CS742 Assignment One

Florian Suess

Tuesday 8th August, 2023

We answer the following questions within the context of the given paper "Workload Characterization of a Large Systems Conference Web Server" by Mahanti et al.[1].

(Q1) Measurement approaches

For clarity, our response is grounded in the definitions provided by RFC7799[2]. The study employs two distinct measurement techniques:

- Examination of the structured server logs created during the study's duration.
- Google Analytics (GA) data—a web analytics service that utilizes a "page tagging" method to gather information about website traffic and user behaviors.

The **analysis of structured server logs is unambiguously a passive measurement**. This conclusion stems from the inferred asynchronous (likely driven by buffer flushing) process of logging data locally. This procedure, inherently, doesn't influence the server-side network IO.

The classification of GA is more contentious. In our course, GA was characterized as an active measurement tool, primarily because its implementation affects edge network transmissions to relay data to GA's analysis server. However, it's crucial to note that the network packets themselves aren't the crux of the measurement. The actual metrics of interest are predetermined by the page tagging mechanism, making **GA fundamentally a passive measurement tool**. To counter this assertion by suggesting GA as active would lead to a paradox. Contemporary server logs, in many production settings, offload logs to persistent storage systems or platforms optimized for text-search, such as Kibana or OpenSearch. This data transfer, akin to GA's process, introduces edge network traffic. Hence, classifying GA as active would inadvertently imply the same for these modern server logs.

Furthermore, Google Analytics supports the bulk uploading of aggregate result[3]. This feature allows data to be collected, aggregated, and then uploaded at specific intervals rather than in real-time. Such a mechanism accentuates

the passive nature of the measurement, as data is collected without continuous network interactions. This mode of operation reduces the immediate impact on the edge network and underscores GA's capability to function predominantly as a passive measurement tool.

(Q2-3) Measurement vantage points

(Q2) Type of measurement vantage used

Given the client-side and server-side source of both measurement mechanisms, we can confidently narrow say these measurement vantage points to network edge as opposed to core.

(Q3) Quantity of viewpoints considered

The server-side logs provide a single viewpoint, whilst the client-side measurements provide a very large quantity of viewpoints as most clients (with JS execution enabled) visiting will collect their own metrics.

(Q4) Hardware and software tools used

For metric collection and aggregate metric analysis; there were no physical measurement devices deployed. The logs can be assumed to be a natural bi-product of the server running that we collected after the study for analysis using some sort of programming language and GA is by definition a software as a service offering.

(Q5) Online and/or offline analyses performed

For all practical purposes, metrics collected were analysed post the server study period which as per lecture posits the "analysis" done falls into the "offline" criteria. This is at mercy to the implementation details of GA however, perhaps there is real-time analysis done on the client side that aggregates results before shipping them off to GA analytics platform, in which case implies a component of "online" analytics.

(Q6) Attributing metrics to Active and/or Passive measurements

The result in Q1 moots this question, as we have asserted that both measurement mechanisms used are considered "passive". Implies that all metrics materialised in this study are a result of "passive" measurements.

(Q7) Modern approach on workload characterization of "foreign" internet servers

When dealing with systems, especially those we do not own or have direct access to, it is essential to respect privacy and legal boundaries. Any unauthorized access or data collection may infringe on privacy rights and laws, leading to both ethical and legal consequences.

Given this ethical standpoint, if we aim to perform a workload characterization study of a server we cannot access, we must adopt a non-intrusive approach, ensuring we don't infringe on any rights or disrupt the server's normal operations.

1. Passive vs. Active Measurements:

- **Passive Measurement:** Since direct access to the server is restricted, passive measurement becomes the primary choice. This approach involves observing traffic without actively injecting or altering any data.
- **Active Measurement:** This would involve sending traffic or requests to the server to monitor its responses. Given our lack of access and the need for an ethical approach, it's prudent to avoid active measurements unless we can guarantee they won't disrupt or degrade the server's performance.

