Oracle Event Processing and Coherence

Patterns for Distributed Event Processing

Philip Aston

Coherence SIG
15th November 2012

Outline

Introduction

Continuous Aggregation

Product Integration

Coherence

- In-memory Data Grid
- Simple event model for change notification
- Mature clustering
  - Partitioned scheme scales to large data sets
  - Transactional changes
  - High availability / recovery

Oracle Event Processing (OEP)

- Domain-specific Application Server
  - Event processing framework
  - Adapters
  - Tooling
- Continuous Query Language (CQL)
- Good Coherence integration
- Limited OOTB support for distributed event processing

The Problem

- Continuously reduce an event stream, grouped by some attributes

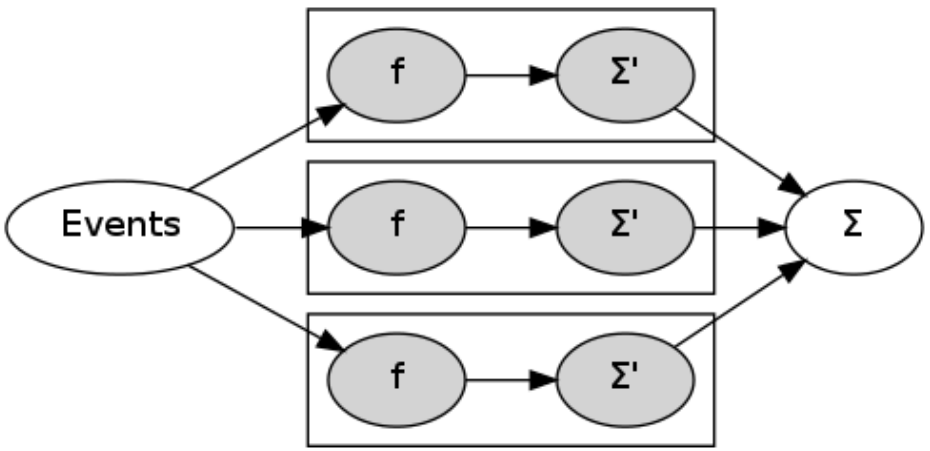
$$\mathrm{E} : (\mathrm{a},\ \mathrm{b},\ \mathrm{c},\ \mathrm{v})$$

```
select a, b, sum(v)
  from E
  group by a, b
```

- Do so in a *distributed*, *efficient*, and *resilient* manner

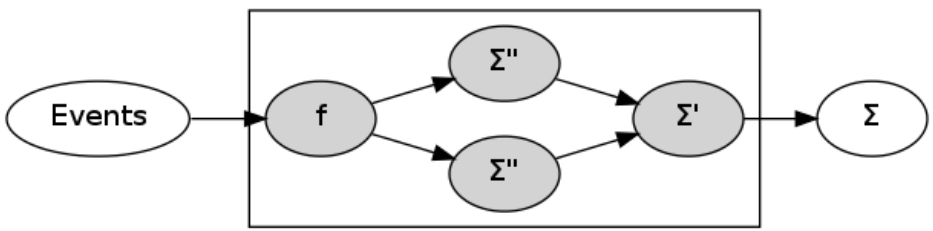
Processing partition ($\equiv$ shard)

- Large volume of events
- The event stream is distributed over hardware



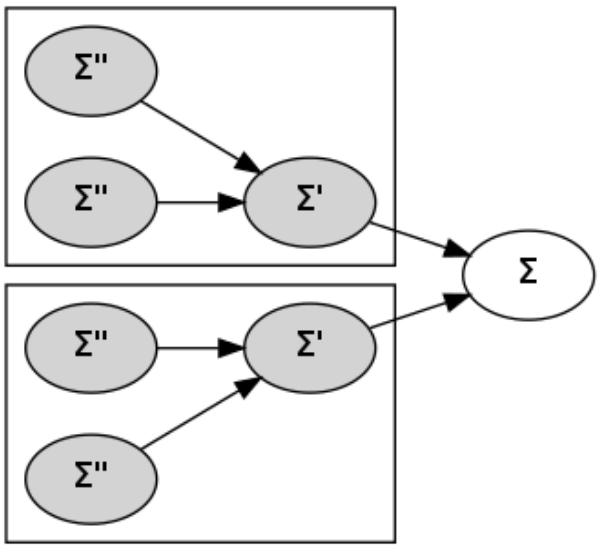
Entity-level aggregation

- Events typically refer to identifiable objects or *entities*
- Changes are calculated at entity level



