Reactors: A Case for Predictable, Virtualized Actor Database Systems

Vivek Shah, Marcos Antonio Vaz Salles
University of Copenhagen(DMS GROUP@DIKU)

UNIVERSITY OF COPENHAGEN

Motivation

New OLTP Application Trends

Increasing application logic complexity
Latency is critical
Scalability matters

Is the programming interface of stored procedures good enough for modern OLTP databases?

- Performance Challenges
  - Locality and intra-transaction parallelism
  - Low-level deployment control
- Software Engineering Challenges
  - Modularity and isolation
  - Abstractions to reason about performance

ERLANG akka Orleans

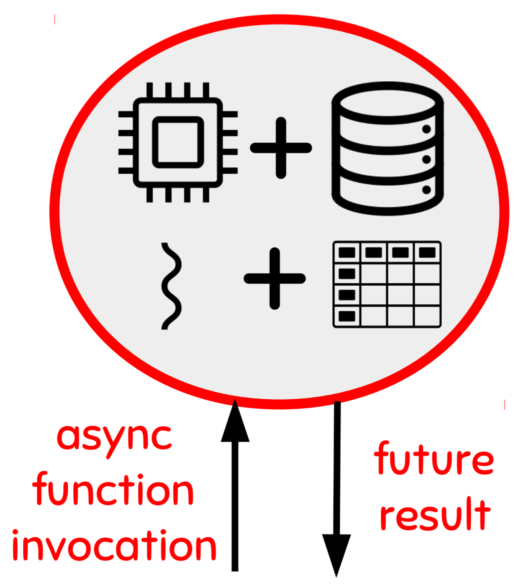
Modular, Concurrent and Asynchronous

How can we integrate actor programming models in modern relational databases to improve programmability of stored procedures?

Relational Actor (Reactor) Programming Model

Reactor Programming Model Concepts

- **A reactor is an application-defined actor encapsulating the state as relations**
- **Declarative queries are used to access the encapsulated reactor state**
- **Communication across reactors only through asynchronous function calls**
- **Computations across reactors provide ACID guarantees**