2. Vantage Points:

The location from which we observe traffic can provide different insights:

- **Core Network:** Observing from a core network location gives a macroscopic view of the traffic, capturing large flows and general trends. However, this vantage point might miss out on fine-grained details.
- **Edge Network:** Being closer to the end-users and the server, the edge network provides a microscopic view. It's ideal for capturing detailed interactions but might not reflect the broader patterns. However, given our lack of access to the server and end-user, placing observation tools near the edge may be too difficult.

3. Utilize Public Data:

Often, servers or websites have public metrics, logs, or performance dashboards. While these might not provide a comprehensive view, they can offer a starting point or a means to validate our passive measurements.

4. Collaborate with Others:

Engaging with the broader research community can provide additional insights. Others might have partial data, past studies, or expertise that could benefit the characterization study.

5. Understand Limitations:

Whatever our approach;W ithout direct access, our study will inherently have limitations. Recognizing these and being transparent about them ensures that our findings are taken in the correct context.

The following sections provide an approach for the analysis of the `ls -lR` output file. Full set of scripts available are provided [1].

Notes on querying `ls` output methodology

The following questions given to me suggest the use of a queryable interface can be advantageous, whilst the plotting requirements to me speak intervention needed by something like `python`'s `matplotlib` hence a well interoperable store is favourable. We are dealing with a very small amount of data. With all of this in mind; we can sufficiently depend on something lightweight like a local `sqlite` instance. Hence we will;

1. Process the `ls` output into a `csv` format.
2. Spin up a `sqlite` instance and load a simple table with this `csv`.
3. Perform migrations if nessecary (such as extracting the file extension).

One can utilize `awk`[2] to parse this output into a `csv` with the following two pattern match blocks.

```
/:$/{gsub(":",""); dir=$0; next}
$9{print $9 "," dir "," $5 "," $6 " " $7 "," $8}
```

First one `/:$/` matches any lines with a colon and updates the `dir` variable for interpolation in the next block. Second match ensures there's 9 string parts in the line, if so, converts the ls output line into a csv line of shape; "file name", "dir", "bytes", "day month", "year".

This somewhat awkward looking choice is made because of some lazy date stamps given;

```
-rw-------   1 carey    www2007       0 Aug 28 13:24 allfiles.out
```

Without a year provided, we need to infer it. We shall assume that year represents 2007 (end of study period). We write a simple `go` script that looks for a colon in that field, if so, replace it with 2007. We then combine the "day month" and fixed "year" and parse the date into a unix time stamp[3].

In our `sqlite` you can bulk upload via `.mode csv` specifying the seperator as a comma into the following table;

```
CREATE TABLE files (
    file_name TEXT,
    directory TEXT,
    bytes INTEGER,
    last_modified INTEGER
);
```

[1] Private link here, also feel free to reach me and we can work through these together.

[2] Would in practice probably use something more elaborate like a `go` script.

[3] `sqlite` doesn't offer a native date type hence we will use the `INTEGER` type in combination with the `date` function for readability

Q8

Quantity of different files

```
sqlite> SELECT COUNT(*) FROM files;
6062
```

Additional commentary; inline with `Table 1` total presented in the paper.

Aggregate sum of file bytes

```
sqlite> SELECT SUM(size_in_bytes) from files;
1107569732
```

Which comes to about 1.107GB.

Additional commentary; inline with `Table 1` total size presented in the paper.

Q9

Largest file on site

```
sqlite> SELECT MAX(size_in_bytes) FROM files;
59381544
```

Which comes to about 59.382MB.

Number of empty files

```
sqlite> SELECT COUNT(*) FROM files
WHERE size_in_bytes = 0;
15
```

Smallest non-empty file

```
sqlite> SELECT MIN(size_in_bytes) FROM files
WHERE size_in_bytes != 0;
2
```

Q10

Mean file size

```
sqlite> SELECT AVG(size_in_bytes) FROM files;
182706.98317387
```

Which comes to about 182.707KB.

Additional commentary; inline with `Table 1` average size presented in the paper.

Standard deviation of file size

```
sqlite> SELECT
    SQRT(
        AVG(size_in_bytes*size_in_bytes) -
        (AVG(size_in_bytes) * AVG(size_in_bytes))
    )
FROM files;
2192180.71338199
```

Which comes to about 2.192MB.

