Virtual Consensus in Delos

Mahesh Balakrishnan, Jason Flinn, Chen Shen, Mihir Dharamshi, Ahmed Jafri, Xiao Shi, Santosh Ghosh, Hazem Hassan, Aaryaman Sagar, Rhed Shi, Jingming Liu, Filip Gruszczynski, Xianan Zhang, Huy Hoang, Ahmed Yossef, Francois Richard, Yee Jiun Song

Facebook, Inc.
the Facebook stack

turtles all the way down...
the Facebook stack

turtles all the way down...

```
the data plane

NoSQL  Databases  Web  AI/ML  ...

the control plane

Scheduler  Config  Naming  Sharding

control plane storage

fault-tolerant
[ zero-dependency, durable, highly available ]

rich API
[ transactions, range queries, secondary indices ]

[Twine, OSDI 2020]
```
the need for a new storage system

why not use an existing system?

MySQL  
ZippyDB  
ZooKeeper

rich API  
fault-tolerance
the need for a new storage system

why not use an existing system?

why not modify an existing system?

hard to change database API (e.g., add TXes to ZooKeeper)

hard to change consensus protocol (e.g., MySQL over ZAB)
the need for a new storage system

why not use an existing system?

why not modify an existing system?

problem statement circa 2017:
can we build a zero-dependency, fault-tolerant system with a rich API... *in months*?
the Delos storage system

or: “how to build a production-ready storage system in eight months”
the Delos storage system

or: “how to build a production-ready storage system in eight months”

database (materialized state)

shared log (consensus)

complex distributed protocol

the shared log is an API for consensus

CORFU (NSDI 2012), Tango (SOSP 2013), Hyder (SIGMOD 2015), CorfuDB, LogDevice, Scalog (NSDI 2020)…
the Delos storage system: **above** the log

- Client
- DelosTable
- DelosRuntime
- database
  (materialized state)
- shared log
  (consensus)
the Delos storage system: above the log

database (materialized state)

shared log (consensus)
the Delos storage system: **above** the log

- **Client**
- **DelosTable**
- **DelosRuntime**
- **database** (materialized state)
- **shared log** (consensus)
the Delos storage system: **above the log**

Database (materialized state)

Shared log (consensus)
the Delos storage system: **above** the log

Database (materialized state)

Shared log (consensus)
the Delos storage system: above the log

database (materialized state)

shared log (consensus)
the Delos storage system: above the log

- Client
- DelosTable
- DelosRuntime
- database (materialized state)
- shared log (consensus)

logical updates: e.g. “put x=5 if x==4”
the Delos storage system: above the log

Client

DelosTable

DelosRuntime

database (materialized state)

shared log (consensus)

logical updates: e.g. “put x=5 if x==4”

simple protocols above the log
the Delos storage system: above the log

Client

DelosTable

DelosRuntime

database (materialized state)

shared log (consensus)

aware of upstream API

oblivious to upstream API

logical updates: e.g. “put x=5 if x==4”

simple protocols above the log

easy to support new APIs
the Delos storage system: below the log

- database (materialized state)
- shared log (consensus)

Actions:
- append
- checkTail
- readNext
the Delos storage system: below the log

- database (materialized state)
- shared log (consensus)
- (ZooKeeper shim)
- appendcheckTail
- readNext
- /log
  - /log/entry0
  - /log/entry1
  - ...

(ZooKeeper shim)
the Delos storage system: below the log

- **database** (materialized state)
- **shared log** (consensus)

Pros:
- Fast to build/deploy
- Highly reliable

(ZooKeeper shim)

/log
- /log/entry0
- /log/entry1
- ...

Append, checkTail, readNext
the Delos storage system: below the log

Pros:
- fast to build/deploy
- highly reliable

Cons:
- very inefficient and slow
- service dependency

- database (materialized state)
- shared log (consensus)
- (ZooKeeper shim)

append
checkTail
readNext

/prod
- /log
- /log/entry0
- /log/entry1
- ...

(ZooKeeper shim)
how do we develop a new shared log?
(without re-implementing MultiPaxos...)
how do we **develop** a new shared log? (without re-implementing MultiPaxos...)

how do we **deploy** a new shared log? (without service downtime...)
how do we develop a new shared log? (without re-implementing MultiPaxos...)

how do we deploy a new shared log? (without service downtime...)

