ZIL support in Lustre*

Alexey Zhuravlev

Sept 22, 2015

Async writes are great, but…

- Sync is required sometimes
  - fsync(2), fdatasync(2)
  - Commit on Share (OST, DNE)
  - OOM
- Regular commit procedure (TXG commit) is very expensive
  - Seconds or dozen seconds
- Something much faster is needed

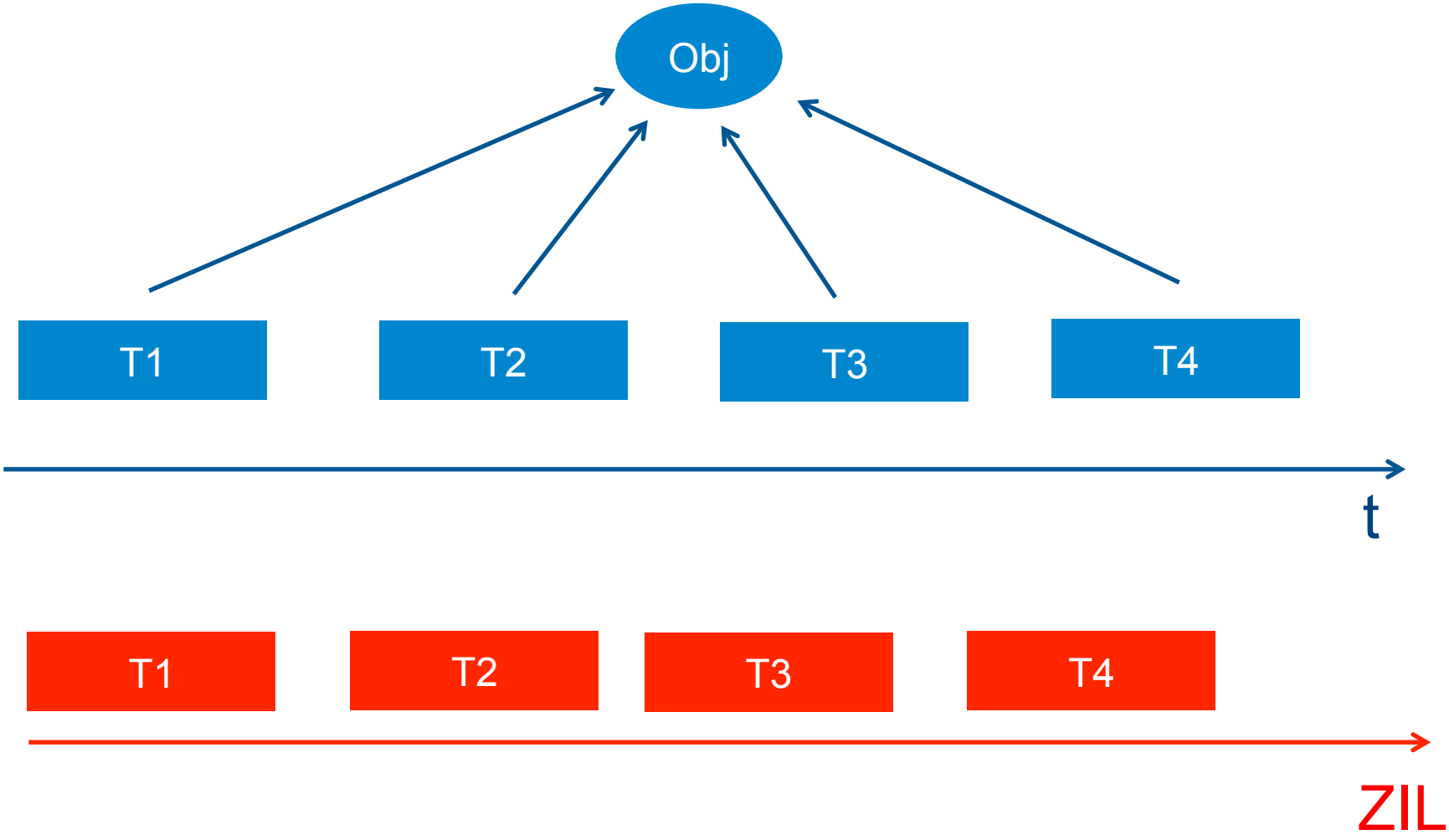
We aren’t alone

- Native ZFS faces the same issue
- They invented a mechanism to improve this – ZIL
  - ZFS Intent Log
  - a chain of blocks with records can be written out of regular TXG
  - a record describes regular POSIX op, very few types - transaction
  - can start to go out early and keep going along with running TXG
  - data can be written out of original order and only when required
    - not metadata though
  - the log is discarded when corresponding TXG is committed
  - If the system crashes, then the log is replayed at mount

