Kronos

The Design and Implementation of an Event Ordering Service

Robert Escriva†
Ayush Dubey†
Bernard Wong‡
Emin Gün Sirer†

†Department of Computer Science
  Cornell University
‡School of Computer Science
  University of Waterloo

EuroSys ’14
April 14, 2014
Distributed Systems

Distributed systems are difficult to build because changes are happening across many computers.
1977: Han shot first
1977: Han shot first
1997: Greedo shot first

We need some way to know who shot first!
Order in Distributed Systems

- Lamport Timestamps:
  Institute a total order across requests
- Vector Clocks:
  Requires agreement on membership and format
- Consensus Protocols
  Serialized execution which limits concurrency
Kronos: A Time Oracle for Distributed Systems

A time oracle maintains the global timeline for the system:

- Tell the oracle the order in which things happen
- Ask the oracle to recall this information later
Managing Dependencies in a Social Network

Alice

Bob

FS

KVS
Managing Dependencies in a Social Network

Alice

Bob

Increase Privacy

FS

KVS
Managing Dependencies in a Social Network

Alice

Increase Privacy

Bob

View Alice’s Album

FS

KVS
Managing Dependencies in a Social Network

Alice

Increase Privacy

Bob

View Alice’s Album

Get ACL

FS

KVS

Get Photos

Upload

Embarrassing Photo

Store Photo

Which photos come before this ACL?
Managing Dependencies in a Social Network

Alice

Bob

Increase Privacy

Update ACL

View Alice’s Album

Alice

Bob

FS

KVS
Managing Dependencies in a Social Network

Alice

Upload
Embarrassing Photo

Bob

View Alice’s Album

Kronos

KVS

FS

Get ACL
Update ACL

Get Photos
Increase Privacy

Upload

Store Photo

Photo comes after old ACL

Which photos come before this ACL?

KVS

Robert Escriva, Ayush Dubey, Bernard Wong, Emin Gün Sirer
Managing Dependencies in a Social Network

Alice

Bob

Get Photos

View Alice’s Album

Get ACL

Update ACL

Upload Embarrassing Photo

Store Photo

Which photos come before this ACL?
Managing Dependencies in a Social Network

Alice

Bob

Kronos

FS

KVS
Managing Dependencies in a Social Network

Alice

Increase Privacy

Bob

FS

Kronos

KVS
Managing Dependencies in a Social Network

Alice

Increase Privacy

Bob

View Alice’s Album
Managing Dependencies in a Social Network

Alice

Increase Privacy

Bob

View Alice’s Album

Get ACL

Increase Privacy

View Alice’s Album

Get ACL

Robert Escriva, Ayush Dubey, Bernard Wong, Emin Gün Sirer
Managing Dependencies in a Social Network

Alice

Increase Privacy

Bob

View Alice’s Album

Update ACL

Kronos

 FS

KVS

Get Photos

Get ACL

Upload

Embarrassing Photo

Photo comes after old ACL

Which photos come before this ACL?
Managing Dependencies in a Social Network

Alice

Bob

Upload Embarrassing Photo

View Alice’s Album

Photo comes after old ACL
Managing Dependencies in a Social Network

Alice

Upload Embarrassing Photo

Bob

View Alice’s Album

Store Photo

FS

Kronos

KVS

Photo comes after old ACL

Robert Escriva Ayush Dubey, Bernard Wong, Emin Gün Sirer
Managing Dependencies in a Social Network

Alice

Bob

View Alice’s Album

Get Photos

Kronos

KVS

FS

Get ACL

Update ACL

Upload

Embarrassing Photo

Photo comes after old ACL

Store Photo

Get Photos

Which photos come before this ACL?
Managing Dependencies in a Social Network

Which photos come before this ACL?

Alice

Bob

View Alice’s Album

Kronos

KVS

FS

Upload

Embarrassing Photo

Get Photos

Get ACL

Update ACL

Increase Privacy

View Alice’s Album

Photo comes after old ACL

Store Photo
What is an Event?

