IOZone Application 4

Introduction

TargetFS-LKM has been run on IOZone. IOZone is a file system benchmark application used to test various file operations for Linux file systems. Download information and more details about the benchmark itself can be found at: http://www.iozone.org.

The following modifications were done to the IOZone Makefile to make it run in Blunk's test environment:

1) CC, GCC and CFLAGS were modified to reflect the tool chain and architecture options specific for Blunk:

   ```
   CC = $(TARGET_CROSS_COMPILE)gcc
   CFLAGS = -Wall –g –D_FILE_OFFSET_BITS=64 -march=armv5t -mtune=arm926ej-s
   ```

2) Disabled tool chain dependent feature, Async I/O, by removing the relevant compilation file (libsync.c) and taking out the ASYNC_IO switch.

Blunk's test environment for the IOZone benchmark consists of the LPC3250 developer's kit from Embedded Artists, running Ubuntu 11.04 and Linux kernel version 2.6.39.2. The board is equipped with a 1Gbit NAND SLC flash memory from Samsung. The part number for the memory is K9F1G08U0A.

The IOZone benchmark was run on TargetFS-LKM, JFFS2 and UBIFS. JFFS2 is a log structured Linux native file system designed specifically for flash devices – i.e. it does not need a flash translation layer. The Unsorted Block Image File System (UBIFS) is a Linux native successor to JFFS2.

For additional information on UBIFS consult http://www.linux-mtd.infradead.org/index.html.

For additional information on JFFS2 consult http://sourceware.org/jffs2/.

IOZone was run for all three available TargetFS-LKM configurations: asynchronous, background synchronization and synchronous to emphasize the strength and purpose of each of these configurations and how well it fares against the native Linux solutions in both read and write performance. Each configuration section, besides the specific reads/writes graphs, contains a CPU load graph to show the system load imposed by each file system.

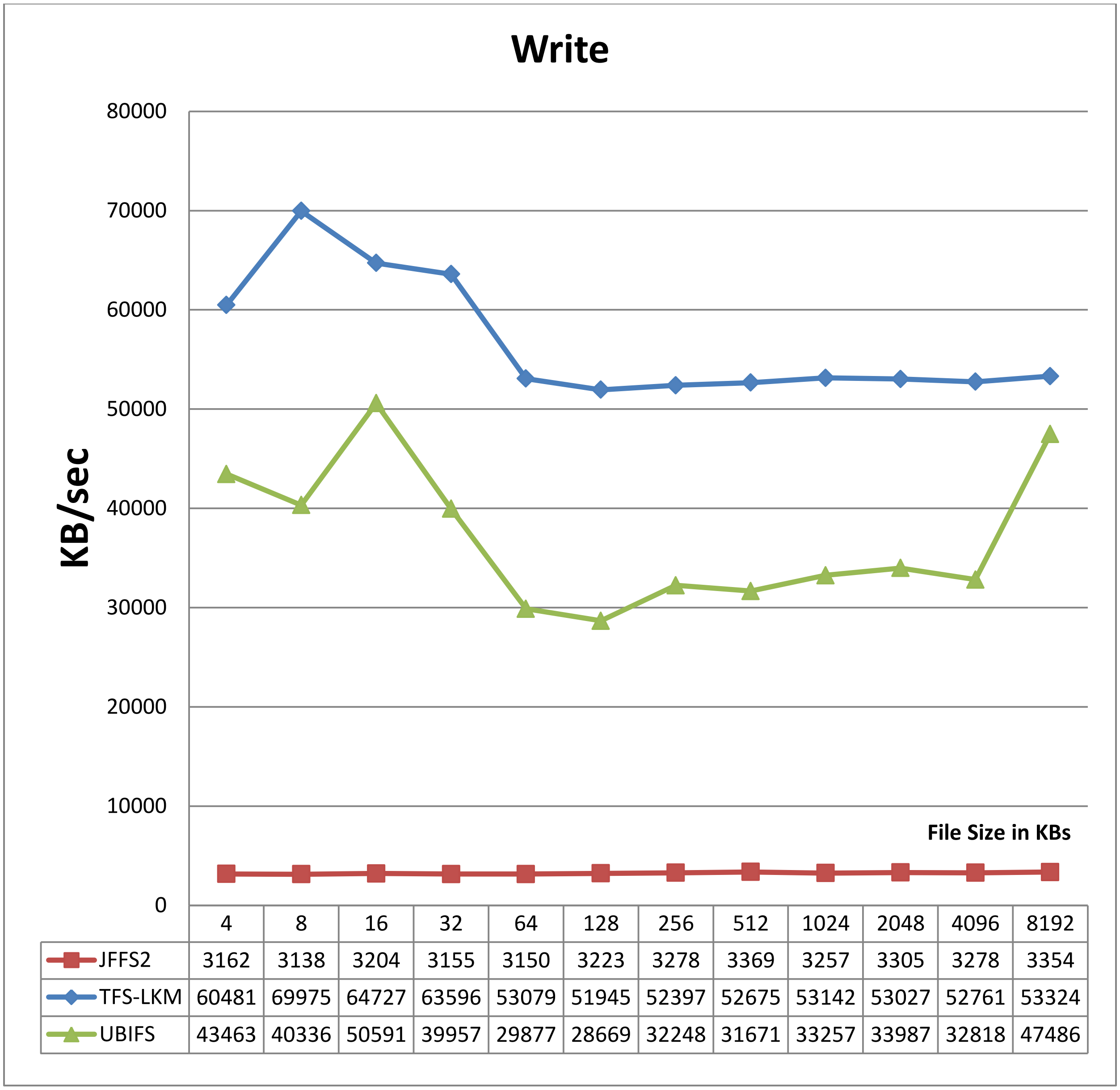
Asynchronous Configuration

In this configuration, the file system responds instantaneously to application requests. This configuration is ideal for writing files that can be cached without being throttled by the kernel.

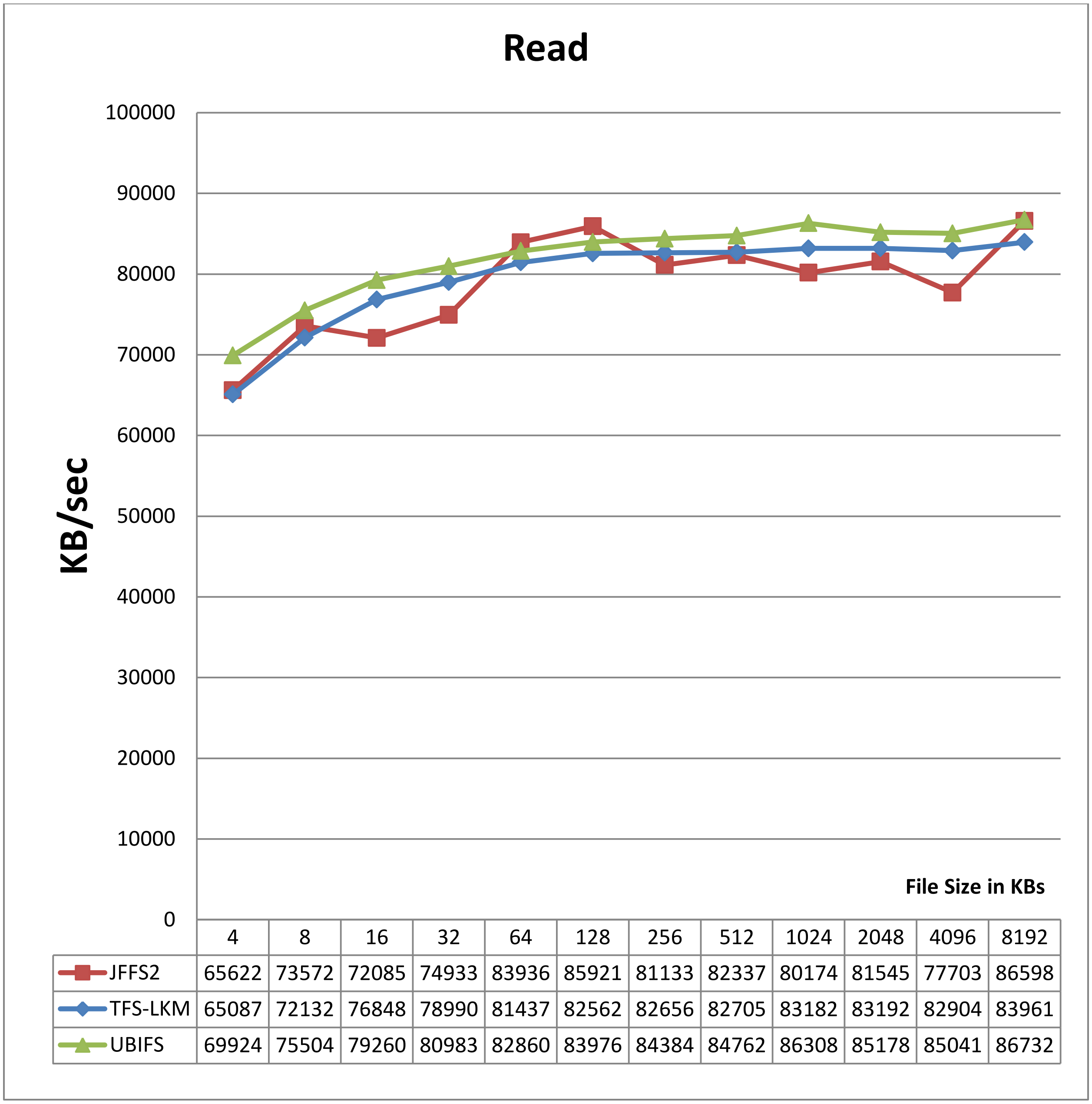
The defining strength of this configuration is its effective use of the Linux page cache in its write-back mode. Caching data in RAM results in quick writes to the cache. The kernel will spawn a background flusher task to sync the dirty cache pages to the storage media. Sustained throughput in this configuration depends on the flusher task being able to keep the number of dirty pages within the allowed limits.

If the application issues write requests at a rate significantly higher than the flusher write-back rate, the kernel will throttle the application by forcing it into sleep state which can prove counter-productive. An effective adoption of this configuration will need to ensure that the rate and size of content created for writing is within the bounds of the dirty page quota assigned for each process, something which is true for any Linux file system which implements write-back, including UBIFS. The quota value is a kernel configuration option.