Additional commentary; considering the average file size is 182KB, this speaks to a wider spread distribution with our first bit of evidence suggesting a positive skew.

Median file size

Little more tricky given `sqlite3`'s limitation of no built-in support for median. Recall;

```
sqlite> SELECT count(*) from files;
6062
```

Hence the median by definition is the average of the middle two file sizes when ordered.[4]

```
sqlite> WITH sorted AS (
    SELECT size_in_bytes,
           ROW_NUMBER() OVER(ORDER BY size_in_bytes) AS rn,
           (SELECT COUNT(*) FROM files) AS cnt
    FROM files
)

SELECT
    AVG(size_in_bytes)
FROM sorted
WHERE rn IN (cnt / 2, cnt / 2 + 1);
1471.0
```

Which comes to about 1.471KB.

Additional commentary; From this result, we can compare the average and median file size. With the median being significantly smaller than the average, we have further evidence suggesting a positively skewed distribution.

File size mode (most frequently occurring file size)

```
sqlite> SELECT size_in_bytes, COUNT(*) AS frequency
FROM files GROUP BY size_in_bytes
ORDER BY frequency DESC, size_in_bytes ASC
```

[4] We could dynamically calculate the median using a `CASE` on file count also.

```
LIMIT 1;
4096|102
```

4096B sized files occured a whooping 102 times!

Notes on the plotting methodology

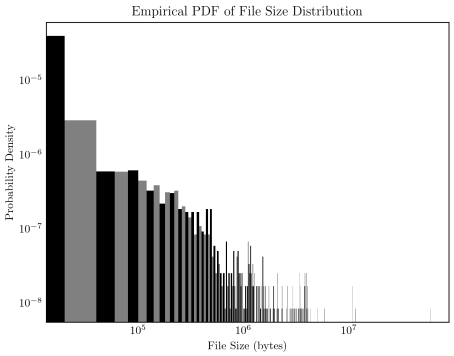
`python` in my eyes is the defacto wizard when it comes to graphing due to well-traversed libraries such as `matplotlib`. Hence we will use the standard library `sqlite` module in `python`. Scripts are here to use for further exploration of the plotted graphs including the offering of a interactive/zoomable view [5].

Q11 - Plotting file size distribution

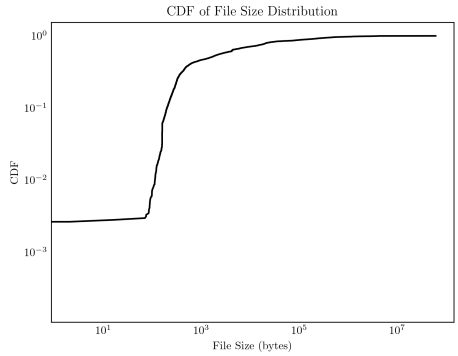
We effectively just handle the full list of bytes via;

```
SELECT size_in_bytes FROM files
```

(a) Probability density graph



(b) Cumulative distribution graph



We used a log scale on both of these graphs in order to effectively capture and graph the positively-skewed distribution. **Additional commentary;** this wasn't a surprise considering the analytics we gathered prior to mapping these results out. The CDF plot in particular provides a bit of an insight into the typical file size appearance, most files sitting in the $10^2 B$ range, hence the elbow shape of the curve.

Q12 - Table of top 10 file types

We can broadly determine the filetype based on the extension of the filename in our database (defined as the letters after the last dot). Hence we perform the following migration;

[5] Private link here, also feel free to reach me and we can work through these together.

```
ALTER TABLE files ADD COLUMN extension TEXT;

UPDATE files SET extension =
REPLACE(file_name, RTRIM(file_name, REPLACE(file_name, '.', '')), '');
```

The latter migration is a common workaround for `sqlite`'s lack of a last index function[6], but where effectively extracting the substring from the position of the **last** dot.

```
UPDATE files SET extension = LOWER(extension);
```

