IIT Madras
BSc Degree

Performance

Performance

- Speed
  - Single user performance
  - User experience
  - Network
- Scaling
  - Multi-user performance
  - Server load
  - Cost

Speed

User Experience

UI: User Interface

vs

UX: User Experience

Speed

- Quick response
  - Waiting for page load / response is bad / confusing
- Contributing factors
  - Network: mobile vs broadband, distance from server, congestion
  - Number of requests
  - Size of response
  - HTTP 1 vs 2 (vs 3?) - pipelining, keepalive, optimizations
  - Compression

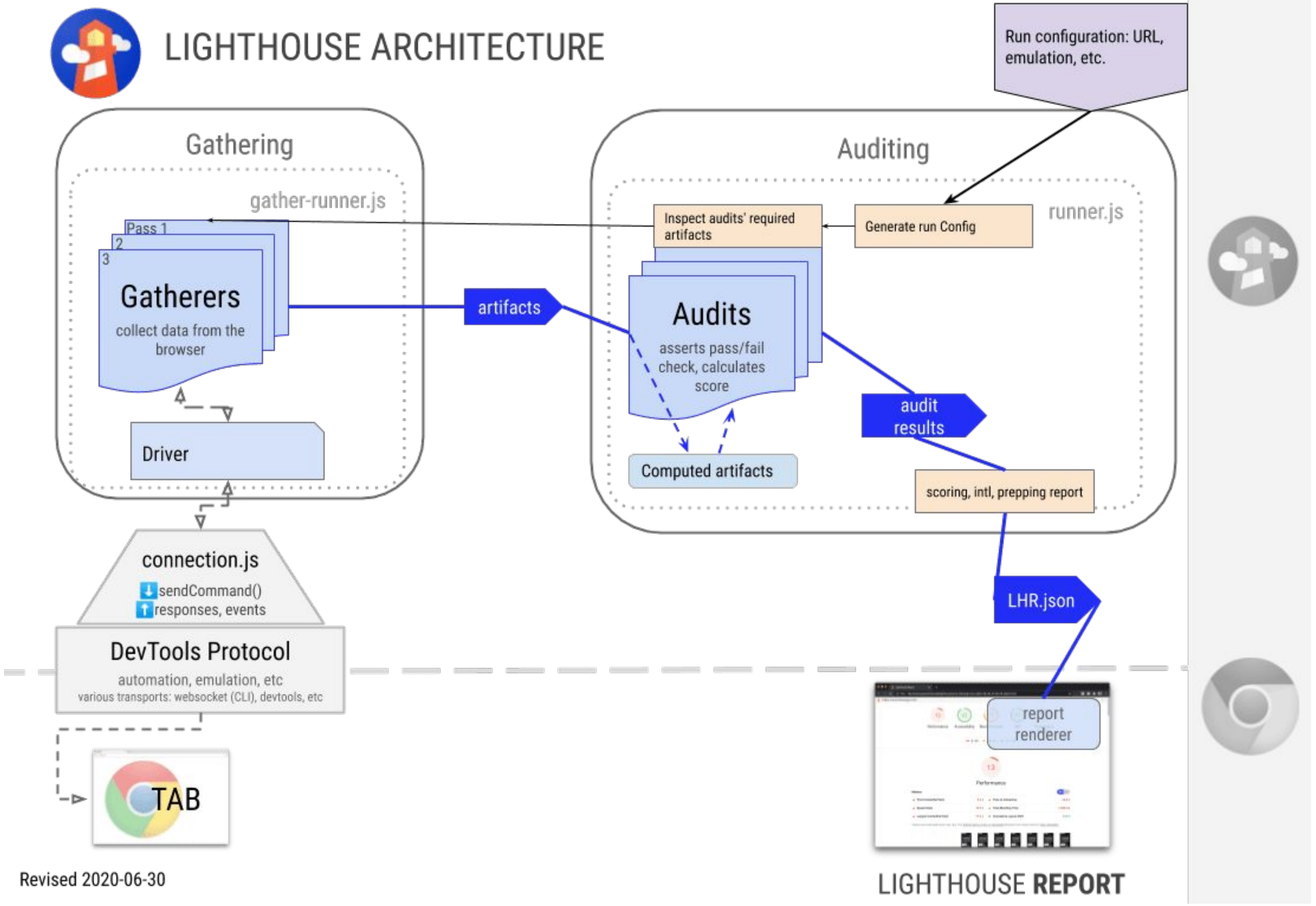
Tools

“You can’t optimize what you can’t measure”

- source unknown, attributed to Peter Drucker among others

How do we measure performance of a website?