Tiered aggregation

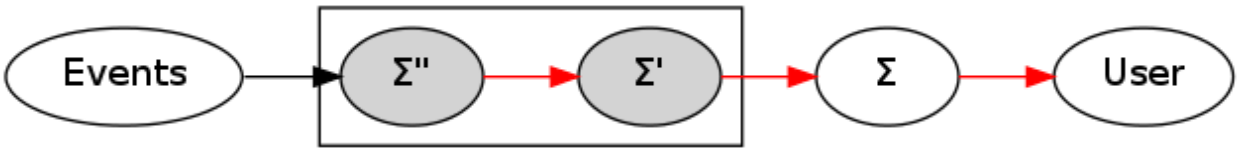
We've identified three levels of aggregation



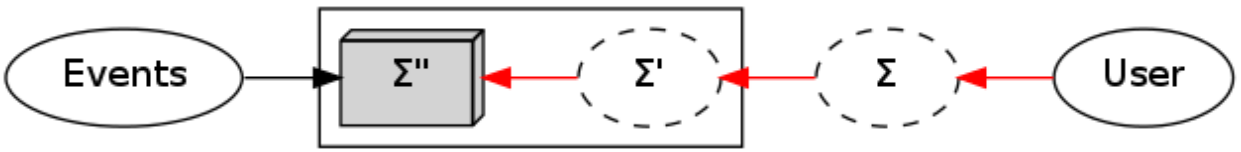
- Σ″ Entity
- Σ′ Processing partition
- Σ Total result

Push or Pull?

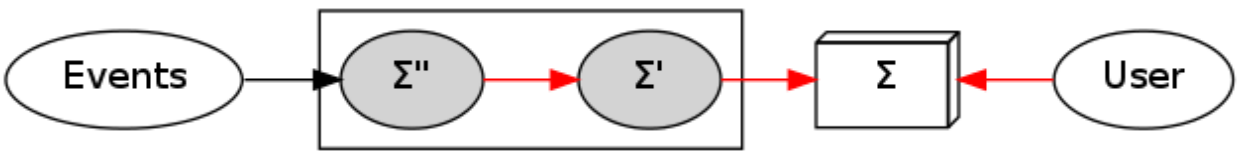
- Continuous aggregation



- Query



- Hybrid



Coherence features?

Distributed aggregation queries, but not continuous aggregation

- `EntryAggregator`
  - *Reduce cache entries to a result*
  - Pull
- Continuous Query Cache
  - *Filtered view of a cache*
  - Push
  - Not an aggregation mechanism

OEP features?

Continuous aggregation, but not aggregation by query

- Continuous Query Language

```
select a, b, myaggr(v) from E group by a, b
```

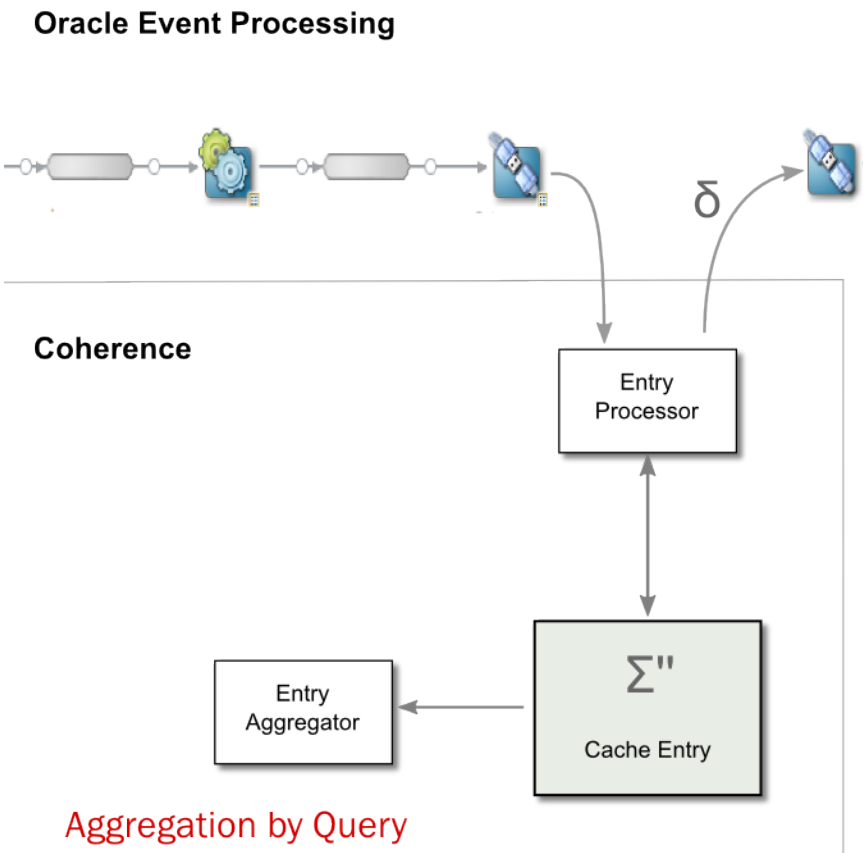
  - Aggregations are incrementally evaluated
  - Parallel evaluation, by group
  - Supports *User Defined Aggregations* in Java
- Highly stateful
  - State is implicit and cannot be replicated atomically
  - Distributed aggregation is an exercise for the user

