FINE-GRAINED DISTRIBUTED CONSISTENCY GUARANTEES WITH EFFECT ORCHESTRATION
STRUCTURE OF THE TALK

▸ Background
▸ Problem Definition
▸ Our Solution
▸ Evaluation Results
Low latency and high availability are vital for modern web-scale applications.
Low latency and high availability are vital for modern web-scale applications.

1 second delay in page response can result in a 7% reduction in sales.
- Low latency and high availability are vital for modern web-scale applications

- 1 second delay in page response can result in a 7% reduction in sales

- Usually rely on the underlying data stores for consistency, integrity, durability and availability of the data.
Scalability and availability are achieved by geographic replication of data.
Scalability and availability are achieved by geographic replication of data.

Unfortunately, in real world, inter-continent connections are very slow and links are broken frequently.
Scalability and availability are achieved by geographic replication of data.

Unfortunately, in real world, inter-continent connections are very slow and links are broken frequently.
Scalability and availability are achieved by geographic replication of data.

Unfortunately, in real world, inter-continent connections are very slow and links are broken frequently.
Scalability and availability are achieved by geographic replication of data.

Unfortunately, in real world, inter-continent connections are very slow and links are broken frequently.
- Scalability and availability are achieved by geographic replication of data
- Unfortunately, in real world, inter-continent connections are very slow and links are broken frequently
- Modern data stores trade off strong forms of data consistency in favor of low response time and high availability (CAP theorem)
Eventual Consistency (EC) has become a prominent consistency guarantee offered in many data stores by different vendors.
- **Eventual Consistency (EC)** has become a prominent consistency guarantee offered in many data stores by different vendors.

- There has been a constant growth in popularity of EC data stores (and other NoSQL databases).
Eventual Consistency (EC) has become a prominent consistency guarantee offered in many data stores by different vendors.

There has been a constant growth in popularity of EC data stores (and other NoSQL databases).

Building correct applications on top of EC stores, is “unnatural” and error-prone and sometimes impossible.
Eventual Consistency (EC) has become a prominent consistency guarantee offered in many data stores by different vendors.

There has been a constant growth in popularity of EC data stores (and other NoSQL databases).

Building correct applications on top of EC stores, is “unnatural” and error-prone and sometimes impossible.

Additional weaker forms of consistency guarantees are introduced, e.g. Causal Consistency (CC) and session guarantees (D. Terry et.al.)
PROBLEM: LACK OF A GENERIC ENFORCEMENT MECHANISM
PROBLEM: LACK OF A GENERIC ENFORCEMENT MECHANISM

- There are many useful weak consistency guarantees
PROBLEM: LACK OF A GENERIC ENFORCEMENT MECHANISM

- There are many useful weak consistency guarantees

- Combinations of these guarantees are also required sometimes
There are many useful weak consistency guarantees

Combinations of these guarantees are also required sometimes

OTS data stores only offer a handful of them (at most)
PROBLEM: LACK OF A GENERIC ENFORCEMENT MECHANISM

- There are many useful weak consistency guarantees
- Combinations of these guarantees are also required sometimes
- OTS data stores only offer a handful of them (at most)
- Designing new data stores offering every single combination is not feasible (e.g. Cassandra has 360'000 LOC)
PROBLEM: LACK OF A GENERIC ENFORCEMENT MECHANISM

- There are many useful weak consistency guarantees
- Combinations of these guarantees are also required sometimes
- OTS data stores only offer a handful of them (at most)
- Designing new data stores offering every single combination is not feasible (e.g. Cassandra has 360'000 LOC)
- System developers end up using guarantees stronger than what they need
PREVIOUS ATTEMPTS
A generic tool offering fine-grained consistency guarantees does not exist
- A generic tool offering fine-grained consistency guarantees does not exist

- We introduce SYNCOPE to address this problem
A generic tool offering fine-grained consistency guarantees does not exist.

We introduce SYNCOPE to address this problem.

SYNCOPE extends off the shelf EC data stores into a tunable multi-consistent data store, offering all known fine-grained consistency guarantees (and beyond!)
A generic tool offering fine-grained consistency guarantees does not exist

We introduce SYNCOPE to address this problem

SYNCOPE extends off the shelf EC data stores into a tunable multi-consistent data store, offering all known fine-grained consistency guarantees (and beyond!)