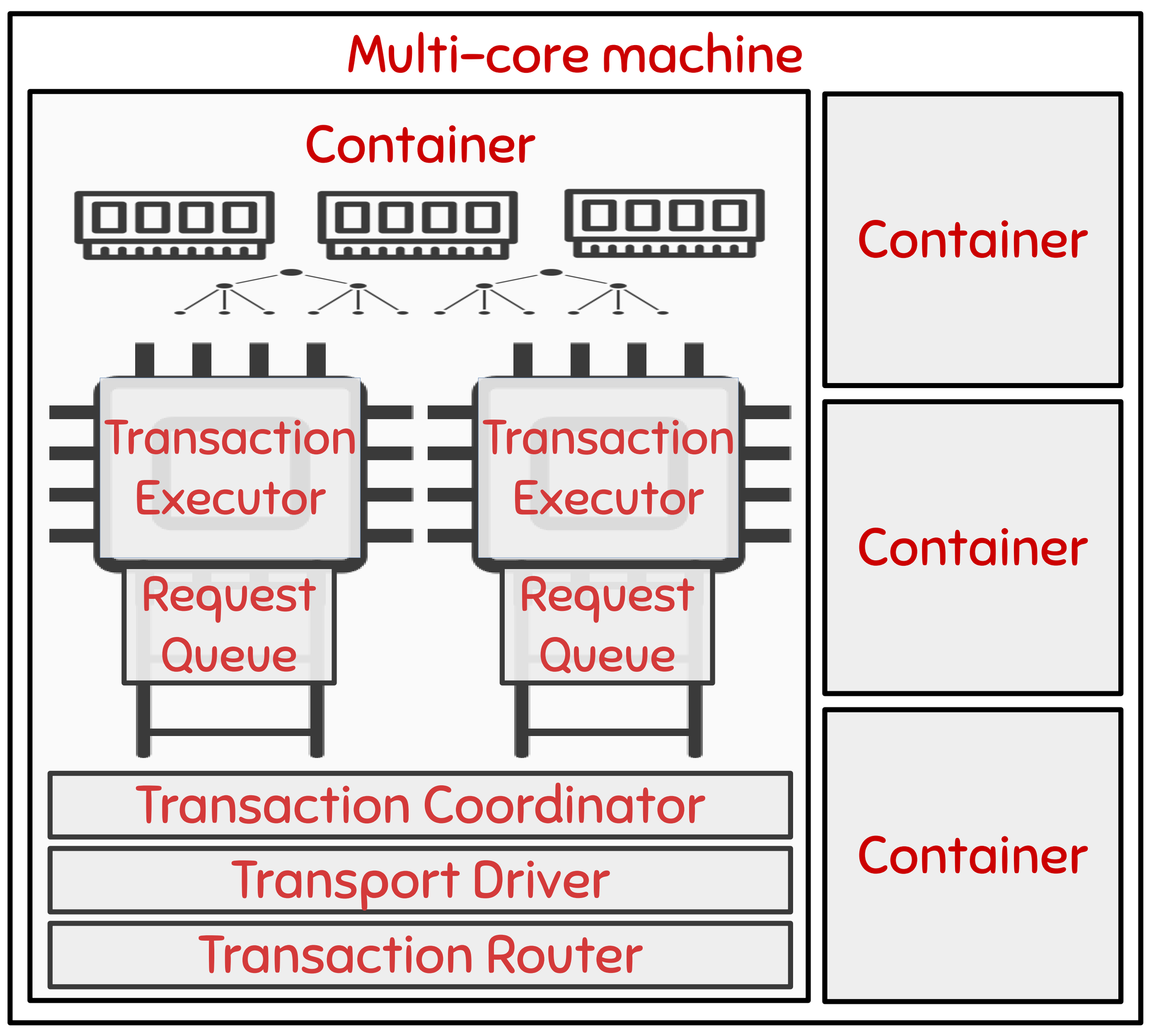
- Reactor Type
  - Schema
  - Methods
- Reactor Instance
  - Based on type
  - Unique name for addressing

```
reactor Provider {
  Relation orders { wallet int, value float,
            settled char(1) };
  Relation provider_info { risk float,
                     time timestamp,
                     window interval };

  float calc_risk(p_exposure) {
    SELECT SUM(value) INTO exposure
    FROM orders WHERE settled = 'N';
    if exposure > p_exposure then abort;
    SELECT risk, time, window
         INTO p_risk, p_time, p_window
    FROM provider_info;
    if p_time < now() - p_window then
     p_risk := sim_risk(my_name(),exposure);
     UPDATE provider_info SET risk = p_risk,
                      time = now();
    end if;
    return p_risk;
  }
  void add_entry(wallet, value){
    INSERT INTO orders VALUES (wallet, value, 'N');
  }
}
```

