Testing

Each server improvement was tested twice under each of the workloads. The workloads are as follows:

- **balance_test**: User sends requests for file100, file1000, and file10000 every 5-15 seconds. file100 is requested 3x as often as file10000, and file1000 is requested 2x as often as file10000. Tests were capped at 1500 users spawning at 150 per second, ending at 5000 requests
- **dogpile_test**: User sends request for file 1000 every half second. Tests were capped at 1500 users spawning at 150 per second, ending at 5000 requests.
- **nuke_test:** Same requests as **balance_test**, but capped at 5000 users spawning at a rate of 500 per second, ending at 10,000 requests.

The results of the tests are as follows:

balance_test

Run 1

	Avg Response Time (ms)	Min Response Time(ms)	Max Response Time (ms)
Caching	12	5	94
Priority	16	5.7	325
Baseline	15	4.9	3011.6

Run 2

	Avg Response Time (ms)	Min Response Time(ms)	Max Response Time (ms)
Caching	139	5	9253
Priority	28	4.5	3077.7
Baseline	13	4	163

dogpile_test

Run 1

	Avg Response Time (ms)	Min Response Time(ms)	Max Response Time (ms)
Caching	826	6.8	13175
Priority	1028	4.9	14935.7
Baseline	606	6.7	11033.4

Run 2

	Avg Response Time (ms)	Min Response Time(ms)	Max Response Time (ms)
Caching	854	7	13065
Priority	857	6	13709.4
Baseline	965	5.94	15889

nuke_test

Run 1

	Avg Response Time (ms)	Min Response Time(ms)	Max Response Time (ms)
Caching	1378	7	12495.7
Priority	209	7	5894
Baseline	329	14	6107

Run 2

	Avg Response Time (ms)	Min Response Time(ms)	Max Response Time (ms)
Caching	86	7	3073.2
Priority	88	7	2863.3
Baseline	483	6	10065.2

Analysis

Note: A basic implementation of the server model that incrementally sends data was partially completed. The incremental reading/writing seemed to work, but there was an issue with the stream closing early that I could not resolve that resulted in much of the data not being received. As the requests finished after having sent a few hundred bytes, its results in testing were inordinately fast. I have omitted this testing data due to it being the result of a lack of integrity in the data sent.

I must admit I’m a bit bemused by the results. At times, my improvements seemed to outperform the baseline implementation by a respectable degree, but on subsequent runs were utterly crushed by it under the same workload. There seems to be a massive variance in performance between runs across the board, making it difficult to draw any major conclusions.

On the whole, while it did have the lowest average response times in half of the tests (by a small margin, usually), I expected the caching model to perform much better than it did. I was most surprised to see the priority model perform so close to it on *dogpile_test*, in which the priority queue affords no advantage. My first explanation would be that the cache model involves more overhead and thus has an inherent performance bottleneck, but since very few files are involved in this testing environment, that should be a non issue after the first few runs once all of them are cached.

As for the wild variance in results, my only other explanation would be that it was due to the campus network, which has been notably uncooperative over the past few days.