PRODUCT | RELEASED DECEMBER 20, 2021 |

The Ultimate JSON Library: JSON.simple vs. GSON vs. Jackson vs. JSONP

By Harness

JSON is the accepted standard these days for transmitting data between servers and web applications. We often don't think about the JSON libraries we use, but there are some differences between them.

JSON is often used to transport and parse big files. This is a scenario that is common in data processing applications running in Hadoop or Spark clusters. Given the size of these files, you can be looking at significant differences in parsing speed between libraries.

all files come up all the time as incoming requests at high throughput, and parsing them appens quickly, so the differences in performance may not seem to be a big deal at first. But the differences add up, as often you need to parse lots of small files in rapid succession during times of heavy traffic. Microservices and distributed architectures often use JSON for transporting these kinds of files, as it is the de facto format for web APIs.

Not all JSON libraries perform the same. Picking the right one for your environment can be critical. We ran a benchmark test to see how fast four of the most popular JSON libraries for Java parse different sizes of files. This benchmark can help you decide.

The JSON Libraries

JSON.simple vs GSON vs Jackson vs JSONP For the benchmark tests, we looked at four major JSON libraries for Java: JSON.simple, GSON, Jackson, and JSONP. All of these libraries are popularly used for JSON processing in a Java environment, and were chosen according to their popularity in Github projects. Here are the ones we tested:

- Yidong Fang's JSON.simple – JSON.simple is a Java toolkit for encoding and decoding JSON text. It's meant to be a lightweight and simple library that still performs at a high level.
- Google's GSON – GSON is a Java library that converts Java Objects into JSON and vice versa. It provides the added benefit of full support for Java Generics, and it doesn't require you to annotate your classes. Not needing to add annotations makes for simpler implementation and can even be a requirement if you don't have access to your source code.
- FasterXML's Jackson Project – Jackson is a group of data processing tools highlighted by its streaming JSON parser and generator library. Designed for Java, it can also handle other non-JSON encodings. It's the most popular JSON parser, according to our findings on Github usages.
- Oracle's JSONP – JSONP (JSON Processing) is a Java API for JSON processing, namely around consuming and producing streaming JSON text. It's the open source reference implementation of JSR353.