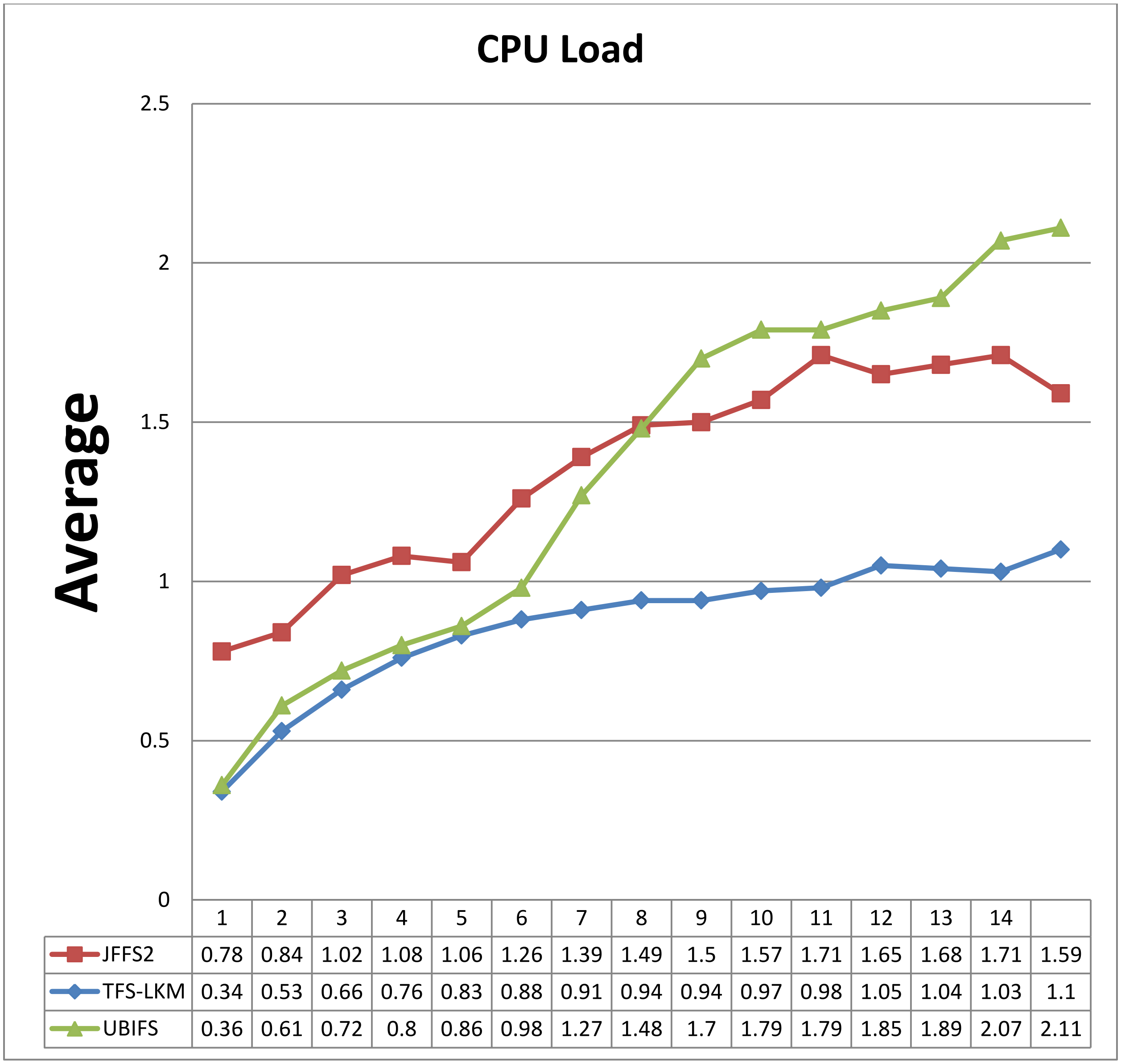
TargetFS-LKM induces less CPU load and is more deterministic when mapping dirty pages to storage memory blocks. These are some of the reasons for its better throughput numbers and its scalability.



	4	8	16	32	64	128	256	512	1024	2048	4096	8192
■ JFFS2	3162	3138	3204	3155	3150	3223	3278	3369	3257	3305	3278	3354
◆ TFS-LKM	60481	69975	64727	63596	53079	51945	52397	52675	53142	53027	52761	53324
▲ UBIFS	43463	40336	50591	39957	29877	28669	32248	31671	33257	33987	32818	47486



	4	8	16	32	64	128	256	512	1024	2048	4096	8192
JFFS2	65622	73572	72085	74933	83936	85921	81133	82337	80174	81545	77703	86598
TFS-LKM	65087	72132	76848	78990	81437	82562	82656	82705	83182	83192	82904	83961
UBIFS	69924	75504	79260	80983	82860	83976	84384	84762	86308	85178	85041	86732



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
■ JFFS2	0.78	0.84	1.02	1.08	1.06	1.26	1.39	1.49	1.5	1.57	1.71	1.65	1.68	1.71	1.59
◆ TFS-LKM	0.34	0.53	0.66	0.76	0.83	0.88	0.91	0.94	0.94	0.97	0.98	1.05	1.04	1.03	1.1
▲ UBIFS	0.36	0.61	0.72	0.8	0.86	0.98	1.27	1.48	1.7	1.79	1.79	1.85	1.89	2.07	2.11

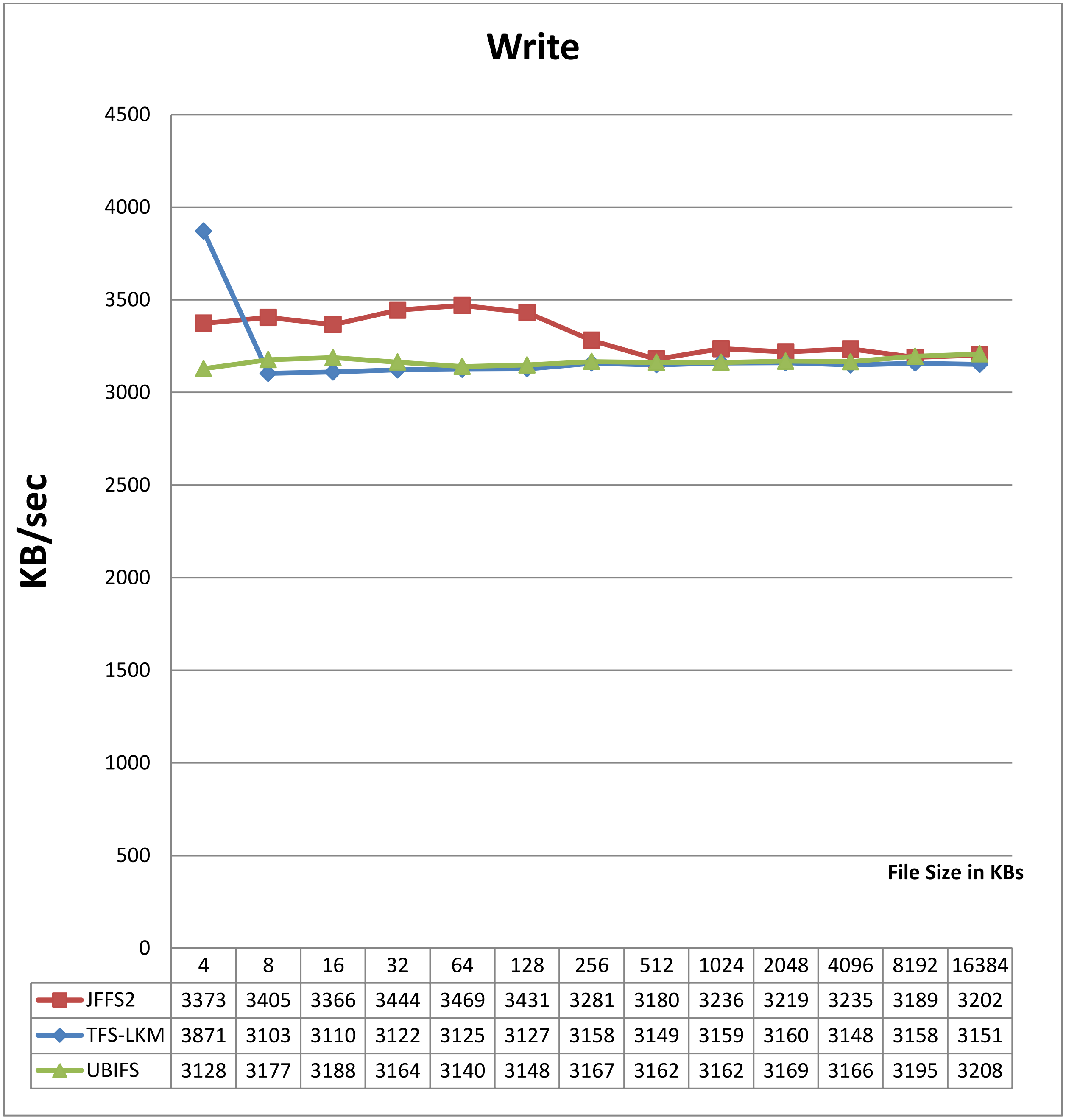
Background Synchronization Configuration

For large file sizes, file systems such as UBIFS, which are dependent on the Linux page cache, are at a disadvantage because the files cannot be effectively cached. TargetFS-LKM is capable of direct access to flash in this configuration which leads to lower system loads and faster reads/writes.