Virtual Consensus!
virtualizing consensus via the VirtualLog
virtualizing consensus via the VirtualLog
virtualizing consensus via the VirtualLog

- MetaStore
- VirtualLog
- Loglet
- ZKLoglet
- ZooKeeper

Operations:
- append
- checkTail
- readNext
- seal
virtualizing consensus via the VirtualLog

MetaStore

VirtualLog

Loglet

ZKLoglet

ZooKeeper

append
checkTail
readNext

append
checkTail
readNext

seal
virtualizing consensus via the VirtualLog

MetaStore

[ZKLoglet

Loglet

ZooKeeper

VirtualLog

append
checkTail
readNext

append
cHECKTail
readNext

seal
virtualizing consensus via the VirtualLog

MetaStore

[ver 1]
0 → 3: ZKLoglet
3 → inf: LDLoglet

VirtualLog

Loglet

ZKLoglet

ZooKeeper

LDLoglet

LogDevice
virtualizing consensus via the VirtualLog

we can deploy a new Loglet without downtime!
difficult to build a log that is simple, fast, fault-tolerant
difficult to build a log that is **simple, fast, fault-tolerant**
difficult to build a log that is simple, fast, fault-tolerant
difficult to build a log that is **simple, fast, fault-tolerant**

![Diagram](image)

**MetaStore**

- [ver 0]
- 0 → inf : Loglet

**VirtualLog**

**Loglet**

- no fault-tolerant consensus;
- only fault-tolerant seal

**simple, fast**

**simple, fault-tolerant**
difficult to build a log that is simple, fast, fault-tolerant

MetaStore:
- necessary and sufficient source of fault-tolerant consensus...
- [ver 0]
- 0 -> inf : Loglet

VirtualLog:
- simple, fault-tolerant

Loglet:
- simple, fast
- no fault-tolerant consensus; only fault-tolerant seal
difficult to build a log that is \textit{simple, fast, fault-tolerant}.

The \textit{VirtualLog} handles all \texttt{reconfiguration} (including leader election); the \textit{Loglet} provides failure-free \texttt{ordering}.

the \textit{VirtualLog} handles all \texttt{reconfiguration} (including leader election); the \textit{Loglet} provides failure-free \texttt{ordering}.
difficult to build a log that is simple, fast, fault-tolerant

the **VirtualLog** handles all reconfiguration (including leader election);
the **Loglet** provides failure-free ordering
the NativeLoglet

Delos Runtime (client) sequencer LogServer
the NativeLoglet

Delos Runtime (client) → sequencer → LogServer

1 quorum appends
the NativeLoglet

Delos Runtime (client)  sequencer  LogServer

1 quorum appends  2 quorum checkTail
the NativeLoglet

Delos Runtime (client) – LogServer

1. quorum appends  
2. quorum checkTail  
3. fast local reads
the NativeLoglet

1. quorum appends
2. quorum checkTail
3. fast local reads
4. fault-tolerant seal
switching logs mid-flight
deploying Loglets: converged vs. disaggregated
deploying Loglets: **converged** vs. **disaggregated**

- fast local log reads
- fate-sharing
deploying Loglets: **converged** vs. **disaggregated**

- **log+DB on each server:**
  - fast local log reads
  - fate-sharing

- **separate log and DB:**
  - less I/O contention
  - independent scaling
deploying Loglets: **converged** vs. **disaggregated**

**Converged** is preferred in production: the DB wants **fate-sharing** with the log...

- fast local log reads
- fate-sharing

- less I/O contention
- independent scaling
deploying Loglets: **converged** vs. **disaggregated**

**Converged**
- log+DB on each server:
  - fast local log reads
  - fate-sharing

**Disaggregated**
- separate log and DB:
  - less I/O contention
  - independent scaling

**Converged** is preferred in production:
the DB wants **fate-sharing** with the log...
(unless its own fate is bad...)
**deploying Loglets:** **converged** vs. **disaggregated**

- **converged** is preferred in production: the DB wants fate-sharing with the log...
  (unless its own fate is bad...)

