Universität
Augsburg
University

Evaluation of Parallel Design Patterns for Message Processing Systems on Embedded Multicore Systems

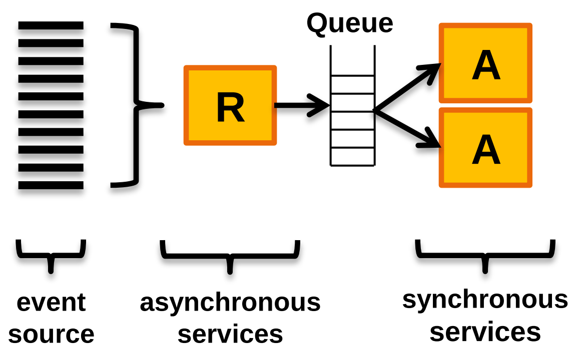
Ronald Strebelow
Institute of Computer Science
University of Augsburg

Christian Prehofer
Fraunhofer Institute for
Communication Systems ESK

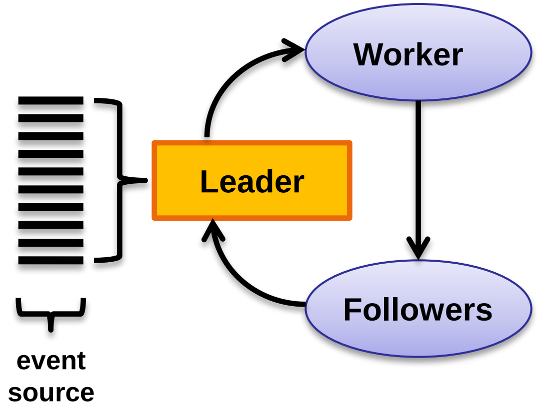
Motivation

- Design patterns reflect engineering practice and experience, but
- Complexity and performance implications often unknown
- Several patterns exist for event processing:
  - Reactor
  - Half-Sync/Half-Async
  - Leader/Followers
  - Proactor
- All patterns have been evaluated before
  - But never in the same context
  - Always under a specific application
  - Not on multicore systems
- Our aim:
  - Evaluation of all patterns in the same application-agnostic context
  - On embedded multicore system

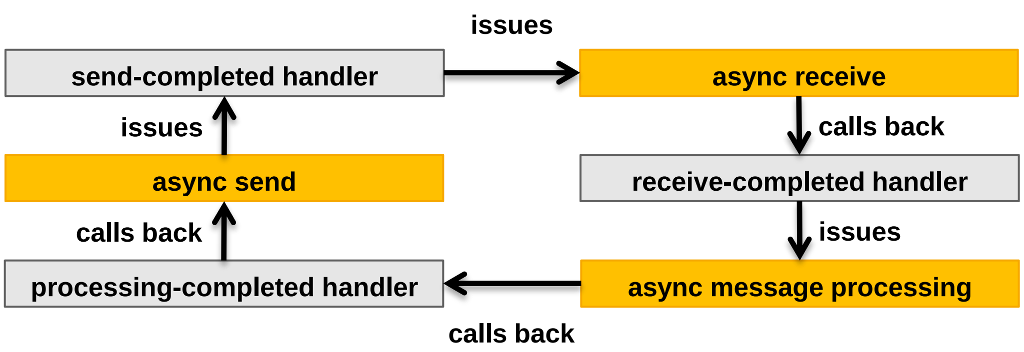
- Thread-per-Connection (multi-threading strategy)
  - Each thread serves one connection exclusively
  - Thread terminates after connection was torn down
- Reactor (pattern)
  - Single threaded → avoids all multithreading overhead
  - Used in Half-Sync/Half-Async & Leader/Followers
- Half-Sync/Half-Async (pattern)
  - Distinguish between asynchronous and synchronous services
  - Asynchronous services are triggered by external event sources
  - Synchronous services poll a queue and processes data further



- Leader/Followers (pattern)
  - Threads take turn accessing the set of event sources
  - At most one thread is Leader
  - Idle threads are Followers waiting to become Leader