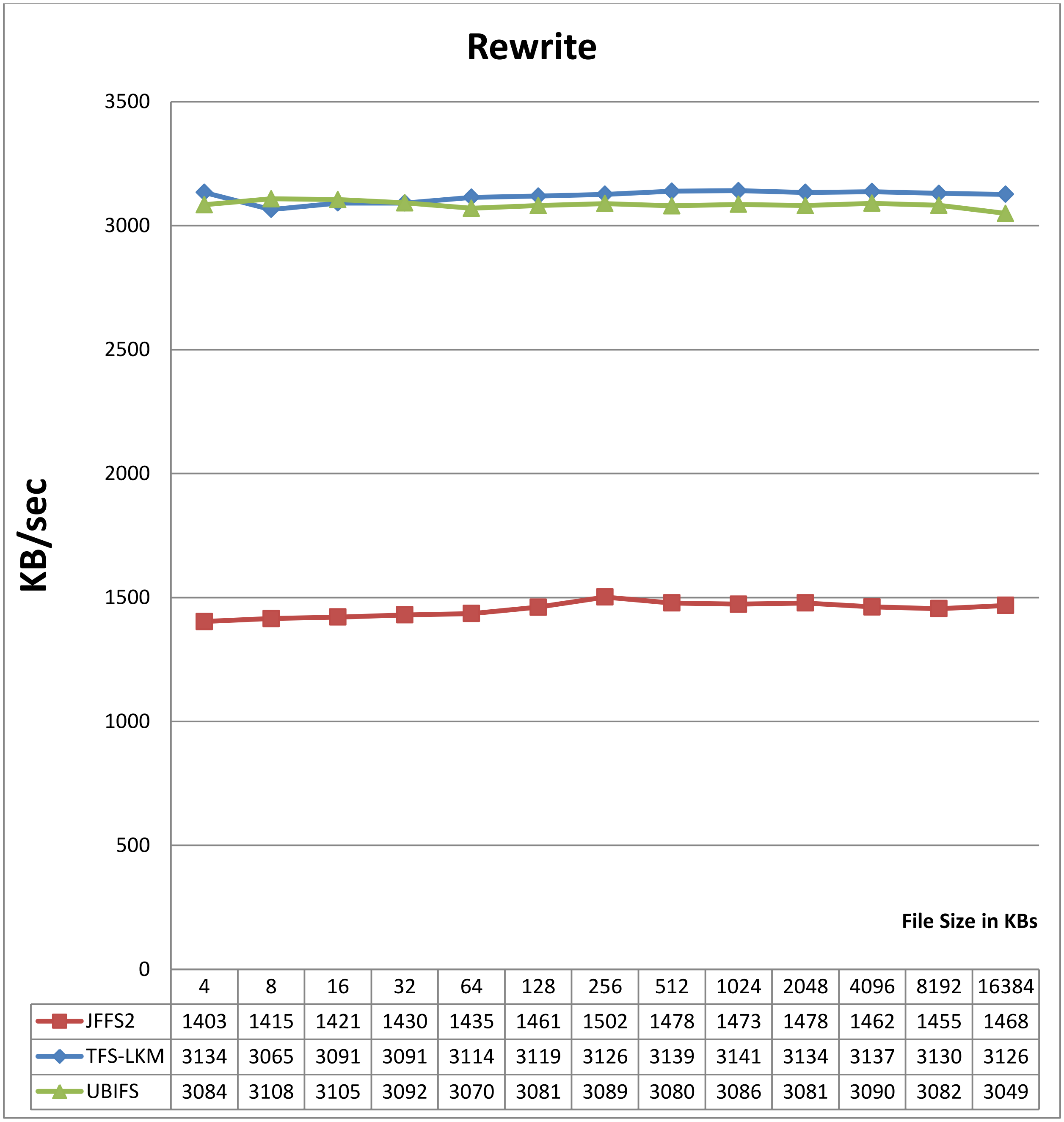
Unlike the asynchronous configuration, in this configuration TargetFS-LKM employs the Linux page cache in a write-through fashion, so it does not actually decouple the programming of file data to the storage medium from the application context. It does however synchronize the file system metadata in a background task. This configuration is frugal in its demand for CPU time and does not create dirty cache pages. This configuration is thus ideal for background I/O of file sizes significantly larger than the application page cache quota.

File systems similar to UBIFS which insist on using the page cache at all times in write-back mode will find it difficult to justify their performance when we consider the memory and CPU load they exact for handling huge files within the constraints of available RAM.

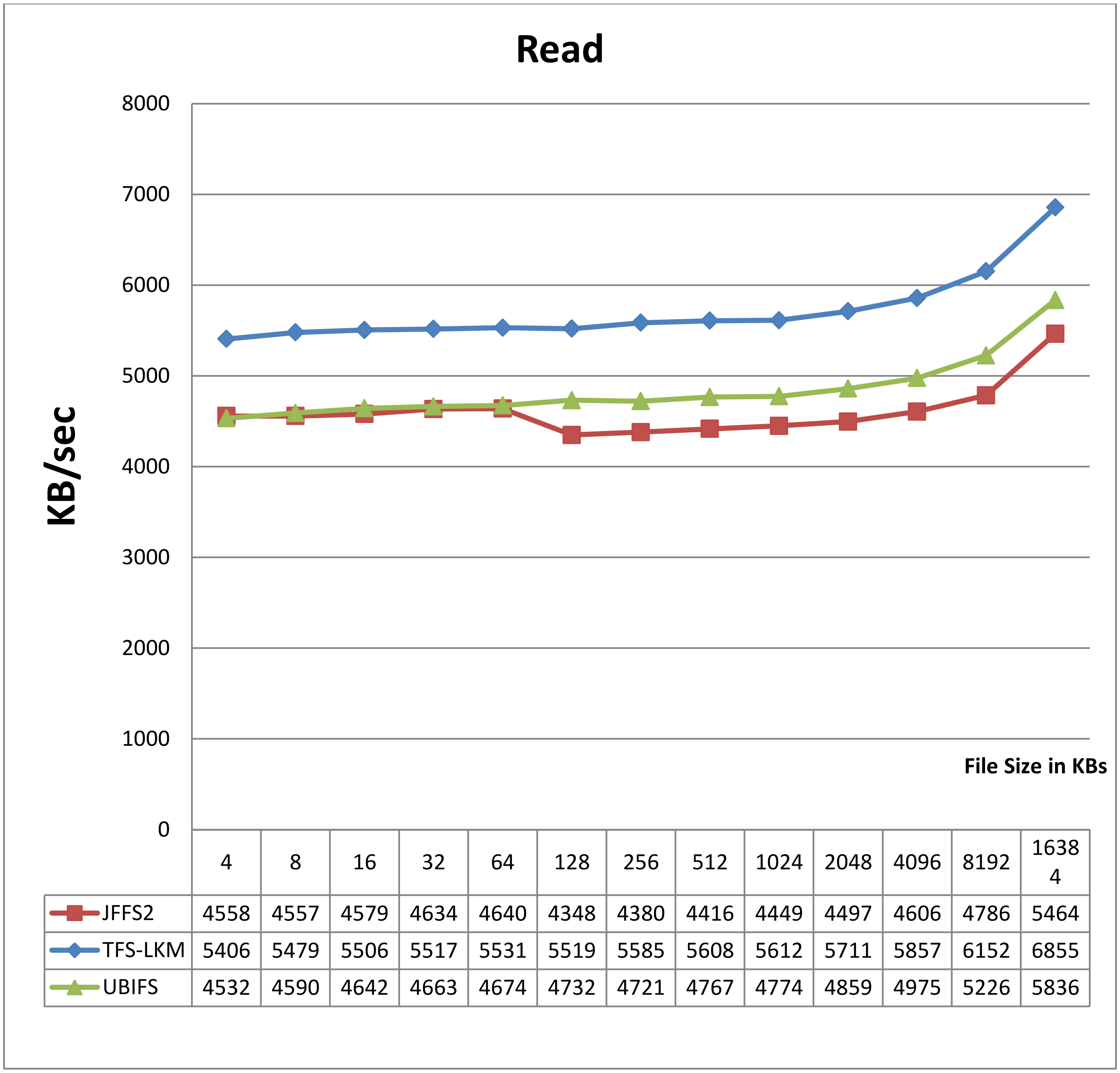
For such use cases TargetFS-LKM in background sync mode performs just as fast and, at the same time, consumes minimal system resources. This is quite evident from the graphs provided below that show TargetFS-LKM matching and sometimes exceeding UBIFS throughput when handling large file writes/reads.



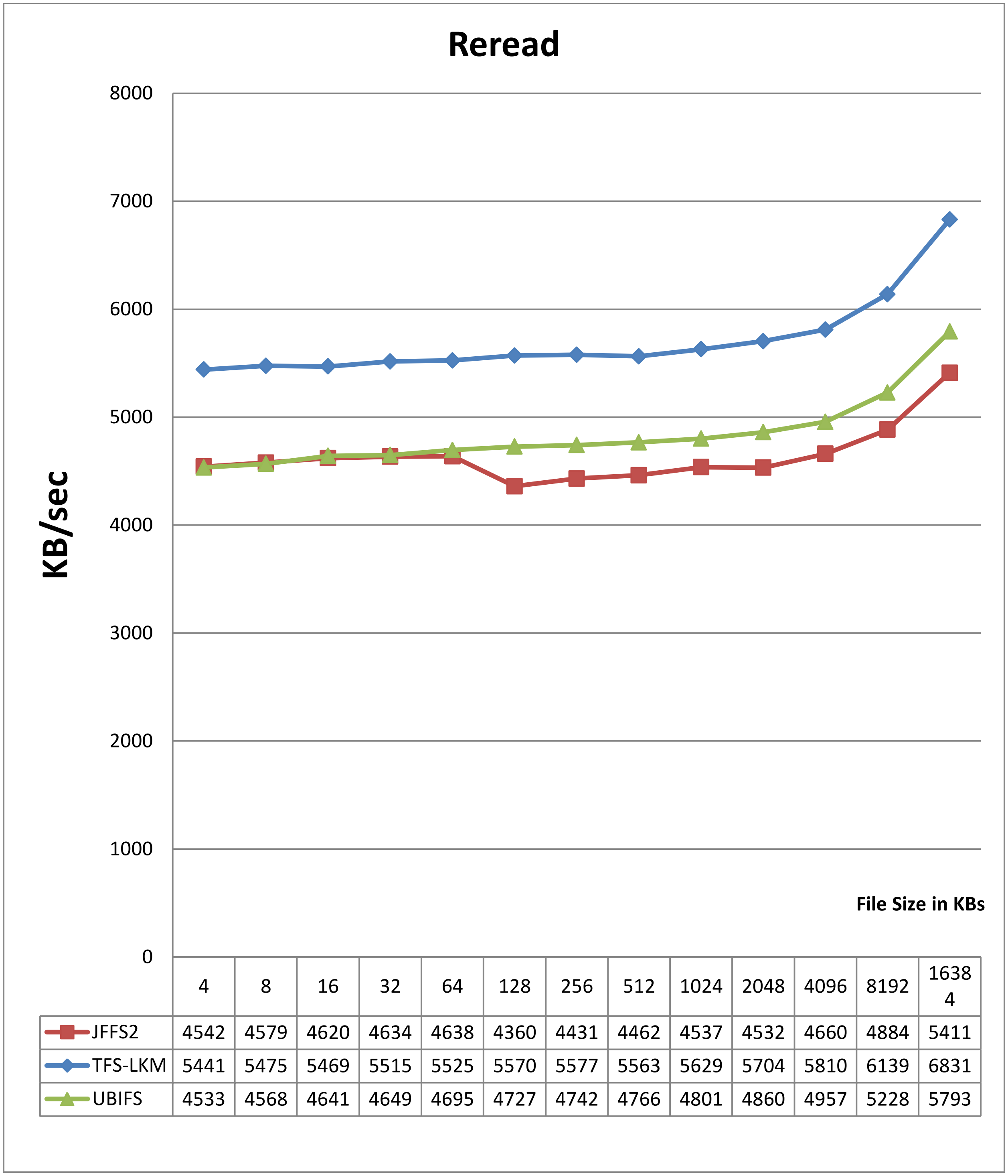
	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
■ JFFS2	3373	3405	3366	3444	3469	3431	3281	3180	3236	3219	3235	3189	3202
◆ TFS-LKM	3871	3103	3110	3122	3125	3127	3158	3149	3159	3160	3148	3158	3151
▲ UBIFS	3128	3177	3188	3164	3140	3148	3167	3162	3162	3169	3166	3195	3208