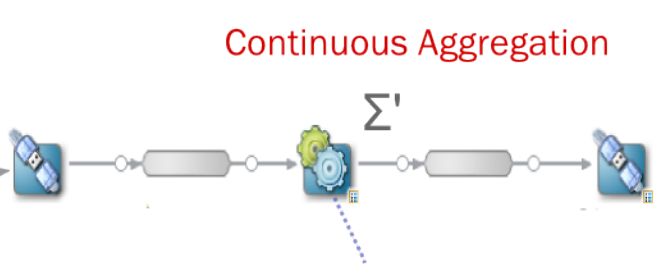
OEP + Coherence = Win

- Use Coherence partitions for processing partitions
  - Cheap event joins through key affinity
  - Partition-level transactions
- Store entities in Coherence
  - No duplication of entity state in OEP
  - Coherence provides resilience
  - Allows aggregation by query
- Use CQL for partition-level continuous aggregation
  - Declarative rules
  - Results can be re-calculated after failure

ORACLE

Entity and partition-level aggregation






```
<view id="Totals">
  istream(select date, book,
            sum(risks) as total
          from InputChannel
          group by date, book)
</view>

<query id="AggregationOutput">
  select date, book, total
  from Totals [range 15 slide 15]
  order by element_time desc rows 1
  partition by date, book
</query>
```

Final aggregation

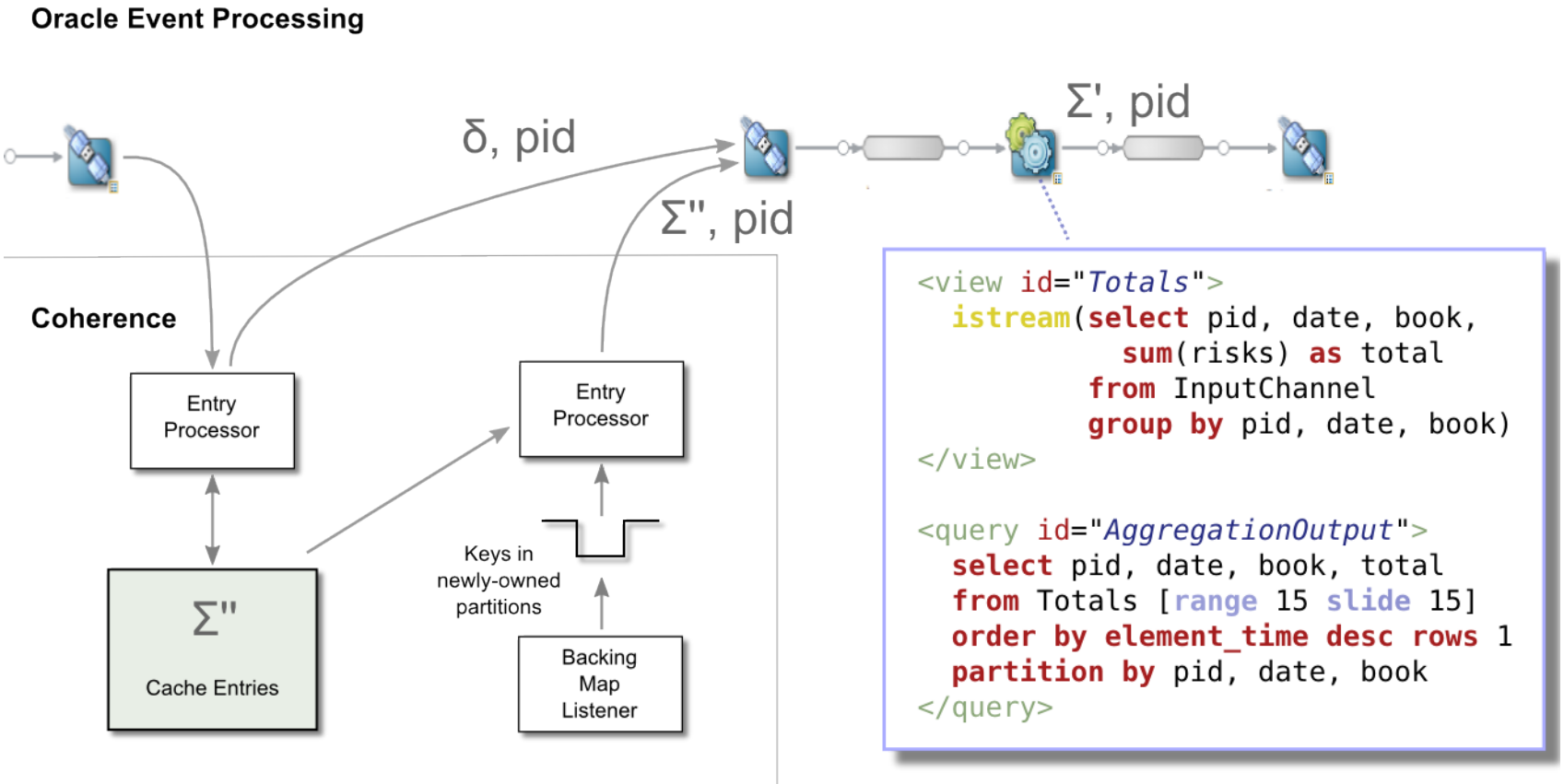
- Partition-level results are placed in a Coherence cache
  - Convenient publishing mechanism
- We query to produce the total
  - Results cache partitioned by logical key for cheap aggregation across processing partitions
- Alternative: continuously aggregate total using CQL
  - Appropriate if end-point is event-driven
  - E.g. message bus, or dynamic user interface