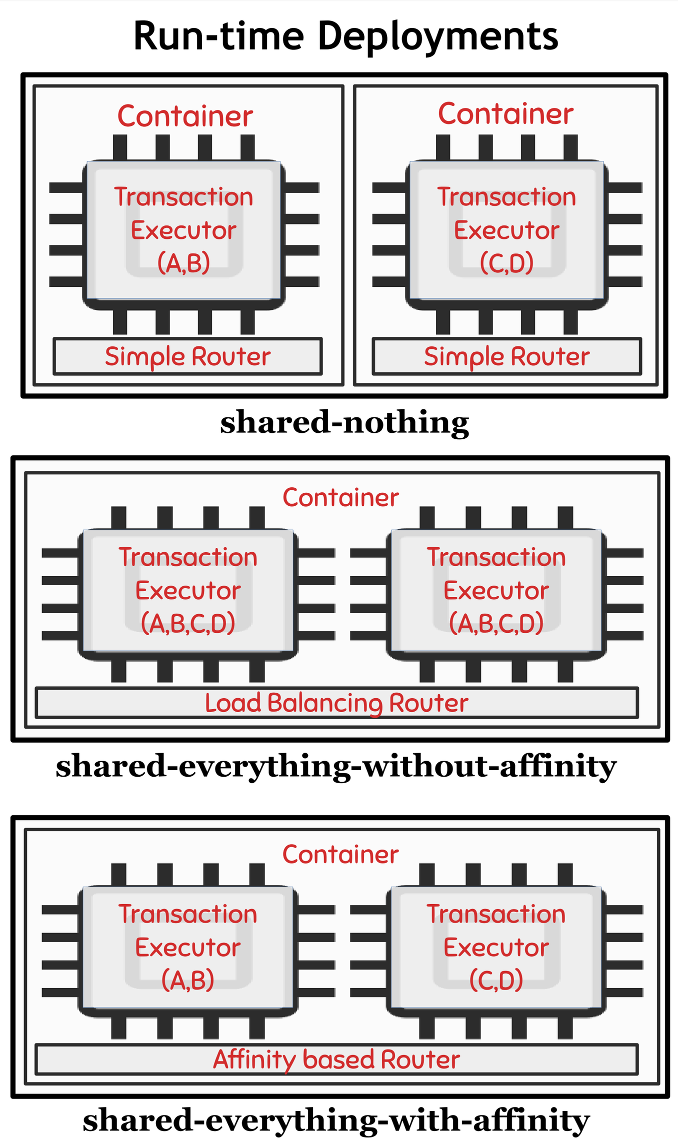
```
reactor Exchange {
  Relation settlement_risk { p_exposure float,
                  g_risk float},
  Relation provider_names  { value varchar(32) };

  void auth_pay(pprovider, pwallet, pvalue) {
    SELECT g_risk, p_exposure INTO risk, exposure
    FROM settlement_risk;
    results := [];
    foreach pprovider in
           (SELECT value FROM provider_names) {
      res := calc_risk(exposure) on reactor pprovider;
      results.add(res);
    }

    total_risk := 0;
    foreach res in results
      total_risk := total_risk + res.get();

    if total_risk + pvalue < risk then
      add_entry(pwallet,pvalue) on reactor pprovider;
    else abort;
    end if;
  }
}
```

A Simplified Digital Currency Exchange Application using Reactors

Design and Implementation of ReactDB



ReactDB Architecture



Implementation Overview

Thread Management in Transaction Executors:
- Thread pool uses cooperative multitasking
- Configurable multi-programming level

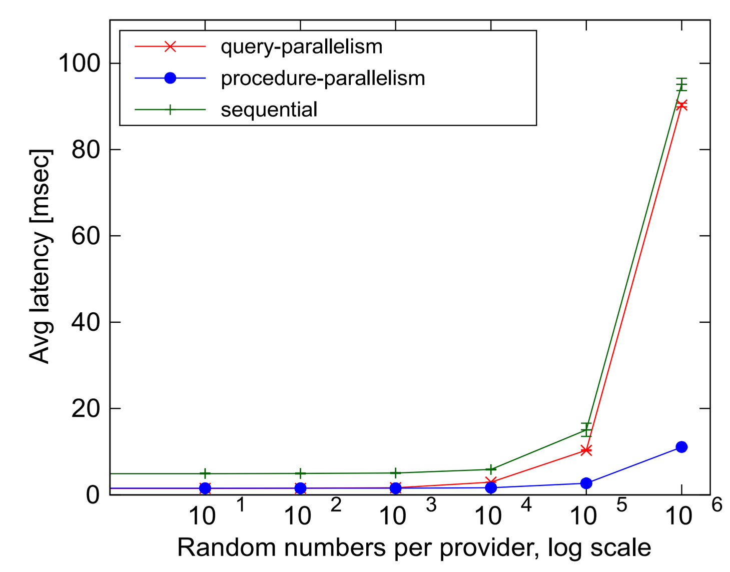
Storage Layer:
- Primary Masstree indices (Mao et al. [EuroSys 2012])

Concurrency Control:
- **Single Container**
  - OCC protocol (Silo, Tu et al. [SOSP 2013])
  - Single transaction context, sequential execution to leverage shared memory
- **Multi-Container**
  - OCC + 2PC protocol
  - Multiple transaction contexts, asynchronous execution across containers