- ...we can decouple fate on demand by reconfiguring to a **disaggregated** log
deploying Loglets: **converged** vs. **disaggregated**

log+DB on each server:
- fast local log reads
- fate-sharing

separate log and DB:
- less I/O contention
- independent scaling

**Converged** is preferred in production:
the DB wants fate-sharing with the log… (unless its own fate is bad…)

... we can **decouple** fate on demand by reconfiguring to a **disaggregated** log

10X higher throughput via **disaggregation**
composing Loglets: the **StripedLoglet**
composing Loglets: the *StripedLoglet*
composing Loglets: the **StripedLoglet**
composing Loglets: the **StripedLoglet**

![Diagram of DelosRuntime and StripedLoglet interaction]
composing Loglets: the **StripedLoglet**
composing Loglets: the **StripedLoglet**
composing Loglets: the **StripedLoglet**
composing Loglets: the **StripedLoglet**
composing Loglets: the **StripedLoglet**

DelosRuntime

DelosRuntime

0 1 2 3 6

0 3 6

1

2

rotating sequencer
composing Loglets: the **StripedLoglet**

Diagram showing the composition of DelosRuntime components and sharded acceptors.
composing Loglets: the **StripedLoglet**

![Diagram showing the StripedLoglet architecture with DelosRuntime and sharded acceptors.](image)

- DelosRuntime
- 1KB appends/s with 30 stripes
- Sharded acceptors
trimming the VirtualLog

ZKLoglet

NativeLoglet
trimming the VirtualLog

trim cold segments

ZKLoglet

NativeLoglet
trimming the VirtualLog

- trim cold segments
- remap cold segments
  - InfiniteLog → PiT restore
  - more durability
trimming the VirtualLog

- trim cold segments
- remap cold segments
  - InfiniteLog → PiT restore
  - more durability
trimming the VirtualLog

- trim cold segments
- remap cold segments
  - InfiniteLog \rightarrow PiT restore
  - more durability
- remap single slots
  - delete poison pill entries
  - less durability

Diagram:
- BackupLoglet
- NativeLoglet
Delos as a platform

original goal: can we build a zero-dependency, fault-tolerant system with a rich API... in months?
original goal: can we build a zero-dependency, fault-tolerant system with a rich API... \textit{in months}?

Delos as a platform

rich API

- DelosTable

fault-tolerant

- DelosRuntime
- VirtualLog

...in months

- ZKLoglet
original goal: can we build a zero-dependency, fault-tolerant system with a rich API... *in months*?
original goal: can we build a zero-dependency, fault-tolerant system with a rich API... in months?

Delos as a platform

rich API
- DelosTable

fault-tolerant
- DelosRuntime
- VirtualLog

...in months
- ZKLoglet
- NativeLoglet

2 years in production
1.8B TXes per day

zero-dependency
Delos as a platform

original goal: can we build a zero-dependency, fault-tolerant system with a rich API... in months?

DelosTable  DelosZK  extensible APIs

DelosRuntime

VirtualLog

2 years in production  1.8B TXes per day

ZKLoglet  NativeLoglet  zero-dependency

...in months
Delos as a platform

original goal: can we build a zero-dependency, fault-tolerant system with a rich API... in months?

DelosTable

DelosRuntime

VirtualLog

ZKLoglet

NativeLoglet

LDLoglet

DelosZK

extensible APIs

extensible Loglets

2 years in production

1.8B TXes per day
original goal: can we build a zero-dependency, fault-tolerant system with a rich API... \textit{in months}?

Delos as a platform

rich API
- \texttt{DelosTable}
- \texttt{DelosZK}

fault-tolerant
- \texttt{DelosRuntime}
- \texttt{VirtualLog}

common platform
- \texttt{ZKLoglet}
- \texttt{NativeLoglet}
- \texttt{LDLoglet}

deliver extensible Loglets

\begin{align*}
\text{2 years in production} & \quad \text{1.8B TXes per day} \\
\end{align*}
Delos is a new storage system at the bottom of the Facebook stack. Virtualizing consensus allowed us to develop and deploy new protocols. Production benefits immediately from new research... new research can reach production quickly.
thank you!

contact: mbalakrishnan at fb.com