The Watchman

Monday, 17 April 2023 | Edition 1 | Do not go gentle into that good night

wget is much faster than scp or rsync

@edent , 17/4/23

I needed to copy 3TB of data from my old homeserver to my new one. I decided to spend as much time "sharpening my axe" as possible. I spent ages dicking around with ZFS configs, tweaking BIOS settings, flashing firmware, and all the other yak-shaving necessary for convincing yourself you're doing useful work. Then I started testing large file transfers. Both `scp` and `rsync` started well - transferring files at around 112MBps. That pretty much saturated my Gigabit link. Nice! This was going to take no time at all... And then, after a few GB of a single large file had transferred, the speed slowed to a crawl. Eventually dropping to about 16MBps where it stayed for the majority of the transfer. I spent ages futzing around with the various options. Disabling encryption, disabling compression, flicking obscure switches. I tried using an SSD as a ZIL. I rebuilt my ZFS pool as a MDADM RAID. I mounted disks individually. Nothing seemed to work. It seemed that something was filling a buffer somewhere when I used `scp` or `rsync`. So I tried a speed test. Using `curl` I could easily hit my ISP's limit of 70MBps (about 560Mbps). It was getting late and I wanted to start the backup before going to bed. Time for radical action! On the sending server, I opened a new `tmux` and ran: cd /my/data/dir/ python3 -m http.server 1234 That starts a webserver which lists all the files and folders in that directory. On the receiving server, I opened a new `tmux` and ran: wget --mirror http://server.ip.address:1234/ That downloads all the files that it sees, follows all the directories and subdirectories, and recreates them on the server. After running overnight, the total transfer speed reported by `wget` was about 68MBps. Not exactly saturating my link - but better than the puny throughput I experienced earlier. ##

The Watchman

Monday, 17 April 2023 | Edition 1 | Do not go gentle into that good night

Downsides There are a few (minor) problems with this approach. * No encryption. As this was a LAN transfer, I didn't really care. * No preservation of Linux attributes. I didn't mind losing metadata. * No directory timestamps. Although file timestamps are preserved. * An `index.html` file is stored every directory. I couldn't find an option to turn that off. * They can be deleted with `find . -newermt '2023-04-01' \! -newermt '2023-04-03' -name 'index.html' -delete` * It all just feels a bit icky. But, hey, if it's stupid and it works; it isn't stupid. I'm sure someone in the comments will tell me exactly which obscure setting I needed to turn on to make `scp` work at the same speed as `wget`. But this was a quick way to transfer a bunch of large files with the minimum of fuss.

curl speaks HTTP/2 with proxy

Daniel Stenberg, 14/4/23

In September 2013 we merged the first code into curl that made it capable of using HTTP/2: HTTP version 2. This version of HTTP changed a lot of previous presumptions when it comes to transfers, which introduced quite a few challenges to HTTP stack authors all of the world. One of them being that with version 2 there can be more than one transfer using the same connection where as up to that point we had always just had one transfer per connection. In May 2015 the spec was published. ## 2023 Now almost eight years since the RFC was published, HTTP/2 is the version seen most frequently in browser responses if we ask the Firefox telemetry data. 44.4% of the responses are HTTP/2. ## curl This year, the curl project has been sponsored by the Sovereign Tech Fund, and one of the projects this funding has covered is what I am here to talk about: Speaking HTTP/2 with a proxy. More

The Watchman

Monday, 17 April 2023 | Edition 1 | Do not go gentle into that good night

specifically with what is commonly referred to as a "forward proxy." Many organizations and companies have setups like the one illustrated in this image below. The user on the left is inside the organization network A and the website they want to reach is on the outside on network B. ## HTTP/2 to the proxy When this is an HTTPS proxy, meaning that the communication to and with the proxy is itself protected with TLS, curl and libcurl are now capable of negotiating HTTP/2 with it. It might not seem like a big deal to most people, and maybe it is not, but the introduction of this feature comes after some rather heavy lifting and internal refactors over the recent months that have enabled the rearrangement of networking components for this purpose. ## Enable To enable this feature in your libcurl-using application, you first need to make sure you use libcurl 8.1.0 when it ships in mid May and then you need to set the proxy type to `CURLPROXY_HTTPS2`. In plain C code it could look like this: curl_easy_setopt(handle, CURLOPT_PROXYTYPE, CURLPROXY_HTTPS2); curl_easy_setopt(handle, CURLOPT_PROXY, "https://hostname"); This allows HTTP/2 but will proceed with plain old HTTP/1 if it can't negotiate the higher protocol version using ALPN. The old proxy type called just `CURLPROXY_HTTPS` remains for asking libcurl to stick to HTTP/1 when talking to the proxy. We decided to introduce a new option for this simply because we anticipate that there will be proxies out there that will not work correctly so we cannot throw this feature at users without them asking for it. ## command line tool Using the command line tool, you use a HTTPS proxy exactly like before and then you add this flag to tell the tool that it may try HTTP/2 with the proxy: `--proxy-http2`. This also happens to be curl's 251st command line option. ## Shipping and credits This implementation has been

The Watchman

Monday, 17 April 2023 | Edition 1 | Do not go gentle into that good night

done by Stefan Eissing. These features have already landed in the master branch and will be part of the pending curl 8.1.0 release, scheduled for release on May 17, 2023.

