Università
della
Svizzera
italiana

**Faculty
of Informatics**

Actor Profiling on the JVM

Andrea Rosà
PhD candidate
Dynamic Analysis Group
Università della Svizzera italiana (USI)
Lugano, Switzerland

VMM 2016
September 1st, 2016

Overview

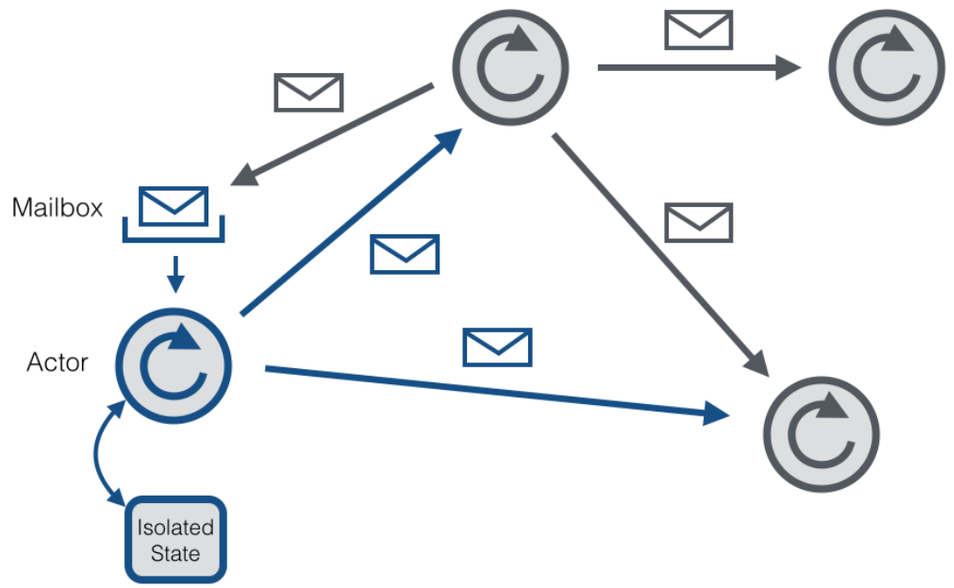
- In this talk, we will discuss and answer the following questions:
  - Why is it necessary to **profile actors**?
  - Which **metrics** should we focus on when profiling?
  - **How** to profile actors?
  - Why is it **useful**?

- Focus of this talk:
  - Java Virtual Machine
  - **Akka** actor library

Background

Actors

- Atomic entities communicating via message exchange
- Continuously listen for incoming messages
- Execute work in response to a message



- Properties:
  - Cannot share state
  - Communicate only via asynchronous messages
  - Opaque addressing

Actors in practice

- Applications:
  - Computing workers (e.g., Signal/Collect)
  - Communication endpoints (e.g., Apache Spark, Apache Flink)
  - Used in several commercial products (e.g., Amazon's SimpleDB, Facebook Chat System, WhatsApp)
  - …
- In general, there is a shortage of profilers for actors
  - Not much effective when analyzing actors

Actor profiling

Metrics

- Collect most useful metrics according to how actors are used
  - Executed computations
  - Initialization cost
  - Actor utilization

  } Useful for computing workers

  - Messages sent
  - Messages received

  } Useful for communication endpoints

- Focus of other profilers: mailbox size, time in mailbox, errors, dispatchers, …

Akka actor 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)

Challenges

```
1  actor A {...}
2  actor B {
3      B() {...}                    // constructor of B
4  }
5  actor C subtype of B {
6      C() {                        // constructor of C
7          B();                     // account to current actor
8          A a = new A();           // account to new actor
9          ...
10     }
11  }
```

- Multiple constructors
- Nested actor creation
- Multiple sending methods
- Multiple receiving methods
- …

} Incorrect accounting

Use cases

Actor utilization

- Goal:
  - Analyze the effectiveness of parallelism in an application using only actors to obtain concurrency
- Why profiling actors?
  - Concurrency depends on actors, not on other constructs (`Runnable`, `Future`, etc.)