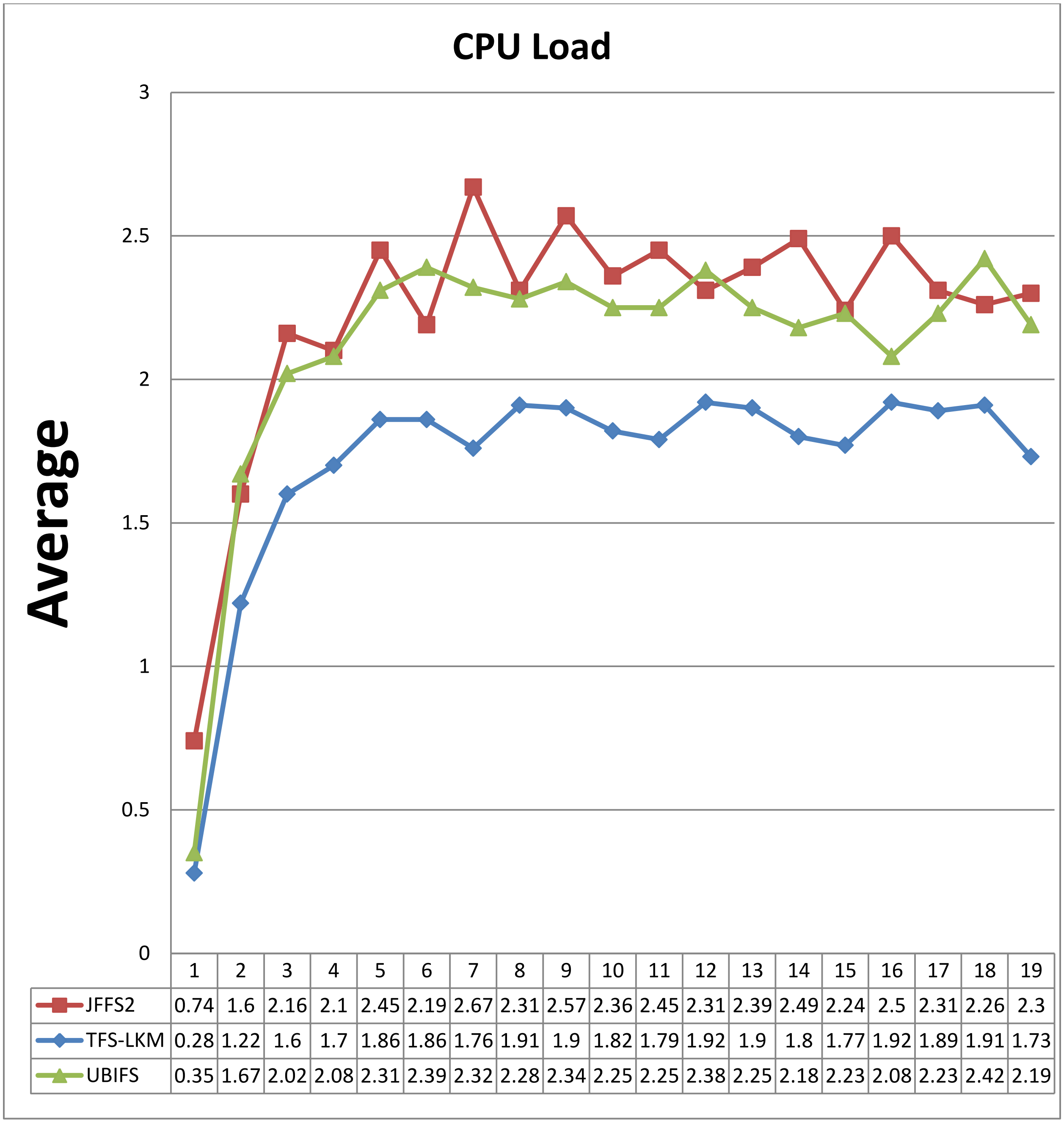
	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
■ JFFS2	1403	1415	1421	1430	1435	1461	1502	1478	1473	1478	1462	1455	1468
◆ TFS-LKM	3134	3065	3091	3091	3114	3119	3126	3139	3141	3134	3137	3130	3126
▲ UBIFS	3084	3108	3105	3092	3070	3081	3089	3080	3086	3081	3090	3082	3049



	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
JFFS2	4558	4557	4579	4634	4640	4348	4380	4416	4449	4497	4606	4786	5464
TFS-LKM	5406	5479	5506	5517	5531	5519	5585	5608	5612	5711	5857	6152	6855
UBIFS	4532	4590	4642	4663	4674	4732	4721	4767	4774	4859	4975	5226	5836



	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
■ JFFS2	4542	4579	4620	4634	4638	4360	4431	4462	4537	4532	4660	4884	5411
◆ TFS-LKM	5441	5475	5469	5515	5525	5570	5577	5563	5629	5704	5810	6139	6831
▲ UBIFS	4533	4568	4641	4649	4695	4727	4742	4766	4801	4860	4957	5228	5793

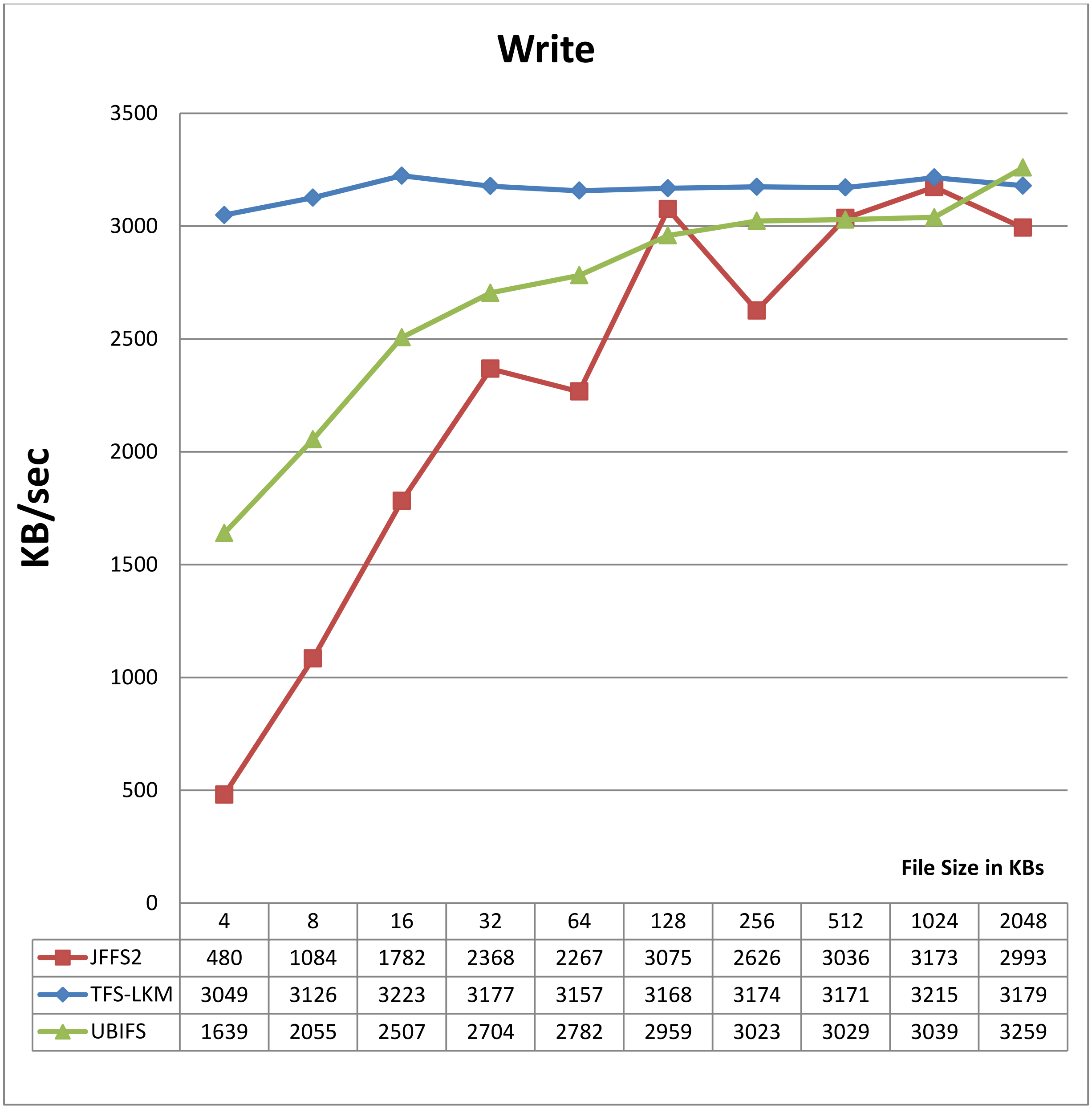


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
JFFS2	0.74	1.6	2.16	2.1	2.45	2.19	2.67	2.31	2.57	2.36	2.45	2.31	2.39	2.49	2.24	2.5	2.31	2.26	2.3
TFS-LKM	0.28	1.22	1.6	1.7	1.86	1.86	1.76	1.91	1.9	1.82	1.79	1.92	1.9	1.8	1.77	1.92	1.89	1.91	1.73
UBIFS	0.35	1.67	2.02	2.08	2.31	2.39	2.32	2.28	2.34	2.25	2.25	2.38	2.25	2.18	2.23	2.08	2.23	2.42	2.19

