ulm university universität ulm

Benjamin Erb, Dominik Meißner,
Jakob Pietron, Frank Kargl
Ulm University, Germany

Barcelona, Spain
June 21st, 2017

Chronograph — A Distributed Processing Platform for Online and Batch Computations on Event-sourced Graphs

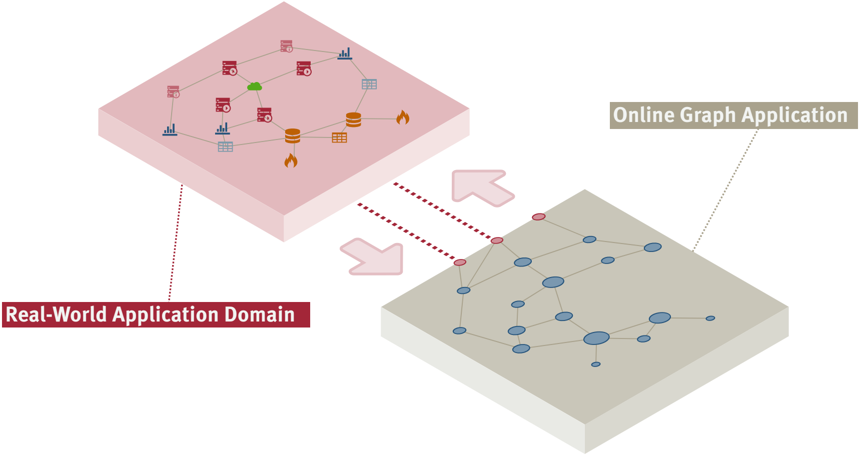
11th ACM International Conference on
Distributed and Event-Based Systems

"Many applications today are data-intensive, as opposed to compute-intensive. Raw CPU power is rarely a limiting factor for these applications — bigger problems are usually the amount of data, the complexity of data, and the speed at which it is changing."

— Martin Kleppmann

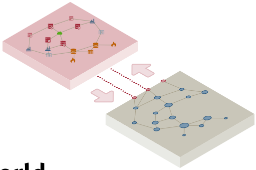
Source: M. Kleppmann (2017), *Designing Data-Intensive Applications*. O'Reilly.

Initial Challenge



Live Graph Applications

Characteristics



1. a graph **captures** a model of the **real world**
   - mapping of connectedness
2. a graph **enables computations**
   - style and granularity of processing operations
3. **changes occur** in the real world & in the system
   - progress via events
4. the **system interacts** with the real world (bidirectionally!)
   - system also influences the real world
5. the **evolution of state** is relevant for the applications
   - data lineage, time-series, and retrospective insights

Designated Computations & Operations

Requirements

- *How does the graph currently evolve?*
  - topology and state updates
  - execution of **online computations**
- *What has been the graph state at a specific point in time?*
  - graph state retrospection
  - basis for various **offline computations**
- *How has the status of a single vertex changed over time?*
  - retrospection of vertex history
  - basis for **time-series computations**
- *How do graph states of the evolving graph differ?*
  - comparison of different graph states
  - basis for **temporal batch computations**

Related Work

Existing Approaches and Partial Solutions

- graph computing systems
  - batch: e.g., Pregel, PowerGraph,
  - temporal: e.g., GoFFish, ImmortalGraph
- online processing systems
  - event processing engines: e.g., Storm
- general purpose data processing systems
  - w/ graph libraries: e.g., Spark+GraphX, Flink+Gelly
  - iterative dataflow: e.g., Naiad
- hybrid online & offline computation models
  - Lambda/Kappa architectures
- storage systems
  - event stores & time-series databases
  - graph databases

Live Graph (Actor-based System)

asynchronicity, distribution, liveness

“Freezed” Graph (Decoupled Snapshots)

