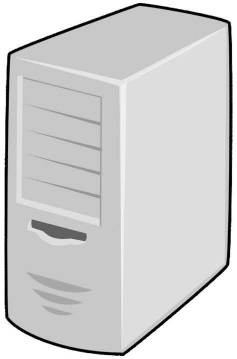
Comparing Recent Shared Memory Concurrency Models

Ivan Valkov
Supervisors: Professor Phil Trinder, Dr Natalia Chechina

Introduction

- Servers must deal with many concurrent requests
- The programming language used is important
- It must support massive concurrency
- Erlang, Go, and Scala with Akka are compared in this context



Selecting Languages to Compare

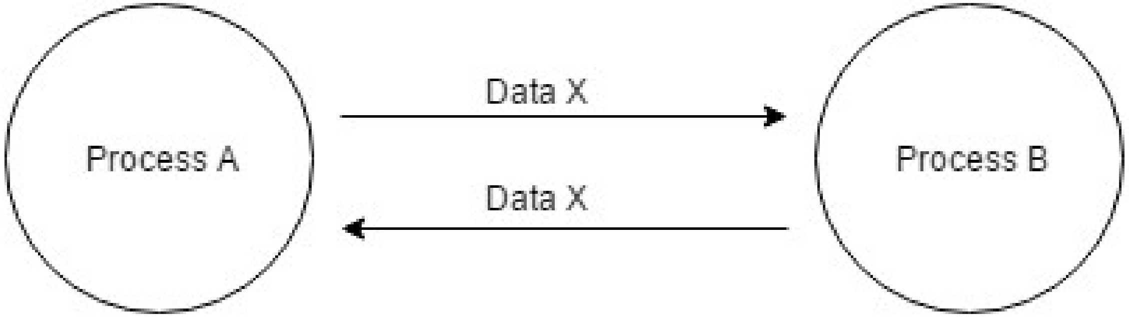
	Computation			*Coordination*			**Compilation**	*Popularity*	
Language	**Model**	**Typing**	**Abstraction**	**Model**	**Abstraction**	**Determinism**	**Runtime env**	**TIOBE Sep 2016**	**The Red Monk June 2016**
Java	OO	Strong Static	High	Explicit	High	N	JVM	1	2
C + OpenMP	Procedural	Weak Static	Low	Annotation	High	N	Native	2	9
C + PThreads	Procedural	Weak Static	Low	Explicit	Low	N	Native	2	9
Haskell	Pure Functional	Strong Static	High	Eval Strat	High	Y	Native/ GHCi	40	16
Erlang	Functional	Strong Dynamic	High	Actors	High	N	Erlang VM	42	26
Scala + AKKA	Functional/OO	Strong Static	High	Actors	High	N	JVM	32	14
Go	Procedural	Strong Static	High	CSP	High	N	Native	19	15
Elixir	Functional	Strong Dynamic	High	Actors	High	N	Erlang VM	50+	~40
Clojure	Functional	Strong Dynamic	High	STM/Agents	High	N	JVM	50	20
Rust	Procedural	Strong Static	High	Explicit	High	N	Native	45	~40
F#	Functional/ OO	Strong Static	High	Explicit	High	N	Native	29	~40
C#	OO	Strong Static	High	Explicit	High	N	Native	4	5

Selected Languages

	Computation			***Coordination***			**Compilation**	***Popularity***	
Language	**Model**	**Typing**	**Abstraction**	**Model**	**Abstraction**	**Determinism**	**Runtime env**	**TIOBE Sep 2016**	**The Red Monk June 2016**
Erlang	Functional	Strong Dynamic	High	Actors	High	N	Erlang VM	42	26
Scala + AKKA	Functional/OO	Strong Static	High	Actors	High	N	JVM	32	14
Go	Procedural	Strong Static	High	CSP	High	N	Native	19	15

Benchmarks (1/3) – Process Communication Latency

- Design based on Intel’s MPI Benchmark PingPing
- Measures time to exchange data between two processes
- Important in systems with high number of messages, such as servers