[6] `https://stackoverflow.com/a/38330814`

```
WITH FileTypeAggregates AS (
    SELECT
        extension,
        COUNT(*) as file_count,
        SUM(size_in_bytes) as total_bytes
    FROM files
    WHERE extension IS NOT NULL AND extension != ''
    GROUP BY extension
),

Top10 AS (
    SELECT extension, file_count, total_bytes
    FROM FileTypeAggregates
    ORDER BY file_count DESC
    LIMIT 10
),

OtherAggregates AS (
    SELECT
        'OTHER' AS extension,
        SUM(file_count) as file_count,
        SUM(total_bytes) as total_bytes
    FROM FileTypeAggregates
    WHERE extension NOT IN (SELECT extension FROM Top10)
),

Combined AS (
    SELECT * FROM Top10
    UNION ALL
    SELECT * FROM OtherAggregates
),

TotalAggregates AS (
    SELECT
        SUM(file_count) as total_files,
        SUM(total_bytes) as total_bytes
    FROM Combined
)

SELECT
    c.extension AS "Type",
    c.file_count AS "Count",
    printf("%.2f", (c.file_count * 100.0 / ta.total_files))
        AS "Count Percentage",
    c.total_bytes AS "Total Bytes",
    printf("%.2f", (c.total_bytes * 100.0 / ta.total_bytes))
        AS "Percentage of Bytes"
FROM Combined c, TotalAggregates ta
ORDER BY CASE WHEN
c.extension = 'OTHER' THEN 1 ELSE 0 END, c.file_count DESC;
```

This SQL query aggregates file statistics by extension, identifying the top 10 file extensions by count. It provides a breakdown of file counts and their sizes both for these top 10 extensions and for all other extensions combined under "OTHER". The final output displays each file type, its count, its percentage of the total file count, its total bytes, and its percentage of the total bytes, with the "OTHER" category displayed last. Now with `.headers on` and `.mode column` we can yield the following table.

Type	Count	Count Percentage	Total Bytes	Percentage of Bytes
png	2339	38.58	12220808	1.10
jpg	905	14.93	337287389	30.45
gif	656	10.82	3003468	0.27
php	521	8.59	1947964	0.18
pdf	362	5.97	545146440	49.22
txt	249	4.11	32067913	2.90
svn-base	190	3.13	437769	0.04
svn-work	190	3.13	29472	0.00
wmz	133	2.19	81007	0.01
html	114	1.88	6767416	0.61
OTHER	403	6.65	168580086	15.22

Additional commentary: what we find here is enlightening with respect to filenames. `svn-base` and `svn-work` extensions seem unfamiliar, and upon a re-investigation of the raw `ls` output we see this extension usually added to the end of other file extensions such as `html` and `png`. In usual circumstance we'd query the web server insiders the origin of this extension to build a better understanding.

We can also immediately start building contrast with the file type breakdowns provided by the research paper re: `Table 1.` The ordering and relative size of each category presented here is inline with what is presented in the paper, however, there seems to be an omission of these extensions such as `svn-base` and `svn-work` spoken to above.

Q13 - Plotting file size distribution restricted to `./papers` and `./posters`

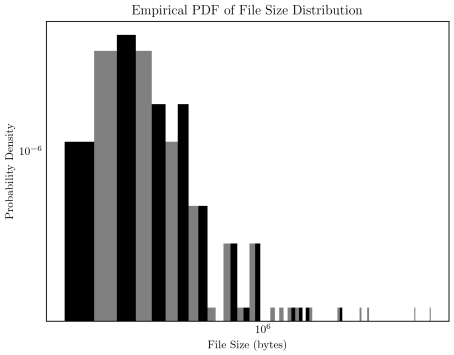
In contrast to Q11 when we had to plot PDF functions; we effectively just handle the list of bytes via additional `WHERE` clauses narrowing down the files to within a certain directory.

```
SELECT size_in_bytes FROM files WHERE directory = "./papers"
```

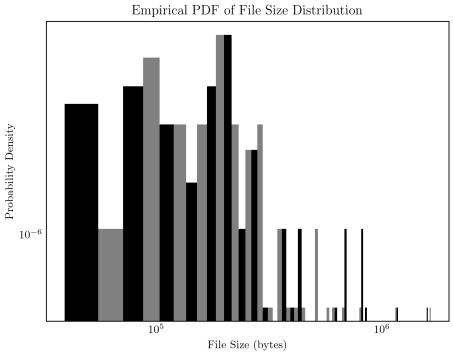
And

```
SELECT size_in_bytes FROM files WHERE directory = "./posters"
```

