WiscAFS

Asket Agarwal Preetham Srinivas Kikkeri Aanandita Dhawan

DESIGN

- The file system is implemented with an AFS-like protocol.
- Upon file open, the client will request the entire file from the server and store it in the client's local cache.
- Subsequent read and write requests will be redirected to the local cached copy. On close, the dirty file will be flushed back to the server. If the file was opened in read-only mode then it is not pushed to server.
- Client replicates the server directory structure.
- Server is stateless in our design.

Client Design (cache)

- Client does a whole file caching where it fetches the files from the server.
- Whenever a user opens a file, the client does a getattribute call to the server to see if the file was modified. If yes, it fetches it.
- All reads/writes happens to the local copy with no server interaction to increase performance.
- Upon close the entire file is flushed to the server.

Filebench Metrics

Filecreate.f

```
finish               432ops       43ops/s   0.0mb/s    0.000ms/op [0.000ms - 0.001ms]
append-file          433ops       43ops/s  43.2mb/s   23.036ms/op [3.648ms - 25.253ms]
11.050: IO Summary:   433 ops 43.296 ops/s 0/43 rd/wr  43.2mb/s 23.036ms/op
```

Filemicro_createfiles

```
finish               2436ops      244ops/s   0.0mb/s    0.000ms/op [0.000ms - 0.001ms]
closefile1           2436ops      244ops/s   0.0mb/s    0.134ms/op [0.086ms - 0.257ms]
writefile1           2436ops      244ops/s   0.2mb/s    0.333ms/op [0.186ms - 0.511ms]
createfile1          2436ops      244ops/s   0.0mb/s    3.608ms/op [2.964ms - 6.718ms]
409.681: IO Summary:  7308 ops 730.716 ops/s 0/244 rd/wr   0.2mb/s 1.358ms/op
```

Filemicro_createrand

```
11.248: Per-Operation Breakdown
finish               80ops        8ops/s   0.0mb/s    0.000ms/op [0.000ms - 0.002ms]
sync                 80ops        8ops/s   0.0mb/s   15.627ms/op [8.553ms - 132.780ms]
append-file          808ops       81ops/s  39.7mb/s   10.791ms/op [0.171ms - 24.412ms]
11.248: IO Summary:   888 ops 88.791 ops/s 0/81 rd/wr  39.7mb/s 11.227ms/op
```

Filemicro_rwritedsync.f

```
finish               3261ops      652ops/s   0.0mb/s    0.000ms/op [0.000ms - 0.001ms]
write-file           3262ops      652ops/s   1.3mb/s    1.521ms/op [0.360ms - 2105.543ms]
29.532: IO Summary:  3262 ops 652.317 ops/s 0/652 rd/wr   1.3mb/s 1.521ms/op
```

Filemicro_seqread.f

```
seqread-file         44853ops     4485ops/s 4371.6mb/s    0.221ms/op [0.138ms - 2510.954ms]
33.897: IO Summary: 44853 ops 4484.658 ops/s 4485/0 rd/wr 4371.6mb/s 0.221ms/op
```

Filemicro_seqwrite.f

```
finish               0ops        0ops/s   0.0mb/s    0.000ms/op [0.000ms - 0.000ms]
write-file           444ops       44ops/s  44.3mb/s   22.471ms/op [3.680ms - 24.810ms]
11.300: IO Summary:   444 ops 44.395 ops/s 0/44 rd/wr  44.3mb/s 22.471ms/op
```

Filemicro_statfile.f

```
statfile1           112714ops    11271ops/s   0.0mb/s    1.755ms/op [0.061ms - 10.018ms]
168.727: IO Summary: 112714 ops 11270.545 ops/s 0/0 rd/wr   0.0mb/s 1.755ms/op
```

Filemicro_writefsync.f

```
finish              14ops        1ops/s   0.0mb/s    0.001ms/op [0.000ms - 0.006ms]
sync-file           14ops        1ops/s   0.0mb/s   32.960ms/op [23.206ms - 145.686ms]
append-file         15226ops     1522ops/s  11.9mb/s    0.621ms/op [0.179ms - 3.420ms]
11.280: IO Summary: 15240 ops 1523.843 ops/s 0/1522 rd/wr  11.9mb/s 0.651ms/op
```