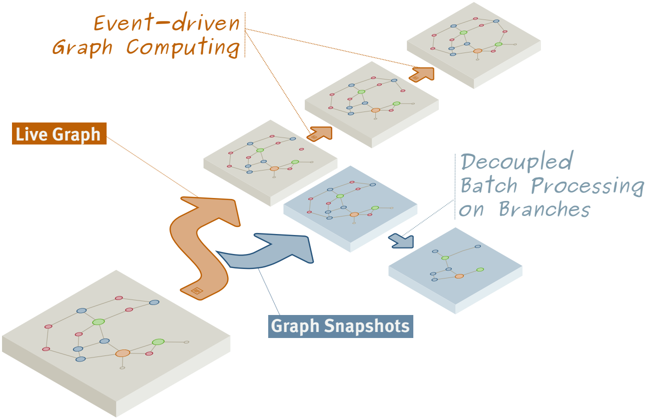
consistency, global state

Graph Evolution (Event Sourcing)

graph lineage

Decoupling Online & Offline Processing

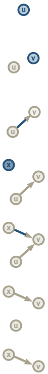
Event Sourcing, Snapshotting & Branching



Event Sourcing of Graphs: Topology Changes

Command Log		Event Log
addVertex(G,u)	⇢	VERTEX_ADDED(G,u)
addVertex(G,v)	⇢	VERTEX_ADDED(G,v)
addEdge(G,u,v)	⇢	EDGE_ADDED(G,u,v)
addVertex(G,x)	⇢	VERTEX_ADDED(G,x)
addEdge(G,x,v)	⇢	EDGE_ADDED(G,x,v)
removeEdge(G,u,v)	⇢	EDGE_REMOVED(G,u,v)
removeVertex(G,u)	⇢	VERTEX_REMOVED(G,u)

Graph State



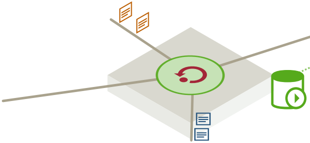
CHRONOGRAPH: *dedicated log for each vertex, tracking of vertex state*

Vertices: Event-sourced Actors

Asynchronous, message-driven, vertex-centric, decentralized

Incoming messages from adjacent vertices

Event-sourced vertex state



$$(S_{t+1}, f_{t+1}, [m_{out}]) = f_t(m_{in}, S_t)$$

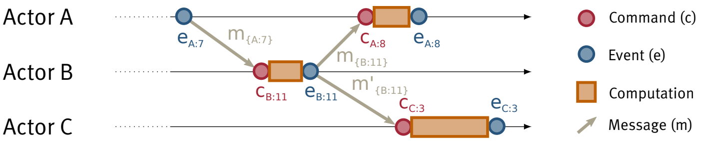
Outgoing messages to adjacent vertices

Behavior function

Delta Events: $E_t = \Delta(S_t, S_{t+1})$ (e.g., JSON Patch)

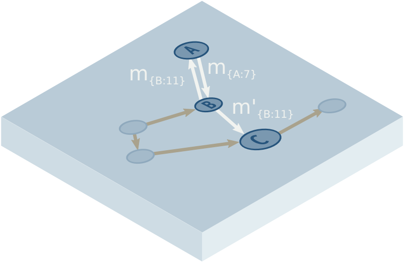
Command & Event Sourcing of Vertices

Versioning & Causality Tracking



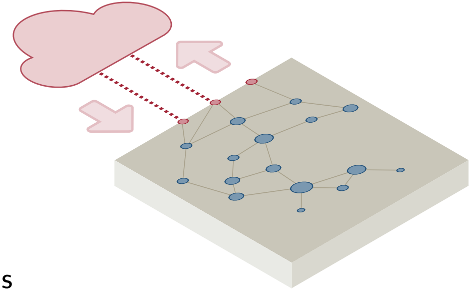
Snapshots

- causally consistent
- snapshot types
  - retrospective
  - asynchronous/latent
  - periodical