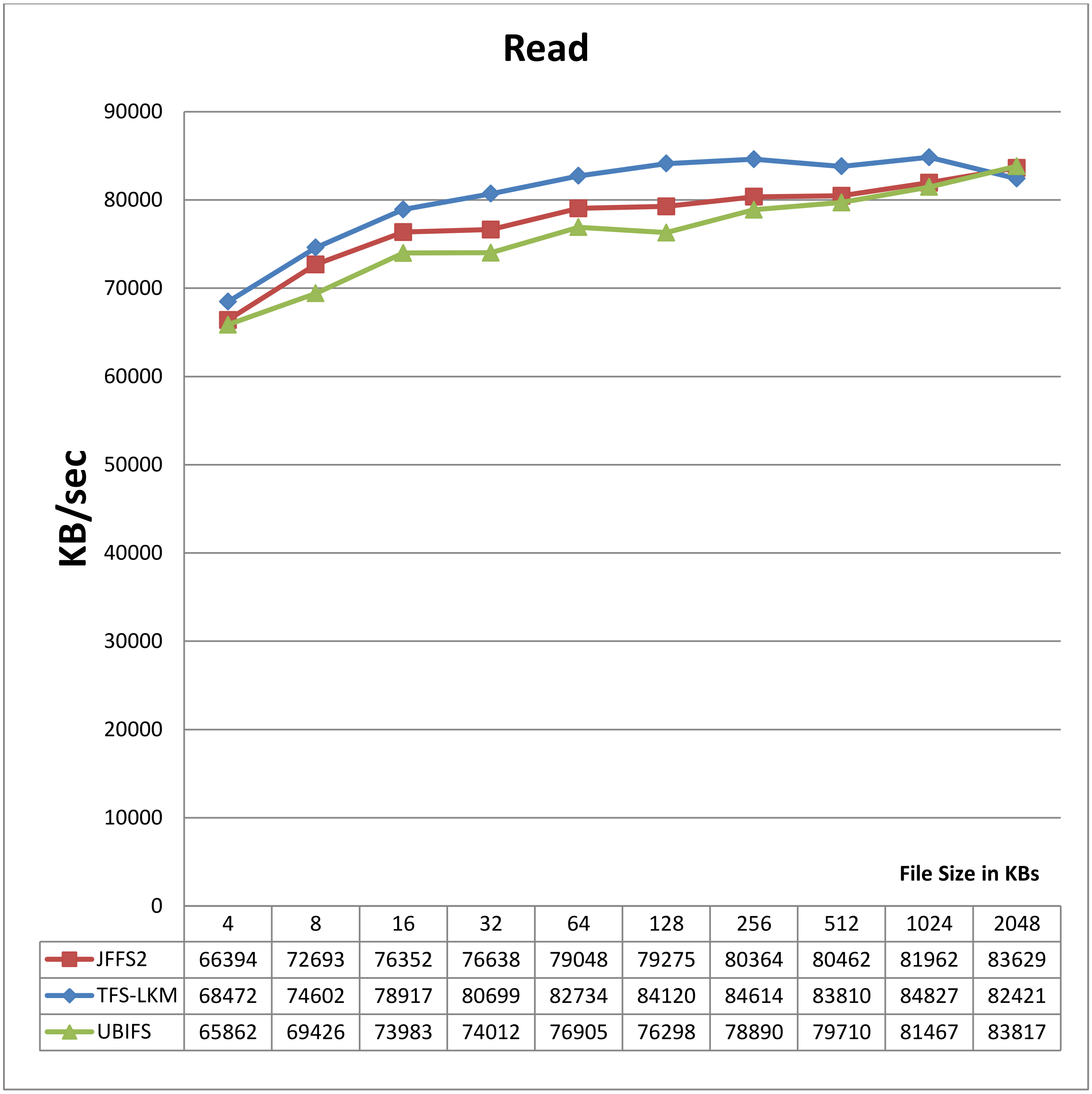
Synchronous Configuration

In this configuration all file systems were mounted with syncs enabled. TargetFS-LKM is faster especially for file sizes ranging up to 2MB.

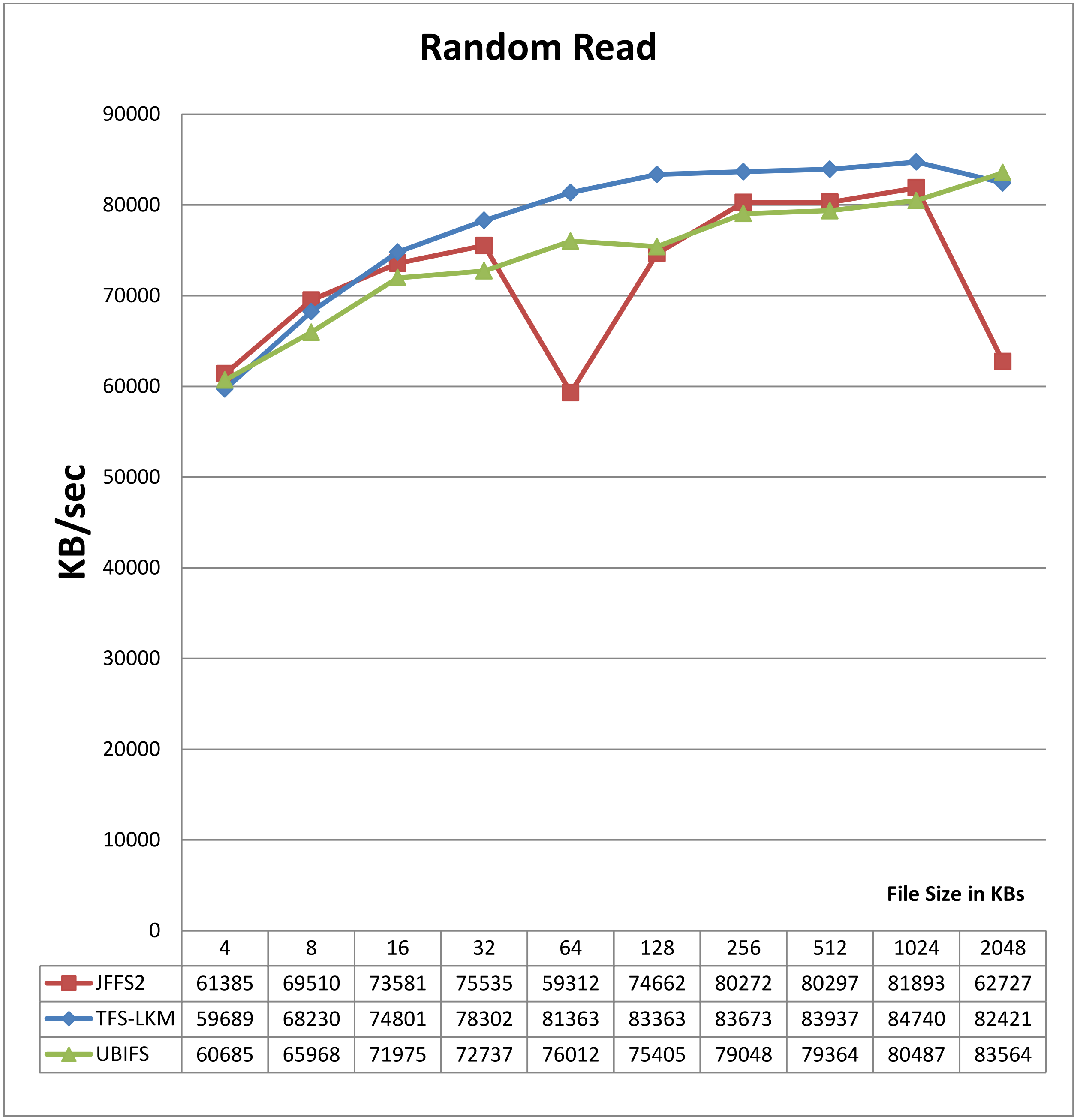
In this configuration, every application API request that changes the file system metadata will trigger a volume sync for TargetFS-LKM. This means that, upon recovery from an unexpected power loss, on the next mount, TargetFS-LKM will revert to the state immediately prior to the interrupted API request, or, in other words, all completed application API requests are immediately saved to the storage media.



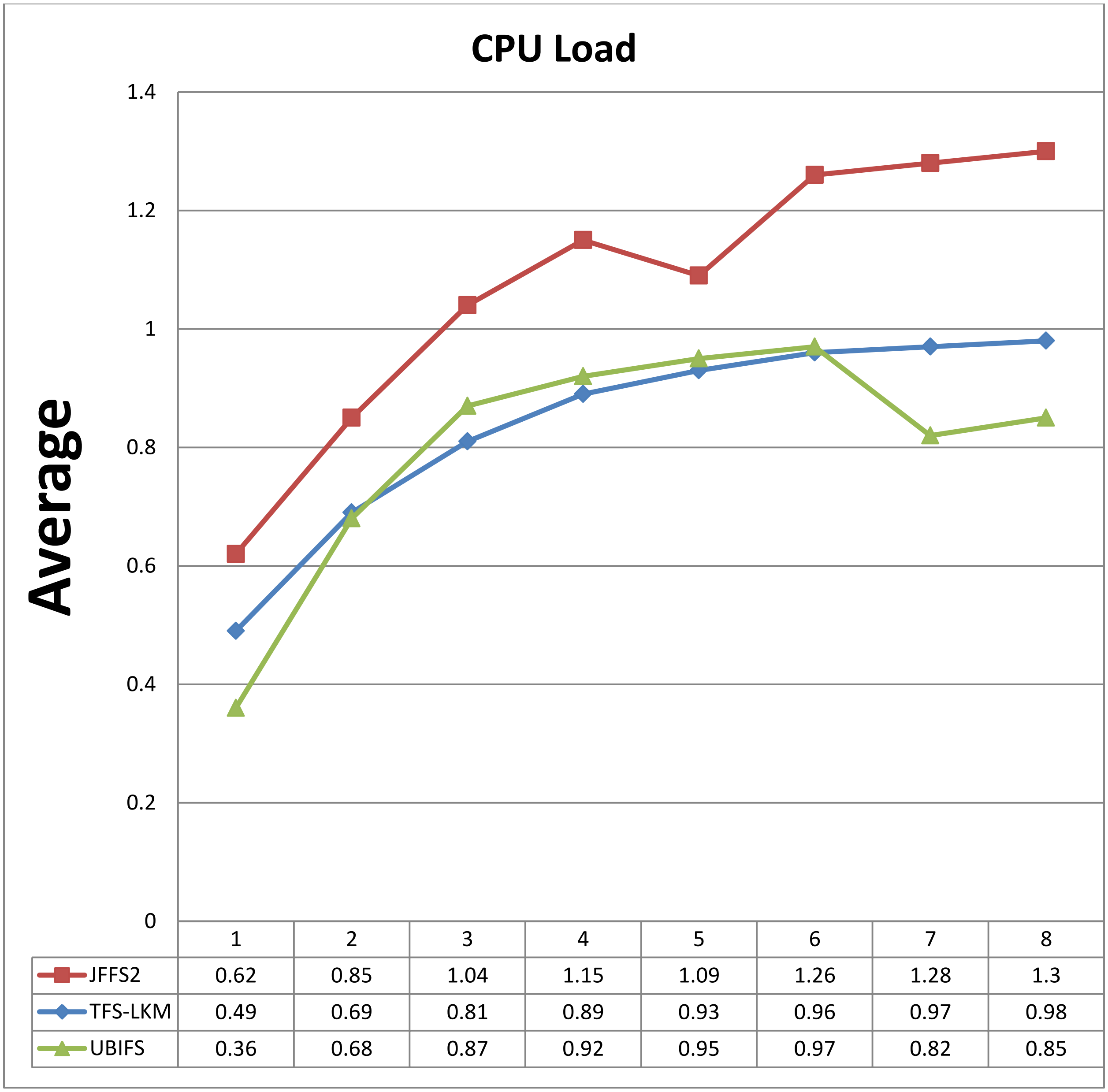
	4	8	16	32	64	128	256	512	1024	2048
■ JFFS2	480	1084	1782	2368	2267	3075	2626	3036	3173	2993
◆ TFS-LKM	3049	3126	3223	3177	3157	3168	3174	3171	3215	3179
▲ UBIFS	1639	2055	2507	2704	2782	2959	3023	3029	3039	3259



