The Delos Fuzzy Log Project (part 1)

ABSTRACT

This report introduces the Delos Fuzzy Shared Log abstraction and its implementation. The Delos project is aimed at building a distributed datastore for control plane applications such as coordination services, SDN controllers, filesystem namespaces, and big data schedulers which have strong requirements for consistency as well as performance. The Fuzzy Log provides the abstraction of a Directed Acyclic Graph of mutations with dependencies on previous mutations by way of directed edges, allowing applications to construct and access a durable, iterable partial order of updates in the system.

In particular, the report details the contributions of the author to the project, specifically the design and implementation of the colour-enhanced directed acyclic graph API, the creation of a distributed multiput protocol and the conception of an end-to-end use case for mixed consistency guarantees.
INTRODUCTION

The fuzzy log project is an attempt to create a better abstraction for distributed metadata storage, in particular for use by control plane services, including but not limited to SDN controllers (ONOS, NSX), filesystem namespaces (HDFS namenode), big data runtime schedulers (Google Omega), and general-purpose coordination services (e.g. ZooKeeper, Chubby). All of these services need to ensure that state is managed with strong consistency, concurrency control, fault-tolerance and performance.

Currently, your typical service of this nature might use ARIES for failure atomicity and durability with regard to reboots, some version of MultiPaxos for availability and durability despite machine crashes and some form of two-phase commit protocol for transactional isolation and failure atomicity across partitioned data, requiring the services of a number of competent distributed systems engineers to comprehend, implement and manage. The ultimate aim of the project is to abstract away the snarl of complex protocols, each providing, for the most part, a small subset of these properties separately, that are currently utilized by such services, allowing the application developer to concentrate on the application and leave the distributed systems stuff completely to the experts.

A relatively recent development in this field is the use of the shared log abstraction to simplify the creation and maintenance of control plane services. Projects such as Professor Balakrishnan’s own CorfuDB [corfudb] and Tango [tango2013] and others such as Hyder[hyder2011], Calvin[calvin2012], and Confluent [confluent] have leveraged a
single, globally shared log to derive properties such as consistency, durability, failure atomicity and transactional isolation via simple append/read operations. This is considerably more simple, from the view of the application developer, than mucking about with the distributed protocols mentioned above - it is possible to build a shared log version of ZooKeeper in only a few thousand lines of code, and order of magnitude less than the Paxos-based version.

Enter the fuzzy shared log: a new abstraction allowing for applications to define arbitrary partial orders on updates, allowing it to, in principle, pick exactly the level of consistency it needs in every case.

The weaker consistency guarantee we’re interested in is causal consistency, similar to the definition in the COPS [cops2011] and Eiger [eiger2013] papers. In our version, we will allow client apps to specifically define the causal dependencies of each mutation by way of edges on the DAG.

THE FUZZY LOG ABSTRACTION

A fuzzy log is a type of directed acyclic graph (DAG) that can be constructed and traversed concurrently by multiple clients. Each node in the DAG is tagged with a colour attribute; multiple nodes can have the same color, and a single node can be tagged with more than one color.
In its simplest form, the abstraction allows clients to add a new node with the property that it is tagged with some set of colours which depends on previous nodes of other colours.

These colours can be mapped to a number of things depending on the requirements of the data structure implemented on top of this abstraction. The three most obvious mapping are as follows: a single colour could be mapped to a single writer, to a single type of operation (add = red, subtract = blue, etc.), or to a specific instance of a data structure (each map gets its own colour, for example).

These dependencies can either be stale dependencies representing things this client has seen, to provide merely causal consistency, or all previous updates up till that point on the colours chosen, providing linearizability with respected to those colours.

**SPECIFIC CONTRIBUTIONS TO THE PROJECT**

1. DAG API Design
2. Mapping colours to chains on the multi-chain log
3. Mixed consistency use case
4. Distributed multiput protocol
DAG API Design

I designed the api which is exposed to client applications of the DAG so as to most simply allow the application to express its partial orders, and to specify its consistency requirements for both readers and writers.

newdag(colours) // return dag instance pointer based on colours of interest

readnext(dag, snapshot/null) // read from a dag instance

writenode(dag, colours, seen_colours, succeed_colours, payload)

snapshot(dag)

The 'seen_colours' field essentially exposes a causal guarantee to the application: the node will have happens after edges from the last nodes of a colour that the app has seen (which the colour layer knows about - it has seen all of the apps reads).

