Type-Level Latency Tracking with Placement Types

Tobias Reinhard

TECHNISCHE UNIVERSITÄT DARMSTADT

Software across Geo-Distributed Data Centers

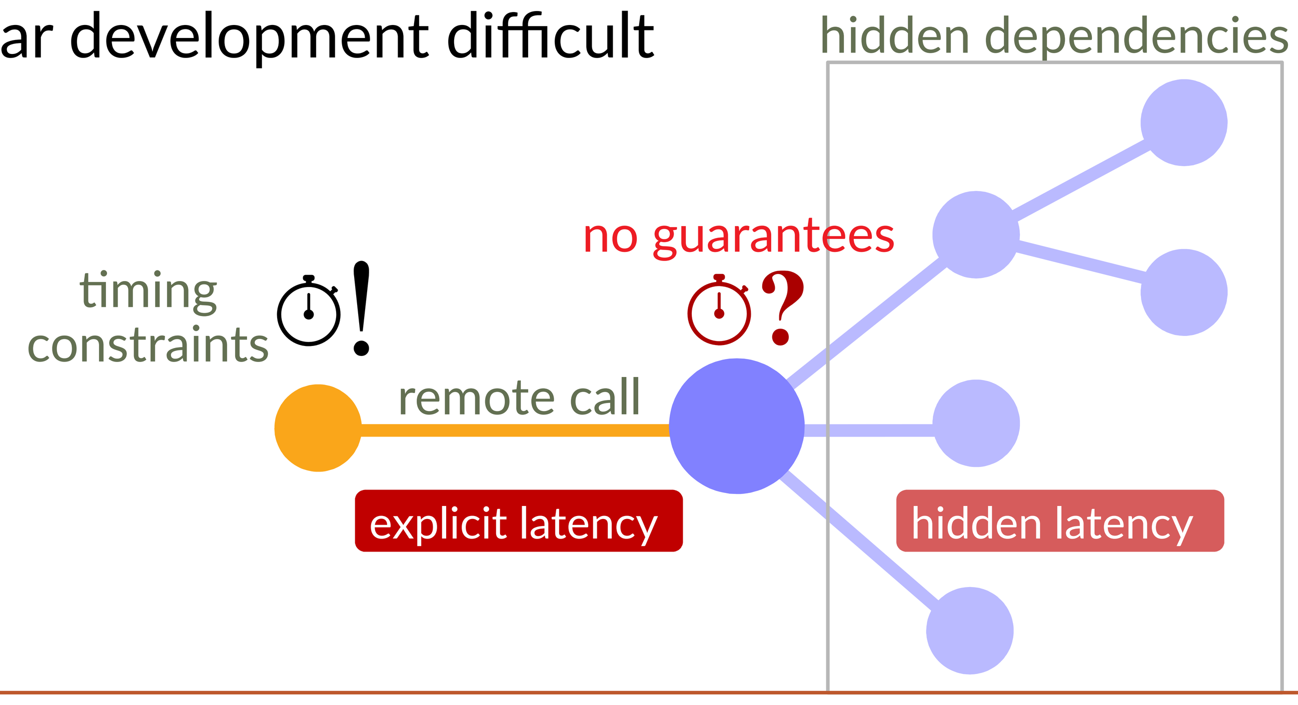
- Fixed locations
- Predictable latency between servers:
  - within same data center < 2ms
  - in different data centers > 100ms

→ Location matters

Latency is Hard to Predict in Distributed Applications

- Remote computations with hidden dependencies
- No static guarantees towards timing constraints
- Need global view to reason about latency

→ Modular development difficult



Goals

Make latency and locations explicit

Static guarantees towards latency

⇒ Allows latency-aware software development & refactoring

Extending ScalaLoci's Placement with Latency

- Types encode location and number of remote communication steps.
- Makes locations and latency explicit.