	4	8	16	32	64	128	256	512	1024	2048
■ JFFS2	66394	72693	76352	76638	79048	79275	80364	80462	81962	83629
◆ TFS-LKM	68472	74602	78917	80699	82734	84120	84614	83810	84827	82421
▲ UBIFS	65862	69426	73983	74012	76905	76298	78890	79710	81467	83817



	4	8	16	32	64	128	256	512	1024	2048
JFFS2	61385	69510	73581	75535	59312	74662	80272	80297	81893	62727
TFS-LKM	59689	68230	74801	78302	81363	83363	83673	83937	84740	82421
UBIFS	60685	65968	71975	72737	76012	75405	79048	79364	80487	83564



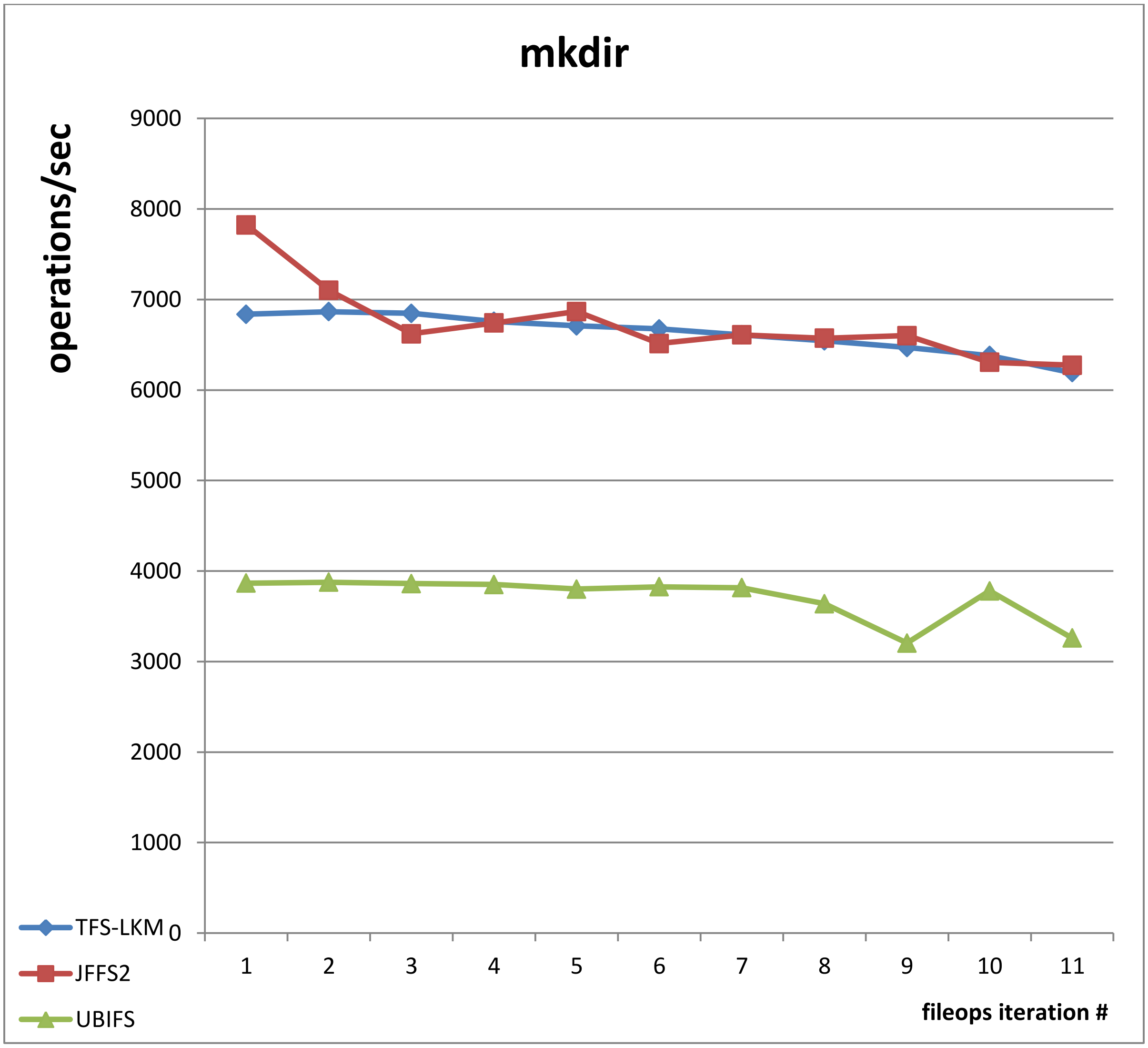
	1	2	3	4	5	6	7	8
■ JFFS2	0.62	0.85	1.04	1.15	1.09	1.26	1.28	1.3
◆ TFS-LKM	0.49	0.69	0.81	0.89	0.93	0.96	0.97	0.98
▲ UBIFS	0.36	0.68	0.87	0.92	0.95	0.97	0.82	0.85

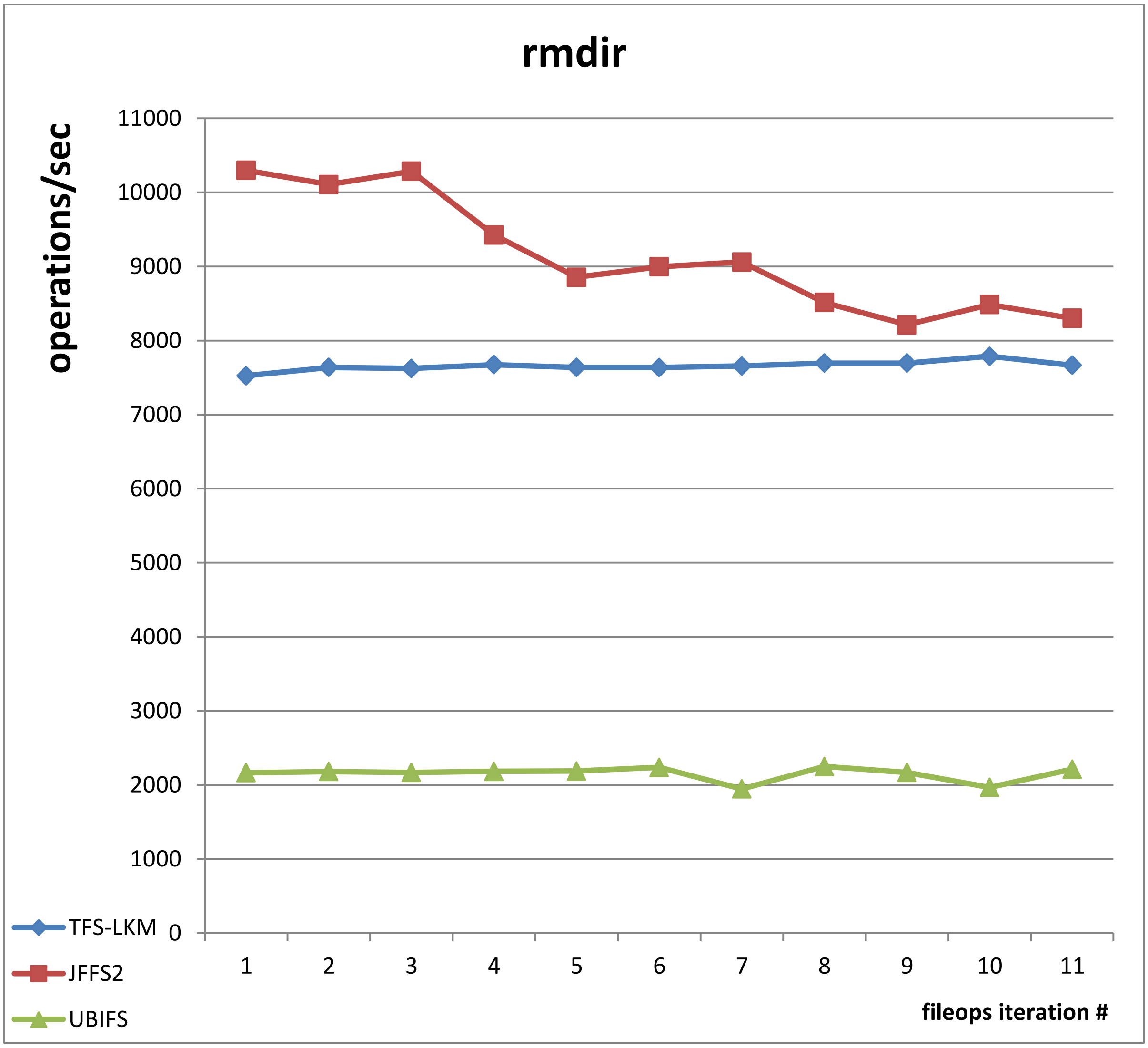
File/Directory Operations Time Measurements