An event is an application-determined set of state changes that take place atomically, associated with a unique identifier, e.g.:

- Reads or writes in a distributed filesystem
- Transactions in a key value store
- Queries on a graph store
The event dependency graph captures happens-before relationships and enables queries over the timeline.
The Coherency Invariant

Ensures the timeline makes logical sense by preventing cycles.
The Coherency Invariant

Ensures the timeline makes logical sense by preventing cycles.
The Monotonicity Invariant

Relationships, once established by Kronos, are incontrovertible.
The Monotonicity Invariant

Relationships, once established by Kronos, are incontrovertible.
The Monotonicity Invariant

Relationships, once established by Kronos, are incontrovertible.
The Monotonicity Invariant

Relationships, once established by Kronos, are incontrovertible.
The Monotonicity Invariant

Relationships, once established by Kronos, are incontrovertible.
API: Event Creation

- `create_event()`
  \[ \Rightarrow A \]

`create_event()`
Create a new event and return a unique identifier \( e \).
API: Event Creation

- `create_event()`
  - $\Rightarrow A$

- `create_event()`
  - $\Rightarrow B$

`create_event()`
Create a new event and return a unique identifier $e$. 
API: Event Creation

```plaintext
create_event() ⇒ A
create_event() ⇒ B
create_event() ⇒ C
```

`create_event()` Create a new event and return a unique identifier $e$. 
API: Dependency Creation

- \[\text{assign\_order}(A, B) \Rightarrow True\]

\[
\begin{array}{c}
\begin{array}{c}
\text{A} \\
\downarrow \\
\text{C} \\
\text{B}
\end{array}
\end{array}
\]

\[\text{assign\_order}(e_i, e_j)\]
Create the relationship \(e_i \leadsto e_j\) if possible.
API: Dependency Creation

- $\text{assign\_order}(A, B) \Rightarrow True$
- $\text{assign\_order}(B, C) \Rightarrow True$

$\text{assign\_order}(e_i, e_j)$
Create the relationship $e_i \rightsquigarrow e_j$ if possible.
API: Dependency Creation

- $\text{assign\_order}(A, B) \Rightarrow True$
- $\text{assign\_order}(B, C) \Rightarrow True$
- $\text{assign\_order}(C, A) \Rightarrow False$

assign_order($e_i, e_j$)
Create the relationship $e_i \rightsquigarrow e_j$ if possible.
Batch operations execute atomically. Either all operations complete, or none have any effect.
API: Atomic Batch Operations

- `create_event(3)`
  \[ \Rightarrow [A, B, C] \]

Batch operations execute atomically. Either all operations complete, or none have any effect.
API: Atomic Batch Operations

- `create_event(3)`
  \[ \Rightarrow [A, B, C] \]

- `assign_order([(A, B), (B, C)])`
  \[ \Rightarrow True \]

Batch operations execute atomically. Either all operations complete, or none have any effect.
Batch operations execute atomically. Either all operations complete, or none have any effect.
API: Atomic Batch Operations

- **create_event(3)**
  \[ \Rightarrow [A, B, C] \]

- **assign_order([[A, B], (B, C)])**
  \[ \Rightarrow True \]

- **assign_order([[A, B], (C, A), prefer])**
  \[ \Rightarrow False \]

By default, all dependencies in a batch **must** be consistent with the timeline, or the operation and batch will fail.
API: Atomic Batch Operations

- `create_event(3)`
  \[ \Rightarrow [A, B, C] \]
- `assign_order([[A, B], (B, C)])`
  \[ \Rightarrow True \]
- `assign_order([[A, B], (C, A)])`
  \[ \Rightarrow False \]
- `assign_order([[A, B], (C, A, prefer)])`
  \[ \Rightarrow True \]

