Università
della
Svizzera
italiana

Faculty
of Informatics

Profiling Actor Utilization and Communication in Akka

Andrea Rosà*, Lydia Y. Chenˆ, and Walter Binder*

*Università della Svizzera italiana (USI), Faculty of Informatics, Lugano, Switzerland
ˆIBM Research Lab Zurich, Rüschlikon, Switzerland

Erlang 2016
September 23rd, 2016
Nara, Japan

Introduction

- In this talk, we present a **profiler** for Akka actors
  - Based on **bytecode instrumentation**
  - Centered on:
    - actor **utilization**
    - **communication** between actors

Metrics

- Target actor usages:
  - Computing workers
    - Executed computations
    - Initialization cost

    } Utilization
  - Communication endpoints
    - Messages sent
    - Messages received

    } Communication
- Akka profilers do not focus on utilization or communication
  - Profiled metrics: mailbox size, time in mailbox, time to process messages, errors, dispatchers, …

Profiler

- Relies on the DiSL bytecode instrumentation framework [1]
  - Guarantees full bytecode coverage
- Instrumentation:
  - Actors → Subtypes of `akka.actor.Actor`
    - Constructors
    - Send methods → `tell [!] / ask [?]`
    - Receive methods → Receive `PartialFunction`
  - Thread-local bytecode counters
  - Basic blocks (for maintaining counters)

[1] L. Marek, A. Villazon, Y. Zheng, D. Ansaloni, W. Binder, and Z. Qi. DiSL: A Domain-specific Language for Bytecode Instrumentation. In AOSD, pages 239–250, 2012.

Evaluation

- Use cases:
  1. Actor utilization (Savina)
  2. Load balancing (Signal/Collect)
  3. Communication (Spark, Flink)

Actor utilization

- Goal:
  - Analyze the effectiveness of parallelism in an application using only actors to obtain concurrency
- Target application:
  - Savina benchmark suite [2]
    - 30 benchmarks
    - 10 different actor libraries for the JVM
    - Uses only actors to obtain concurrency

[2] S. M. Imam and V. Sarkar. Savina - An Actor Benchmark Suite: Enabling Empirical Evaluation of Actor Libraries. In AGERE!, pages 67–80, 2014.

Actor utilization

Low utilization (U < 10)

Benchmark	Actors		Messages		Utilization				
	#	# types	#	# types	AVG	STD	20th perc.	50th perc.	80th perc.
barber	5007	7	41474	10	304	14844	4	4	4
bitonicsort	190525	16	2674789	8	12	127	6	6	7
count	6	5	1000008	7	150864	292090	0	315	341271
facloc	1370	5	743792	9	253	6314	2	4	21
fib	150052	4	450149	6	285	915	4	22	289
filterbank	66	14	1419465	11	20819	114765	5	580	3784
fjcreate	40004	4	80003	5	3	3	3	3	3
pingpong	6	5	120006	10	28394	45128	0	321	77835
recmatmul	25	5	1818	8	4969990	10166347	4	5	11649055

- 20% of actors are little utilized in 9 benchmarks
- 50% of actors are little utilized in 5 benchmarks
- 80% of actors are little utilized in 3 benchmarks
- Number of actors spawned is high
- Number of messages is high

Actor utilization

Low utilization (U < 10)

Benchmark	Actors		Messages		Utilization				
	#	# types	#	# types	AVG	STD	20th perc.	50th perc.	80th perc.
barber	5007	7	41474	10	304	14844	4	4	4
bitonicsort	190525	16	2674789	8	12	127	6	6	7
count	6	5	1000008	7	150864	292090	0	315	341271
facloc	1370	5	743792	9	253	6314	2	4	21
fib	150052	4	450149	6	285	915	4	22	289
filterbank	66	14	1419465	11	20819	114765	5	580	3784
fjcreate	40004	4	80003	5	3	3	3	3	3
pingpong	6	5	120006	10	28394	45128	0	321	77835
recmatmul	25	5	1818	8	4969990	10166347	4	5	11649055

- Possible optimizations:
  - Remove some actors
  - Redesign assignment of work to actors

Actor utilization

High utilization (U > 100000)

Benchmark	Actors		Messages		Utilization				
	#	# types	#	# types	AVG	STD	20th perc.	50th perc.	80th perc.
bndbuffer	85	6	160204	10	700944	222883	757762	769162	783645
count	6	5	1000008	7	150864	292090	0	315	341271
nqueenk	25	5	29140	9	1060159	542017	615780	1303146	1368435
piprecision	25	5	8673	9	1858180	949326	1105397	2309469	2358476
recmatmul	25	5	1818	8	4969990	10166347	4	5	11649055
sieve	15	5	91343	8	145413	152496	315	96522	303587
uct	199977	5	879898	13	572591	95530	491944	573138	651467

- 7 benchmarks show high average actor utilization
- Possible optimization (depending on available resources):
    - Add more actors