Actor/Vertex Types

Stateful & I/O Actors

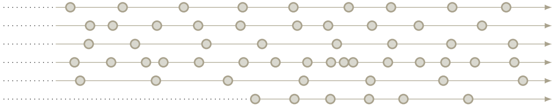


- stateful vertices/actors
  - typed by behavior
  - stateful
  - cannot cause side-effects
  - fully event-sourced
- I/O vertices/actors
  - allow real world interactions
  - either ingress or outgoing communication
  - event sourcing only for messages within CHRONOGRAPH
  - currently TCP-based socket to outside process

Programming & Processing Models

Models Supported by the CHRONOGRAPH Concept

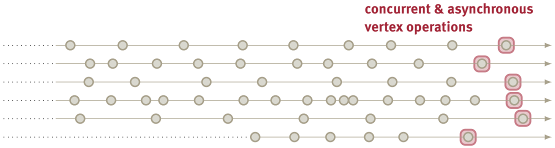
Model	Initial Data Set	Locality	Time Model
Main CHRONOGRAPH model	live graph	vertex-local	continuous
MapReduce (vertex-based)	graph snapshot	vertex-local	snapshotted
MapReduce (edge-based)	graph snapshot	vertex-local	snapshotted
Pregel	graph snapshot	vertex-local	snapshotted
Event Folding	log sequence	single-vertex	time series
Command Folding	log sequence	single-vertex	time series
MapReduce (temporal)	graph snapshot	vertex-local	snapshotted/iterative
Pause/Shift/Resume	graph snapshot	vertex-local	iterative



Programming & Processing Models

Models Supported by the CHRONOGRAPH Concept

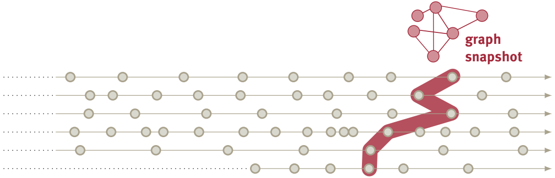
Model	Initial Data Set	Locality	Time Model
Main CHRONOGRAPH model	live graph	vertex-local	continuous
MapReduce (vertex-based)	graph snapshot	vertex-local	snapshotted
MapReduce (edge-based)	graph snapshot	vertex-local	snapshotted
Pregel	graph snapshot	vertex-local	snapshotted
Event Folding	log sequence	single-vertex	time series
Command Folding	log sequence	single-vertex	time series
MapReduce (temporal)	graph snapshot	vertex-local	snapshotted/iterative
Pause/Shift/Resume	graph snapshot	vertex-local	iterative



Programming & Processing Models

Models Supported by the CHRONOGRAPH Concept

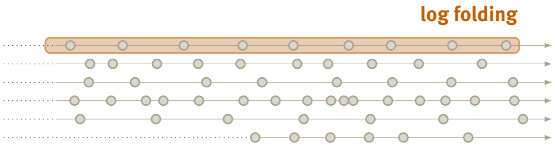
Model	Initial Data Set	Locality	Time Model
Main CHRONOGRAPH model	live graph	vertex-local	continuous
MapReduce (vertex-based)	graph snapshot	vertex-local	snapshotted
MapReduce (edge-based)	graph snapshot	vertex-local	snapshotted
Pregel	graph snapshot	vertex-local	snapshotted
Event Folding	log sequence	single-vertex	time series
Command Folding	log sequence	single-vertex	time series
MapReduce (temporal)	graph snapshot	vertex-local	snapshotted/iterative
Pause/Shift/Resume	graph snapshot	vertex-local	iterative



Programming & Processing Models

Models Supported by the CHRONOGRAPH Concept

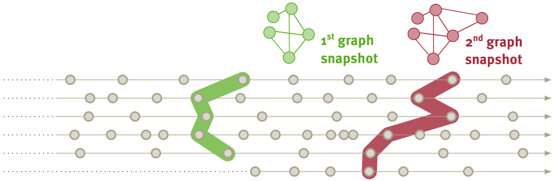
Model	Initial Data Set	Locality	Time Model
Main CHRONOGRAPH model	live graph	vertex-local	continuous
MapReduce (vertex-based)	graph snapshot	vertex-local	snapshotted
MapReduce (edge-based)	graph snapshot	vertex-local	snapshotted
Pregel	graph snapshot	vertex-local	snapshotted
Event Folding	log sequence	single-vertex	time series
Command Folding	log sequence	single-vertex	time series
MapReduce (temporal)	graph snapshot	vertex-local	snapshotted/iterative
Pause/Shift/Resume	graph snapshot	vertex-local	iterative