The 'succeed_colours' field exposes a sequential ordering guarantee: the new node will be ordered immediately after the latest node of the succeed_colours.

readnext can either operate over a snapshot (in which case it operates over a linearizable view of the DAG instance wrt the snapshot) or not, in which case it operates over a causally consistent view of the bundle. It returns the next node you need to read, in DAG order, based on all dependencies you need to see. There is an implicit fairness guarantee here - you are guaranteed to make progress one colour of interest in an instance approximately as often as you are each other colour modulo dependencies.
Causal consistency use case

As part of building the ability to request weaker consistency into the system, I was tasked with finding end-to-end application use cases which need both strong and weak consistency for separate kinds of operations but can still make progress safely during a network partition.

The example I came up with was that of changing album permissions on a Social Network.

Story:

India, China and the US are friends on Fuzzybook. They each access Fuzzybook through different datacenters. India creates an album which initially has global permissions. As creator of the album, only India can change permissions, but anyone with access can see and upload pictures. The US and China both see this empty, globally accessible gallery. At some future point, he changes the permissions to only include himself and China. China sees this, but the lazy US is currently idle so doesn't yet know. Suddenly, there is a network partition and India is cut off from the world. China, relying on the changed permission, uploads military secrets to the album. China wants to keep making progress, but also wants to make sure that the US can't see the pictures which are being uploaded. This requires that the underlying datastore be causally consistent - the new picture upload needs to have the permission change as a dependency.
Implementation:

We define a permission-map as follows:

ops:

create() //returns map id

put(map, picture) //returns key

get(map, key) //returns picture

change_permissions(map, client_ids/null) //null is global permission

Only the creator client can change permissions.

In the colour api, with one writer per colour, this maps to the following:

create-> singleton append

put-> singleton put with seen_colours = creator colour

get-> causal read (readnext(null))

change_permission-> make it more privates, singleton, make it more public, multiput (could both be multiput, but strictly worse performance)
Mapping Colours to Chains

The Delos project’s implementation of the fuzzy log abstraction is built on top of a multi-chain log, similar to multiple parallel instances of a Corfu log. In order to implement the colour layer, I had to develop a mapping from colours to these chains.

The trivial implementation involves mapping each colour to a chain. This means that there is no parallelism on writes to a single colour. To mitigate this, I’m developing a mapping from a colour to a tunable number of chains, allowing for app-specific parallelism guarantees. An alternate mapping also being developed is mapping each colour to a dynamic number of chains, with frequent checkpointing to garbage collect whole chains. A third mapping being considered is the mapping of one colour to a number of chains which are distributed across datacenters for greater locality.

The performance of these mappings is currently being considered from a theoretical perspective, and will be tested more practically in the coming months.

Distributed Multiput Protocol

In order to implement the strong consistency guarantees offered by the sequential ordering part of the colour API, we needed a distributed multi-put protocol to atomically place a node (or multiple nodes logically equivalent to one node) across multiple chains. I
developed a chain-based scale-out transaction scheme for this purpose. It is detailed below.

**Pyramid Scheme**

This transaction scheme is based on a k-level hierarchy (in the form of a tree) of which the lowest level will consist of chain servers (data servers) and the k-1 higher levels will consist of lock servers (metadata servers) arranged in pyramidal fashion up to the root.

We will first consider the 2-level case, then generalize to the k-level case.

**2-level case:**

Each lock server has one counter per node it serves (this can be modified to one counter per chain, but that would introduce complications when scaling up layers). The lock-server supports multi-counter atomic increments. We intend to implement this on top of the multi-chain log abstraction so that a multi-counter increment can be represented by a multi-chain transaction. This also allows us to bind the client transaction data to the timestamp.

Each node maintains atomic lock and unlock counters (these can be implemented as a chain/separate chains as well).

When a client wants to perform a multi-node transaction it performs a multi-counter increment on the lock server across all relevant nodes, retaining a vector timestamp of the counter values for the relevant nodes immediately after said increment. The data which is to be written for the client transaction is attached to the transaction representing
the increment. At this point, any client can complete this transaction, so if the client fails
at any point, some other client can finish it off eventually.

The client then goes to each of the chain servers in parallel, attempting to increment to
counter to their timestamp.