- Controlled and measured access to all elements of site
- Lighthouse: popular tool built in to Google Chrome



https://github.com/GoogleChrome/lighthouse/blob/master/docs/architecture.md

What does it do?

- Loads a page and all resources while monitoring time taken
  - Flush caches if necessary
- Measure time, memory metrics
- Emulate network bottlenecks, throttling etc.
  - Separate projects on how to make this realistic!
- Emulate devices: mobile vs desktop
- Compute weighted average score
  - Performance
  - Accessibility
  - Best practices
  - Search Engine Optimization (SEO)
  - Progressive Web App (if relevant)

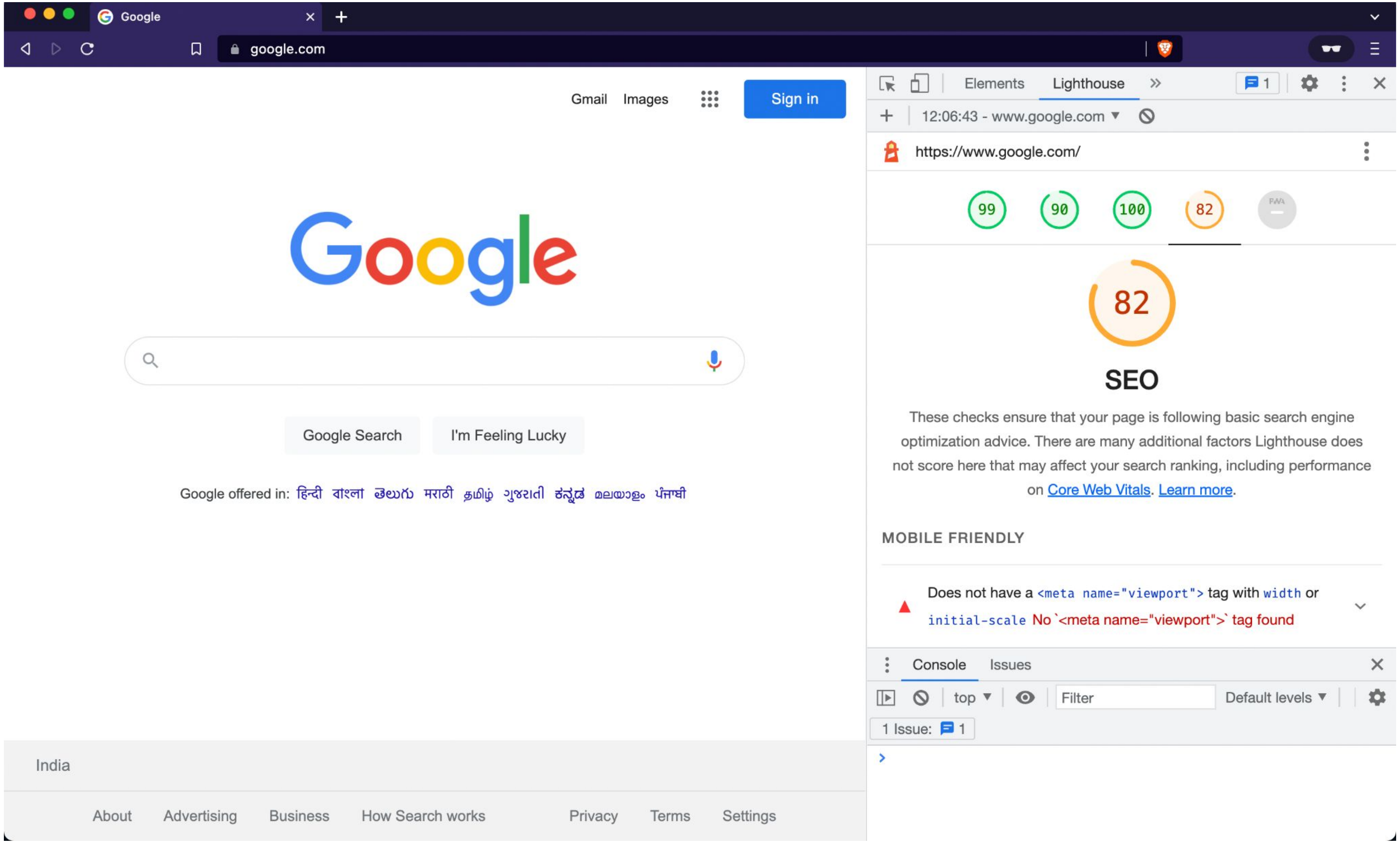
Performance Metrics

- First Contentful Paint
  - Something displayed on screen
  - First meaningful paint - useful info
- Speed Index
  - Capture video of page loading and analyze
- Largest Contentful Paint
  - “Most of page” rendered
- Time to Interactive
  - Usability of page
- Total Blocking Time
  - Time blocked from responding to user
  - Mostly JS related
- Cumulative Layout Shift
  - Rearrangement of page after some more data loaded - shift existing parts

