The Design and Implementation of a Scalable, Concurrent Virtual Machine

Robert Virding
Principle Language Expert at Erlang Solutions Ltd.
Concurrency vs. Parallelism

- Parallelism is what the system provides
- Concurrency is a property of the problem/solution

(as good a definition as any)
• Will look at
  - Memory management + GC
  - Concurrency primitives
  - Communication

• Will not look at
  - Code generation
  - Virtual code vs. native code
Erlang is COP

Concurrency Oriented Programming
Problem domain/Erlang properties

- Light-weight concurrency
- Asynchronous message passing
- Process isolation
- Error handling
- Soft real-time
- Continuous evolution of system
- Immutable data
Communication

- Asynchronous message passing
- Each process has a single message queue
- Selective receive
- Suitable as a building block
The BEAM

The main Erlang implementation
Will cover:
  • Processes/scheduling
  • Memory management
  • Communication
  • Error handling

(Bogdan’s Erlang Abstract Machine, now Björn’s)
Processes and scheduling

- Want light-weight concurrency
- Must have processes (isolation)

➡ Implement processes ourselves ("green" processes)

- They are really light-weight
- Complete control of scheduling
- Not a general machine
Processes and scheduling

- Pre-emptive scheduling
  - Scheduled as cooperating coroutines
- One run-queue per thread
- Multi-core/multi-thread ➔ multi-queues
  - Process stealing
- No really viable alternatives
Memory management/GC

• Soft real-time requires unobtrusive GC
• Must ensure process isolation

• Stop-the-world collector
  vs.
  interactive/concurrent collector
  - independent of parallelism
  - independent of collector algorithm
Memory management/GC

• Three main algorithms
  - reference counting
  - mark-sweep
  - copying

• Generational collectors
  - most objects die young
  - Big Win™ in most cases
  - algorithm independent
Memory management/GC

Alternatives:

- Shared heap
- Separate process heaps
- Hybrid
Memory management/GC

Shared heap

- Reduces copying of data
  - Messages shared
- Complicates GC
  - must use incremental/concurrent collector
- Less well-suited to multi-core
- Complicates handling of processes
Memory management/GC

Separate process heaps

- Increases copying of data
  - messages copied
- Simplifies GC
  - can use stop-the-world per process collector
- Fits well on multi-core
- Scales well
- Simplifies handling of processes
Memory management/GC

Hybrid heaps

- Reduces copying of data
  - messages on shared heap
- May in part be determined at compile-time
- Complicates GC
- Less well-suited on multi-core
Memory management/GC

Erlang does **NOT** require separate heaps and message copying!

- Immutable data means we can safely share
Memory management/GC

Erlang BEAM uses very restricted hybrid heaps

- Mainly separate heaps
  - most of the benefits of separate heaps
- Some shared data
  - get some benefits of shared data
  - without too many of the complications
- Generational collector per heap
Memory management/GC

JVM uses shared heap

- Uses generational collectors
  - young heap(s), tenured heap

- Has multiple collectors
  - Stop-the-world collectors, serial and throughput
  - Concurrent collector

- Cannot guarantee soft real-time behaviour
Concurrency communication alternatives

- Message passing
  - Suits multi-core
    - needs only limited synchronisation
  - Scales well
Concurrency communication alternatives

- **Shared memory communication**
  - synchronisation woes - locking, mutexes ...
  - scalability?
  - error handling
- **STM (Software Transactional Memory)**
  - wrap access to shared memory in a transaction
  - only allows undoable operations
  - scalability?
Concurrency communication alternatives

• Go concurrency motto
  - "Do not communicate by sharing memory; instead, share memory by communicating"
Erjang - Erlang on the JVM

A real Erlang on the JVM
Will cover:
- Why the JVM?
- Concurrency/scheduling
- Memory management/GC
Erjang - Why the JVM?

- Erjang is different from the BEAM
  - it has different properties
- Very mature VM
- Wide-spread VM
  - Erlang can run in MANY more places
- Improved interface between Erlang and other JVM languages
- Access to libraries

(Kresten wanted to learn Erlang)
Erjang - Concurrency / scheduling

- Uses Kilim to provide concurrency and message passing
  - ultra light-weight threads
  - message passing with shared messages
- Cooperative coroutines
  - pre-emptive scheduling
- One run-queue per thread
- Multi-core/multi-thread ➔ multi-queues
Erjang - Memory management/GC

- Uses standard JVM memory management/GC

➡ Cannot guarantee soft real-time behaviour (yet?)
Thank you

Robert Virding

robert.virding@erlang-solutions.com

@rvirding