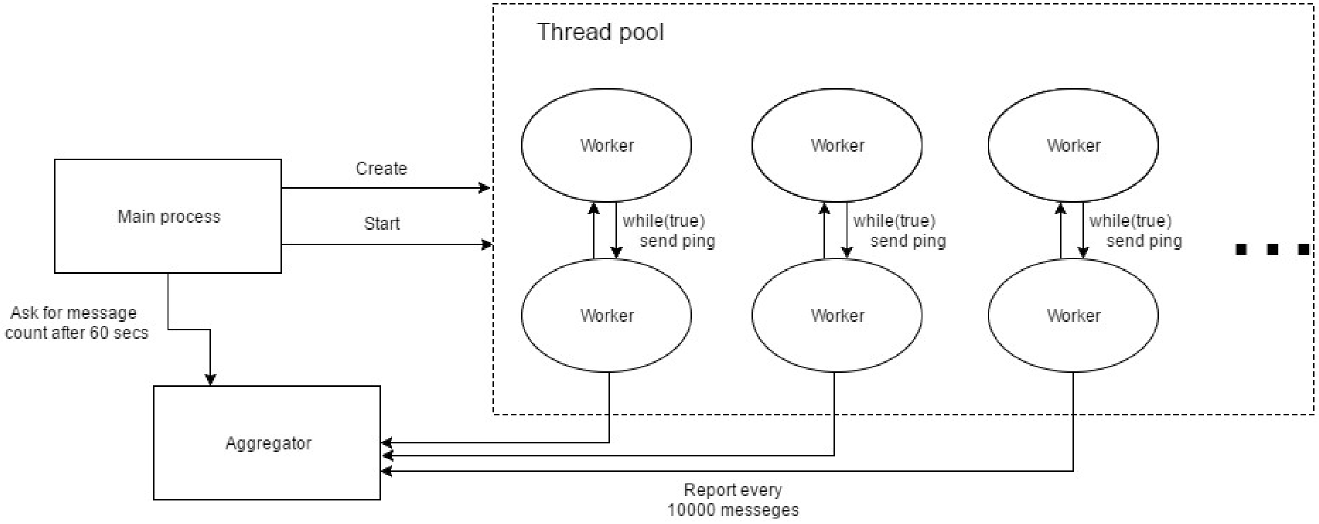


Benchmarks (2/3) – Process Creation Latency

- Measures time to spawn ***N*** processes
- Given a big enough ***N***, the maximum number processes is found
- Important in systems where:
  - a lot of processes are constantly being spawned
  - a lot of processes need to be supported

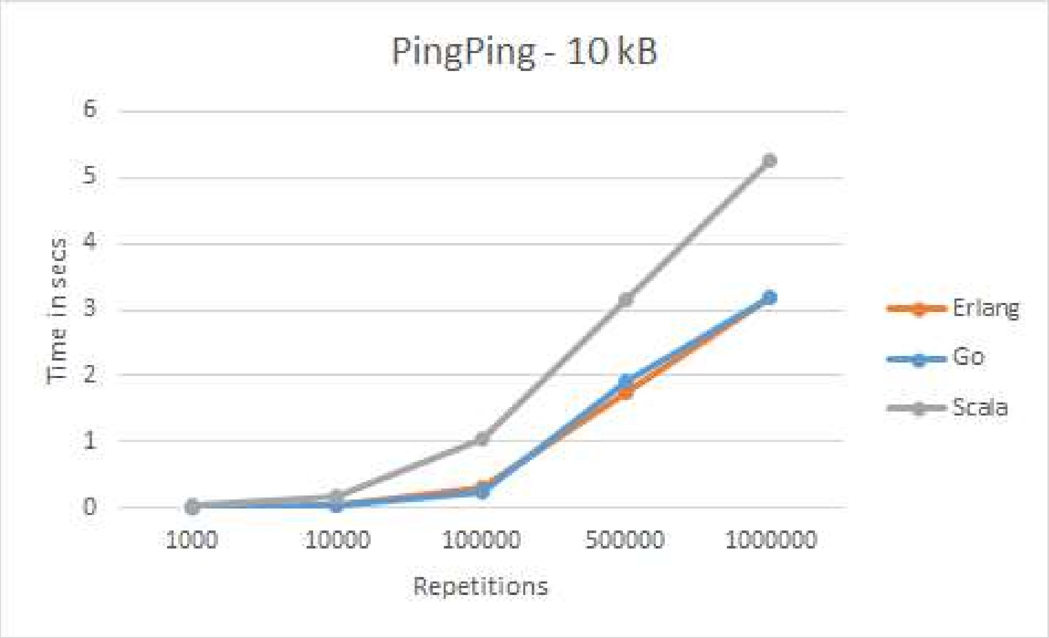
Benchmarks (3/3) – Concurrent Processes Throughput