Programming & Processing Models

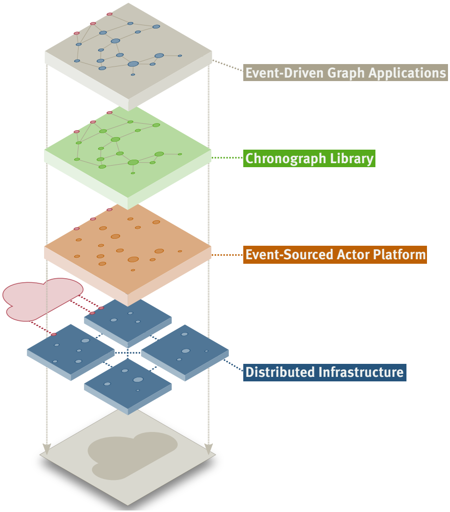
Models Supported by the CHRONOGRAPH Concept

Model	Initial Data Set	Locality	Time Model
Main CHRONOGRAPH model	live graph	vertex-local	continuous
MapReduce (vertex-based)	graph snapshot	vertex-local	snapshotted
MapReduce (edge-based)	graph snapshot	vertex-local	snapshotted
Pregel	graph snapshot	vertex-local	snapshotted
Event Folding	log sequence	single-vertex	time series
Command Folding	log sequence	single-vertex	time series
MapReduce (temporal)	graph snapshot	vertex-local	snapshotted/iterative
Pause/Shift/Resume	graph snapshot	vertex-local	iterative



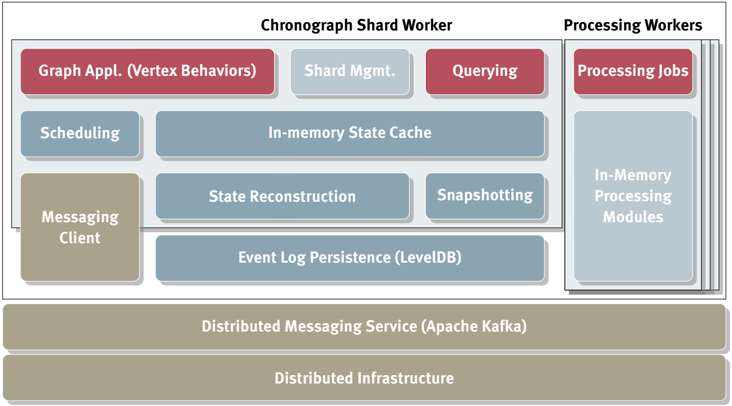
CHRONOGRAPH System

Conceptual Overview



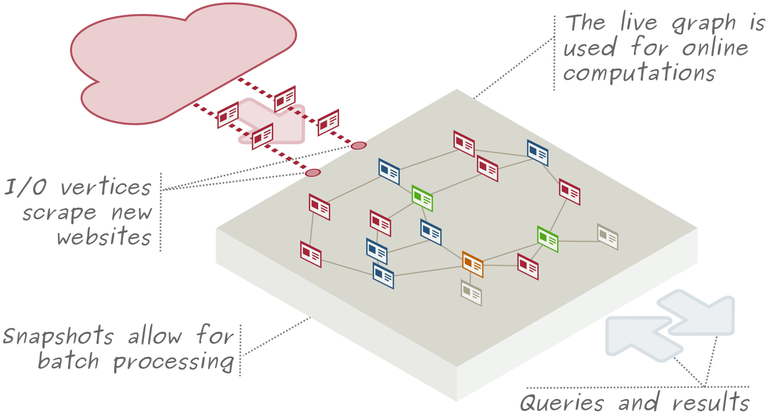
CHRONOGRAPH Worker Architecture

Shard Worker & Processing Worker(s)