Previous attempts have offered weak consistency extensions with a much coarser granularity: e.g. Bolt-on Causal Consistency
A generic tool offering fine-grained consistency guarantees does not exist

We introduce SYNCOPE to address this problem

SYNCOPE extends off-the-shelf EC data stores into a tunable multi-consistent data store, offering all known fine-grained consistency guarantees (and beyond!)

Previous attempts have offered weak consistency extensions with a much coarser granularity: e.g., Bolt-on Causal Consistency

Multi-consistent shims have also been used before (e.g., Quelea) but only focusing on the relationship between the application-level requirements to a pre-defined set of weak consistency guarantees
1. How prevalent these fine-grained consistency requirements are?
1. How prevalent these fine-grained consistency requirements are?

2. Is it worth it to differentiate between all these fine-grained consistency levels?
HOW FREQUENTLY ARE THEY USED? (1)
HOW FREQUENTLY ARE THEY USED? (1)

Analyzer 7 benchmark applications
Analyzed 7 benchmark applications

All applications had operations that required some form of weak consistency guarantee
HOW FREQUENTLY ARE THEY USED? (1)

- Analyzed 7 benchmark applications
- All applications had operations that required some form of weak consistency guarantee
- Due to the lack of available underlying store implementations, all operations were originally mapped to CC
HOW FREQUENT ARE THEY USED? (2)
All examined benchmark applications can be shown to require weaker forms of guarantees than CC.
All examined benchmark applications can be shown to require weaker forms of guarantees than CC.

The requirements are expressible as a logical combination of known weak consistency levels.
ARE THEY REALLY THAT DIFFERENT?

- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC
Over faulty networks the difference between fine-grained consistency requirements can be substantial.

Example: Read-My-Writes (RMW) and CC.
Over faulty networks the difference between fine-grained consistency requirements can be substantial.

Example: Read-My-Writes (RMW) and CC.
Over faulty networks the difference between fine-grained consistency requirements can be substantial.

Example: Read-My-Writes (RMW) and CC.
Over faulty networks the difference between fine-grained consistency requirements can be substantial.

Example: Read-My-Writes (RMW) and CC
Over faulty networks the difference between fine-grained consistency requirements can be substantial.

Example: Read-My-Writes (RMW) and CC.
Over faulty networks the difference between fine-grained consistency requirements can be substantial.

Example: Read-My-Writes (RMW) and CC.
Over faulty networks the difference between fine-grained consistency requirements can be substantial.

- Example: Read-My-Writes (RMW) and CC

- CC incurs additional constraints

- Not only my previous writes, but everything visible to them should also be visible to me!
APPLICATION AND DATABASE CONSISTENCY MISMATCH

- Over faulty networks the difference between fine-grained consistency requirements can be substantial.
- Example: Read-My-Writes (RMW) and CC.
- CC incurs additional constraints.
- Not only my previous writes, but everything visible to them should also be visible to me!
Over faulty networks the difference between fine-grained consistency requirements can be substantial.

Example: Read-My-Writes (RMW) and CC.

CC incurs additional constraints.

Not only my previous writes, but everything visible to them should also be visible to me!
APPLICATION AND DATABASE CONSISTENCY MISMATCH

- Over faulty networks the difference between fine-grained consistency requirements can be substantial.
- Example: Read-My-Writes (RMW) and CC.
- CC incurs additional constraints.
- Not only my previous writes, but everything visible to them should also be visible to me!
Over faulty networks the difference between fine-grained consistency requirements can be substantial.

Example: Read-My-Writes (RMW) and CC.

CC incurs additional constraints.

Not only my previous writes, but everything visible to them should also be visible to me!
Over faulty networks the difference between fine-grained consistency requirements can be substantial.

Example: Read-My-Writes (RMW) and CC.

CC incurs additional constraints.

Not only my previous writes, but everything visible to them should also be visible to me!
Over faulty networks the difference between fine-grained consistency requirements can be substantial.

Example: Read-My-Writes (RMW) and CC.

CC incurs additional constraints.

Not only my previous writes, but everything visible to them should also be visible to me!
Over faulty networks the difference between fine-grained consistency requirements can be substantial.

Example: Read-My-Writes (RMW) and CC

CC incurs additional constraints.