ORACLE

Resilience

- Processing partitions are Coherence partitions
- Each processing partition produces an absolute result
  - Output is a function of the partition contents
- A backing map listener tracks newly-owned entity keys
  - Scheduled EP flushes entity-level values for aggregation

Resilience



The actual CQL is a little more involved to discard aggregations for partitions we no longer own.

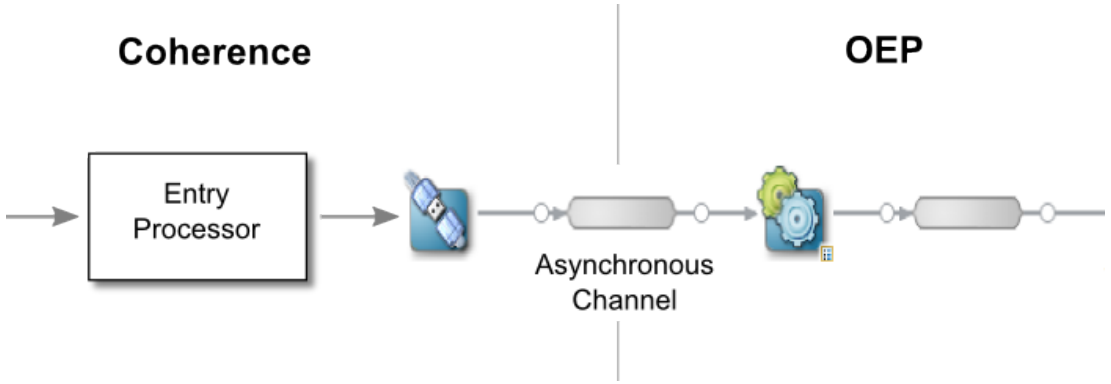
Two threading models

- Coherence service threads
  - Blocking an `EntryProcessor`, `EntryAggregator`, or a `BackingMapListener` will result in failure
- OEP threads
  - Uses WebLogic's self-tuning thread pool/work managers
  - Adjusts thread count for I/O or CPU bound work

Listening to a cache

Dispatch to the EPN must not result in I/O for every event

- OK? - use an `EntryProcessor` + asynchronous channel



- Otherwise, poll using the `NamedCache` API

Local processing

- Single distribution point when event enters system
- Subsequent processing kept within the JVM
  - `storageenabled=true`
- Our events make idempotent changes
  - Enables resiliency through *upstream backup*

Queue Cache