Web Crawling with CHRONOGRAPH

Example Application



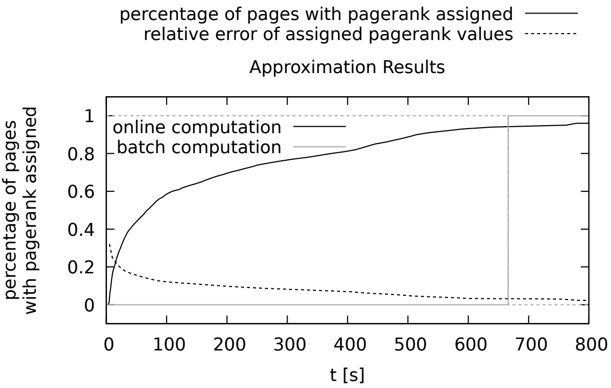
Performance Evaluation

Scenarios and Methodology

- evaluation workloads
  - small web graph (based on a SNAP data set)
    - 60,826 vertices, 143,766 edges
  - DEBS'16 Grand challenge: stream of social graph events
    - 42,934 vertices, 1,241,381 edges
- methodology
  - repeated and isolated runs of all workloads (5x)
  - collection and descriptive analysis of measurements
  - test setup
    - four bare-metal machines in dedicated LAN (1 GigE)
    - Intel Xeon E31220 (4x3.10 GHz); 16 GB RAM; Ubuntu 14.04 LTS;
  - measurements
    - application and worker metrics
    - process and system metrics

Performance Results: Fast vs. Exact Results

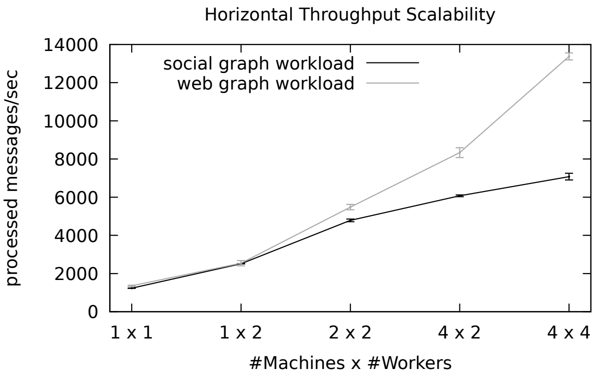
Online vs. Batch Computations



- example web graph (60,826 vertices, 143,766 edges)
- BSP/Pregel PageRank algorithm (offline) on completed graph
- online algorithm on evovling graph (based on Sankaralingam et al., 2003)

Performance Results: Scalability

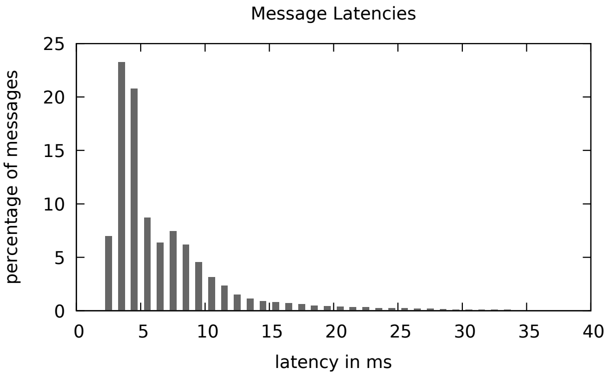
Application Messages per Second



- different workload characteristics: reactive vs. reactive + active

Performance Results: Message Latencies

Application-level Latencies between Vertices



- workload: social graph example; 4x4 setup
- distribution of application-level message latencies (vertex-to-vertex)
- latency statistics: mean=5.02 ms, P_{95}=15.16 ms, P_{99}=29.18 ms

Performance Results: Graph Reconstructions

Snapshotting & Reconstruction Times

	Average	**SD**
Graph snapshotting	3.65 ms	0.80 ms
Graph reconstruction	2701.48 ms	64.31 ms
Event restore rate	75728.49 events/s	1766.01 events/s