What's in the RedPajama-Data-1T LLM training set

http://simonwillison.net/2023/Apr/17/redpajama-data/#atom-everything, 17/4/23

RedPajama is "a project to create leading open-source models, starts by reproducing LLaMA training dataset of over 1.2 trillion tokens". It's a collaboration between Together, Ontocord.ai, ETH DS3Lab, Stanford CRFM, Hazy Research, and MILA Québec AI Institute. They just announced their first release: RedPajama-Data-1T, a 1.2 trillion token dataset modelled on the training data described in the original LLaMA paper. The full dataset is 2.67TB, so I decided not to try and download the whole thing! Here's what I've figured out about it so far. #### How to download it The data is split across 2,084 different files. These are listed in a plain text file here: https://data.together.xyz/redpajama-data-1T/v1.0.0/urls.txt The dataset card suggests you could download them all like this - assuming you have 2.67TB of disk space and bandwith to spare: wget -i https://data.together.xyz/redpajama-data-1T/v1.0.0/urls.txt I prompted GPT-4 a few times to write a quick Python script to run a `HEAD` request against each URL in that file instead, in order to collect the `Content-Length` and calculate the total size of the data. My script is at the bottom of this post. I then processed the size data into a format suitable for loading into Datasette Lite. #### Exploring the size data Here's a link to a Datasette Lite page showing all 2,084 files, sorted by size and with some useful facets. ![Datasette showing the rows, faceted by top_folder and top_folders. The largest file is wikipedia/wiki.jsonl at 111GB, then book/book.jsonl at 100GB, then stackexchange/stackexchange.jsonl at 74GB, then various filtered GitHub fil

The Watchman

Monday, 17 April 2023 | Edition 1 | Do not go gentle into that good night

es](https://static.simonwillison.net/static/2023/redpajama-sizes.jpg) This is already revealing a lot about the data. The `top_folders` facet inspired me to run this SQL query: select top_folders, cast (sum(size_gb) as integer) as total_gb, count(*) as num_files from raw group by top_folders order by sum(size_gb) desc Here are the results: top_folders | total_gb | num_files ---|---|--- c4 | 806 | 1024 common_crawl/2023-06 | 288 | 175 common_crawl/2020-05 | 286 | 198 common_crawl/2021-04 | 276 | 176 common_crawl/2022-05 | 251 | 157 common_crawl/2019-30 | 237 | 153 github | 212 | 98 wikipedia | 111 | 1 book | 100 | 1 arxiv | 87 | 100 stackexchange | 74 | 1 There's a lot of Common Crawl data in there! The RedPajama announcement says: > * CommonCrawl: Five dumps of CommonCrawl, processed using the CCNet > pipeline, and filtered via several quality filters including a linear > classifier that selects for Wikipedia-like pages. > * C4: Standard C4 dataset > It looks like they used CommonCrawl from 5 different dates, from 2019-30 (30? That's not a valid month - looks like it's a week number) to 2022-05. I wonder if they de-duplicated content within those different crawls? C4 is "a colossal, cleaned version of Common Crawl's web crawl corpus" - so yet another copy of Common Crawl, cleaned in a different way. I downloaded the first 100MB of that 100GB `book.jsonl` file - the first 300 rows in it are all full-text books from Project Gutenberg, starting with The Bible Both Testaments King James Version from 1611. The data all appears to be in JSONL format - newline-delimited JSON. Different files I looked at had different shapes, though a common pattern was a `"text"` key containing the text and a `"meta"` key containing a dictionary of metadata. For example, the first line of `books.jsonl` looks like this (after pretty- printing using `jq`): { "meta": { "short_book_title": "The Bible Both Testaments King James Version", "publication_date": 1611, "url": "http://www.gutenb

The Watchman

Monday, 17 April 2023 | Edition 1 | Do not go gentle into that good night

erg.org/ebooks/10" }, "text": "\n\nThe Old Testament of the King James Version of the Bible\n..." } There are more details on the composition of the dataset in the dataset card. #### My Python script I wrote a quick Python script to do the next best thing: run a `HEAD` request against each URL to figure out the total size of the data. I prompted GPT-4 a few times, and came up with this: import httpx from tqdm import tqdm async def get_sizes(urls): sizes = {} async def fetch_size(url): try: response = await client.head(url) content_length = response.headers.get('Content-Length') if content_length is not None: return url, int(content_length) except Exception as e: print(f"Error while processing URL '{url}': {e}") return url, 0 async with httpx.AsyncClient() as client: # Create a progress bar using tqdm with tqdm(total=len(urls), desc="Fetching sizes", unit="url") as pbar: # Use asyncio.as_completed to process results as they arrive coros = [fetch_size(url) for url in urls] for coro in asyncio.as_completed(coros): url, size = await coro sizes[url] = size # Update the progress bar pbar.update(1) return sizes I pasted this into `python3 -m asyncio` \- the `-m asyncio` flag ensures the `await` statement can be used in the interactive interpreter - and ran the following: >>> urls = httpx.get("https://data.together.xyz/redpajama-data-1T/v1.0.0/urls.txt").text.splitlines() >>> sizes = await get_sizes(urls) Fetching sizes: 100%|██████████████████████████████████████| 2084/2084 [00:08<00:00, 256.60url/s] >>> sum(sizes.values()) 2936454998167 Then I added the following to turn the data into something that would work with Datasette Lite: output = [] for url, size in sizes.items(): path = url.split('/redpajama-data-1T/v1.0.0/')[1] output.append({ "url": url, "size": size, "size_mb": size / 1024 / 1024, "size_gb": size / 1024 / 1024 / 1024, "path": path, "top_folder": path.split("/")[0], "top_folders": path.rsplit("/", 1)[0], }) open("/tmp/sizes.json", "w").write(json.dumps(output, indent=2)) I pasted the result

The Watchman

Monday, 17 April 2023 | Edition 1 | Do not go gentle into that good night

into a Gist.