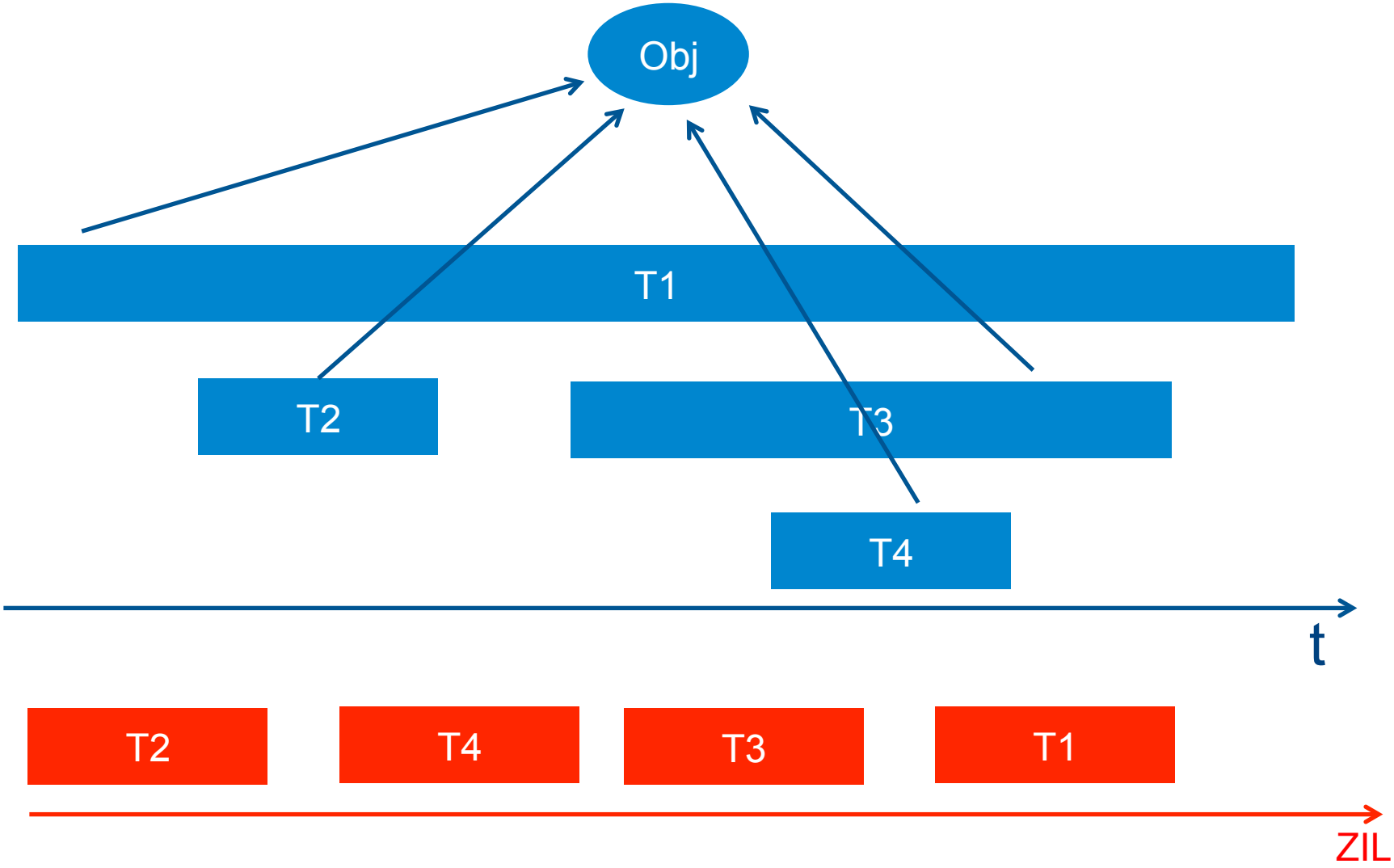
Lustre is different

- many different types of operations
  - regular, hsm, changelogs, dne, lfsck
- the operations can be much more complicated:
  - Include llogs, last_rcvd, lov_objids, changelogs, hsm
- hard to describe all this with a single type operand
  - Especially in the multi-layer stack
- Lustre transactions race each another
  - few concurrent transactions may change same bits
  - and depend each on another (llog, lov_objids)

Transactions in ZFS: no races by locks



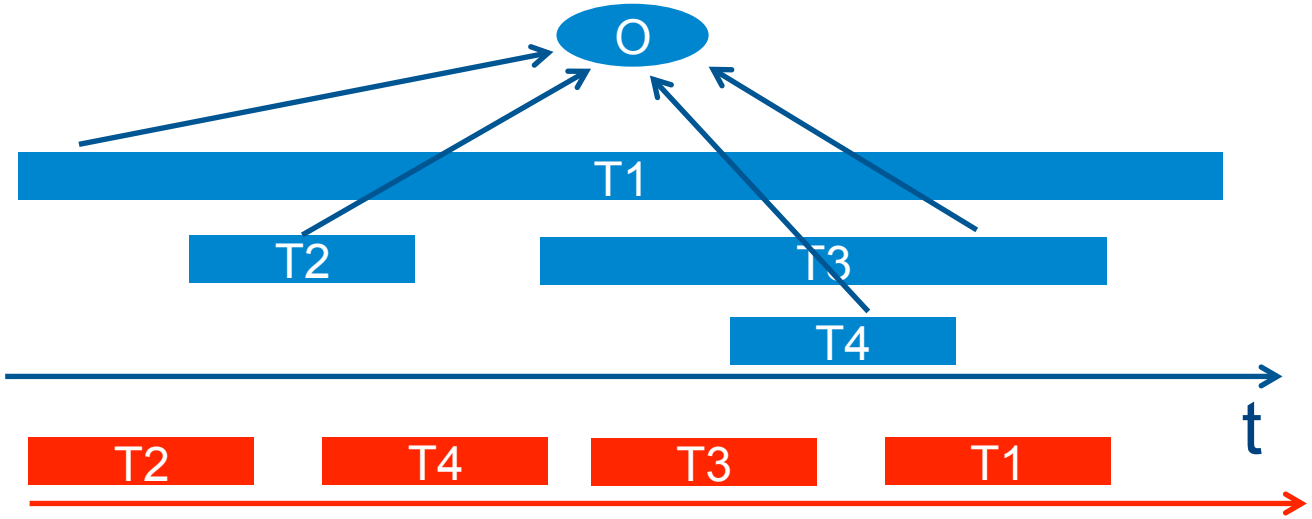
Transactions in Lustre: races all the time



Races in Lustre

- Gives performance
  - Big transactions don’t need to serialize due to a single object
- Keep simplicity
- Bring a lot of headache in cases like ZIL
  - end of transaction doesn’t mean the preceding are completed
  - replay in the same order is not possible (no control over TXG)
- Ideally we want the model used in ZFS:
  - no races within transaction: by locks or by structures
  - help ZIL and DNE
  - a lot of changes to Lustre core (ldiskfs-based are affected)
  - will take some time to productize

Out of order transactions



- transactions get into ZIL once completed
- see on the picture: T3 get into ZIL before T1
- but T3 depends on specific state of the object
  - which was introduced by T1
- we need barriers similar to TXG commit point
  - there is no running transaction and all previous transactions got into ZIL
  - this barrier can be a separate record generated when sync is requested