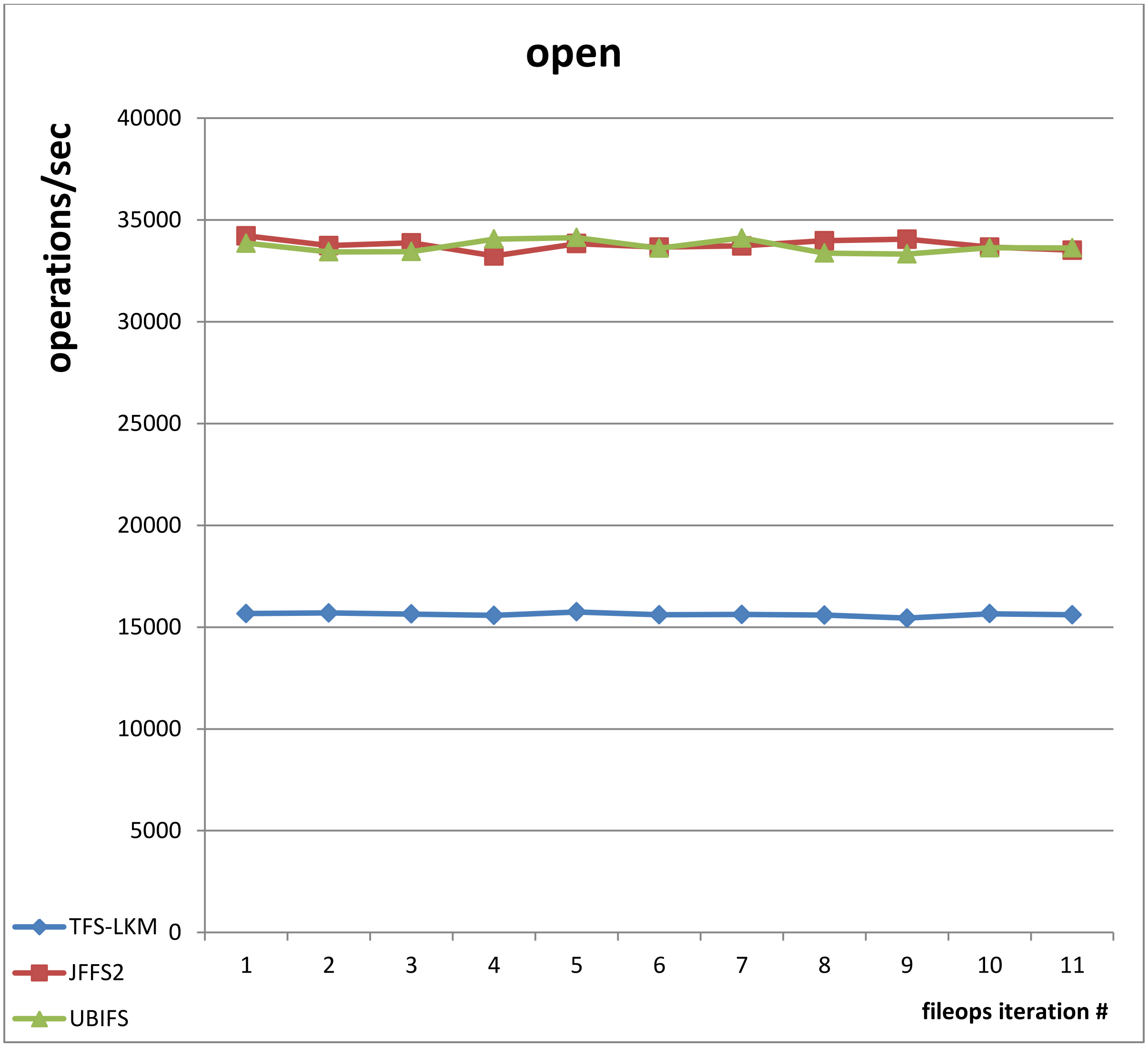
Besides read/write throughput, the IOZone benchmark measures average execution times for various file and directory operations such as: mkdir, rmdir, file open, file access, link, unlink, etc.

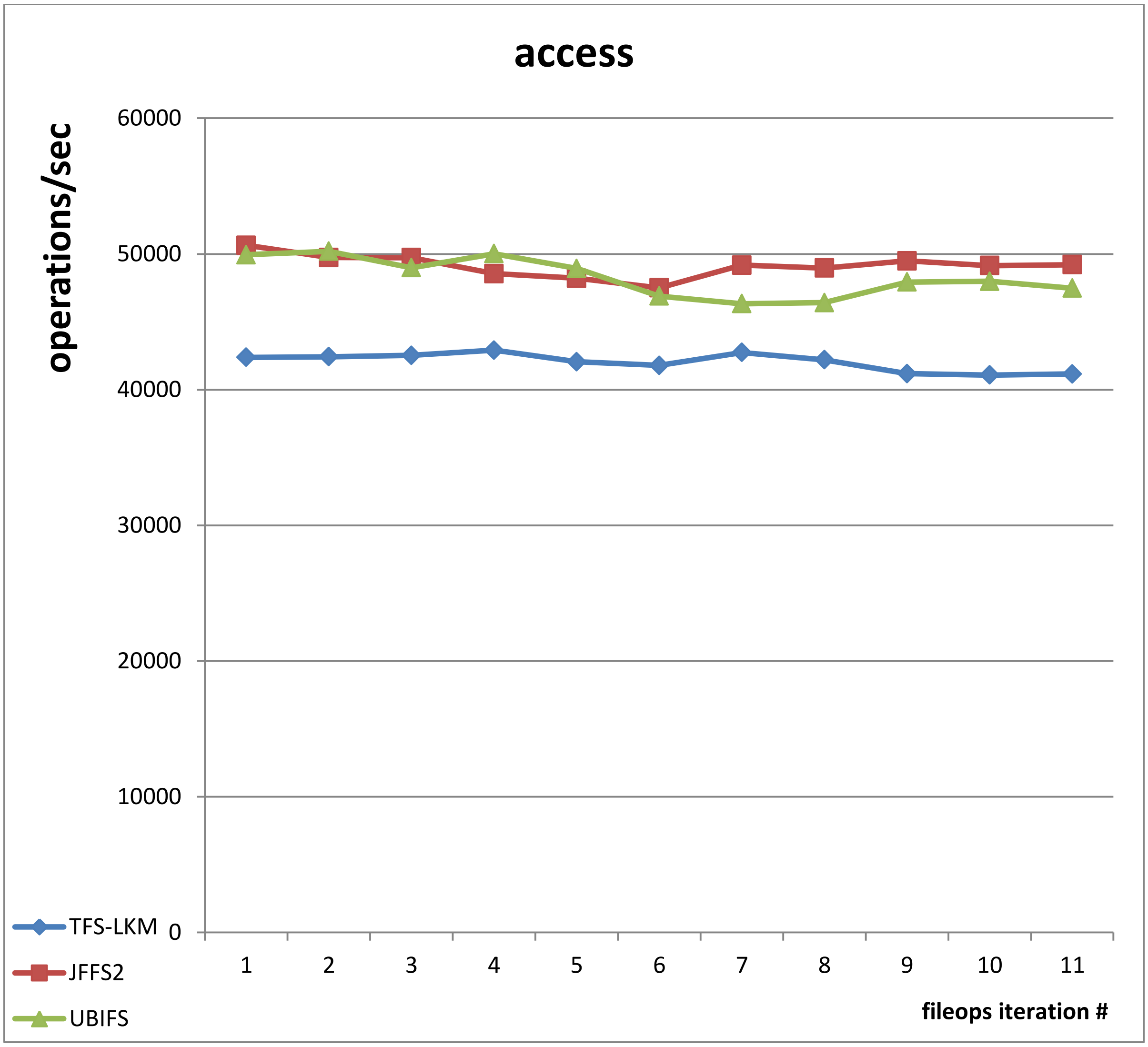
For both JFFS2 and UBIFS, some of these operations, like file open, are performed solely inside the Linux VFS layer without any other work done by the specific native Linux file system, and thus, they exceed TargetFS-LKM's performance since it has to do additional internal work to its meta information.

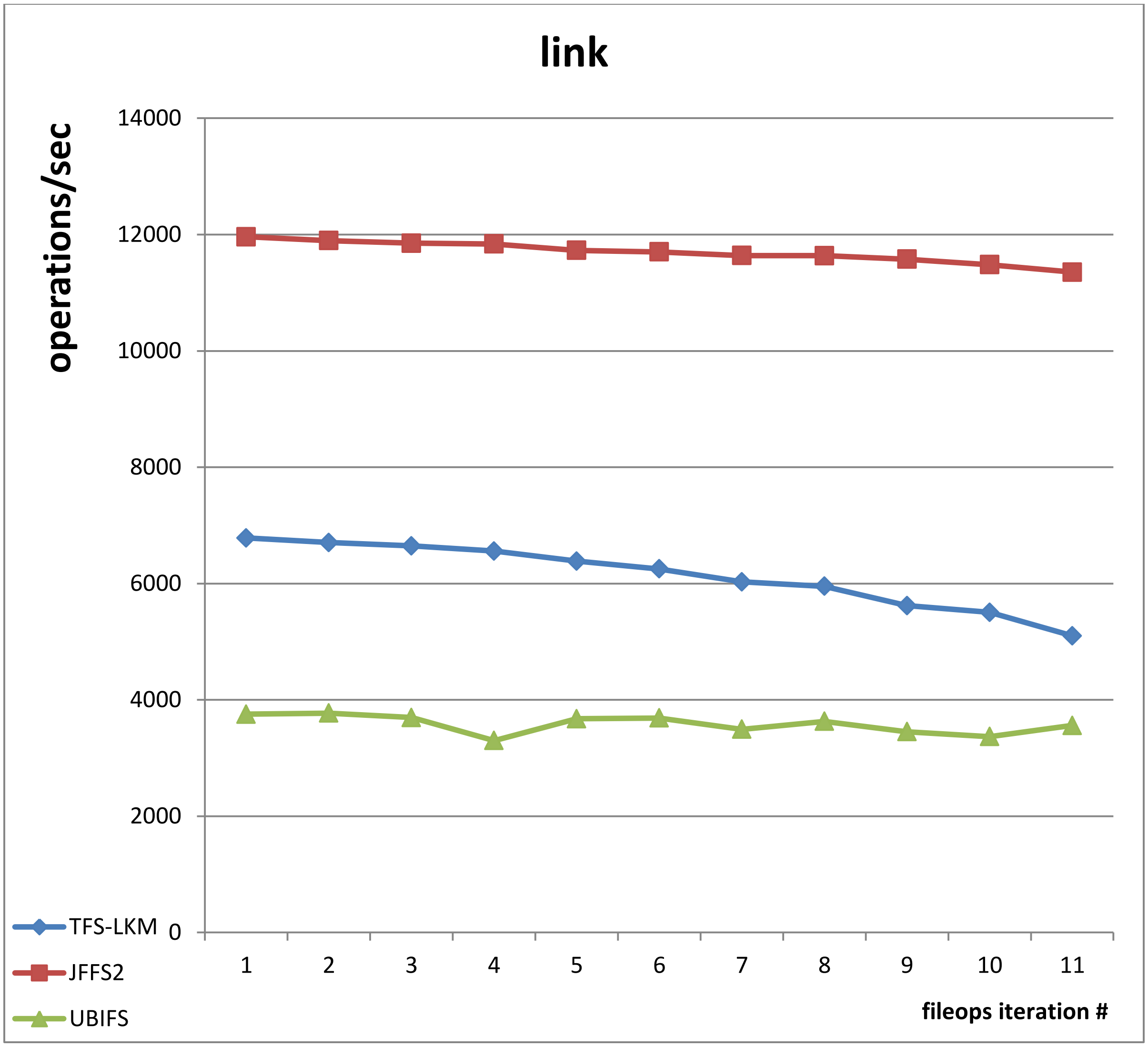
Furthermore, file systems such as JFFS2 keep all of their meta information in RAM, so, even for operations that result in a change of meta state, the execution time is faster than for file systems like TargetFS-LKM and UBIFS which have to cache theirs. While this is an advantage for small volumes, it is a big detriment to scalability.