The Benchmark: 2017

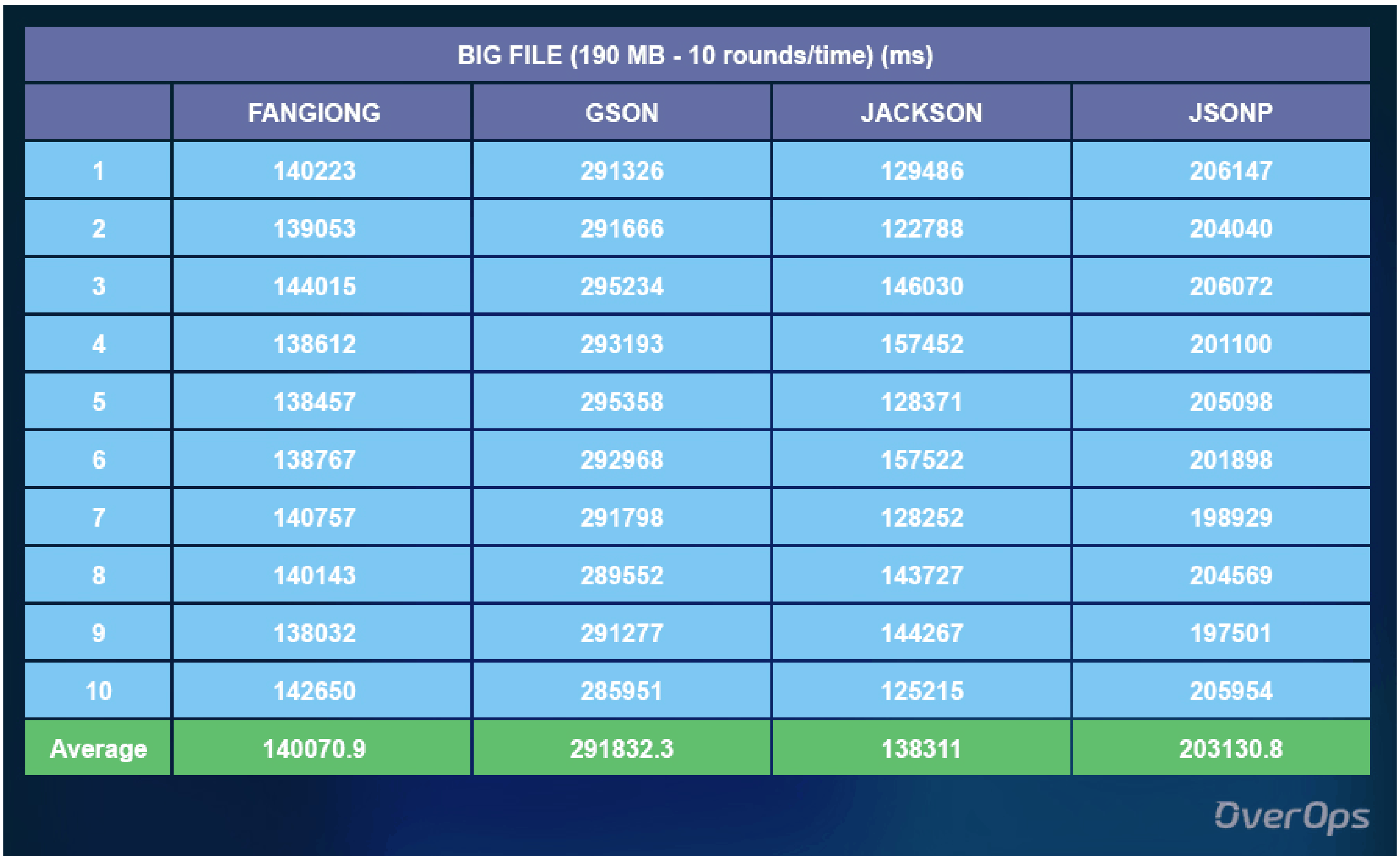
We ran a benchmark test on the libraries for both big files and small files. The requirements (and therefore performance) for handling different file sizes are different, as are the environments in which the need to parse these files arise.

The benchmark tested two key scenarios: parsing speed for big files (190 MB) and parsing speed for small files (1 KB). The big files were taken from GitHub. The small files were randomly nerated from here the JSON Generator.

For both big and small files, we ran each file 10 times per library. Given the size of the big file, we did 10 iterations per run for each library. Each small file was iterated 10,000 times per run for each library. For the small files test, we didn't retain the files in memory between iterations and the test was run on a c3.large instance on AWS.

The results for the big file are shown in full below, but I've further averaged the results for the small files in the interest of space. To view the extended results, go here. If you want to view the source code for the small files or the libraries, go here.