Actor utilization

- Target application:
  - Savina benchmark suite [1]
    - 30 benchmarks
    - 10 different actor libraries for the JVM
    - Uses only actors to obtain concurrency

[1] 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

- 2 benchmarks utilize actors scarcely on average
- 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

Load balancing

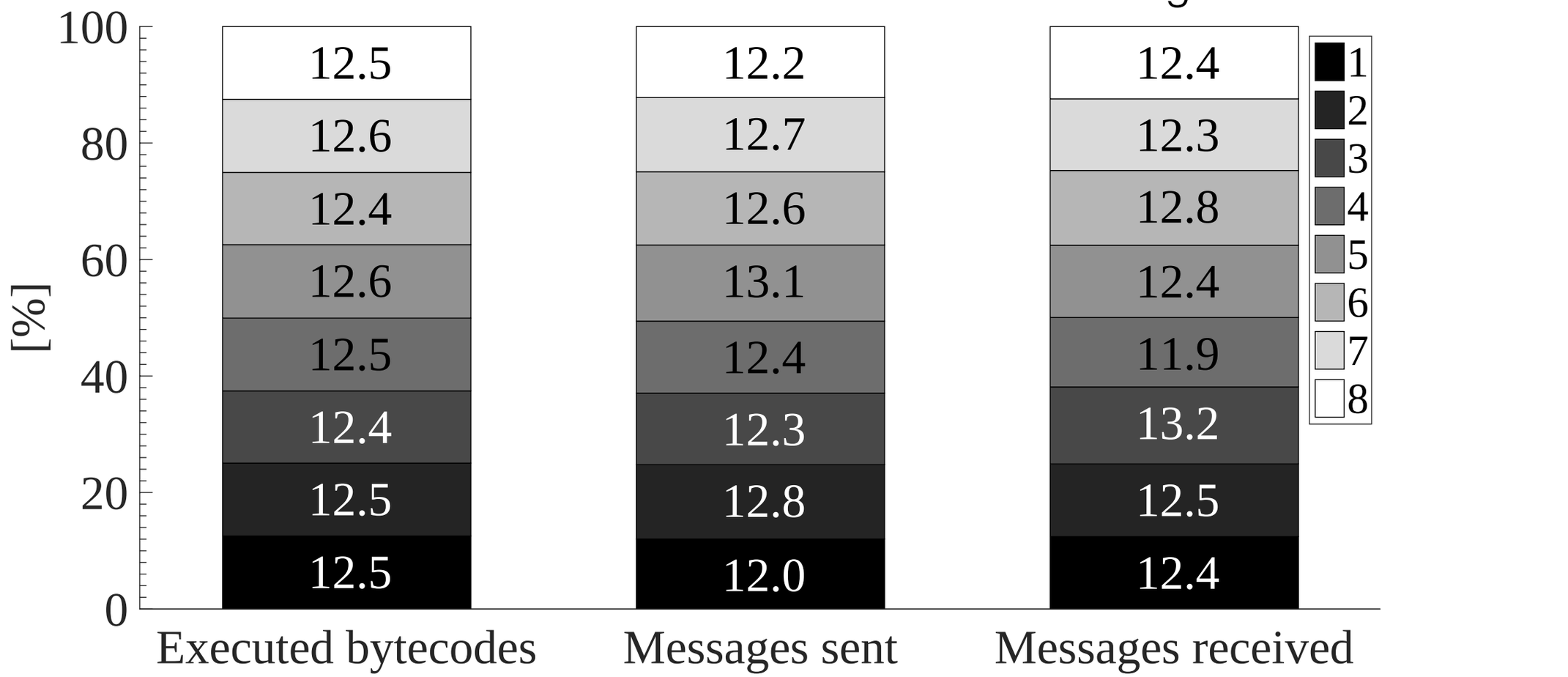
- Goal:
  - Understand if load is well balanced among workers in parallel processing frameworks
- Why profiling actors?
  - Actors are the key entities carrying on computations

Load balancing

- Target framework:
  - Signal/Collect [2]
    - Framework for graph computations
    - Uses Akka actors as computing workers
    - Vertices = computational entities
    - Edges = messages used by vertices to interact

[2] P. Stutz, A. Bernstein, and W. Cohen. Signal/Collect: Graph Algorithms for the (Semantic) Web. In *ISWC*, pages 764–780, 2010.