Let the client have timestamp x. It then goes to chain server α, which has timestamp y. If if
x > y + 1, then it goes to the lock server and attempts to complete the x - y missing
transactions before re-attempting it's own transaction. If x == y + 1, it starts its transaction,
writing to node α locking it until it receives unlock y. If x <= y, then transaction has been
started (and possibly) by someone else (by pulling the data from the lock server). While a
node is locked, no writes return - all transactions and singletons queue until the unlock.
No reads beyond the transaction tied to the current unlock value return either - a reader
can attempt to complete the transactions blocking its reads if it so desires. Once the
transaction write returns on all relevant chain servers, the client sends the unlock to all of
them. If the unlock counter on any node hasn't reached the value needed, the client
tries to complete the transactions and unlock forward until the right value is reached.

Ideal Time:

1 Rtt to write to per-node logs

1 Rtt to lock

1 Rtt to unlock

-------
3 Rtts

**k-level (k>2) case**

If a lock-server gets saturated or for latency reasons you can divide its clients between two lock-servers 'regions' and add an additional lock-server to deal with cross-region transactions in the same way as above. This can, in principle, scale out infinitely, with the client acquiring timestamps from lock-servers from level 2 up to the minimal level in the hierarchy for cross-node transactions.

Ideal Time (for a write touching k levels):

1 Rtt to write to per-node logs

k-1 Rtt to lock

k-1 Rtt to unlock

--------

k+1 Rtts (we can do the unlocks in parallel)

The k-level hierarchy allows for region-specific locking, ensuring that the system as a whole doesn’t freeze up due to specific sets of nodes and greatly reducing system wide latency (in theory) as compared to a single global lock-server due the possibility of optimizing locations of specific region lock-servers. In addition, this protocol ensures that scale-out can be infinite (in theory).
This protocol is currently being implemented by Josh Lockerman for use in the Delos project.

**CURRENT PROJECT STATUS**

As of the present moment, I have written an implementation of the colour layer on top of the chain layer which maps one colour to one chain. This implementation is not currently functional and will be rewritten completely.

The current implementation of the chain layer is buggy and does not support distributed transactions. It will likely be patched up and rewritten.

We have recently acquired a number of high-performance servers to test scale-out performance with. These are being set up over the next few weeks.

The fuzzy log team was originally gearing up for a submission to OSDI ‘16, but will now likely submit to SOSP ‘17 in hopes of having an extremely strong, perhaps even award-winning paper.

**FUTURE WORK**

Over the course of next semester, I will build out a robust implementation of the colour layer of the fuzzy shared log which will be tunable parallel for commuting writes.
I will then write a number of general-use transaction-supporting data structures, including maps, counters and queues of various kinds on top of this layer.

I will then implement a number of end-to-end applications on top of the colour layer with the help of the data structures mentioned above to (1) catalog the lines of code these applications take and (2) measure performance with respect to the same or similar applications implemented on top of other distributed datastore.

These applications will likely include the causal use cases mentioned above, SDN controllers and HDFS.

The team will continue to work on fleshing out the theoretical implications of the colour-enhanced DAG abstraction.

As mentioned earlier, this all leads up to a submission to SOSP in a little under one year.

**ACKNOWLEDGEMENTS**

This paper is based on work done in close collaboration with Jose Faleiro and Joshua Lockerman, both Yale CS PHD students, and Professor Mahesh Balakrishnan. Professors James Aspnes and Daniel Abadi are advisors on the project.

I'd like to thank Jose for being gentle to an ignorant undergrad, Josh for most emphatically *not* being gentle, Professor Aspnes for showing me all the terrible things the adversary can do as well as letting me do this project at all, and finally, most importantly, Mahesh for reviving my faith in C.S. academia and showing me at least one true path to Zen enlightenment.
BIBLIOGRAPHY ATTACHED BELOW
Bibliography

References

agiga .................................................................
Agiga tech raid cache.

corfudb ............................................................
Corfudb.
https://github.com/corfudb.

fcntlmanpage ..........................................
fcntl man page.

fusepage .............................................................
Filesystem in userspace.

fusionio ............................................................
Fusion-io.

fusionioatomic ..........................................
Fusion-io atomic multi-block writes.

fusionioacm ..................................................
Fusion-io auto-commit memory.

iozone ............................................................
Iozone filesystem benchmark.

leveldbbench ...................................................
LevelDB benchmarks.

seagatekinetic ...................................................
Seagate kinetic open storage platform.

storagespaces ...................................................
Storage spaces.
winfs  .................................................................
Winfs.

abaditensorflow  ................................................
Tensorflow: Large-scale machine learning on heterogeneous systems, 2015.
Software available from tensorflow.org.

shingled  ............................................................
A. Aghayev and P. Desnoyers.
Skylight a window on shingled disk operation.
In USENIX FAST, pages 135–149, 2015.

sinfonia2007  ......................................................
Sinfonia: a new paradigm for building scalable distributed systems.

openreplica2012  ...................................................
D. Altınbükün and E. G. Sirer.
Commodifying replicated state machines with openreplica.