Webserver.f

```
appendlog            1449ops      145ops/s   1.1mb/s    8.365ms/op [0.317ms - 64.755ms]
closefile10          1399ops      140ops/s   0.0mb/s    2.200ms/op [0.078ms - 7.810ms]
readfile10           1399ops      140ops/s   2.2mb/s    3.726ms/op [0.127ms - 13.845ms]
openfile10           1403ops      140ops/s   0.0mb/s   27.577ms/op [2.791ms - 76.815ms]
closefile9           1403ops      140ops/s   0.0mb/s    2.151ms/op [0.073ms - 7.715ms]
readfile9            1403ops      140ops/s   2.1mb/s    3.924ms/op [0.106ms - 14.556ms]
openfile9            1407ops      141ops/s   0.0mb/s   27.865ms/op [4.029ms - 61.863ms]
closefile8           1407ops      141ops/s   0.0mb/s    2.178ms/op [0.071ms - 9.028ms]
readfile8            1407ops      141ops/s   2.2mb/s    3.868ms/op [0.101ms - 13.931ms]
openfile8            1413ops      141ops/s   0.0mb/s   27.304ms/op [3.617ms - 73.206ms]
closefile7           1414ops      141ops/s   0.0mb/s    2.129ms/op [0.077ms - 7.272ms]
readfile7            1414ops      141ops/s   2.1mb/s    3.787ms/op [0.147ms - 16.676ms]
openfile7            1418ops      142ops/s   0.0mb/s   26.949ms/op [3.449ms - 57.598ms]
closefile6           1419ops      142ops/s   0.0mb/s    2.153ms/op [0.083ms - 9.235ms]
readfile6            1419ops      142ops/s   2.2mb/s    3.744ms/op [0.162ms - 13.687ms]
openfile6            1424ops      142ops/s   0.0mb/s   27.247ms/op [3.435ms - 72.481ms]
closefile5           1425ops      142ops/s   0.0mb/s    2.218ms/op [0.066ms - 9.082ms]
readfile5            1425ops      142ops/s   2.2mb/s    3.987ms/op [0.149ms - 13.474ms]
openfile5            1427ops      143ops/s   0.0mb/s   27.231ms/op [3.997ms - 67.959ms]
closefile4           1427ops      143ops/s   0.0mb/s    2.166ms/op [0.077ms - 8.009ms]
readfile4            1427ops      143ops/s   2.2mb/s    3.917ms/op [0.121ms - 13.664ms]
openfile4            1430ops      143ops/s   0.0mb/s   27.162ms/op [3.688ms - 68.930ms]
closefile3           1430ops      143ops/s   0.0mb/s    2.137ms/op [0.077ms - 8.985ms]
readfile3            1430ops      143ops/s   2.2mb/s    4.070ms/op [0.143ms - 13.176ms]
openfile3            1438ops      144ops/s   0.0mb/s   27.058ms/op [3.575ms - 74.976ms]
```

Some Takeaways:

Sequential reads/writes perform better than their random counterparts as we expect in a local filesystem as all reads/writes are performed locally.

Workloads with more FS opertions like create and mkdir are typically more slower than workloads doing only file operations as filesystem operations in our design always goes to the server.

Also sync operations (for example flusing to disk after every write) performs poorly because after every write the data has to be flushed to the server as opposed to async operations where it can happen in the background.

Consistency

The data is flushed to disk after every close operation. We ensure that the client which closes the file last "wins" the write. Here is a small demo showing that :

https://drive.google.com/file/d/1zqRifms4HjGXSbSg96NgvsrR9uTCtWlm/view?usp=share_link

We have tested the functionality using the test case provided and also ran on 2 of our own test cases.

1) Close at the end : The 2 clients opens a local copy and do not close the file till the end. We here see that all reads/writes only happen to the local copy and the clients run independently.

2) Interleaved close : Both clients start writing to a file simultaneously, one client closes the file before the other (ensured using signals). When the first client opens the file, he sees the contents of the other client and not its own.