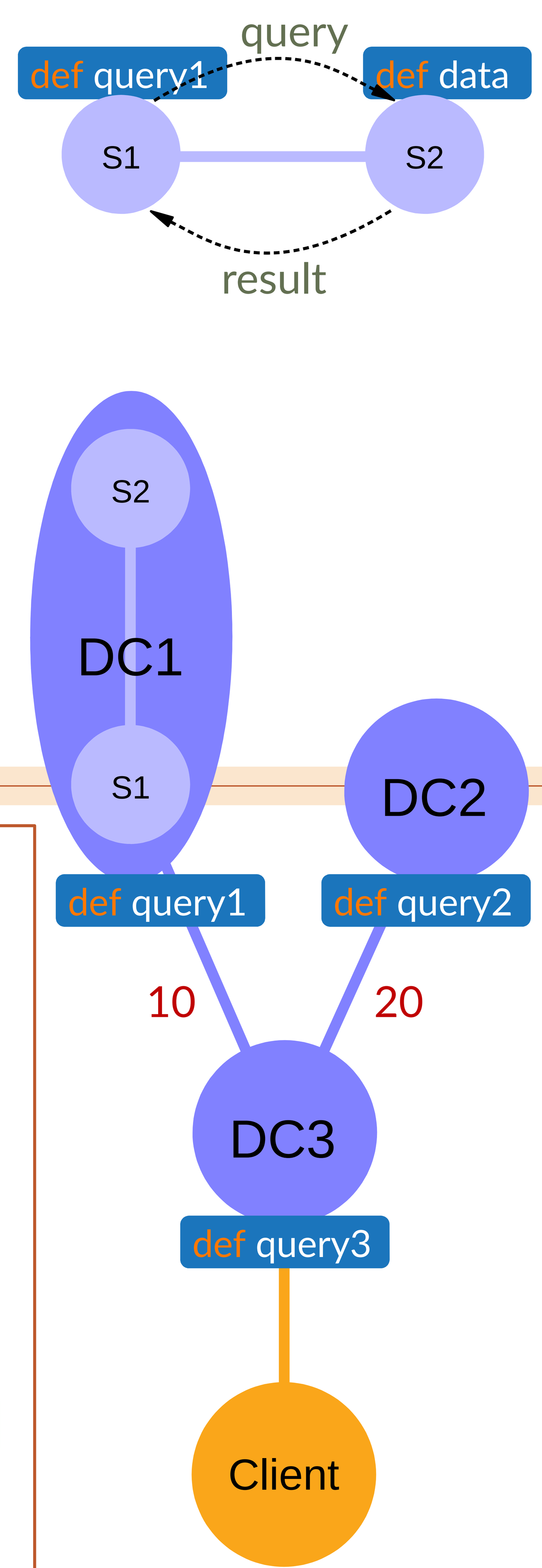
→ No hidden dependencies

placement | latency

```
def data(q: Query): Result on S2 withLat _0
  = placed[S2] { computeResult(q) }
```

specify location | # messages

```
def query1(q: Query): Result on S1 withLat _2
  = placed[S1] { remoteCall data(q) }
```



Latency Weights & Bounds

- Weights on connections approximate latency predictions.
  - DC1 ⇔ DC3 : 10
  - DC2 ⇔ DC3 : 20
- Latency is upper bound on all paths through the program.

Bound on all paths

```
def query3(q: Query): Result on DC3 withLat Max[_40, _22]
  = placed[DC3] {
      if (cond(q)) remoteCall query2(q)
      else remoteCall query1(q) }
```

latency: 40

latency: 20 + 2

Static Guarantees

Type system rejects wrong assumptions:

```
placed[Client] { val r: Result withLat _2
  = remoteCall query3(myQuery); ... }
```

latency: 2 + 40

Allows explicit over-approximation:

```
placed[Client] { val r: Result withLat _50
  = remoteCall query3(myQuery); ... }
```

Type signatures encode location and latency:

```
query3 : Query => Result on DC3 withLat _40
```

→ Allows modular development.