Not only my previous writes, but everything visible to them should also be visible to me!
Over faulty networks the difference between fine-grained consistency requirements can be substantial.

Example: Read-My-Writes (RMW) and CC.

CC incurs additional constraints.

Not only my previous writes, but everything visible to them should also be visible to me!
Over faulty networks the difference between fine-grained consistency requirements can be substantial.

Example: Read-My-Writes (RMW) and CC

CC incurs additional constraints.

Not only my previous writes, but everything visible to them should also be visible to me!
Over faulty networks the difference between fine-grained consistency requirements can be substantial.

Example: Read-My-Writes (RMW) and CC.

CC incurs additional constraints.

Not only my previous writes, but everything visible to them should also be visible to me!
Over faulty networks the difference between fine-grained consistency requirements can be substantial.

Example: Read-My-Writes (RMW) and CC

CC incurs additional constraints

Not only my previous writes, but everything visible to them should also be visible to me!
APPLICATION AND DATABASE CONSISTENCY MISMATCH

- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC
- CC incurs additional constraints
- Not only my previous writes, but everything visible to them should also be visible to me!
Over faulty networks the difference between fine-grained consistency requirements can be substantial.

Example: Read-My-Writes (RMW) and CC.

CC incurs additional constraints.

Not only my previous writes, but everything visible to them should also be visible to me!
APPLICATION AND DATABASE CONSISTENCY MISMATCH

- Over faulty networks the difference between fine-grained consistency requirements can be substantial.
- Example: Read-My-Writes (RMW) and CC.
- CC incurs additional constraints.
- Not only my previous writes, but everything visible to them should also be visible to me!
EVENTUALLY CONSISTENT DATA STORES
EVENTUALLY CONSISTENT DATA STORES

INTERNET

ADDITIONAL CONSISTENCY GUARANTEES

Startup #1
- BASIC EC
- STRONG

Project #2
- BASIC EC
- CAUSAL
- STRONG

Company #3
- MONOTONIC WRITES
- MONOTONIC READS
- CAUSAL CONSISTENCY

Paper #4
- BASIC EC
- MONOTONIC WRITES
- MONOTONIC READS
- READ-MY-WRITES
- WRITES FOLLOW READS
EVENTUALLY CONSISTENT DATA STORES

INTERNET

BASIC EC

MONOTONIC WRITES

CAUSAL CONSISTENCY

ADDITIONAL CONSISTENCY GUARANTEES

Startup #1

BASIC EC

CAUSAL

MONOTONIC READS

STRONG

Project #2

BASIC EC

CAUSAL

MONOTONIC READS

STRONG

Company #3

MONOTONIC WRITES

CAUSAL CONSISTENCY

Paper #4

BASIC EC

MONOTONIC WRITES

READ-MY-WRITES

WRITE-FOLLOW-READS

SYSTEM DEVELOPER

Startup #1

Project #2

Company #3

Paper #4
EVENTUALLY CONSISTENT DATA STORES

INTERNET

AUTOMATIC ENFORCEMENT OF FINE-GRAINED CONSISTENCY REQUIREMENTS

SYNCOPE

• Light weight run-time system on top of Cassandra
• Multi-consistent shim layer which is specifically maintained for each operation
• Automatic enforcement of consistency requirements per operation
• Provably optimal and correct

SYSTEM DEVELOPER

Per operation consistency declaration

Generic programming model
IT’S ALL ABOUT ENFORCING VISIBILITY
There are two fundamental relations between such effects: \textit{vis} and \textit{so}
There are two fundamental relations between such effects: **vis** and **so**

**vis** relates an effect to all other effects present at the local replica at the time of its creation.
There are two fundamental relations between such effects: **vis** and **so**

- **vis** relates an effect to all other effects present at the local replica at the time of its creation
- **so** relates operations from the same client session
There are two fundamental relations between such effects: \textit{vis} and \textit{so}

\textit{vis} relates an effect to all other effects present at the local replica at the time of its creation

\textit{so} relates operations from the same client session

\textit{vis} might be altered at the executing replica by changing the visible snapshot of the system to each operation
There are two fundamental relations between such effects: **vis** and **so**