- Proactor (pattern)
  - Uses asynchronous I/O and message processing



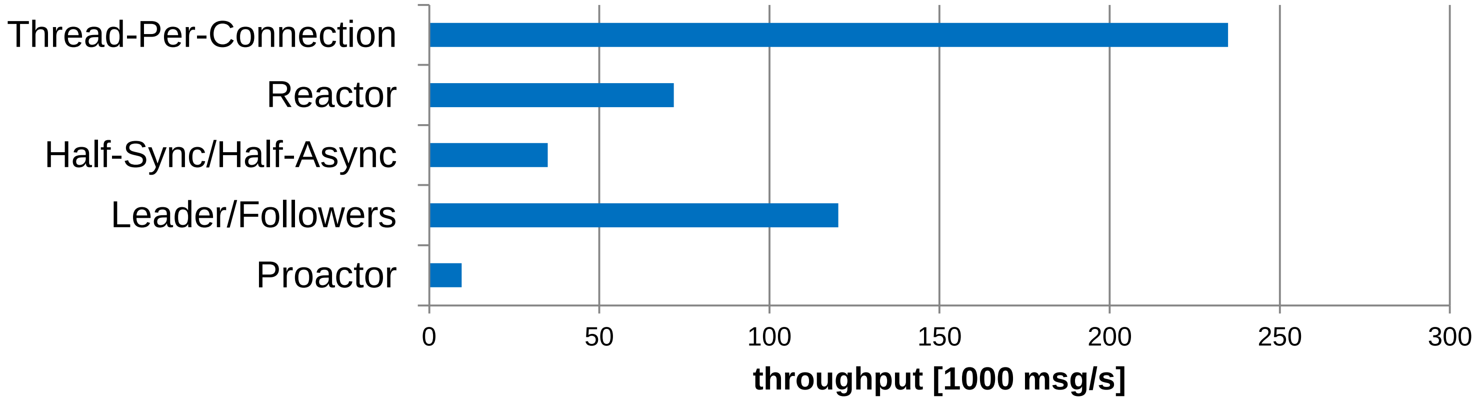
Evaluation Settings

- Evaluation system
  - Cavium Octeon Plus CN5650
    - 12 MIPS cores @ 800MHz
    - Designed for embedded telecommunication applications

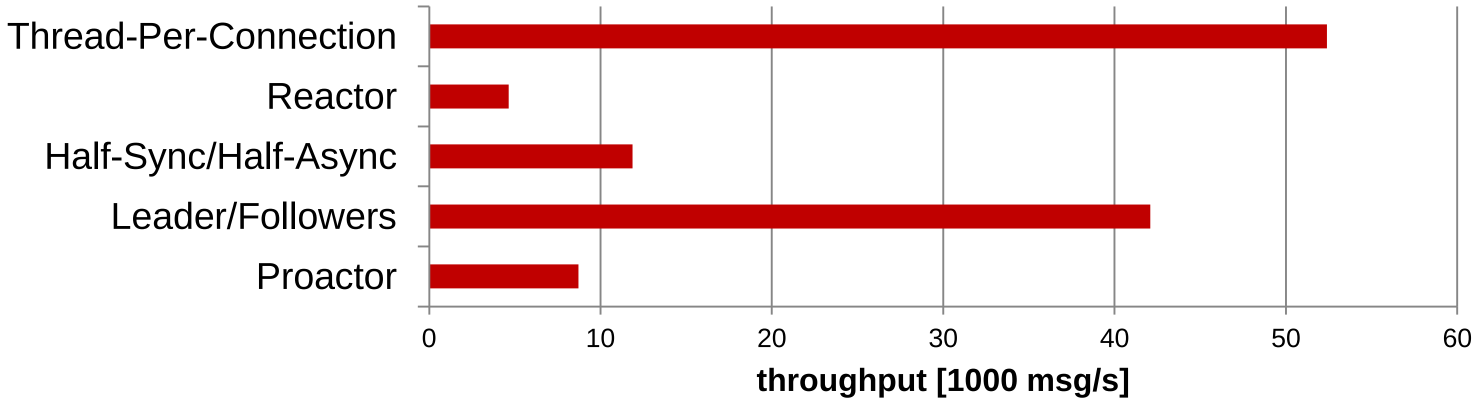
- Measurement settings
  - 128 TCP connections maintained by 2 threads
  - Messages are1 byte long
  - With 1 - 12 threads (Half-Sync/Half-Async, Leader/Followers, Proactor)
  - With 1 – 12 cores (Thread-per-Connection)
  - Additional work load per message of 0 to 200 µs