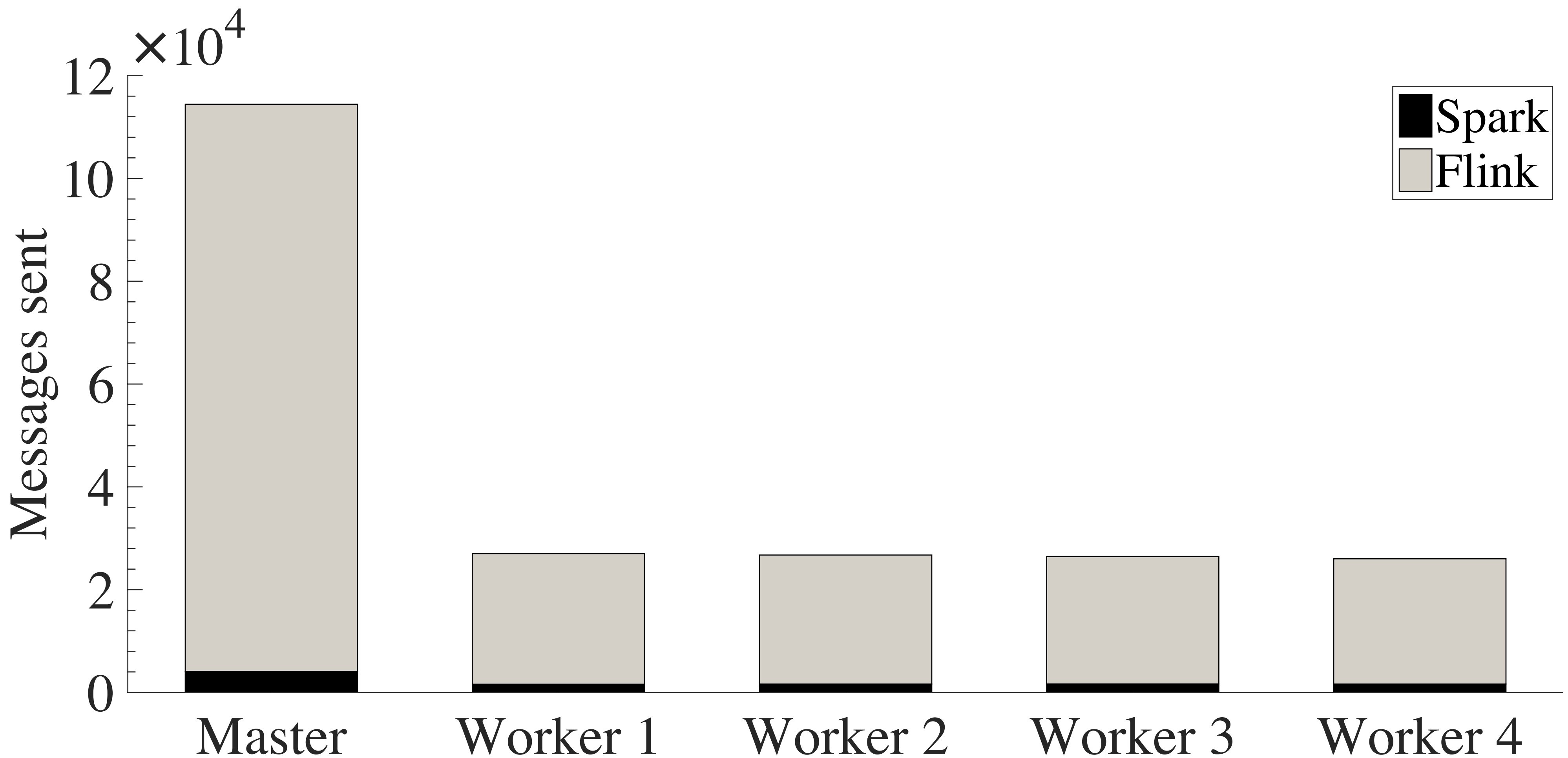
Communication

- Goal:
  - Analyze communication between workers in distributed computing frameworks
- Target frameworks:
  - Apache Spark [3] and Apache Flink [4]
    - Computing frameworks for big-data, machine learning, graphs, streaming, etc.
    - Actors handle communication between master and workers (not computations)

[3] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In *NSDI*, pages 1–14, 2012.
[4] Apache Flink. https://flink.apache.org.

Communication

Kmeans on 10M points



- Great difference between Spark and Flink
  - Worker: ~1.6k (Spark), ~25k (Flink)
  - Master: ~4.1 k (Spark), ~110k (Flink)

Communication

- Other analyses (see paper for more information):
  - Correlation between messages sent and data size
    - Linear in Spark, unclear in Flink
  - Execution time
    - Spark always faster than Flink

Conclusions

- We presented a profiler for Akka actors:
  - Relies on bytecode instrumentation
  - Centered on actor utilization and on the communication between actors
  - Useful for many user needs

Discussion

- Limitation of bytecode count:
  - Cannot track code without bytecode representation (e.g., native methods, JVM internal functions)
  - Work of different complexity is represented with the same unit
  - Susceptible to on-the-fly optimizations
- Bytecode count vs. machine instruction count
  - Accuracy vs. portability

Discussion

- Complementary metrics:
  - Machine instruction count / CPU time
    - Are actors always busy in carrying on computations?
    - However, subjected from instrumentation perturbation, unlike actor utilization
- Expand analysis on use cases
  - Flink: root causes of inefficient communication?

Thank you for the attention

- Contact details:

Andrea Rosà
andrea.rosa@usi.ch
http://www.inf.usi.ch/phd/rosaa

Backup slides

Our profiler in Erlang

- Collect metrics on Erlang processes:
  - Initialization cost: not much useful
    - Process initialization cannot be customized
  - Executed computations: requires bytecode instrumentation
    - No available bytecode instrumentation frameworks for Erlang
  - Messages sent/received: use Erlang tracing facility
    - Some tools already track them (e.g., Percept2)