- **vis** relates an effect to all other effects present at the local replica at the time of its creation
- **so** relates operations from the same client session
- **vis** might be altered at the executing replica by changing the visible snapshot of the system to each operation
- **so** is pre-determined by the clients
There are two fundamental relations between such effects: $\text{vis}$ and $\text{so}$

- $\text{vis}$ relates an effect to all other effects present at the local replica at the time of its creation
- $\text{so}$ relates operations from the same client session
- $\text{vis}$ might be altered at the executing replica by changing the visible snapshot of the system to each operation
- $\text{so}$ is pre-determined by the clients

The difference between all weak consistency guarantees is on how they enforce visibility relations between effects
r ∈ rel.seed := vis | so | r ∪ r
\[ (a \xrightarrow{r_1; r_2; \ldots; r_k} b) \text{ is interpreted as} \]
\[ \exists c. (a \xrightarrow{r_1; r_2; \ldots; r_{k-1}} c \land c \xrightarrow{r_k} b) \]

\[
\begin{align*}
r \in \text{rel.seed} & := vis | so | r \cup r \\
R \in \text{relation} & := r | R; r | \text{null}
\end{align*}
\]
(\(\frac{r_1; r_2; \cdots; r_k}{r} \to b\)) is interpreted as

\[\exists c. (a \frac{r_1; r_2; \cdots; r_{k-1}}{r} \to c \land c \frac{r_k}{b})\]

\(r \in \text{rel.seed} := \, vis | so | r \cup r\)

\(R \in \text{relation} := r | R; r | \text{null}\)

\(\pi \in \text{prop} := \forall a. a \xrightarrow{R} \hat{\eta} \Rightarrow a \xrightarrow{vis} \hat{\eta}\)
SPECIFICATION LANGUAGE: DEFINITION

\[(a \overrightarrow{r_1; r_2; \ldots; r_k} \rightarrow b) \text{ is interpreted as}\]
\[\exists c. (a \overrightarrow{r_1; r_2; \ldots; r_{k-1}} c \land c \overrightarrow{r_k} \rightarrow b)\]

\[
\begin{align*}
\mathbf{r} & \in \text{rel.seed} := \mathit{vis} | \mathit{so} | \mathbf{r} \cup \mathbf{r} \\
\mathbf{R} & \in \text{relation} := \mathbf{r} | \mathbf{R}; \mathbf{r} | \mathit{null} \\
\pi & \in \text{prop} := \forall a. a \overrightarrow{R} \hat{\eta} \Rightarrow a \overrightarrow{\mathit{vis}} \hat{\eta} \\
\psi & \in \text{spec} := \pi | \pi \land \pi
\end{align*}
\]
GENERALITY: SESSION GUARANTEES

READ MY WRITES

\( \forall a.a \xrightarrow{so} \hat{\eta} \Rightarrow a \xrightarrow{vis} \hat{\eta} \)

MONOTONIC WRITES

\( \forall a.a \xrightarrow{so;vis} \hat{\eta} \Rightarrow a \xrightarrow{vis} \hat{\eta} \)

MONOTONIC READS

\( \forall a.a \xrightarrow{vis;so} \hat{\eta} \Rightarrow a \xrightarrow{vis} \hat{\eta} \)
TWO CLASSES
Guarantees can be classified into two groups: Lower Bound (LB) and Upper Bound (UB) contracts.
Guarantees can be classified into two groups: Lower Bound (LB) and Upper Bound (UB) contracts.

Each class naturally maps to an enforcement mechanism.
THE CORE OF LB
LB guarantees specify a *minimal set* of effects that must be visible to an effect at the time of its creation.
LB guarantees specify a minimal set of effects that must be visible to an effect at the time of its creation.

The dependence relation of LB guarantees ends with an so edge: \( R'; so \).
LB guarantees specify a *minimal set* of effects that must be visible to an effect at the time of its creation.