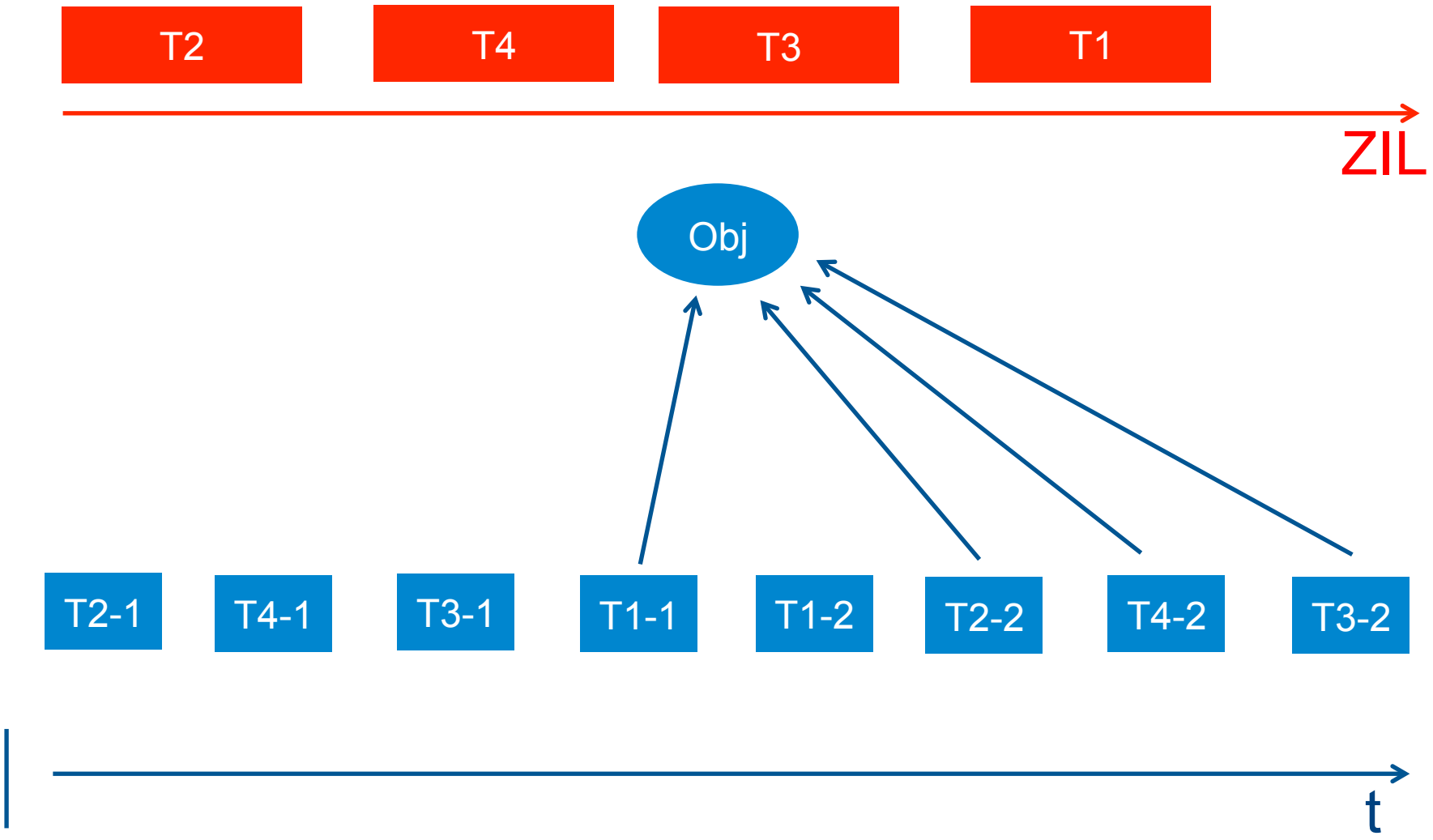
VBR with ZIL

- every object has a version
- every update stores version taken from object
- every update increments object's version
- we can identify how updates depends each on another
- we can apply them in correct order
- but we still need to preserve the property of atomicity
  - we would have to have few transactions open?
  - bad idea – DMU can block any new transaction, so deadlock

Snapshots: a way to mimic TXG

- what if we take a snapshot before we start to replay ?
- we can break all the transactions into a set of single updates
- they are partial ordered by FID + version
- they can be applied one by one
- If the node crashes before completion:
  - we still have the snapshot
  - rollback to that
  - repeat from the beginning