Load balancing



- Computing work is balanced
- Signal distribution is balanced

Communication

- Goal:
  - Analyze communication between workers in distributed computing frameworks
- Why profiling actors?
  - Communication endpoints are frequently implemented by actors

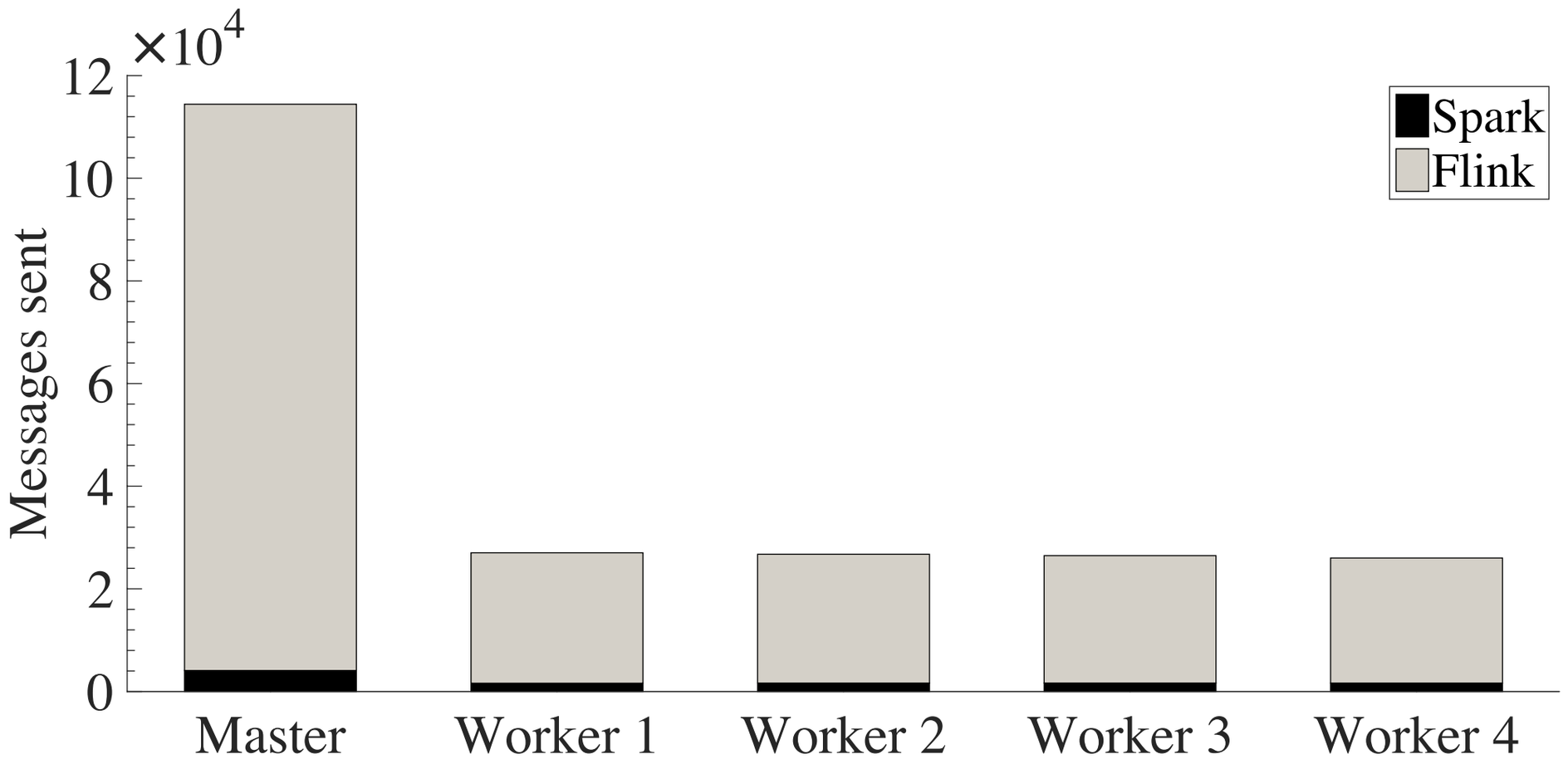
Communication

- 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.