The dependence relation of LB guarantees ends with an *so* edge: $R';so$

The dependency set of the current operation is *pre-determined* at the time of its execution.
HOW TO ENFORCE LB
HOW TO ENFORCE LB

- LB contracts can be enforced by *blocking* the client requests temporarily
HOW TO ENFORCE LB

- LB contracts can be enforced by blocking the client requests temporarily.
- Assuming EC, the dependency set will eventually become available locally.
> LB contracts can be enforced by **blocking** the client requests temporarily

> Assuming EC, the dependency set will eventually become available locally

> Example: RMW
HOW TO ENFORCE LB

- LB contracts can be enforced by **blocking** the client requests temporarily.
- Assuming EC, the dependency set will eventually become available locally.
- Example: RMW
**HOW TO ENFORCE LB**

- LB contracts can be enforced by **blocking** the client requests temporarily.
- Assuming EC, the dependency set will eventually become available locally.
- Example: RMW

```plaintext
wait()
```

Incurs additional latency!
- LB contracts can be enforced by **blocking** the client requests temporarily.
- Assuming EC, the dependency set will eventually become available locally.
- Example: RMW

```java
wait()
```
How to Enforce LB

- LB contracts can be enforced by blocking the client requests temporarily.
- Assuming EC, the dependency set will eventually become available locally.
- Example: RMW
WHAT IS THE CORE OF UB
UB guarantees specify a constraint on the set of effects that *might* be visible to an effect.
UB guarantees specify a constraint on the set of effects that *might* be visible to an effect.

The dependence relation of UB guarantees ends with \( \text{vis} \): \( R';\text{vis} \).
UB guarantees specify a constraint on the set of effects that *might be visible to an effect*

The dependence relation of UB guarantees ends with **vis**: $R';vis$

The dependency set of the current operation can be determined at the running replica by changing the visible snapshot to the current operation.
UB contracts can be satisfied by showing a **filtered** snapshot of the locally available updates to each operation.
UB contracts can be satisfied by showing a *filtered* snapshot of the locally available updates to each operation.

Example: MW
UB contracts can be satisfied by showing a **filtered** snapshot of the locally available updates to each operation.

Example: MW
UB contracts can be satisfied by showing a *filtered* snapshot of the locally available updates to each operation.

Example: MW
UB contracts can be satisfied by showing a **filtered** snapshot of the locally available updates to each operation.

Example: MW
UB contracts can be satisfied by showing a **filtered** snapshot of the locally available updates to each operation.

Example: MW
OVERVIEW OF THE IMPLEMENTATION

- Implemented in Haskell
- Implemented in Haskell
- Backed up by Cassandra
 Implemented in Haskell

 Backed up by Cassandra

 Each operation is executed on it’s own snapshot of the local replica
OVERVIEW OF THE IMPLEMENTATION

- Implemented in Haskell
- Backed up by Cassandra
- Each operation is executed on its own snapshot of the local replica
- Can be realized by keeping multiple local copies of data or by a simple tagging mechanism

![Diagram showing the implementation process with nodes and connections labeled for dependency check, multi-consistent shim layer, and filtration.]
Overview of the Implementation

- Implemented in Haskell
- Backed up by Cassandra
- Each operation is executed on its own snapshot of the local replica
- Can be realized by keeping multiple local copies of data or by a simple tagging mechanism

![Diagram of environment and dependency checks](image1.png)

- Cassandra Node
  - Blocking
  - Multi-Consistent Shim Layer
  - Filtration
- Eventual delivery of all updates
- Durability of writes
- Fault Tolerant
- Periodic fetches
EVALUATION: SETTING
A 3-node cluster backed by Cassandra deployed on Amazon EC2
A 3-node cluster backed by Cassandra deployed on Amazon EC2

Artificial network fault injection: messages are randomly delayed for 1s
EVALUATION: SETTING

- A 3-node cluster backed by Cassandra deployed on Amazon EC2
- Artificial network fault injection: messages are randomly delayed for 1s
- User-Perceived latency (for LB guarantees) and visible snapshot staleness (for UB guarantees) are measured
A 3-node cluster backed by Cassandra deployed on Amazon EC2

Artificial network fault injection: messages are randomly delayed for 1s

User-Perceived latency (for LB guarantees) and visible snapshot staleness (for UB guarantees) are measured

50 Concurrent clients performing sessions of reads and write operations to random replicas on a shared counter object
EVALUATION: RESULTS

LATENCY

STALENESS
We offer a generalized platform for specifying and enforcing fine-grained application-level consistency requirement.

Fine-grained consistency enforcement can result in considerable performance and availability gain in faulty networks.

The next goal is to build more fault resilient distributed systems, where SYNCOPE can be deployed as a secondary defense mechanism to fight unreliable networks without sacrificing correctness.
QUESTIONS?