(a) Papers

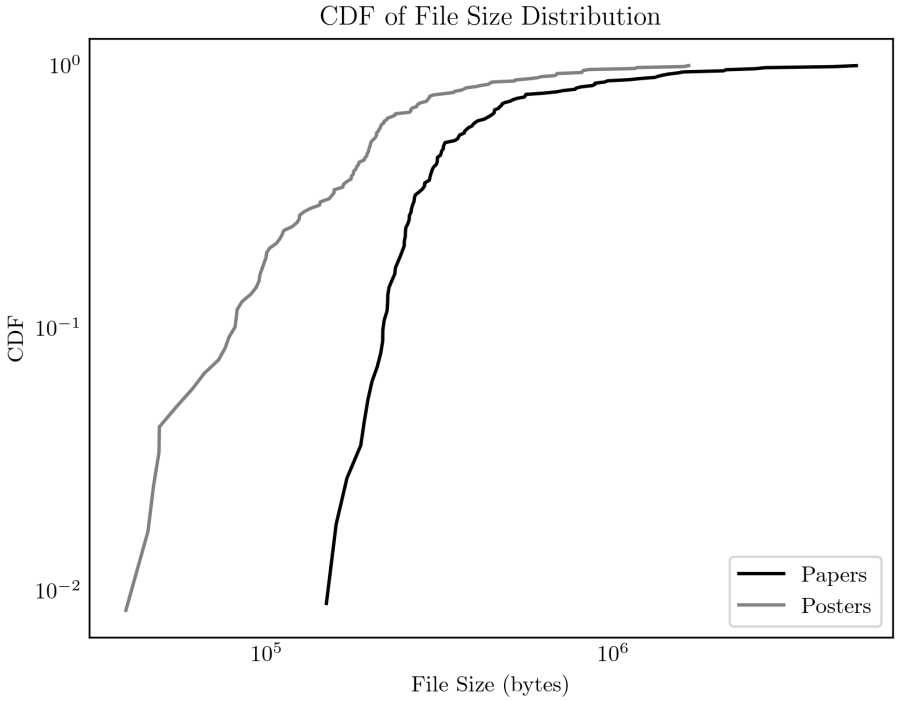


(b) Posters



Separate plots for PDF's of these two directories worth of rows re: size in bytes.

Additional commentary: These PDF functions show a prevalent file size distribution difference. With `./papers` showing a tendency to following a very normal positively skewed distribution, we have something quite different for `./posters`. There is some intuitive lense that could be applied here, that there are significant differences in file types here. `./papers` perhaps being more textual which has a tendency to have sizes that correlate to length of paper, an understandably normal distribution, where `./posters` containing images and renderings that utilise richer file formats. This could be validated by a filetype investigation per directory.



Note: Both of these distributions are positively-skewed hence the log scale approach applied once again.

Q14 - Analysis of file age

Hydrating age of each file

As per original bulk upload, we ensured the date is correctly captured and passed into a `Unix epoch` timestamp (including the edge case timestamps provided[7]) hence we're pre-prepared for this question. Thus when querying for human-readable

The oldest file

```
sqlite> SELECT
    directory || '/' || file_name,
    date(MIN(last_modified), 'unixepoch')
FROM files;
./images.bak/acmlogo.gif|2006-02-23
```

The newest file

```
sqlite> SELECT
    directory || '/' || file_name,
    date(MAX(last_modified), 'unixepoch')
FROM files;
./index.php|2008-02-02
```

Additional commentary: very inline with what we'd expect re: `php` web server files being frequently modified.

