Three Benchmark Workloads:

I made use of the varying size files for these workloads. Some of these didn't get great reads, as I had to run the tests on the same machine that was running the server.

Benchmark 1: File100.html and file1000.html with 100 users spawning 10 users per second. Ran for 30 seconds.

```
1. from locust import HttpUser, task
2.
3. class WebsiteUser(HttpUser):
4.     @task
5.     def f100(self):
6.         self.client.get("/file100.html")
7.
8.     @task
9.     def f1000(self):
10.         self.client.get("/file1000.html")
```

Benchmark 2: File100000.html and file1000000.html with 100 users spawning 10 users per second. Ran for 30 seconds.

```
1. from locust import HttpUser, task
2.
3. class WebsiteUser(HttpUser):
4.     @task
5.     def f100000(self):
6.         self.client.get("/file100000.html")
7.
8.     @task
9.     def f1000000(self):
10.         self.client.get("/file1000000.html")
```

Benchmark 3: All files of varying size with 1000 users spawning 50 users per second. Ran for 60 seconds.

```
1. from locust import HttpUser, task
2.
3. class WebsiteUser(HttpUser):
4.     @task
5.     def f100(self):
6.         self.client.get("/file100.html")
7.
```

```
8.      @task
9.      def f1000(self):
10.         self.client.get("/file1000.html")
11.
12.     @task
13.     def f10000(self):
14.         self.client.get("/file10000.html")
15.
16.     @task
17.     def f100000(self):
18.         self.client.get("/file100000.html")
19.
20.     @task
21.     def f1000000(self):
22.         self.client.get("/file1000000.html")
```

Basic Implementation. Benchmark 1

Name	# Requests	# Fails	Request / sec	Fail / sec
File100.html	5069	72	164.9	2.3
File1000.html	5013	71	163.1	2.3
Aggregated	10082	143	328.1	4.7

Basic Implementation, Benchmark 2

Name	# Requests	# Fails	Request / sec	Fail / sec
File100000.html	1064	11	34.3	0.4
File1000000.html	1030	7	33.2	0.2
Aggregated	2094	18	67.6	0.6

Basic Implementation, Benchmark 3

Name	# Requests	# Fails	Request / sec	Fail / sec
File100.html	1630	5	26.8	0.1
File1000.html	1611	5	26.5	0.1
File10000.html	1678	5	27.6	0.1
File100000.html	1640	10	27.0	0.2
File1000000.html	1532	2	25..2	0.0
Aggregated	8091	27	133.2	0.4

Basic did pretty good, I was shocked. With only about 1.4% fail rate at its highest, I was pretty proud.

Streaming Implementation, Benchmark 1

Name	# Requests	# Fails	Request / sec	Fail / sec
File100.html	4579	0	147.4	0
File1000.html	4545	0	146.3	0
Aggregated	9124	0	29.37	0

Streaming Implementation, Benchmark 2

Name	# Requests	# Fails	Request / sec	Fail / sec
File100000.html	69	0	2.2	0
File1000000.html	0	0	0	0
Aggregated	69	0	2.2	0

Streaming Implementation, Benchmark 3

Name	# Requests	# Fails	Request / sec	Fail / sec
File100.html	203	0	3.0	0
File1000.html	208	0	3.1	0
File10000.html	177	0	2.6	0
File100000.html	132	0	1.9	0
File1000000.html	0	0	0	0
Aggregated	720	0	10.6	0

Streaming I was disappointed in how slow it was, especially with the larger files. I'm not sure why it was unable to process a request for file1000000.html. I do like how it didn't fail with any of the tests though.

Caching Implementation, Benchmark 1

Name	# Requests	# Fails	Request / sec	Fail / sec
File100.html	6007	55	200.1	1.8
File1000.html	6097	49	203.1	1.6
Aggregated	12104	104	403.1	3.5

Caching Implementation, Benchmark 2

Name	# Requests	# Fails	Request / sec	Fail / sec
File100000.html	1282	0	42	0
File1000000.html	1247	0	40.8	0
Aggregated	2529	0	82.8	0

Caching Implementation, Benchmark 3

Name	# Requests	# Fails	Request / sec	Fail / sec
File100.html	1070	3	27.6	0.1
File1000.html	1062	1	27.4	0
File10000.html	1056	2	27.2	0.1
File100000.html	1023	0	26.4	0
File1000000.html	990	2	25.5	0.1
Aggregated	5201	8	134	0.2

Caching worked so much better than I was expecting it to. It was faster than the basic implementation, and gave less errors. It did crash during the third benchmark around 40 seconds in, and it stopped wsl from working on my computer for a bit, so I decided this data was good enough. With a fail rate of nearly half that of the basic implementation I'd call this one a success.

Streaming and Caching Implementation, Benchmark 1

Name	# Requests	# Fails	Request / sec	Fail / sec
File100.html	6061	83	200.2	2.7
File1000.html	6098	62	201.4	2.0
Aggregated	12159	145	401.6	4.8

Caching Implementation, Benchmark 2

Name	# Requests	# Fails	Request / sec	Fail / sec
File100000.html	108	12	9.8	1.1
File1000000.html	12	12	1.1	1.1
Aggregated	120	24	10.9	2.2

Caching Implementation, Benchmark 3

Name	# Requests	# Fails	Request / sec	Fail / sec
File100.html	163	5	2.3	0.1
File1000.html	201	1	2.8	0.0
File10000.html	207	5	2.9	0.1
File100000.html	84	1	1.2	0
File1000000.html	2	2	0	0
Aggregated	657	14	9.1	0.2

I was hoping for the best of streaming and caching to work together on this, but it was mostly just the worst of both. A lot slower with large files and high fail rate, it was a bit disappointing.

Theres certainly some things I can do to improve the performance of both streaming and caching, but I think even with these first implementations I can see why both are extremely useful, although not without certain drawbacks.