Evaluation Results
Peak performance

- No additional load per message

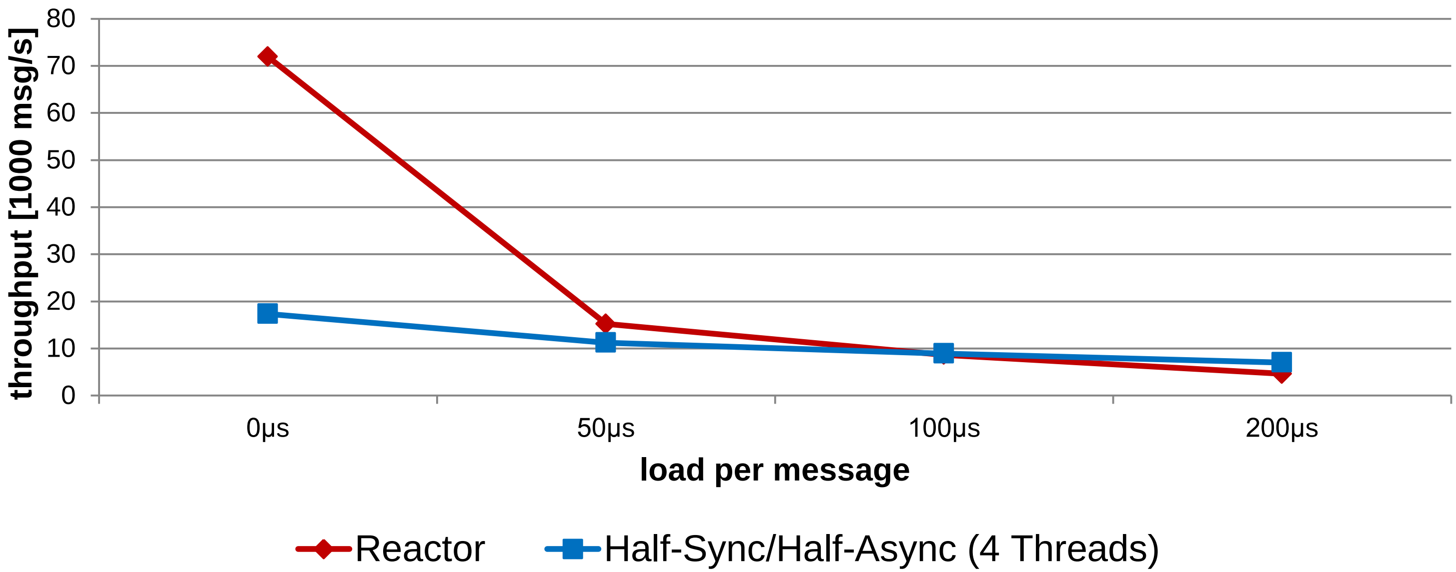


- 200µs load per message



Evaluation Results
Half-Sync/Half-Async pattern

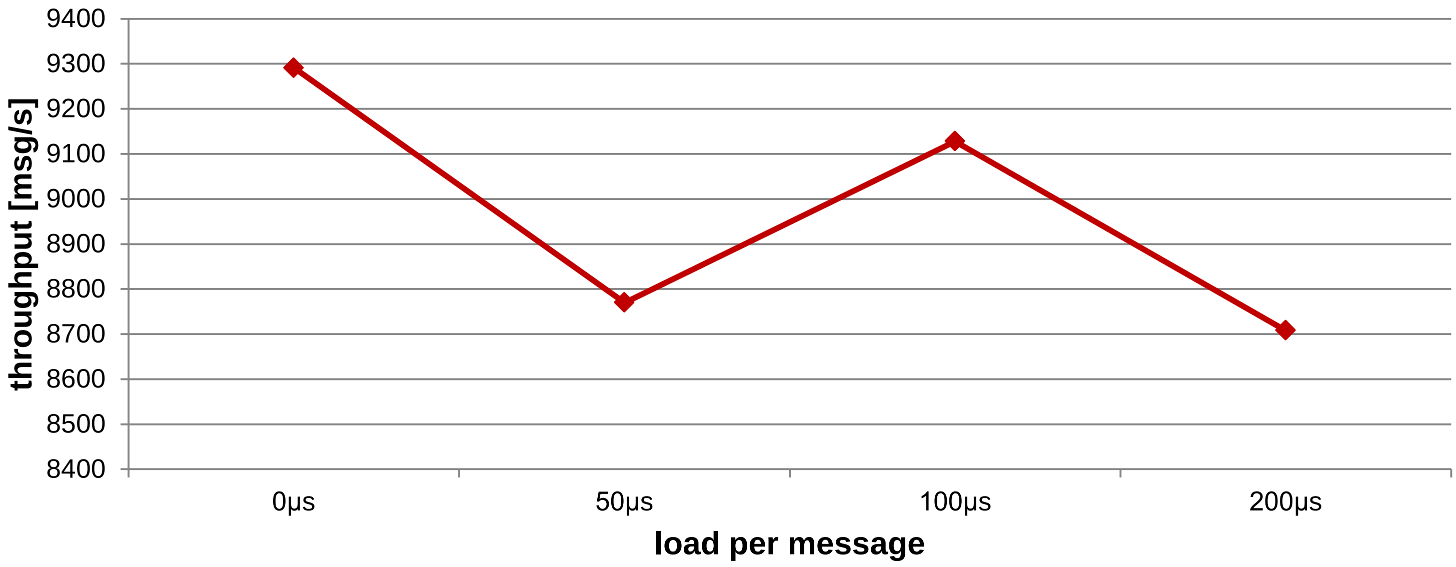
- Comparison of Half-Sync/Half-Async with 4 threads against Reactor pattern
  - Low throughput caused by asynchronous service (implemented using Reactor pattern)
  - →More frequent invocation of event de-multiplexing induces high latency



Evaluation Results
Proactor pattern

- Proactor with 2 threads
  - Increasing load does not decrease throughput → limited by I/O

→One thread created for each I/O completion handler



Lessons learnt & Future Work

- Lessons learnt so far
  - Considerable performance differences between patterns
  - Distribute event de-multiplexing over multiple threads
  - Distribute event sources as well
    →Avoiding the bottleneck of a single thread
  - The wrong multi-threading architecture is worse than none
    →Reactor partly performed better than Half-Sync/Half-Async

- Future Work
  - Include:
    - Shared resources,
    - Connection establishment / termination overhead etc.
  - Use alternative I/O primitives (epoll) and mechanics (POSIX signals)
  - Expand measurement with CPU utilization, cache usage etc.