Mean file age

```
sqlite> SELECT
    date(AVG(last_modified), 'unixepoch')
FROM files;
2007-02-03
```

[7] parsing the "13:24" included timestamp to be contextualised to the year 2007

Median file age

Recall as per Q10;

```
sqlite> SELECT count(*) from files;
6062
```

Hence the median by definition is the average of the middle two epoch timestamps when ordered.[8]

```
sqlite> WITH sorted AS (
    SELECT last_modified,
           ROW_NUMBER() OVER(ORDER BY last_modified) AS rn
    FROM files
)

SELECT
    date(AVG(last_modified), 'unixepoch')
FROM sorted
WHERE rn IN (rn / 2, rn / 2 + 1);
2006-02-23
```

Mode file age

```
sqlite> SELECT
    date(last_modified, 'unixepoch'),
    COUNT(*) AS frequency
FROM files GROUP BY last_modified
ORDER BY frequency DESC, last_modified ASC
LIMIT 1;
2007-02-19|1159
```

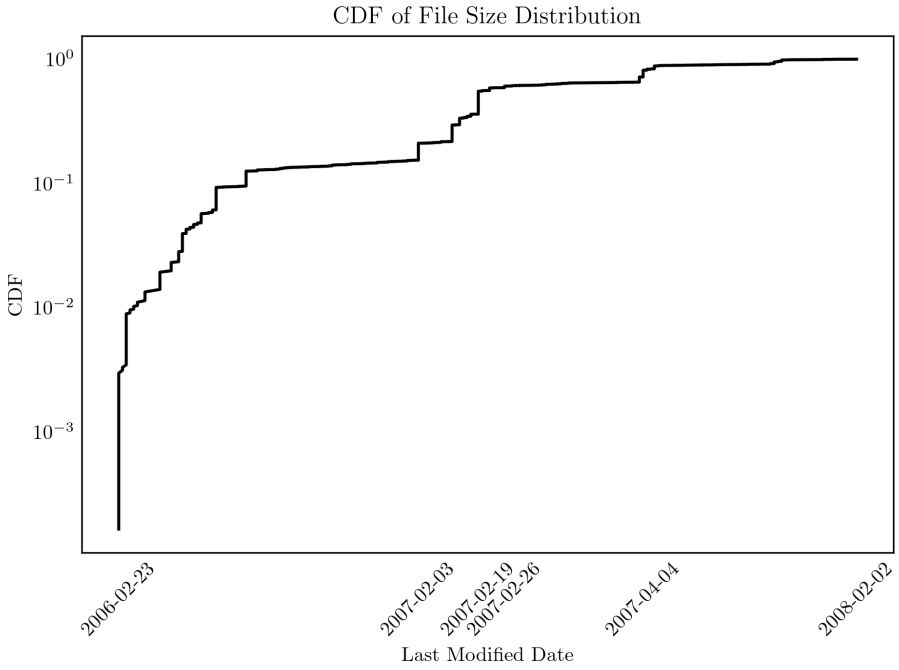
Additional commentary; speaks to the commentary provided in the paper re: conference paper submission deadline.

[8] We could dynamically calculate the median using a `CASE` on file count also.

Q15 - Cumulative distribution function graph of file age

We effectively just handle the full list of rows returned via;

```
SELECT last_modified FROM files
```



We used a log scale on both of these graphs in order to effectively capture and graph the positively-skewed distribution while also converting the unix timestamps to something more familiar for the x-axis labels.

Additional commentary: we can see a lot of files added early on, perhaps speaking to the nature of "bootstrapping" a web server. A relatively calm period up until the week of the paper submission deadline as referenced in the paper. Relatively calm period before the conference date itself. Then a calm period there after, last file modified as found earlier being a `index.php` update.

References

[1] Aniket Mahanti, Carey Williamson, and Leanne Wu. "Workload Characterization of a Large Systems Conference Web Server". In: Seventh Annual Communications Networks and Services Research Conference, 2009.

[2] Internet Engineering Task Force (IETF). *Active and Passive Metrics and Methods (with Hybrid Types In-Between)*. RFC RFC 7799. Available online at `https://www.ietf.org/rfc/rfc7799.txt`. IETF, 2016.

[3] Google. *Batching Requests*. Accessed: 2023-08-08. 2023. URL: `https://developers.google.com/analytics/devguides/reporting/core/v3/batching`.