- web graph workload (60,826 vertices)
- test setup: 4 machines with 2 workers each

Future Work

Remaining Challenges & Future Directions

- in-depth performance analysis and optimizations
  - speed-up of worker performance
  - large-scale setups
  - history pruning and log compaction
- advanced operations on vertex event logs
  - retroactive modifications on branched logs
  - (partial) re-executions
- more real-world use cases and evaluations
  - centralized backend system for IoT applications (LoRaWAN topologies)

Conclusion & Take Aways

CHRONOGRAPH: Online and Batch Processing on Event-sourced Graphs

- event-sourced graph computing
  - asynchronous, message-based **computing model** for vertices
  - distributed **event sourcing** of the entire **graph evolution**
  - **arbitrary reconstructions** on vertex and graph level
- CHRONOGRAPH prototype platform
  - JavaScript-based **runtime environment** for **graph applications**
  - support for various **graph processing** models
  - single platform for **online & offline computations** on graphs

Thanks!

Questions?
Feedback?

Contact

WWW: uulm.de/?erb
Mail: benjamin.erb@uni-ulm.de
Twitter: @b_erb

Sources & Material

- Slide deck license
  -
- Title image
  - *untitled* by milivanily; CC0 License
- Icon sets
  - **PICOL-Icons** made by Melih Bilgil; licensed under CC-BY 3.0
  - **Font Awesome**; licensed under the SIL OFL 1.1
  - **Material Design Icons** by Google; licensed under CC-BY-SA 4.0

Reference

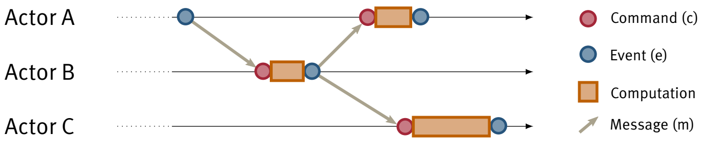
Original Publication

Benjamin Erb, Dominik Meißner, Jakob Pietron, and Frank Kargl. 2017. CHRONOGRAPH — A Distributed Processing Platform for Online and Batch Computations on Event-sourced Graphs. In *Proceedings of DEBS '17, Barcelona, Spain, June 19-23, 2017*, 10 pages.
DOI: 10.1145/3093742.3093913

Backup Slides

Command Sourcing & Event Sourcing of Actors

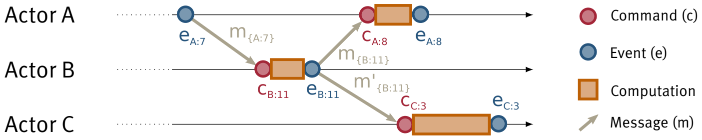
Separation of Log Entries



[1]: B. Erb, G. Habiger, and F. J. Hauck, “On the potential of event sourcing for retroactive actor-based programming”, ACM PMLDC’16

Command Sourcing & Event Sourcing of Actors

Versioning & Causality Tracking



[1]: B. Erb, G. Habiger, and F. J. Hauck, "On the potential of event sourcing for retroactive actor-based programming", ACM PMLDC'16

Event Log: State Modifications as Delta Events

Event-sourced Graph Programming Model

- e.g., JSON Patch (RFC 6902)

Delta State Computation:

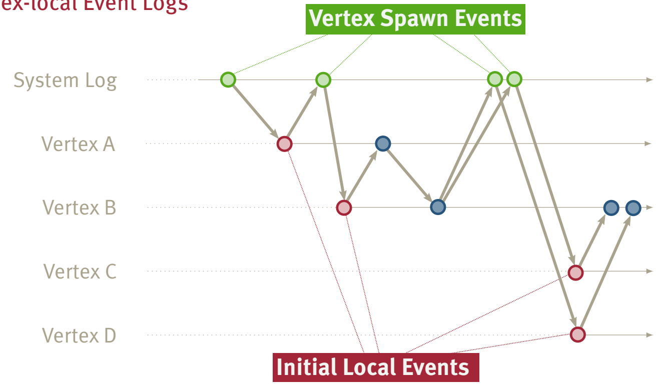
$$\mathit{Event}_t = \Delta(\mathit{State}_t, \mathit{State}_{t+1})$$