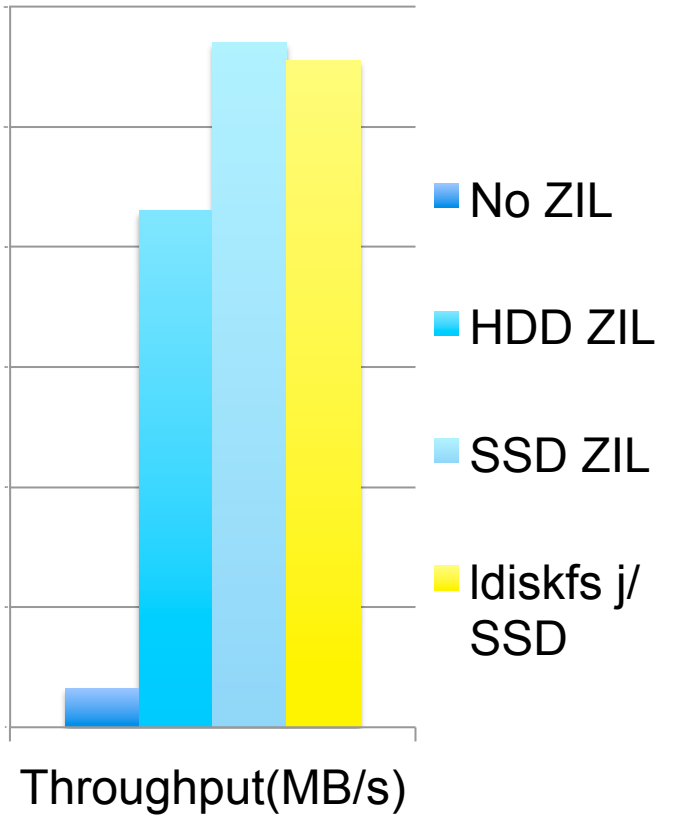
Replay: serialized

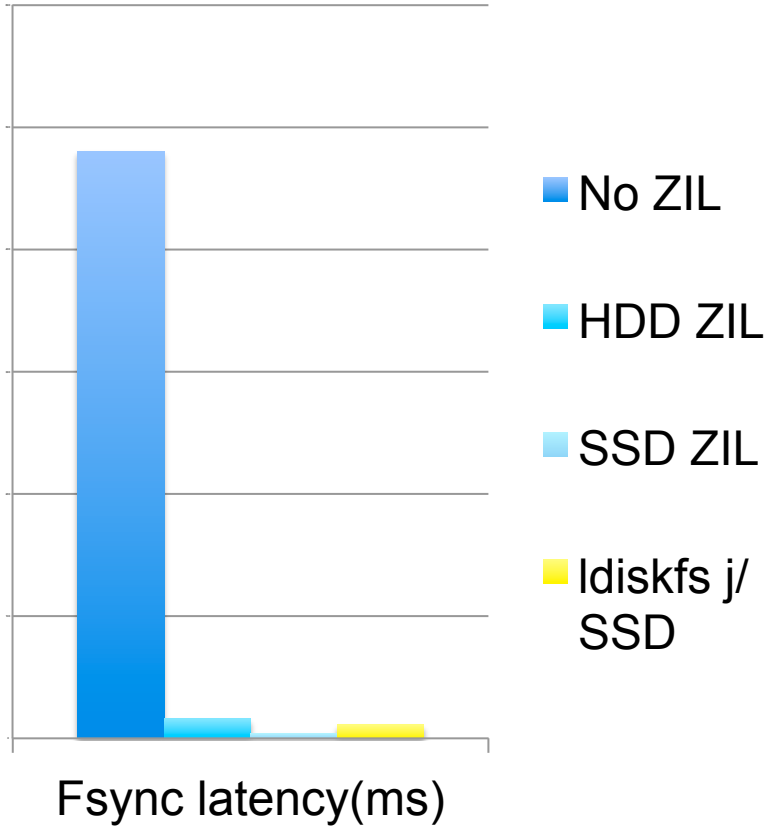


Technical details

- ZIL isn't a subject to snapshot/rollback mechanism
  - it's just discarded
  - so we have to copy ZIL to a stable object at the beginning and then take a snapshot
- we still have to pay some attention to order of replay
  - there are inter-update dependencies
  - e.g. insert need a referenced object to exist
    - an additional issue - compatibility is never beautiful
    - but you can mount directly
  - Indirect writes
    - a special record in ZIL referencing a block