Provably Correct Bounds

Formalization based on λ-calculus with

- Remote communication primitives
- Placement & latency types
- Sized types & size-decreasing recursion

size | latency

$$\frac{\Lambda \Vdash B_t \approx B_f \qquad \Lambda \vdash s_t =_0 s_f \qquad \Delta;\Gamma;\Lambda;P \vdash t_c : (\mathsf{Boolean}, [0], [l_c]) \qquad \Delta;\Gamma;\Lambda;P \vdash t_t : (B_t, [s_t], [l_t]) \qquad \Delta;\Gamma;\Lambda;P \vdash t_f : (B_f, [s_f], [l_f])}{\Delta;\Gamma;\Lambda;P \vdash \mathsf{if}\, t_c \,\{t_t\}\,\{t_f\} : (B_t, [s_t], [l_c + \max(l_t, l_f)])} \;(\text{T-IF})$$

latency bound

→ Correctness proof for inferred latency bounds

- $\Delta;\Gamma;\Lambda;P \vdash t : (B, [s], [l_T])$
- $(\langle t \rangle_{\mathcal{I}}, [0]) \overset{\mathcal{I}}{\rightsquigarrow}_* (\langle v \rangle_{\mathcal{I}}, [l_R])$

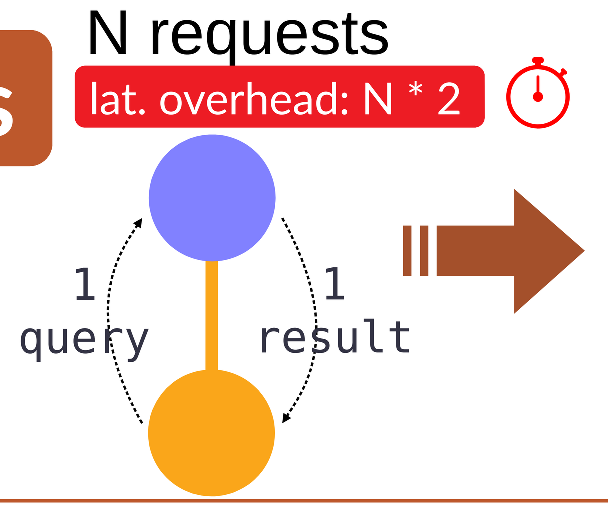
$\Longrightarrow \quad l_R \leq l_T$

Latency-Saving Refactorings

Type system can guide refactoring.

```
val queries: SizedList[Query, N] = ...
queries.map(q => remoteCall query3(q)) }
```

on Client | on DC3

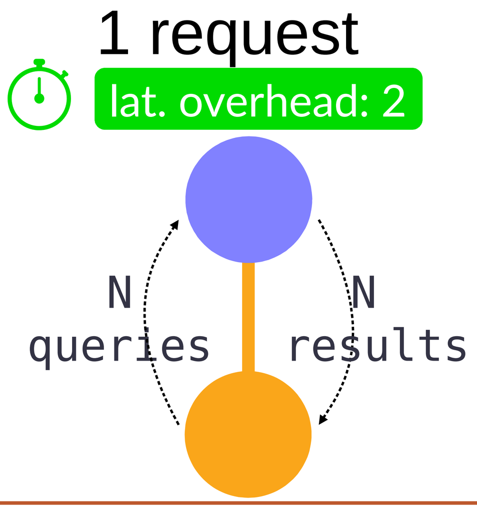


Exploit locality to reduce communication.

```
val queries: SizedList[Query, N] = ...
placedCall[DC3] queries.map(q => query3(q))
```

on DC3

→ Can be automated based on type info.



- P. Weisenburger, T. Reinhard, G. Salvaneschi. Static latency tracking with placement types. FTfJP, 20018.
- P. Weisenburger, M. Köhler, G. Salvaneschi. Distributed system development with ScalaLoci. OOPSLA, 2018.
- S. Jost, H. Loidl, N. Scaife, K. Hammond, G. Michaelson, M. Hofmann. Worst-case execution time analysis through types. ECRTS, 2009.