data file(s) as a parameter
   b) It should enable user to query CPU usage for a specific CPU of a given server in a given time period.
   c) It is an interactive command line tool which read a user's commands from stdin.
   d) The tool should support two commands.
      i. "Query" command will print results to stdout with a syntax as below:
         QUERY IP cpu_id time_start time_end.
         Time_start and time_end should be specified in the format YYYY-MM-DD HH:MM where YYYY is a four digit year, MM is a two digit month (i.e., 01 to 12), DD is the day of the month (i.e., 01 to 31), HH is the hour of the day, and MM is the minute of an hour. For example:
         QUERY 192.168.1.10 1 2014-10-31 00:00 2014-10-31 00:05
      ii. "EXIT" command. It will exit the tool.
   e) Result printing format should be:
      (2014-10-31 00:00, 90%),
      (2014-10-31 00:01, 93%),
      ……
      (2014-10-31 00:06, 78%)

f) The tool may take several minutes to initialize, but the query result should be returned within 1 second.

Clarification of Requirements

1. The generator is a onetime program, will run to generate log data into file(s) and exits instead of always running like a service.
2. According to requirement description, log data should be written into **file(s)** under a specific directory. The example shows that a log file contains log data of multiple servers. Considering there are 1000 servers, which will produce 2000 lines log each minute, 2880000 lines log each day, **a reasonable conjecture is to save log data of each day into a separate file**. And the file name could be "YYYY-MM-DD.log", which enable us to filter irrelevant file(s) with same extension name.
3. The detailed date to generate could be hardcoded as "2014-10-31", but it's better to make it controllable for users.
4. For command tool, it could receive a random time range, if the range covers several dates, then theoretically the query program should check all existing file(s) to get all records covered by input time range.
5. For time format, UNIX time stamp is used for log data, however, human readable string in format "YYYY-MM-DD hh:mm" is used for both input parameters of generator and command tool and output results. The time presented in this format should be local time, in which time zone difference has already been involved.
6. As the example shows, when query records from start time to end time, records of start time should be printed, while records of end time should not be included.
7. According to original requirements, the CPU load should be a random number between 0 and 100. Following practical experience, both 0 and 100 should also be involved, therefore the range of CPU load is [0, 100].
8. Time consumption is required for Query command, it's better to provide detailed measurement of time consumed for both Initialization and Query.

Architecture

1. Generator:
   a) According to 2 in Clarification of Requirements, a strategy that writing log data of each day into a separate file is adopted. With consideration of 3 in Clarification of Requirements, a potential needs could be to generate log file(s) for several days. Therefore a compatible design of either taking only one parameter of path, or taking multiple parameters that defines a time range which might cover several days would be more beneficial.
   b) To improve efficiency while writing multiple log file(s), multi-thread programming is introduced. Each thread will be responsible for writing log data of one day into one specific file.
   c) In default if there is no start/end date/time inputted, there will be only one thread, and the default time range to generate log data is from 2014-10-31 00:00 to 2014-10-31 23:59. If user inputs self-defined start/end time, that default range will be replaced by the range defined by user's parameters.
   d) For each log file to be generated, appending writing is not considered since this is a onetime running program. Each time generate a log file, all contents in this log file will be erased and been written with totally new log data if this log file already exists.
   e) The content of log data will be generated with time order, that is, logs with later timestamps will be written after written after logs with earlier timestamps.
   f) For the same timestamp, log data is written in the order server IP increases. For example, for the same time 2014-10-31 01:01, log of server 192.168.1.10 will appears earlier than log of server 192.168.1.12.
   g) For log of the same server at the same time stamp, record of CPU #0 will be written ahead of record of CPU #1.

2. Command Line Tool:
   a) There are several strategies to load log data for this command line too. One strategy is to dynamically load log file(s) according to user defined start date/time and end date/time and then parse all data for queries.
   b) Therefore multiple file(s) could be stored in the directory user assigned.
   c) Once receive "EXIT" request, all resource allocated will be released and process of command tool will be closed.

3. Performance
    a) To make query as quick as possible, a feasible way is to pre-process log file(s) and store all necessary data in data structure. When a query comes, results will be obtained directly from data structure in memory instead of open log file, read content, and parse data.
    b) Since the command tool needs to support both QUERY and EXIT commands, it should not exit each time and keep taking new commands after finish current QUERY command.
    c) Nested performance statistics would be helpful to evaluate time consumption of each QUERY command. A timestamp should be obtained after taking request of QUERY, and the ending timestamp should be recorded when all result output is finished for this QUERY request. The distance from beginning to ending is treated as time consumed by this QUERY command.
    d) For comparison, similar statistics will be implemented for both log generation and command tool initialization.
4. Exceptions
    a) Will check path/file(s) accessibility and provide protection while creating file(s).
    b) Will check all input parameters, either missing parameter, incorrect characters in parameters will be reported.
    c) In case the information is not complete or with invalid characters, system should mark those records as "N/A" and it will also be returned if this record is covered in the time period of QUERY command.
    d) Should check to avoid memory leaking.
    e) Help information will be printed if user only type in a single command (not request of QUERY or EXIT in command tool) without parameters.