BST2000  ..............................................................
Highly concurrent shared storage.

ssdalloc2011  ......................................................
A. Badam and V. S. Pai.
SSDAlloc: hybrid SSD/RAM memory management made easy.
In USENIX NSDI, 2011.

diffraid2010  ....................................................... 
Differential raid: rethinking raid for ssd reliability.
ACM Transactions on Storage (TOS), 6(2):4, 2010.

corfu2012  ............................................................
M. Balakrishnan, D. Malkhi, V. Prabhakaran, T. Wobber, M. Wei, and J. D. Davis.
CORFU: A shared log design for flash clusters.
In USENIX NSDI, pages 1–14, 2012.
Tango: Distributed data structures over a shared log.
In ACM SOSP, pages 325–340.

Pads: A policy architecture for distributed storage systems.
In NSDI, volume 9, pages 59–73, 2009.

A critique of ansi sql isolation levels.

P. A. Bernstein, S. Das, B. Ding, and M. Pilman.
Optimizing optimistic concurrency control for tree-structured, log-structured databases.

P. A. Bernstein, V. Hadzilacos, and N. Goodman.
Concurrency control and recovery in database systems, volume 370.

P. A. Bernstein, C. W. Reid, and S. Das.
Hyder-a transactional record manager for shared flash.
In CIDR, pages 9–20, 2011.

K. Birman and T. Joseph.
Exploiting virtual synchrony in distributed systems.

T. Blackwell, J. Harris, and M. I. Seltzer.
Heuristic cleaning algorithms in log-structured file systems.

Corey: An operating system for many cores.
In USENIX OSDI, 2008.
chubby2006  
M. Burrows.
The chubby lock service for loosely-coupled distributed systems.

was2011  
B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie,  
Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri,  
A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. ul Haq,  
M. I. ul Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett,  
S. Sankaran, K. Manivannan, and L. Rigas.  
Windows azure storage: a highly available cloud storage service with  
strong consistency.  

chandra2007paxos  
T. D. Chandra, R. Griesemer, and J. Redstone.  
Paxos made live: an engineering perspective.  

bigtable2006  
F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows,  
T. Chandra, A. Fikes, and R. E. Gruber.  
Bigtable: A distributed storage system for structured data.  

chang2008bigtable  
F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows,  
T. Chandra, A. Fikes, and R. E. Gruber.  
Bigtable: A distributed storage system for structured data.  

mime1992  
C. Chao, R. English, D. Jacobson, A. Stepanov, and J. Wilkes.  
Mime: a high performance parallel storage device with strong recovery  
guarantees.  

chidambaram2012  
V. Chidambaram, T. Sharma, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau.  
Consistency without ordering.  
in *FAST*, 2012.
clements2013 ........................................
A. T. Clements, M. F. Kaashoek, N. Zeldovich, R. T. Morris, and
E. Kohler.
The scalable commutativity rule: designing scalable software for multicore
processors.
In ACM SOSP, 2013.

mars2013 ........................................
From ARIES to MARS: Reengineering transaction management for next-
generation, solid-state drives.
In SOSP, 2013.

nvheaps2011 ......................................
J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, R. Jhala,
and S. Swanson.
NV-Heaps: making persistent objects fast and safe with next-generation,
non-volatile memories.

ycsb ................................................
B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears.
Benchmarking cloud serving systems with ycsb.
In ACM SoCC, pages 143–154, 2010.

spanner2013 ......................................
J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman,
S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kan-
thak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle,
S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymiaik, C. Taylor,
R. Wang, and D. Woodford.
Spanner: Google’s globally distributed database.

wayback2004 ....................................
B. Cornell, P. A. Dinda, and F. E. Bustamante.
Wayback: A user-level versioning file system for linux.
In USENIX ATC, FREENIX Track, 2004.