Big File Results: 2017

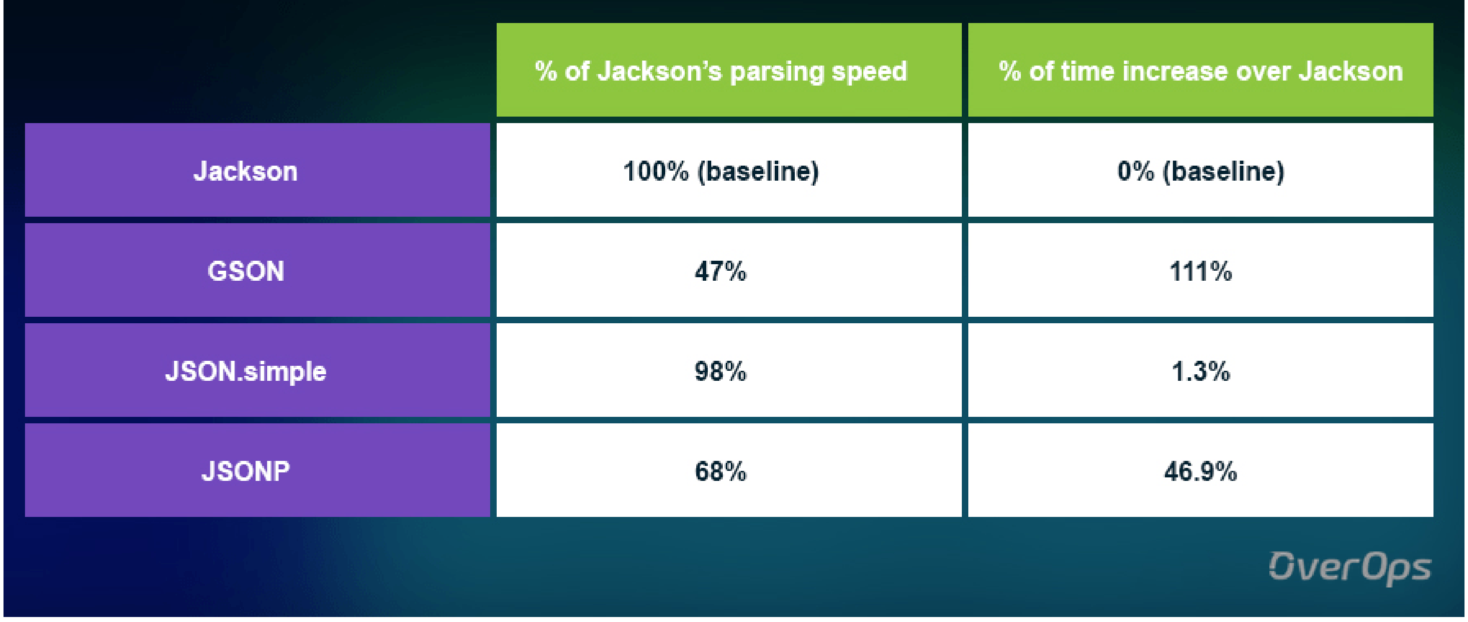


BIG FILE (190 MB - 10 rounds/time) (ms)				
	FANGIONG	GSON	JACKSON	JSONP
1	140223	291326	129486	206147
2	139053	291666	122788	204040
3	144015	295234	146030	206072
4	138612	293193	157452	201100
5	138457	295358	128371	205098
6	138767	292968	157522	201898
7	140757	291798	128252	198929
8	140143	289552	143727	204569
9	138032	291277	144267	197501
10	142650	285951	125215	205954
Average	140070.9	291832.3	138311	203130.8

OverOps

Big differences here! Depending on the run, Jackson or JSON.simple traded fastest times, with Jackson edging out JSON.simple in aggregate. Looking at the average result across all the test runs, Jackson and JSON.simple come out well ahead on the big file, with JSONP a distant third and GSON far in last place.

Let's put that in percentage terms. Jackson is the winner in average time across all the runs. Looking at the numbers from two different angles, here are the percentage results:

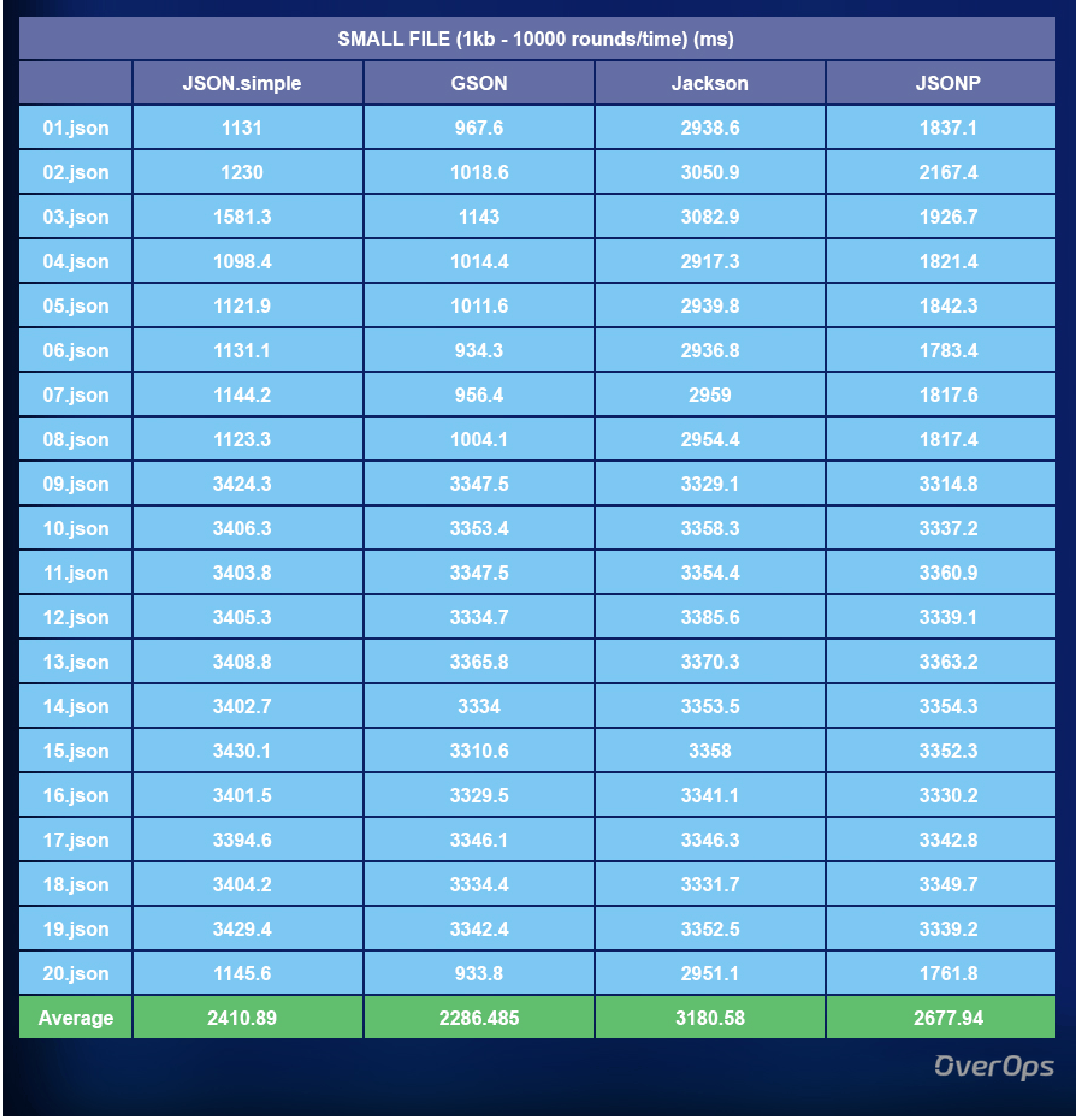


	% of Jackson's parsing speed	% of time increase over Jackson
Jackson	100% (baseline)	0% (baseline)
GSON	47%	111%
JSON.simple	98%	1.3%
JSONP	68%	46.9%

Those are big differences between the library speeds!

Takeaway: It was a photo finish, but Jackson is your winning library. JSON.simple is a nose behind and the other two are in the rearview mirror.

Small Files Results: 2017



SMALL FILE (1kb - 10000 rounds/time) (ms)				
	JSON.simple	GSON	Jackson	JSONP
01.json	1131	967.6	2938.6	1837.1
02.json	1230	1018.6	3050.9	2167.4
03.json	1581.3	1143	3082.9	1926.7
04.json	1098.4	1014.4	2917.3	1821.4
05.json	1121.9	1011.6	2939.8	1842.3
06.json	1131.1	934.3	2936.8	1783.4
07.json	1144.2	956.4	2959	1817.6
08.json	1123.3	1004.1	2954.4	1817.4
09.json	3424.3	3347.5	3329.1	3314.8
10.json	3406.3	3353.4	3358.3	3337.2
11.json	3403.8	3347.5	3354.4	3360.9
12.json	3405.3	3334.7	3385.6	3339.1
13.json	3408.8	3365.8	3370.3	3363.2
14.json	3402.7	3334	3353.5	3354.3
15.json	3430.1	3310.6	3358	3352.3
16.json	3401.5	3329.5	3341.1	3330.2
17.json	3394.6	3346.1	3346.3	3342.8
18.json	3404.2	3334.4	3331.7	3349.7
19.json	3429.4	3342.4	3352.5	3339.2
20.json	1145.6	933.8	2951.1	1761.8
Average	2410.89	2286.485	3180.58	2677.94

OverOps

The table above shows the average of 10 runs for each file, and the total average at the bottom. The tally for fastest library on number of files won is:

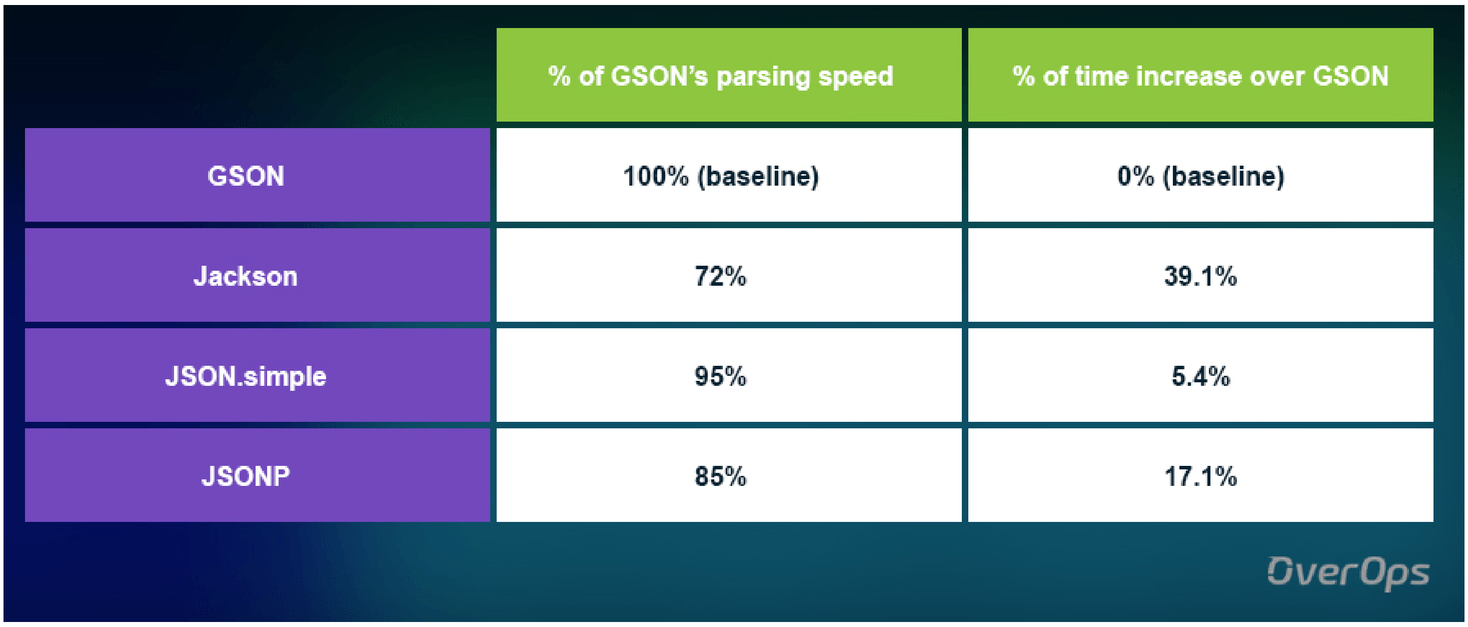
- GSON – 14
- JSONP – 5
- Jackson – 1
- JSON.simple – 0

Looking at the average result for all the test runs across all the files, GSON is the winner here, with JSON.simple and JSONP taking a distinct second and third place, respectively.

ckson came in 2nd to last. So despite not being the fastest on any single file, JSON.simple is second fastest in aggregate. And despite being the fastest on a handful of files, JSONP is well in third place in aggregate.

Jackson is very consistent across all the files, while the other three libraries are occasionally much faster than Jackson, but on some files end up running at about the same speed or even slightly slower.

Let's put the numbers in percentage terms, again looking at the numbers from two different angles:



	% of GSON's parsing speed	% of time increase over GSON
GSON	100% (baseline)	0% (baseline)
Jackson	72%	39.1%
JSON.simple	95%	5.4%
JSONP	85%	17.1%

Compared to the big file tests, these are smaller differences, but still quite noticeable.

Takeaway: Bad luck for JSON.simple, as it again loses a close race, but GSON is your winner. JSONP is a clear third and Jackson brings up the rear.

Conclusion: 2017

Parsing speed isn't the only consideration when choosing a JSON library, but it is an important one. Upon running this benchmark test, what we found was that there is no one library that blows the others away on parsing speed across all file sizes and all runs. The libraries that performed best for big files suffered for small files and vice versa.