Detail Design

1. Generator:
    1. In main function, there is only one input parameter, the path where to store log file(s), thus the main function will try to access this path, and quit directly with indications if failed.
    2. According to "1 b)" in Architecture, multithread is introduced to handle case of generating log data of multiple days. To transfer start/end time of each day to each thread, a vector is used. Each entry of that vector contains a pair of start/end time for an individual day. Each time a new thread created, a distinct pair start/end time will be transmitted as its parameter.
    3. In thread function, it will follow the given format and generate log data for all 1000 servers in each minute. All log data in one thread will be saved into one file since they are all about the same day.
    4. By calling function "gettimeofday()", both beginning and ending timestamp can be obtained. When all log data file(s) are created, generator will exit after printing both job done information and performance statistics.

2. Command Tool:
    1. Similar to generator, it takes only one parameter "path". The main function will also try to access this folder to make sure it exists and accessible.
    2. Better to make codes for entering sub-directory recursively ready, could be inactivated (masked).
    3. The command tool need do pre-processes before taking QUERY or EXIT command. The pre-process is to read all log files under the given path, and load all records into data structure.
    4. A map is used to store all log records of one server, which will enable us to get O(1) complexity while access one record from data structure.
    5. All maps of total 1000 servers are stored in a upper layer map file, which would enable us to access a server's map data structure in O(1) complexity.
    6. Since there is no strong time durance limitation about initialization, multi thread is not used for multi files reading within limited developing time. The performance could be improved by introducing multi threads if there are many log files to read.
    7. For QUERY request, there are several parameters, thus a complicated input check function is designed to check each parameter for both format and semantic. Indications of wrong parameter will be printed if invalid input found.
    8. For EXIT request, will simply release resources and exit while command tool.

3. Class logFile:
    1. A logFile class is designed for log file operations. The command tool will call methods of this class to manipulate log file(s).
    2. All parsed log data will be stored inside this class as private data and won't be able to access directly from outside. The structure used to store all log data, as previously mentioned in "Detail Design 2", is a two level nested map. Detailed usage of this data structure will be described in next section.
    3. Methods provided by logFile class includes:
        - loadLog: to load one logFile with assigned file name, parse all records in this file and then store all parsed data into data structure.
        - writeLog: take detailed information of one log and write it into assigned log file.
        - queryLog: take detailed start/end time, server IP, cpu Id, and check corresponding records in private data structure, generate result vector according to all records meets the query conditions and return this result vector to outside caller. For more convenience, the result vector is a string vector, therefor the outside caller can directly print it out without any extra processes.
    4. Besides methods mentioned above, the class also needs several private functions. For example, getLineData() which takes one record line in log file(s) and parse IP, timestamp, CPU load from record line string.

4. Data Structure:
    1. For each server , a structure RCDDT is used to store the detailed CPU load data in each minute.

```
typedef struct recordData
{
    int cpuLoad_0;      //CPU load data of CPU #0
    int cpuLoad_1;      // CPU load data of CPU #1
} RCDDT;
```

    2. As description above, for a single server, all records of each minute are stored in a map:

```
typedef std::map<time_t, RCDDT> MYRCD;
```

    Here the key of map is UNIX timestamp, which can be treated as an integer therefore can be used to locate a detailed record (value in this map) of a specific minute quickly.
    3. The upper layer of the nested map is designed as:

```
typedef int IPADDR;
std::map< IPADDR, MYRCD > allRecords;
```

    The value is the map which contains all records of a single server.
    The key in this map is actually an integer, which is directly transmitted from IP address:
        - The IP address contains four sections, and each section varies from 0 to 255.
        - An integer contains four bytes, each bytes can be a value from 0 to 255.
        - Thus an IP address can be transmitted to an integer by put value of each section into each byte of an integer.
        - By this way, the integer transmitted from IP address is definitely a unique value, won't be duplicated with other IP address.
        - And we can use this transmitted integer as key in the upper layer map to locate corresponding value, which is the map MYRCD defined above.

5. Common codes:

    There are some common processes used frequently in both Generator and Command Tool, it's better to put those codes into a comm.cpp file and build it a library. Processes includes:
    1. Check process of various data, such as date, time, IP address.
    2. Need handle transfer both local time to UNIX timestamp, and from regular time to daylight saving time.
    3. Translation among different data format, for example, UNIX timestamp from/to local time, integer from/to string

Test Plan

1. Should test inputs with both valid and invalid contents, and also empty input for both Generator and Command Line Tool.
2. Should test with different date/time range while generating log file(s).
3. Should test with valid/invalid/empty input for QUERY request in Command Line Tool.
4. Should test different data/time range for QUERY request, including normal cases and corner cases.
5. Should verify the output results are correct or not, both in data and in format.
6. Should record performance statistics of all tested cases.
7. Should check exception of input file not existing, or output file's folder un-writable.

Remark

1. An assumption is treated as normal in this project: the IP addresses of all servers are continuous.
2. Since time always goes in one direction, the timestamp in log data generated will only increase monotonously.