strata ..............................................
B. Cully, J. Wires, D. Meyer, K. Jamieson, K. Fraser, T. Deegan,
D. Stodden, G. Lefebvre, D. Ferstay, and A. Warfield.
Strata: scalable high-performance storage on virtualized non-volatile
memory.
In Proceedings of the 12th USENIX conference on File and Storage
logicaldisk1993 ..............................................................
  The logical disk: A new approach to improving file systems.

dean2008mapreduce ............................................................
  J. Dean and S. Ghemawat.
  Mapreduce: simplified data processing on large clusters.

dynamo2007 ...........................................................................
  G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman,
  A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels.
  Dynamo: amazon’s highly available key-value store.

demers1987 ............................................................................
  A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker,
  H. Sturgis, D. Swinehart, and D. Terry.
  Epidemic algorithms for replicated database maintenance.
  In *Proceedings of the sixth annual ACM Symposium on Principles of

driscoll1986 ............................................................................
  Making data structures persistent.
  In *Proceedings of the eighteenth annual ACM symposium on Theory of

gsi2005 ..................................................................................
  S. Elnikety, F. Pedone, and W. Zwaenepoel.
  Database replication using generalized snapshot isolation.

loge1992 ................................................................................
  R. M. English and A. A. Stepanov.
  Loge: a self-organizing disk controller.

memc3 ...................................................................................
  B. Fan, D. G. Andersen, and M. Kaminski.
  Memc3: Compact and concurrent memcache with dumber caching and
  smarter hashing.
  *USENIX NSDI*, 2013.

clio1987 ................................................................................
  R. Finlayson and D. Cheriton.
  Log files: An extended file service exploiting write-once storage.
M. Flouris and A. Bilas.
Clotho: Transparent Data Versioning at the Block I/O Level.

M. D. Flouris and A. Bilas.
Violin: a framework for extensible block-level storage.

G. R. Ganger.
*Blurring the line between OSes and storage devices.*

A. K. Goel, J. Pound, N. Auch, P. Bumbulis, S. MacLean, F. Färber,
F. Gropengiesser, C. Mathis, T. Bodner, and W. Lehner.
Towards scalable real-time analytics: An architecture for scale-out of olxp workloads.

J. Gray and L. Lamport.
Consensus on transaction commit.

S. D. Gribble, M. Welsh, R. Von Behren, E. A. Brewer, D. Culler,
The ninja architecture for robust internet-scale systems and services.

R. Guerraoui and M. Kapalka.
On the correctness of transactional memory.

J. G. Hansen and E. Jul.
Lithium: virtual machine storage for the cloud.

T. Harris, J. Larus, and R. Rajwar.
*Transactional Memory.*
zebra1993 .................................
The zebra striped network file system.

waf1994 ..................................
D. Hitz, J. Lau, and M. A. Malcolm.
File system design for an nfs file server appliance.

zookeeper2010 ..........................
P. Hunt, M. Konar, F. P. Junqueira, and B. Reed.
Zookeeper: Wait-free coordination for internet-scale systems.
In USENIX ATC, page 9, 2010.

isard2007dryad ..........................
M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly.
Dryad: distributed data-parallel programs from sequential building
blocks.

flashbackedram ..........................
J. Jose, M. Banikazemi, W. Belluomini, C. Murthy, and D. K. Panda.
Metadata persistence using storage class memory: experiences with
flash-backed dram.
In Proceedings of the 1st Workshop on Interactions of NVM/FLASH with

keckslaw ............................... Is keck’s law coming to an end?
http://spectrum.ieee.org/semiconductors/optoelectronics/is-kecks-law-coming-to-an-end

kungocc .................................
On optimistic methods for concurrency control.

cassandra2010 ..........................
A. Lakshman and P. Malik.
Cassandra: a decentralized structured storage system.

paxos1998 ............................... L. Lamport.
The part-time parliament.
paxos2001  
L. Lamport et al.
Paxos made simple.

hints1983  
B. W. Lampson.
Hints for computer system design.

leveldb  
Leveledb.
http://leveldb.org/.

thor1996  
B. Liskov, A. Adya, M. Castro, S. Ghemawat, R. Gruber, U. Maheshwari,
Safe and efficient sharing of persistent objects in thor.

cops2011  
Don’t settle for eventual: scalable causal consistency for wide-area storage with cops.

eiger2013  
Stronger semantics for low-latency geo-replicated storage.