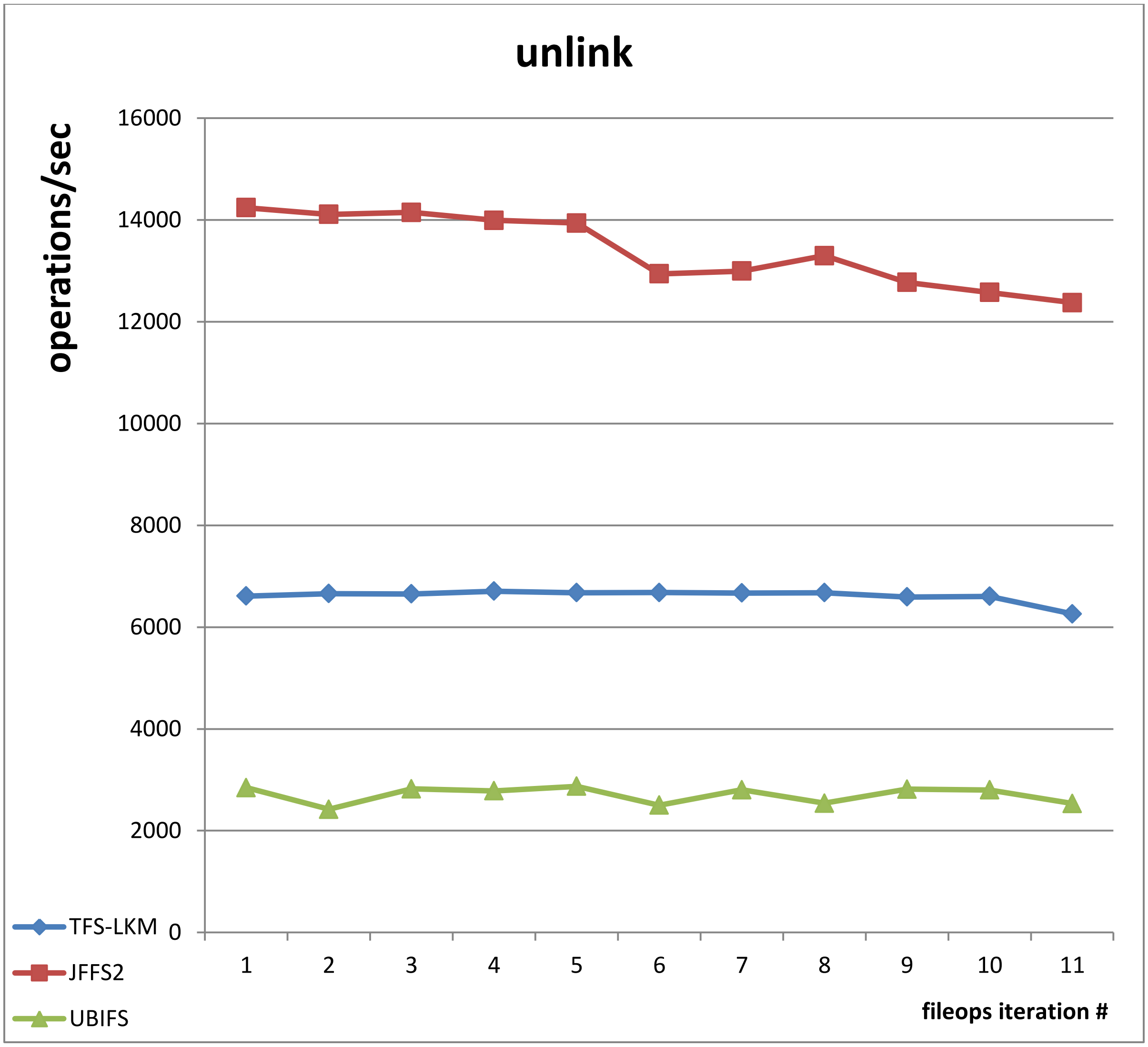



rmdir
operations/sec
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
1
2
3
4
5
6
7
8
9
10
11
fileops iteration #
TFS-LKM
JFFS2
UBIFS


open
operations/sec
40000
35000
30000
25000
20000
15000
10000
5000
0
1
2
3
4
5
6
7
8
9
10
11
fileops iteration #
TFS-LKM
JFFS2
UBIFS


access
operations/sec
60000
50000
40000
30000
20000
10000
0
1
2
3
4
5
6
7
8
9
10
11
fileops iteration #
TFS-LKM
JFFS2
UBIFS


link
operations/sec
14000
12000
10000
8000
6000
4000
2000
0
1
2
3
4
5
6
7
8
9
10
11
fileops iteration #
TFS-LKM
JFFS2
UBIFS


unlink
operations/sec
16000
14000
12000
10000
8000
6000
4000
2000
0
1
2
3
4
5
6
7
8
9
10
11
fileops iteration #
TFS-LKM
JFFS2
UBIFS

Mount Times

One of the strengths of the TargetXFS/TargetFTL-NDM/TargetNDM is its fast, deterministic and constant mount time, even in the case of recovery from unexpected power offs.

To showcase this, we have measured the time to mount a completely full 978 blocks volume on Blunk's test platform for TargetFS-LKM, JFFS2 and UBIFS.

For TargetFS-LKM we timed the kernel module load command and the volume mount command:

```
insmod targetsys.ko
mount -t txfs mtd3 /mnt/txfs_flash
```

For JFFS2 we timed the volume mount command:

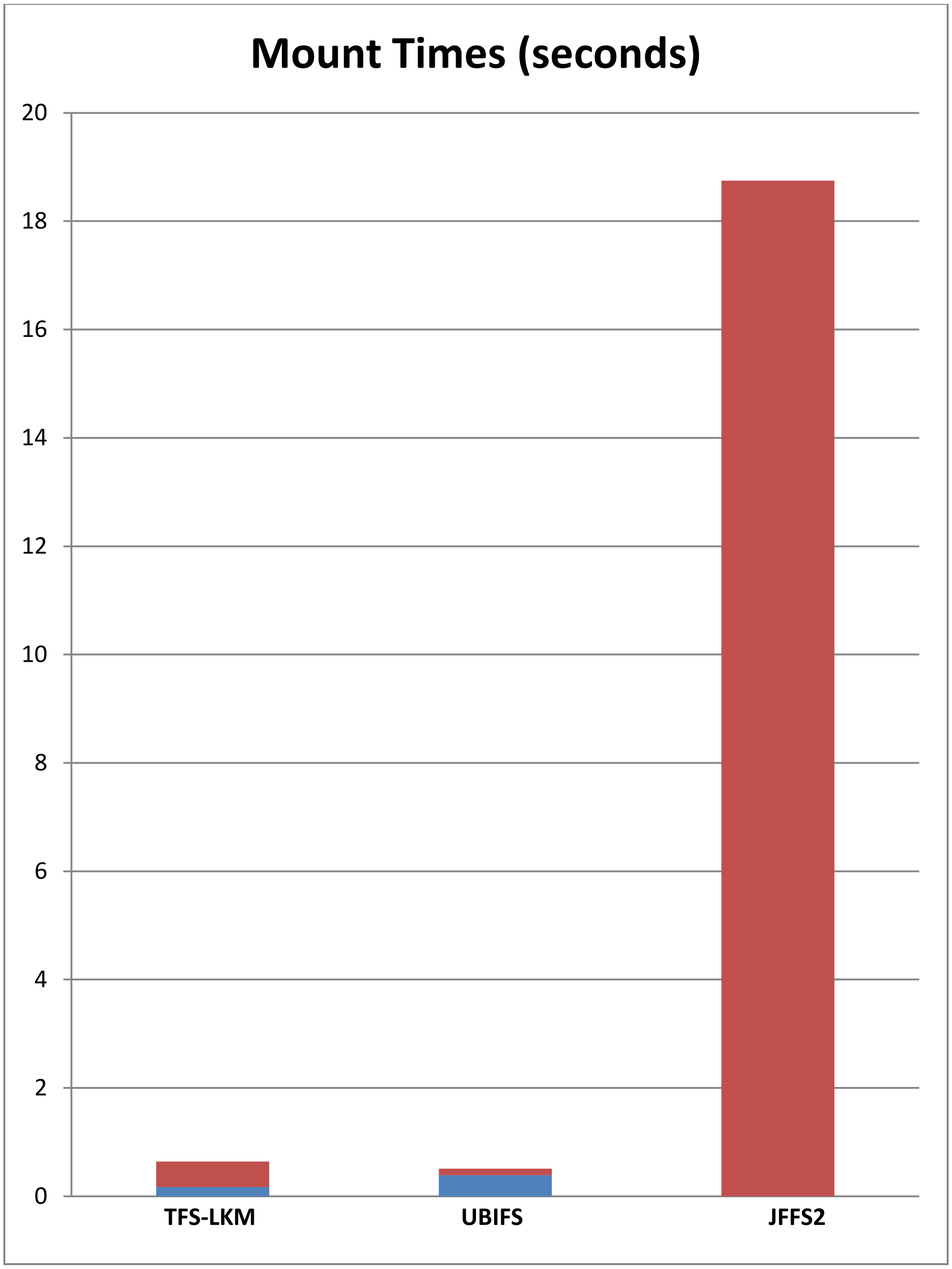
```
mount -t jffs2 mtd3 /mtn/flash_mtd
```

For UBIFS we timed the device add command and the volume mount command:

```
ubiattach /dev/ubi_ctrl -m 3
mount -t ubifs ubi0:benchmark /mnt/ubifs/
```

The table below and its corresponding graph contain the overall mount measurements:

	command	seconds	
TFS-LKM	insmod	0.17	
	mount	0.47	
		0.64	Total
JFFS2	mount	18.75	
		18.75	Total
UBIFS	ubiattach	0.39	
	mount	0.12	
		0.51	Total


Mount Times (seconds)
20
18
16
14
12
10
8
6
4
2
0
TFS-LKM
UBIFS
JFFS2

Intentionally left blank.