The programming continuum — Centre to edge

The Continuum Collaboration
Nova
Erlang Solutions Limited
INESC
Mainflux Tech
Scality
Sorbonne-Université
TU Kaiserslautern
Université catholique de Louvain
Universidad Politècnica de Catalunya

Centre vs edge

Data centre
- Resource-rich, high bandwidth
- Stable, low churn
- Consensus, strong consistency
- Far away, poor availability

Edge
- Local data, short response time
- Autonomy, availability, privacy
- Edge-edge collaboration
- High churn, weak consistency

Communication models

Memory-oriented:
- Read/write from/to database
- Global, flat; wide interface
- Consistency model
- Active process, passive data
- Structured data, unstructured processes
- Dominant in centre

Event-oriented, reactive:
- Structured message-passing graph
- Actor responds to events
- Data local, narrow interface
- Shared-nothing actors (no consistency issues?)
- Dominant at edge

Edge computing has a data problem

Edge-centric: latency, autonomy, availability
- Will grow (conjecture)

Scenarios:
- Collaborations
- Games
- Distributed Learning
- Vehicles

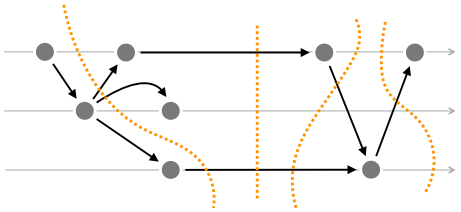
Cloud-mediated
- Aggregation, bandwidth
- “Stateless” services

Wanted: common model

Unified communication model
- Shared data + events
- Uniform semantics, guarantees
- Available first + strongest possible guarantees
  - As concurrent as possible w.r.t. semantics
- Security

Full power of distributed programming
- Abstract, don't hide
- Developer can optimise
- App logic level + operations level

Sweet spot: TCC



Transactional Causal Consistency
- $u \longrightarrow v \wedge v$ visible $\Longrightarrow u$ visible
- *same_bundle (u, v)* $\wedge v$ visible $\Longrightarrow u$ visible
- All *events* that contributed to current *state* are visible
- All *states* that contributed to current *event* are visible

Seamlessly strengthen CC
- SSER = CC + total-ordered snapshots
- Intermediate: some snapshots mutually ordered
- When required by application semantics

Possible API

```
available_ts := available()

commit_ts :=
  txn (available_ts, locks, attributes) {
    ref counter x := db (key_x)
    pre x ≥ 0
    x.inc (10)
    ref set y := db (key_y)
    y.add ("foobar")
    }

subscribe (x, my_callback)

my_callback () returns (ref, new_ts)
                          // not value

Logical time: 1st-class, ≦
```

No wait

Most recent available snapshot

Non-failing

- Consistency
- Centre or edge
- etc.

before-or-concurrent ≡ ¬after

Other highlights

CRDTs

Data invariants
  ⟹ weaker, stronger consistency as required

Availability-compatible access control

End-to-end encryption

Programmer-defined distributed abstractions

Composable abstractions

Integrate SysOps

Single semantics, multiple implementations

The above can be implemented in many different (but mutually compatible) ways, for instance in the core vs. at the far edge.

For instance, we leverage data centres for ensuring consistent communication and backup, and for high-bandwidth computation

Place data/computation where most appropriate

Continuum proposal

Programming continuum: core cloud to far edge

- *Semantics independent of location*
- *Processes communicate and share data consistently*
- *Availability first: local data*
- *Consistent security model*

Methods and tools for application correctness

Principled Systems Operations

- *Orchestration, Elasticity, Placement*

Continuum

Need *mutually-consistent* events and state

- Don't tack one on top of the other!

Causal consistency

- Compatible with availability under partition,
- Snapshots: mutual consistency

Strengthen to total-order when required by app semantics