Choosing which library to use on the merit of parsing speed comes down to your environment then.

- If you have an environment that deals often or primarily with big JSON files, then Jackson is your library of interest. GSON struggles the most with big files.
- If your environment primarily deals with lots of small JSON requests, such as in a micro services or distributed architecture setup, then GSON is your library of interest. Jackson struggles the most with small files.
- If you end up having to often deal with both types of files, JSON.simple came in a very close 2nd place in both tests, making it a good workhorse for a variable environment.

Neither Jackson nor GSON perform as well across multiple file sizes.

As far as parsing speed goes, JSONP doesn't have much to recommend for it in any scenario. It performs poorly for both big and small files compared to other available options. Fortunately, Java 9 is reportedly getting native JSON implementation, which one would imagine is going to be an improvement over the reference implementation.

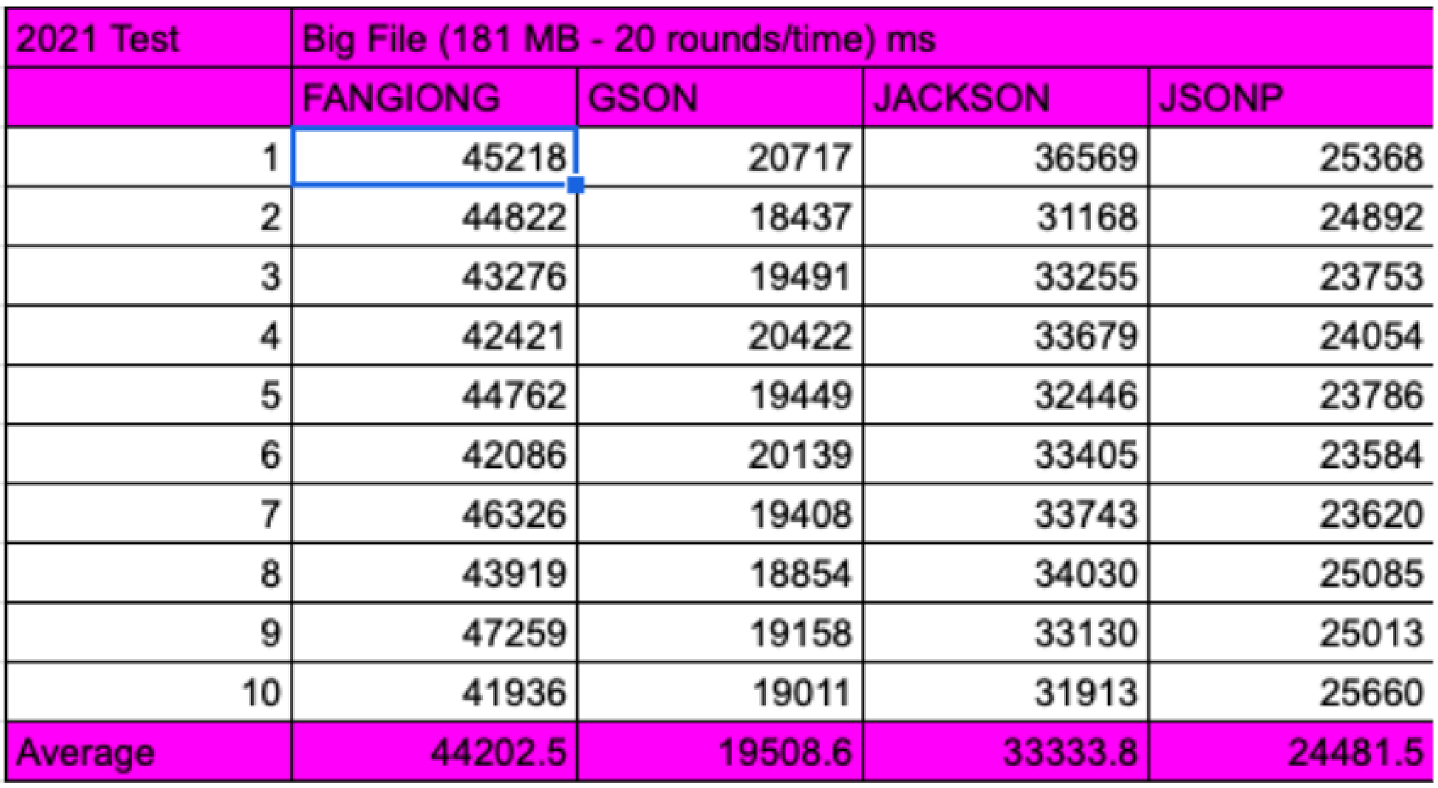
If you're concerned about parsing speed for your JSON library, choose Jackson for big files, GSON for small files, and JSON.simple for handling both.