https://web.dev/performance-scoring/

Other parameters

- Accessibility
  - WCAG, alt links/text, contrast, screen-reader friendly etc.
- Best practices
  - Image resolutions, use HTTPS, allow paste into password fields…
- Search Engine Optimization
  - Document meta info; title, links; alt



https://www.google.com/

82

SEO

These checks ensure that your page is following basic search engine optimization advice. There are many additional factors Lighthouse does not score here that may affect your search ranking, including performance on Core Web Vitals. Learn more.

MOBILE FRIENDLY

Does not have a `<meta name="viewport">` tag with `width` or `initial-scale` No `<meta name="viewport">` tag found

Problems

- Specific set of checks - may not reflect real useability
- Gmail has poor performance score, but is overall a good app
- https://www.matuzo.at/blog/building-the-most-inaccessible-site-possible-with-a-perfect-lighthouse-score/?sfw

Scaling

Static vs Dynamic

Static

- Static text page
- Blog
- Predominantly content that is user neutral

Examples:

- Wikipedia
- W3C guidelines
- MDN

Static vs Dynamic

Static

- Static text page
- Blog
- Predominantly content that is user neutral

Examples:

- Wikipedia
- W3C guidelines
- MDN

Dynamic

- E-commerce
- Learning
- User customization

Examples

- Amazon.in
- Swayam
- Dream11

Scaling

- Response under load: requests per second
    - Constant high rate: Google search, Bing
- Response under sudden ***changes*** in load: rate of change
    - Changes due to circumstances: declare CBSE exam results, IPL final
- Predictable or not?
    - Exam results time known, IPL final time known: prepare in advance
    - Bowler about to take hat-trick, batsman about to score century

Components of an App

- Server:
  - Frontend server - clients connect, receive HTML + CSS + JS
  - Database server - stores models, gets connections from frontend
  - Load balancer - only passes requests to frontends
  - Proxy - handles some queries without loading server
- Network:
  - Mobile - speed, signal quality, movement, congestion
  - Broadband - shared connection, wire quality, upstream provider
- Application:
  - Data intensive
  - Image / script intensive
  - Browser / client functionality

Server: Load Balancing

- Load balancer:
  - Minimum functionality - purely forward requests
  - Round-robin, least load, …
- Commercial offerings
  - Amazon Elastic Load Balancing, Google Cloud Load Balancing, …
  - Distribute load across VM instances, IP addresses, containers, zones, …

Server: Proxy

- Intermediate layer between client and server

“Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable.” - R. Fielding, PhD dissertation

- Caching proxy can be inserted at various stages
    - Close to client for faster response
- Content Delivery Networks (CDN)
    - Form of proxy, but explicitly encoded in URL

Server: DB

- Choice of DB:
  - SQLite, PostgresQL, MySQL, Oracle
  - MongoDB, Cassandra, Amazon Dynamo
- Scaling
  - SQLite difficult to scale for writing, great for reading
  - Synchronization, Replication issues

Server: Language

- Interpreted languages easy to develop
  - Python, JS
- Compiled languages harder, but much faster
  - C, Golang, Java
- Threading and Asynchronous capabilities
  - Goroutines, JS Async, Erlang/Elixir
- Programming paradigms
  - Functional (Haskell/Hasura, Elixir)
  - Declarative
  - Imperative

Monitoring and Measuring

- Highly application and architecture specific
- Server logs used for post-facto analysis
- Tools for live monitoring:
  - ElasticSearch / LogStash / Kibana (ELK stack)
  - Grafana + InfluxDB + Prometheus
  - Overall aspects of server performance, fine grained monitoring

Summary

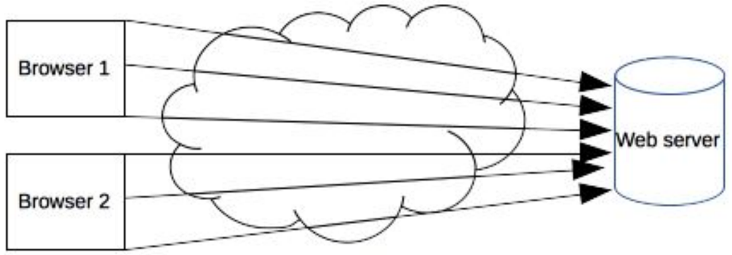
- Measure, then Optimize
- Choice of language, DB, service provider:
    - May not be in your control
    - Adapt to requirements
- Developer
    - Control resource structure, number of requests
    - Images, payload sizes
    - Cacheability

Caching

What is Caching?