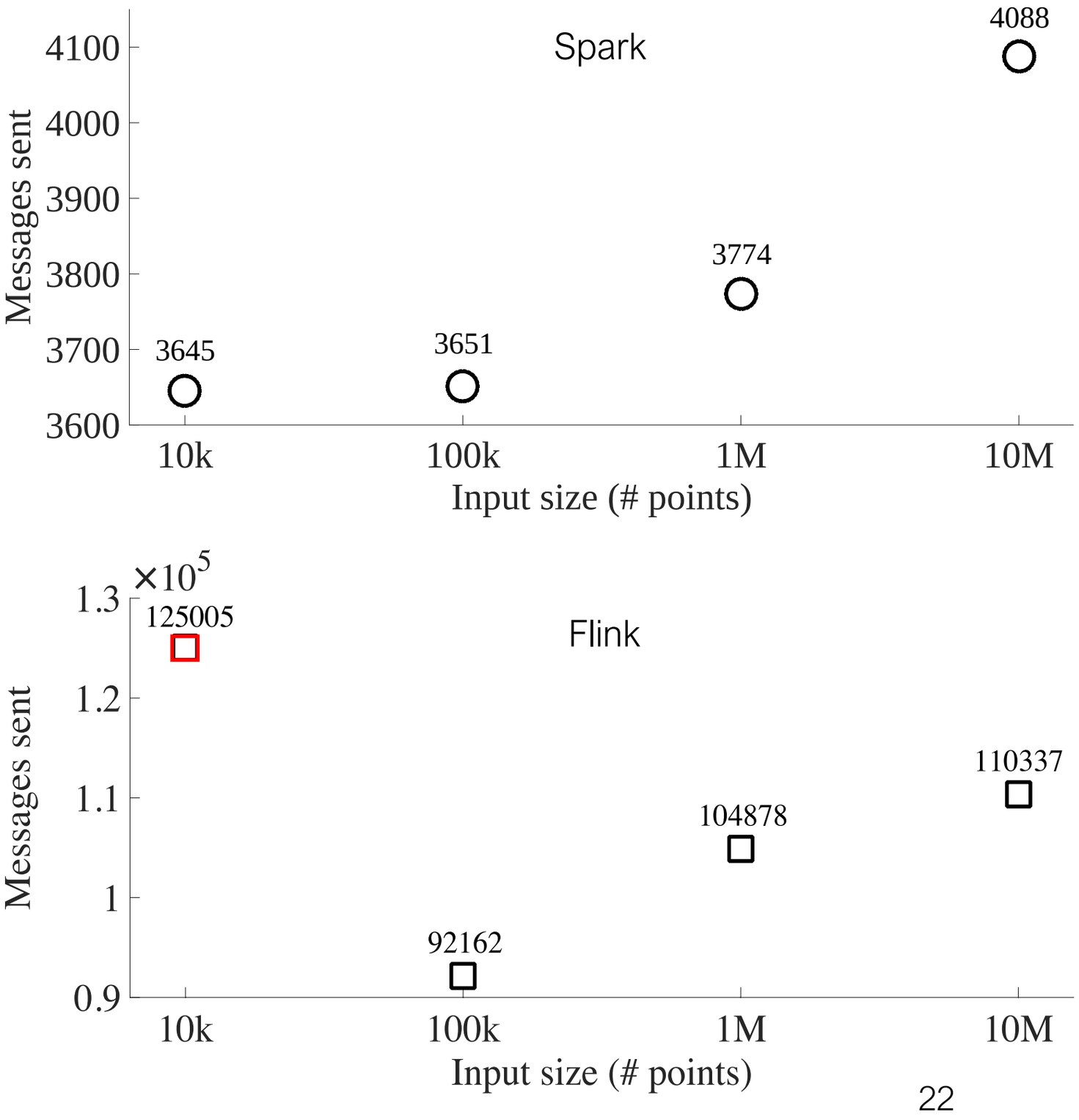
Communication

Kmeans on 10M points



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

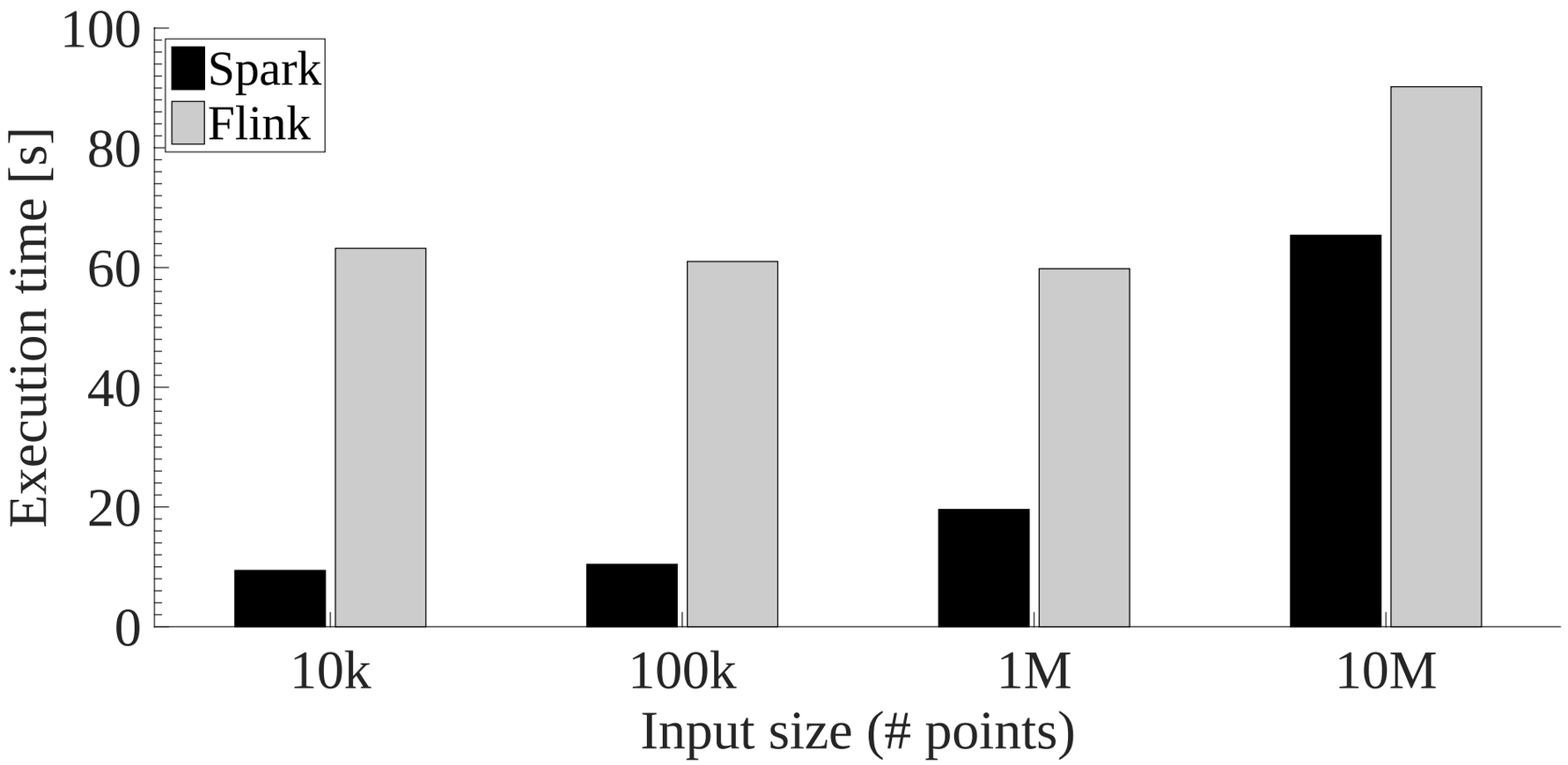
Communication



Master node only

- Exponential behavior in Spark
  - Message size is an indicator of the amount of communication
- Unclear behavior in Flink
  - More investigation is needed

Communication



- Kmeans always faster in Spark
  - Difference as high as 7x
  - Difference gets smaller with increasing data size

Conclusions and discussion

Conclusions

- Actors: many applications, few profilers
- Utilization and communication are key features of actors
  - Profiling them can shed light on actor performance
  - However, pay attention to pitfalls
- Profiling actors helps in the performance analysis of actor-based applications
  - Along several directions

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
  - Signal/Collect: load is balanced, but are actors mostly active or idle?
  - Flink: root causes of inefficient communication?

Thank you for the attention

- More information in:
  - A. Rosà, L. Y. Chen, W. Binder, "*Actor Profiling in Virtual Execution Environments*". In *GPCE'16.*

- Contact details:

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