- Examines a closer to real world scenario
- Measures throughput in a system of multiple process pairs exchanging messages
- Important in systems utilizing high level of concurrency



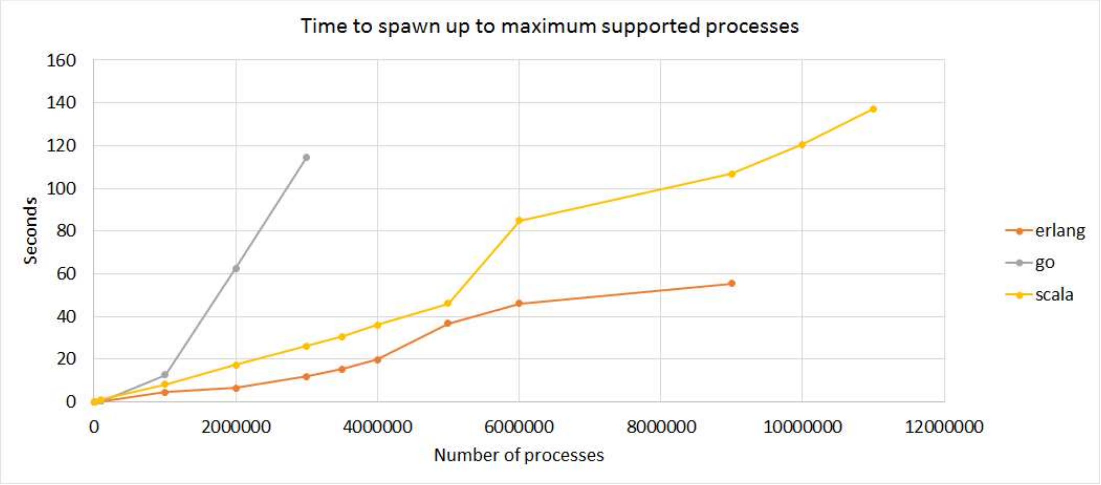
Process Communication Latency Results

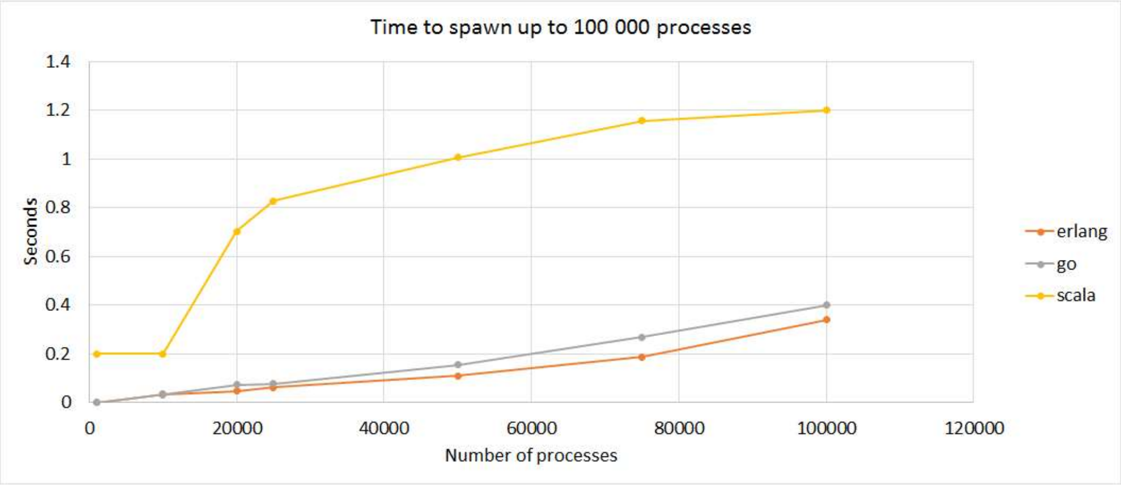
- Ran on Windows 10, 2 cores (Intel Core i5-3230M CPU 2.60GHz), 8Gb RAM
- Erlang and Go perform similarly
- Scala with Akka trails behind