Left Fold of Updates:

$$\mathit{fold}(\{\}, \mathit{Event}_1, \ldots, \mathit{Event}_t) = \mathit{State}_{t+1}$$

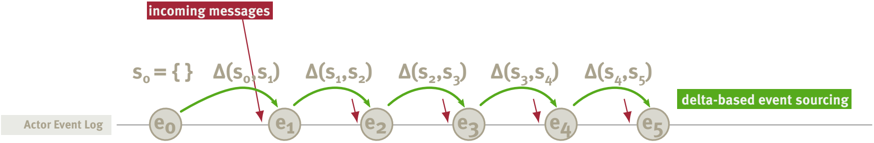
Distibuted Event Sourcing

Vertex-local Event Logs



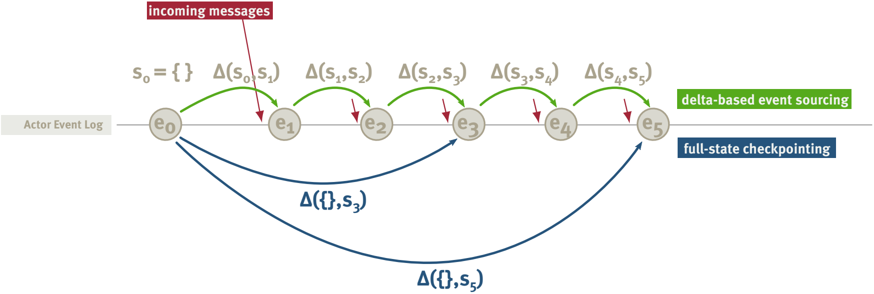
Checkpointing

Speeding up Reconstruction Times



Checkpointing

Speeding up Reconstruction Times



CHRONOGRAPH Programming API

Event-sourced Stateful Vertices

```
// event-sourced vertex behavior function
function vertexBehavior(message: incoming, state:
    currentState)
    sendMessage(vertex: neighbour, message: content);
    spawnVertex(function: behavior);
    spawnEdge(vertex: target);
    removeEdge(vertex: target);
    listOutgoingEdges();
    shutdownVertex();
    return newState;
```

Vertex-based MapReduce API

Snapshot-based Batch Processing

```
// user-defined map function for each vertex
function mapVertex(state: vertex)
    emit(key, value);

// user-defined reduce function
function reduce(string: key, list: values)
    return reducedResult;
```

Edge-based MapReduce API

Snapshot-based Batch Processing

```
// user-defined map function for each edge
function mapEdge(state: vertexOut, state: vertexIn)
    emit(key, value);

// user-defined reduce function
function reduce(string: key, list: values)
    return reducedResult;
```

Pregel-based API

Snapshot-based Batch Processing

```
// user-defined compute function for Pregel-based API
function compute(messages: incoming[], state: currentState)
    sendMessageTo(vertex: neighbour, message: content);
    getOutEdgeIterator();
    getSuperstep();
    voteToHalt();
    spawnVertex(function: behavior);
    spawnEdge(vertex: target);
    removeEdge(vertex: target);
    return newState;
```

Command Folding & Event Folding APIs

Log-based Event Processing

Event Folding

```
// user-defined event folding function
function fold(state: old, state: new, aggregate: foldState)
    return foldState;
```

Command Folding

```
// user-defined command folding function
function fold(message: command, aggregate: foldState)
    return foldState;
```

Temporal MapReduce API

Snapshot-based Temporal Batch Processing

```
// user-defined temporal map function for each vertex
function mapTemporal(state: vertex@s1, state: vertex@s2)
    emit(key, value);

// user-defined reduce function
function reduce(string: key, list: values)
    return reducedResult;
```