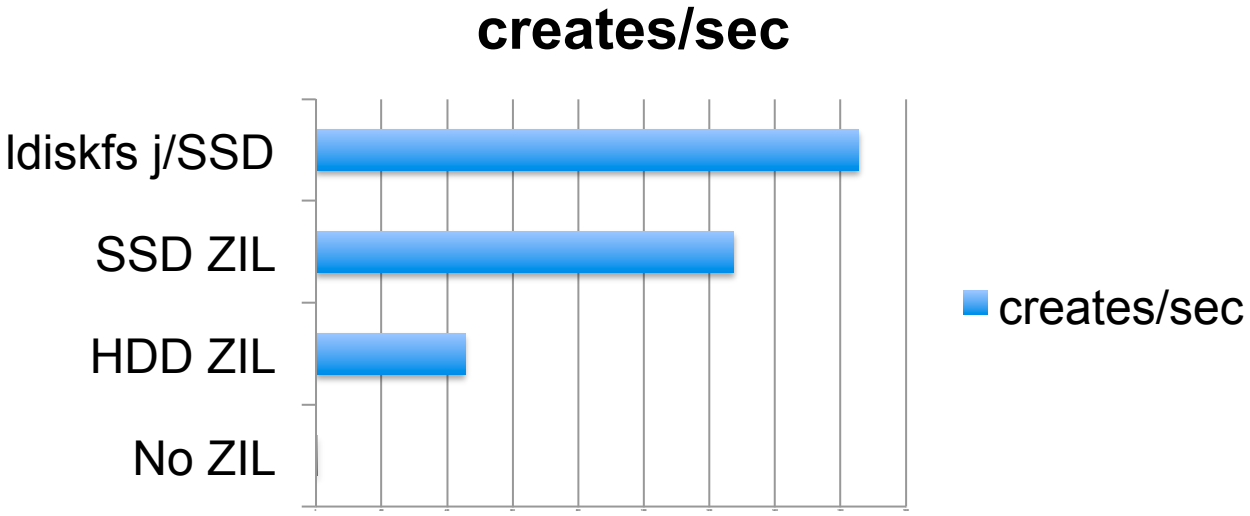
dbench 8 1 client
1 MDT: 10 HDD RAID-0 VDEV + SSD
1 OST: 25 HDD RAID-0 VDEV + SSD





Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

mdtest: 1 task, 2000 files, -y (fsync)
1 MDT: 10 HDD RAID-0 VDEV + SSD
1 OST: 25 HDD RAID-0 VDEV + SSD



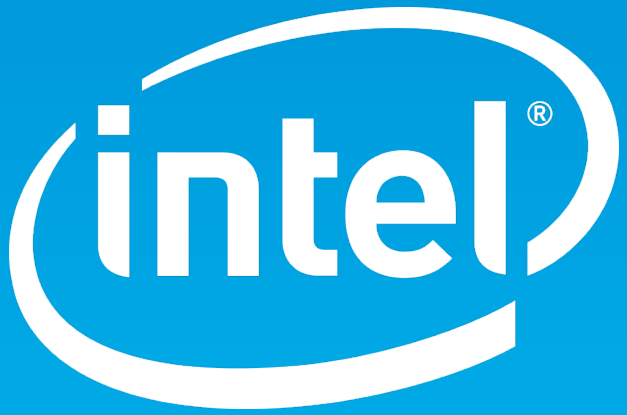
- Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Ideally we want to change Lustre

- Remove races
  - Better structuring in hot paths: e.g. llog to become an index
  - Additional locking is an option for rare cases
- Out-of-order transactions
  - yet another order of magnitude improvement is possible
    - no need to flush the whole ZIL
  - Need support in the protocol
    - A window of transno's the client doesn't know commit status and have to replay, then the server can discard after consulting with a list/index of executed (multislot?)
- DNE would be the first one to utilize all this

Current state

- tracked in LU-4009
- few preparation patches can be landed separately
- targets community 2.9 release


intel