low2014graphlab  

riovista1997  
D. E. Lowell and P. M. Chen.
Free transactions with rio vista.

boxwood2004  
Boxwood: Abstractions as the foundation for storage infrastructure.
tempest2008 .................................................................
T. Marian, M. Balakrishnan, K. Birman, and R. Van Renesse.
Tempest: Soft state replication in the service tier.

turbo2008 .................................................................
J. Matthews, S. Trika, D. Hensgen, R. Coulson, and K. Grimsrud.
Intel® turbo memory: Nonvolatile disk caches in the storage hierarchy
of mainstream computer systems.

objectstore2003 ...........................................................
Object-based storage.

parallax .................................................................
Hutchinson, and A. Warfield.
Parallax: virtual disks for virtual machines.

blockmason2008 ...........................................................
Block mason.
In Workshop on I/O Virtualization, 2008.

aries1992 .................................................................
C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz.
Aries: a transaction recovery method supporting fine-granularity locking
and partial rollbacks using write-ahead logging.

click1999 .................................................................
R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek.
The click modular router.

muniswamy2004 ...........................................................
A versatile and user-oriented versioning file system.

murray2013naiad ...........................................................
Naiad: a timely dataflow system.
In Proceedings of the Twenty-Fourth ACM Symposium on Operating
N. Narula, C. Cutler, E. Kohler, and R. Morris.  
Phase reconciliation for contended in-memory transactions.  

E. B. Nightingale, P. M. Chen, and J. Flinn.  
Speculative execution in a distributed file system.  

Flat datacenter storage.  
In *USENIX OSDI*, 2012.

J. Nomani and J. Szefer.  
Predicting program phases and defending against side-channel attacks using hardware performance counters.  

M. A. Olson.  
The design and implementation of the inversion file system.  

K. Ostrowski, K. Birman, D. Dolev, and J. H. Ahnn.  
Programming with live distributed objects.  

D. A. Patterson, G. Gibson, and R. H. Katz.  
A case for redundant arrays of inexpensive disks (raid).  

Combo drive: Optimizing cost and performance in a heterogeneous storage device.  
In *First Workshop on Integrating Solid-state Memory into the Storage Hierarchy*, volume 1, pages 1–8, 2009.

D. Peng and F. Dabek.  
Large-scale incremental processing using distributed transactions and notifications.  
everythinglocking

A. Pennarun.
Everything you never wanted to know about file locking.
http://apenwarr.ca/log/?m=201012#13.

txos

D. E. Porter, O. S. Hofmann, C. J. Rossbach, A. Benn, and E. Witchel.
Operating system transactions.

prabhakaran2005

V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau.
Analysis and evolution of journaling file systems.
In USENIX ATC, pages 105–120, 2005.

txflash2008

Transactional flash.
In USENIX OSDI, 2008.

venti2002

S. Quinlan and S. Dorward.
Venti: A new approach to archival storage.
In USENIX FAST, 2002.

meld2011

C. Reid, P. A. Bernstein, M. Wu, and X. Yuan.
Optimistic concurrency control by melding trees.

rocksdb

Rocksdb.
http://rocksdb.org/.

LFS1992

M. Rosenblum and J. K. Ousterhout.
The design and implementation of a log-structured file system.
e2epaper

J. H. Saltzer, D. P. Reed, and D. D. Clark.
End-to-end arguments in system design.
elephant1999

Deciding when to forget in the elephant file system.
Lightweight recoverable virtual memory.

Hathi: durable transactions for memory using flash.

Flashtier: a lightweight, consistent and durable storage cache.

F. B. Schneider.
Implementing fault-tolerant services using the state machine approach: A tutorial.

Seagate kinetic.
https://developers.seagate.com/display/KV/Kinetic+Open+Storage+Documentation+Wiki.

R. Sears and E. Brewer.
Stasis: Flexible transactional storage.
In *USENIX OSDI*, 2006.

N. Shavit and D. Touitou.
Software transactional memory.

J.-Y. Shin, M. Balakrishnan, T. Marian, and H. Weatherspoon.
Gecko: Contention-oblivious disk arrays for cloud storage.

Semantically-smart disk systems.


Dryadling: A system for general-purpose distributed data-parallel computing using a high-level language.
In *OSDI*, volume 8, pages 1–14, 2008.

De-indirection for flash-based ssds with nameless writes.
In *USENIX FAST*, 2012.