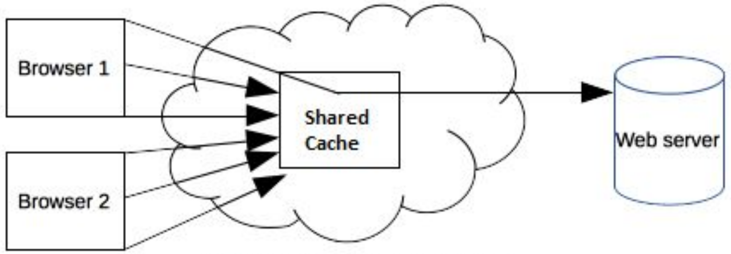
- Store response to requests so they can be reused for future requests
- Largely developer controlled
- Where:
    - server
    - proxy frontend
    - network router
    - client

No cache



All identical requests are
going through to the server.

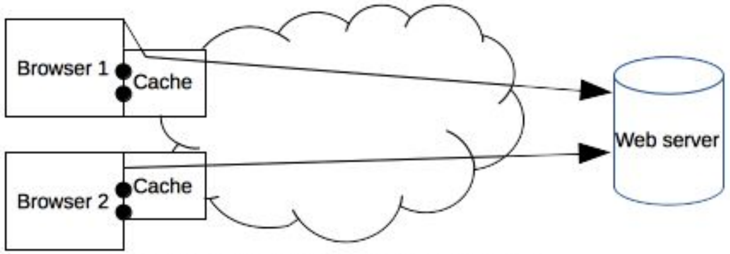
Shared cache



The first request is going through.

Subsequent identical requests are served
by the shared cache.
(more efficient)

Local (private) cache



The first request of each client is going through.

Subsequent identical requests are not even sent, but
served by the local cache.
(most efficient, except for first requests)

https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching

Server support for caching

- HTTP Headers: “cache-control”
  - Note that “no-cache” does not mean no caching…
  - max-age, expiry
  - can specify whether revalidation required etc.
- E-Tag: Entity tag header
  - Unique ID for a resource: cache can identify
- Freshness checking - estimating freshness in absence of Expires etc.
- Explicit revisions on resources:
  - https://cdnjs.cloudflare.com/ajax/libs/vue/3.2.31/vue.cjs.js

Is caching bad for website popularity?

- Reduces hits on server
- Traffic goes to cache / proxy instead of server

Bad way to approach the problem:

- Not all hits should be on server: cache as much as possible
- Use indirect approaches like analytics to update / track visitors

Flask Caching

- Module that integrates directly with Flask

```
@app.route("/")
@cache.cached(timeout=50)
def index():
    return render_template('index.html')
```

Note order: “cached” decorator applied to index, only then “route”

Cache Key

- Cache < - > Python Dictionary
- Key -> Value
- View functions:
  - Key - request path (route)

Cache non-view functions

```
@cache.cached(timeout=50, key_prefix='all_comments')
def get_all_comments():
    comments = do_serious_dbio()
    return [x.author for x in comments]

cached_comments = get_all_comments()
```

Memoization

- Memo: note to be remembered
- Memoize:
  - Create a note (cache entry) based on some function arguments
- Without function arguments, same as “cached”

Memo with function arguments

```
class Person(db.Model):
    @cache.memoize(50)
    def has_membership(self, role_id):
        return Group.query
                   .filter_by(user=self, role_id=role_id)
                   .count() >= 1
```

Jinja caching

```
{% cache 60*5 %}
<div>
    <form>
    {% render_form_field(form.username) %}
    {% render_submit() %}
    </form>
</div>
{% endcache %}
```

Caching Backends

- NullCache: don’t cache - just for testing
- SimpleCache: Local Python dictionary - not thread safe
- FileSystemCache: store to files on disk
- RedisCache (and variants):
    - store in Redis key-value
    - requires separate Redis instance, possible to reuse same server as Celery etc.

Summary

- Caching is under app developer control
- Implement wherever possible to improve performance
- Transparent vs Explicit
    - Client code needs to know about freshness OR
    - Client HTML points to new versions of resources
- Very important for scalability of applications
    - Core part of the REST architecture
    - Prefers static content over JS intensive dynamically generated content