

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

Far edge



Fast reads
 Replicate updates
 consistent \cap available = \emptyset

[The programming continuum, from core to edge]

X

[Dagstuhl PL for Dist. Sys. 2019-10-28]

Programming distributed systems

Lots of issues / silos

- Business logic of a service
- Composing services
- Sharing data
- Reacting to events
- Security
- Deployment, placement, monitoring
- etc.

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

Diverse memory models

Previous work, such as CRDTs [125], TCC [7] and Proteus [138] show the direction for a unification. Abstractly, the system can be modelled as a partial order of states or of update events. Processes can exchange events or messages, which may contain references to state. At any point in time, the state of the database observed by a process is causally consistent with the set of events that it received. This model can be seen as equally event- and memory-centric, provides a familiar consistency guarantee, remains asynchronous and does not reduce availability.

Tension: deterministic vs. available

Models:

- Single sequence of versions
- Multiple version, per process + convergence
- Not monotonic: Branching/Rollback + stable prefix

Replication:

- Core: full replication + ops
- Edge: partial replication + state

Why?

No profound reason for proliferation of incompatible models.

Full-featured computers at the edge

Simplify from different perspectives

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

Unifying the data model

CRDTs

- synchronisation-free
- high availability, local access

Key: a base data/consistency model

- Consistent snapshot, data + events
- CRDTs + TCC + optional stronger
- multi-value concurrency control MVCC

Unifying data & events (1)

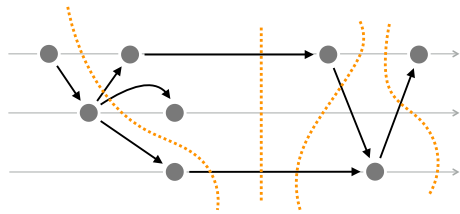
Communication stream =

- Updates, events
- States
- Metadata
 - system: VC, transaction, etc.
 - arbitrary: debugging, flows, etc.

Local state

- CRDTs, Versioned
- = previous state + updates received
- = set of updates that led to this snapshot

Sweet spot: TCC



Transactional Causal Consistency

- $u \rightarrow v \wedge v \text{ visible} \Rightarrow u \text{ visible}$
- $\text{same_bundle}(u, v) \wedge v \text{ visible} \Rightarrow u \text{ visible}$
- 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

[The programming continuum, from core to edge]

6

[Dagstuhl PL for Dist. Sys. 2019-10-28]