The Benchmark: 2021

We ran the same benchmarks again but are now using the latest Libraries and ran the tests on Java 11.

The numbers look quite different from when we ran these benchmarks previously. In general the performance increased drastically.

Big File Results: 2021

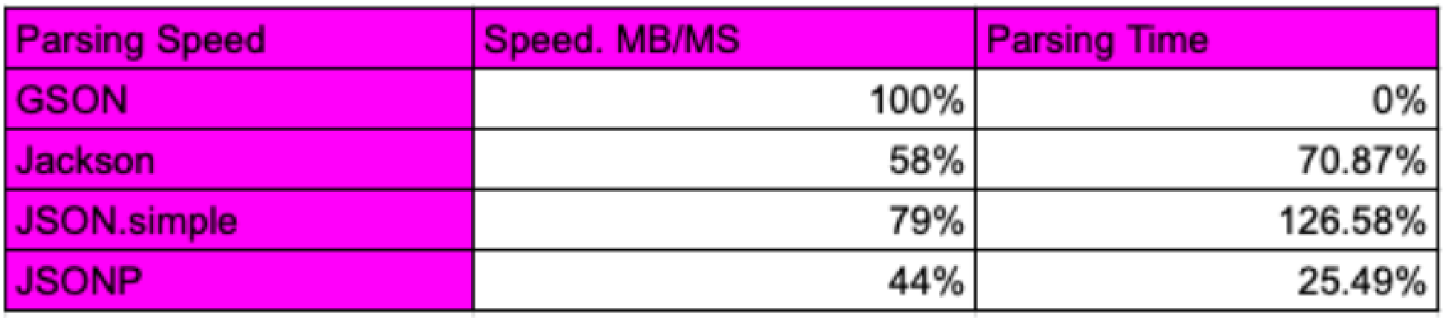


2021 Test	Big File (181 MB - 20 rounds/time) ms			
	FANGIONG	GSON	JACKSON	JSONP
1	45218	20717	36569	25368
2	44822	18437	31168	24892
3	43276	19491	33255	23753
4	42421	20422	33679	24054
5	44762	19449	32446	23786
6	42086	20139	33405	23584
7	46326	19408	33743	23620
8	43919	18854	34030	25085
9	47259	19158	33130	25013
10	41936	19011	31913	25660
Average	44202.5	19508.6	33333.8	24481.5

Last benchmarks Jackson was the winner, but this time GSON is by far the fastest, with JSONP being a close second and then followed by Jackson and then JSON.simple last.

d again looking at the numbers from two different angles, here are the percentage results:

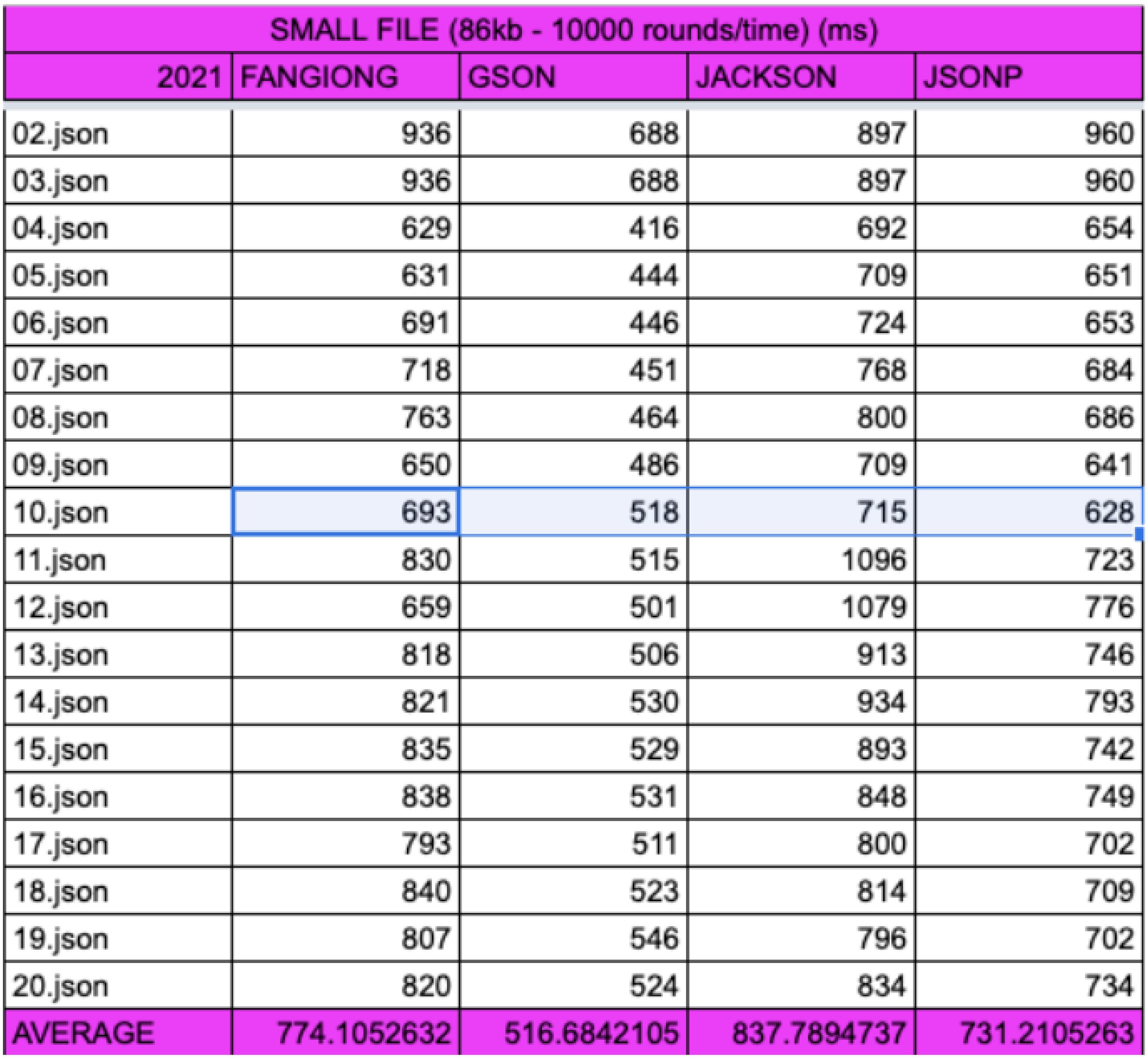
GSON is a clear winner.



Parsing Speed	Speed. MB/MS	Parsing Time
GSON	100%	0%
Jackson	58%	70.87%
JSON.simple	79%	126.58%
JSONP	44%	25.49%

Small Files Result: 2021

In the small File Results Gson is still the winner. Clearly beating the other 3 contenders.



SMALL FILE (86kb - 10000 rounds/time) (ms)				
2021	FANGIONG	GSON	JACKSON	JSONP
02.json	936	688	897	960
03.json	936	688	897	960
04.json	629	416	692	654
05.json	631	444	709	651
06.json	691	446	724	653
07.json	718	451	768	684
08.json	763	464	800	686
09.json	650	486	709	641
10.json	693	518	715	628
11.json	830	515	1096	723
12.json	659	501	1079	776
13.json	818	506	913	746
14.json	821	530	934	793
15.json	835	529	893	742
16.json	838	531	848	749
17.json	793	511	800	702
18.json	840	523	814	709
19.json	807	546	796	702
20.json	820	524	834	734
AVERAGE	774.1052632	516.6842105	837.7894737	731.2105263

re are the results:

1. Gson
2. Jsonp
3. Json.simple
4. Jackson

Gson is processing those files faster by almost 200+ ms before the next candidate.

Conclusion: 2021

The obvious elephant in the room is that java and the json libraries got faster. Like way faster then back when the first benchmarks were run.

It is also obvious that GSON stepped up big and won both benchmarks for Big and small files. In both cases very clearly.