Applications may specify individual dependencies with a preferred ordering that Kronos may reverse if necessary.
API: Atomic Batch Operations

- `create_event(3)`
  \[ [A, B, C] \]

- `assign_order([(A, B), (B, C)])`
  \[ True \]

- `assign_order([(A, B), (C, A)])`
  \[ False \]

- `assign_order([(A, B), (C, A), prefer])`
  \[ True \]

A dependency with a **preferred** ordering will never cause the batch to fail, as Kronos may always align the dependency with the graph.
API: Query Operations

Queries discover happens-before relationships within the graph by performing a standard breadth-first search (BFS).

\[
\text{query\_order} (A, B) ;
\]

\[
A \leadsto B
\]
API: Query Operations

If there is no happens-before relationship between two events, Kronos will return that the events are concurrent.

- `query_order(A, B);`
  - $A \rightsquigarrow B$

- `query_order(A, D);`
  - $A \sim D$ // concurrent
API: Query Operations

Queries may be submitted in bulk, retrieving multiple results simultaneously.

- `query_order(A, B);`  
  \( A \leadsto B \)

- `query_order(A, D);`  
  \( A \not\leadsto D \) // concurrent

- `query_order([ (A, B), (C, A) ]);`  
  \[ A \leadsto B, \ A \leadsto C \]
Twitter Clone

Alice posts a message to the social network, which stores it in the key-value store.
Bob’s reply gets stored in the key-value store, and its dependency on Alice’s message gets stored in Kronos.
def post_message(user, message):
    e = kronos.create_event()
    for friend in friends_of(user):
        enqueue_in_timeline(timeline=friend,
                            source=user,
                            message=message,
                            event=e)
```python
def reply_to_message(user, message, in_reply_to):
    e = kronos.create_event()
    kronos.assign_order(
        [(in_reply_to, '->', e, 'must')]
    )
    for friend in friends_of(user):
        enqueue_in_timeline(timeline=friend,
                            source=user,
                            message=message,
                            event=e)
```
def render_timeline(user):
    messages = get_enqueued_for(timeline=user)
    pairs = all_pairs([m.id for m in messages])
    orderings = kronos.query_order(pairs)
    return topological_sort(messages, orderings)
Transactional Key-Value Store

- ACID transactions: update multiple objects atomically
- Transactions that read/write the same keys ordered by Kronos
- Kronos ensures a serializable order across all transactions
Transactional Key-Value Store

$T_1 \rightarrow \text{read } A \rightarrow \text{write } B \rightarrow T_2$

Robert Escriva, Ayush Dubey, Bernard Wong, Emin Gün Sirer
Transactional Key-Value Store

$T_1 \rightarrow$ read A

write B

$T_2 \rightarrow$ read B

write C
Transactional Key-Value Store

$T_1 \rightarrow \text{read } A \rightarrow \text{write } B \rightarrow T_1 \leadsto T_2 \rightarrow \text{write } C \rightarrow T_2 \rightarrow \text{read } B \rightarrow \text{write } C$

Robert Escriva Ayush Dubey, Bernard Wong, Emin Gün Sirer
KronoGraph

KronoGraph is an online graph store.

- The graph may change as queries execute
- The correctness of queries relies upon seeing a correct state of the graph
- Because the graph may be quite big, queries or updates could span multiple hosts
KronoGraph Motivation

In a networked environment, it sometimes is necessary to change the network configuration.
KronoGraph Motivation

In a networked environment, it sometimes is necessary to change the network configuration.
KronoGraph Motivation

The control platform for this environment could store the topology in a graph store for easy route discover.
KronoGraph Motivation

Such topology changes could be atomic, otherwise it would be possible to discover routes that never existed.
Such topology changes could be atomic, otherwise it would be possible to discover routes that never existed.
KronoGraph Motivation