Experimental Evaluation

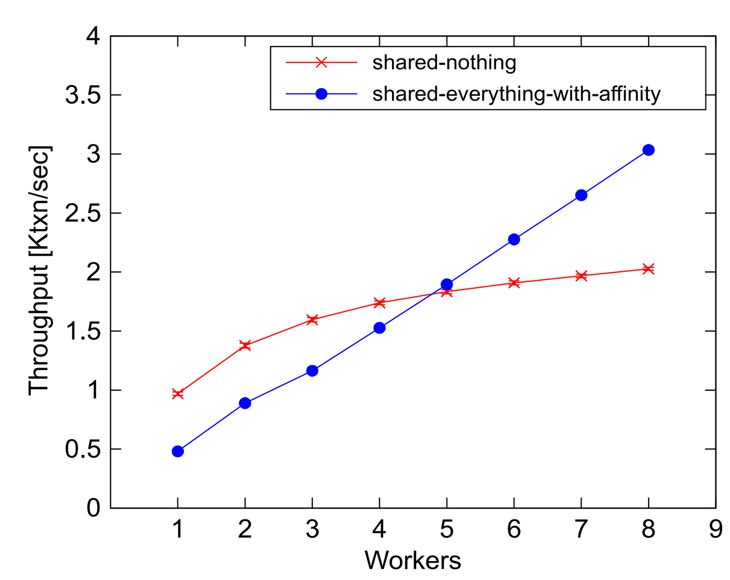
- Simplified Digital Currency Exchange Application
- 1x Exchange reactor, 15x Provider reactors (30,000 orders per provider)
- 1x shared-everything-with-affinity container (sequential) vs 16x shared-nothing containers with partially asynchronous (query-parallelism) or fully asynchronous (procedure-parallelism) auth_pay programs
- Varying complexity of sim_risk simulated by random number generation, single worker only

Asynchronicity gains manifest with increasing complexity of application logic



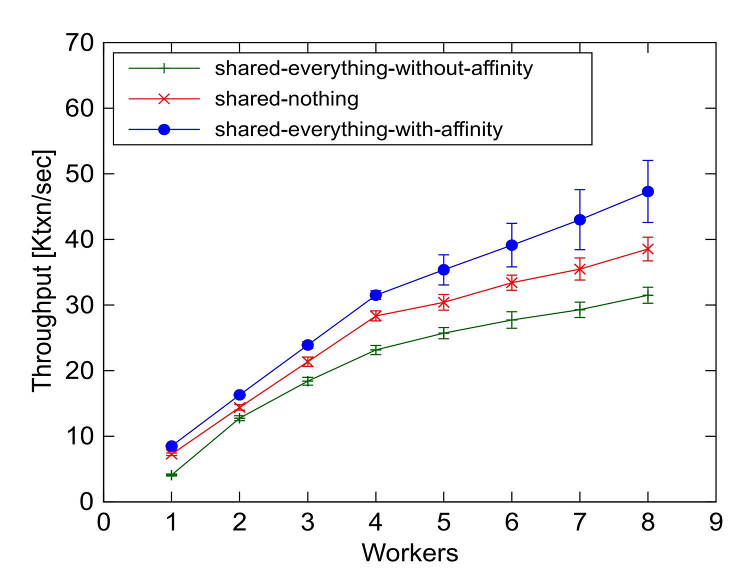
- 100% TPC-C new-order + artificial delay of 300-400 µsec (random number generation to simulate complex stock replenishment logic) per warehouse
- Each warehouse is a reactor
- Fixed scale factor of 8 with varying workers

Asynchronicity gains diminish with increasing load



- Standard TPC-C benchmark mix
- Each warehouse is a reactor
- Fixed scale factor of 4 with varying load on databases as a whole (workers > 4 simulate overload on the database)

Memory access affinity matters for sequential execution of classic OLTP workloads



Machine

2x AMD Opteron 6274 with 8 cores @ 2.1 GHz
4x 32 GB DDR3 RAM
64 KB L1, 2 MB L2, 6 MB L3 cache