Possible API

snapshot

Most recent available snapshot
⇒ no wait

```

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

Non-failing

- Consistency
- Centre or edge
- etc.

subscribe (x, my_callback)

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

Logical time: 1st-class, \cong

before-or-concurrent
 $\equiv \neg \text{after}$

[The programming continuum, from core to edge]

7

[Dagstuhl PL for Dist. Sys. 2019-10-28]

Transparent metadata

Attach metadata to information

- CC
- provenance
- flow analysis
- semantic tags
- etc

Not inflate message size ⇒ metadata store

[The programming continuum, from core to edge]

X

[Dagstuhl PL for Dist. Sys. 2019-10-28]

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

[The programming continuum, from core to edge]

8

[Dagstuhl PL for Dist. Sys. 2019-10-28]

Confidentiality

CRDTs: merge in user device

- End-to-end encryption
- Cloud for storage, communication
- Data not exposed
- But operations, concurrency exposed

P2P encryption in dynamic groups

- See Snapdoc [Kleppmann 2019]

[The programming continuum, from core to edge]

X

[Dagstuhl PL for Dist. Sys. 2019-10-28]

Security & isolation → data model

Security without strong consistency?

Don't send to replica not allowed to read
⇒ partial replication

Refuse updates from replica not allowed to write

- $visible(u) \Rightarrow legal(u)$
- $u \rightarrow v \wedge \neg legal(u) \Rightarrow \neg legal(v)$

Data model:

- Branching histories: fork consistency
- Non-monotonic views, rollbacks, fragmented store?

[The programming continuum, from core to edge]

X

[Dagstuhl PL for Dist. Sys. 2019-10-28]

AccGreGate security model

AccGreGate: access control for weak consistency

- No point in hiding already-visible versions; access right change applies to future versions
- Associate security metadata to data
- Check metadata on/after access
- Concurrent changes: most restrictive wins

Conjecture (TBC):

- AccGreGate + TCC \Rightarrow monotonic, no rollback

[The programming continuum, from core to edge]

X

[Dagstuhl PL for Dist. Sys. 2019-10-28]

Securing CC

Forging VCs, manipulate history
Solutions?

[The programming continuum, from core to edge]

X

[Dagstuhl PL for Dist. Sys. 2019-10-28]

Composability

composition of parts

Ex: security layer, metadata layer

Composability is key

The designer should be able to create and reason about distributed abstractions.

Modular, Composable verification techniques

[The programming continuum, from core to edge]

X

[Dagstuhl PL for Dist. Sys. 2019-10-28]

Applications are often developed under the hidden assumption of strong consistency. (AP) must accept concurrent updates,. Alternatives exist, which depend on the application invariants [127]. Some can be maintained purely at the application level; some need multiple operations to execute transactionally; some require the system to order reads; others, to order writes. To address this challenge, we will develop a library of concurrency control protocol abstractions, as well as language-level logics (supported by static and dynamic verification tools), in order to generate the most efficient correct protocol that ensures the application remains correct in an imperfect environment.

[The programming continuum, from core to edge]

X

[Dagstuhl PL for Dist. Sys. 2019-10-28]

SysOps

Objectives. The overarching goal of this package is to develop tools and techniques that help programmers building correct and efficient multicloud applications. These multicloud applications use replicated data in the cloud and edge, and have to address issues as message latency, failures, and concurrency. These difficulties are magnified because nowadays applications, instead of being built as a monolithic entities, they are structured as a set of autonomous and independent services. Ensuring data consistency between these loosely-coupled services raises new and difficult challenges: related data is scattered across services; the storage system of each service might have a different consistency model; operations

[The programming continuum, from core to edge]

X

[Dagstuhl PL for Dist. Sys. 2019-10-28]

Computation, data access and events consistently, wherever located

Deployment, elasticity, placement and location

- Important
- Orthogonal to business logic
- Programmed with the same abstractions as functional program text

Need principled methods and tools for supervising and operating geo-distributed and edge systems

[The programming continuum, from core to edge]

X

[Dagstuhl PL for Dist. Sys. 2019-10-28]

Cloud-to-Edge system operations, deployment and control. This challenge is to automate deployment and control (including placement of data and computation) of highly-dynamic, evolving cloud-edge systems. Users will set service-level objectives (SLOs). SLO metrics are often in tension and require trade-offs; they may include, for instance, energy consumption, response times, atomicity vs. freshness of data, security constraints vs. bandwidth, and monetary cost. We need to support deployment, placement, monitoring, and run-time analysis of large, dynamic systems, and to support seamless evolution of sub-systems and of interfaces between them.

Placement and Migration Although the APIs and semantics are uniform across the whole core-to-far-edge spectrum, the operational costs are different depending on location. Therefore, data and computations should be placed intelligently (possibly requiring transformations), mostly in an automatic fashion (i.e., minimal to no human interaction) and proactively adapting to variations in workloads, resource availability, key performance indicators of applications, etc. Such dynamic placement must respect consistency, correctness, and security requirements, which is a non-trivial set of restrictions to be addressed at large scale. Furthermore, centralized solutions must be avoided, as to enable fast adaptation and

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

Simplified distributed programming?

MapReduce, TensorFlow, Flink

- Very restricted programming model
- Automate deployment, elasticity, etc.
- Well suited to a specific problem area

Orleans:

- Generic programming at app level
- Configuration level: deployment, elasticity for dummies

Ansible, Puppet, Salt, Kubernetes...

- Orchestration, configuration level only

Control = second-class?

Placement is essential for performance

Open universe?

Garbage collection:

- Stable property w.r.t. CC w/o rollbacks
- Global property (requires closed group?)

Causal consistency

- Vector clock:
 - $O(|universe|)$
- Stable Causal Snapshot (can forget anything prior)
 - $O(|universe|^2)$

[The programming continuum, from core to edge]

X

[Dagstuhl PL for Dist.Sys. 2019-10-28]

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*

[The programming continuum, from core to edge]

10

[Dagstuhl PL for Dist.Sys. 2019-10-28]

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

[The programming continuum, from core to edge]

11

[Dagstuhl PL for Dist.Sys. 2019-10-28]

 [Creative Commons Attribution-ShareAlike 4.0 Intl. License](https://creativecommons.org/licenses/by-sa/4.0/)

You are free to:

- Share — copy and redistribute the material in any medium or format
- Adapt — remix, transform, and build upon the material for any purpose, even commercially, under the following terms:



Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.



ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

[The programming continuum, from core to edge]

12

[Dagstuhl PL for Dist.Sys. 2019-10-28]