If the graph changes while the query traverses across the shard boundaries, it could arrive after the change to the graph is made.
Clients issuing requests to multiple servers can see these requests cross and re-order in the network.
KronoGraph Example

Clients issuing requests to multiple servers can see these requests cross and re-order in the network.
Clients issuing requests to multiple servers can see these requests cross and re-order in the network.
Servers may use Kronos to disambiguate the true order of events, preferring the natural arrival order where possible.
KronoGraph Example

Kronos answers the prefer request with a reversal and the true order between the events.
KronoGraph Example

The shard server can re-order the execution of operations when Kronos indicates a reversal.
Garbage Collection

Garbage collection keeps the size of the event dependency graph proportional to the working set of events.
Kronos associates with each event a reference count. Clients manually acquire and release references.
Acquiring a reference increases the reference count.

```c
acquire_ref('A');
```
Garbage Collection

When a reference count goes to zero, the associated event is ready for garbage collection.

- acquire_ref('A');
- release_ref('B');
Events that are ready for garbage collection stay in the graph until they have no more incoming edges.
Garbage Collection

Events that are ready for garbage collection stay in the graph until they have no more incoming edges.

- acquire_ref('A');
- release_ref('B');
- release_ref('C');
- release_ref('D');
Garbage Collection

Events that are ready for garbage collection stay in the graph until they have no more incoming edges.

- acquire_ref('A');
- release_ref('B');
- release_ref('C');
- release_ref('D');
- release_ref('A');
Garbage Collection

- acquire_ref('A');
- release_ref('B');
- release_ref('C');
- release_ref('D');
- release_ref('A');
- release_ref('A');
Garbage Collection

Once a reference count goes to zero, and has no incoming edges, the event will be garbage collected

- acquire_ref('A');
- release_ref('B');
- release_ref('C');
- release_ref('D');
- release_ref('A');
- release_ref('A');
Garbage Collection

Once a reference count goes to zero, and has no incoming edges, the event will be garbage collected

```c
▶ acquire_ref('A');
▶ release_ref('B');
▶ release_ref('C');
▶ release_ref('D');
▶ release_ref('A');
▶ release_ref('A');
```
Garbage Collection

Once a reference count goes to zero, and has no incoming edges, the event will be garbage collected

Robert Escriva  Ayush Dubey, Bernard Wong, Emin Gün Sirer
Garbage Collection

Once a reference count goes to zero, and has no incoming edges, the event will be garbage collected.

- acquire_ref('A');
- release_ref('B');
- release_ref('C');
- release_ref('D');
- release_ref('A');
- release_ref('A');
Fault Tolerance

- Replicated using Chain Replication
- Tolerate $f$ failures with $f + 1$ replicas
- Could easily use any state machine replication technique
Typical Optimizations

- Out-of-date replicas may be used for queries that return a happens-before relationship
- No cache invalidation necessary for happens-before relationships
  - Caching becomes near free
  - Cache within Kronos
  - Cache within clients
- Exploit batching for `assign_order` and `query_order` calls
- Optimistically order events to improve batching by looking ahead at what events may need order
Experimental Setup

- What is the performance of our Kronos applications?
- How scalable is Kronos?
- How large a graph can be stored?
- What are the costs associated with garbage collection?
Experimental Setup

- 14 Machines for applications
- Intel Xeon 2.5 GHz E5420 × 2
- 16 GB RAM
- 500 GB SATA HDD
- Debian 7.0
- Linux 3.2
KronoGraph outperforms Titan, an online graph store that employs lock-based techniques.
KronoKV Evaluation

KronoKV performs better than locking approaches and is on par with popular industry solutions.
Microbenchmark: Scalability

Scalability on a sparse random graph
Memory usage remains proportional to the number of vertices in the graph
Microbenchmark: Garbage Collection

Garbage collection time is proportional to the number of events collected.
Conclusion

- Kronos is a time oracle for distributed systems
- A time oracle allows high performance systems that uphold strong guarantees