Process Creation Latency Results

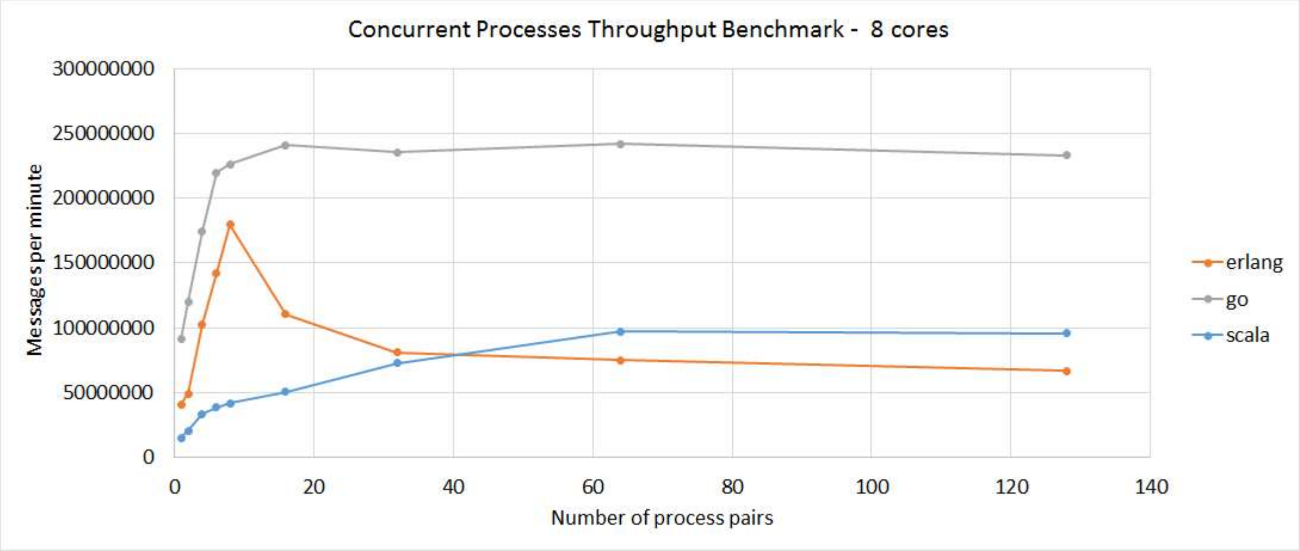
- Ran on Windows 10, 2 cores (Intel Core i5-3230M CPU 2.60GHz), 8Gb RAM
- Scala with Akka spawns the most processes – 11 million
- Erlang and Go have faster spawn time of up to 100,000 processes





Concurrent Processes Throughput Results

- Ran on Scientific Linux 6, 16 cores (2 Intel Xeon E5-2640 2GHz), 64Gb RAM
- Go – best performance; quickly reaches peak and maintains it
- Erlang – quickly reaches peak, but a sudden decay in performance follows
- Scala with Akka – slowly but steadily improves performance with introduction of more processes



Conclusion

If you need:

- Minimising of message latency – Erlang/Go
- Support of many dormant processes – Scala with Akka
- Fast creation time of up to 100,000 processes – Erlang/Go
- High level of concurrency – Go