Tracing of Large-scale Actor Systems

Michał Ciołczyk, Mariusz Wojakowski

Follow slides on:
https://tinyurl.com/lambda2018tracing

About us

Michał:

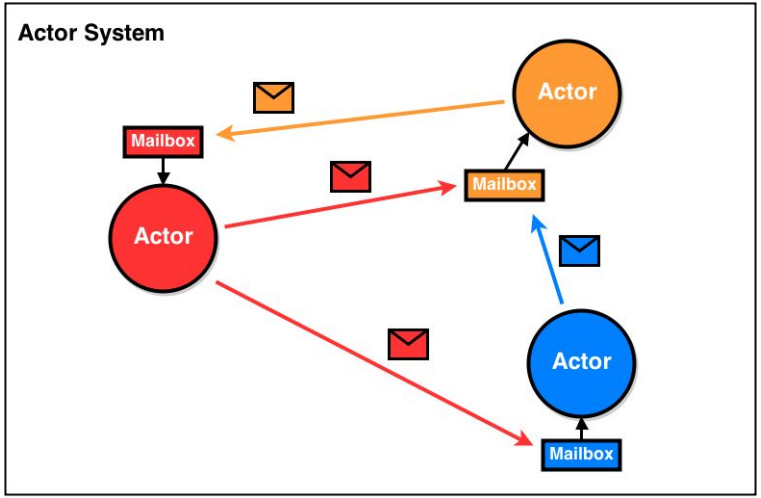
- Computer Science @ AGH
- engineer @ Evidence Prime health care decision tools
- after work: WoW player

Mariusz:

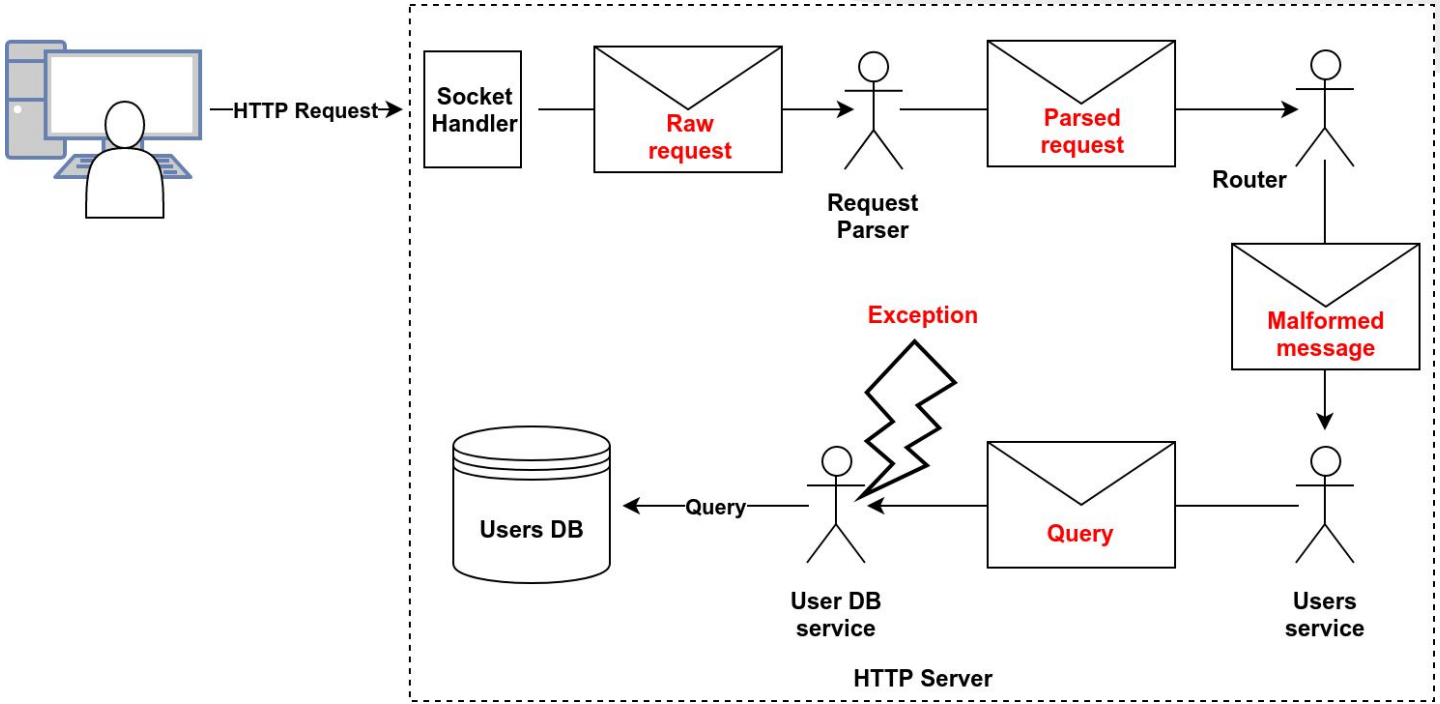
- Computer Science @ AGH
- engineer @ allegro advertisement platform
- after work: distributed systems

Actor model

- mathematical model for concurrent computation
- actor as a computation primitive
- what actors can do?
  - send message (asynchronously)
  - create new actors
  - decide how to answer on next message



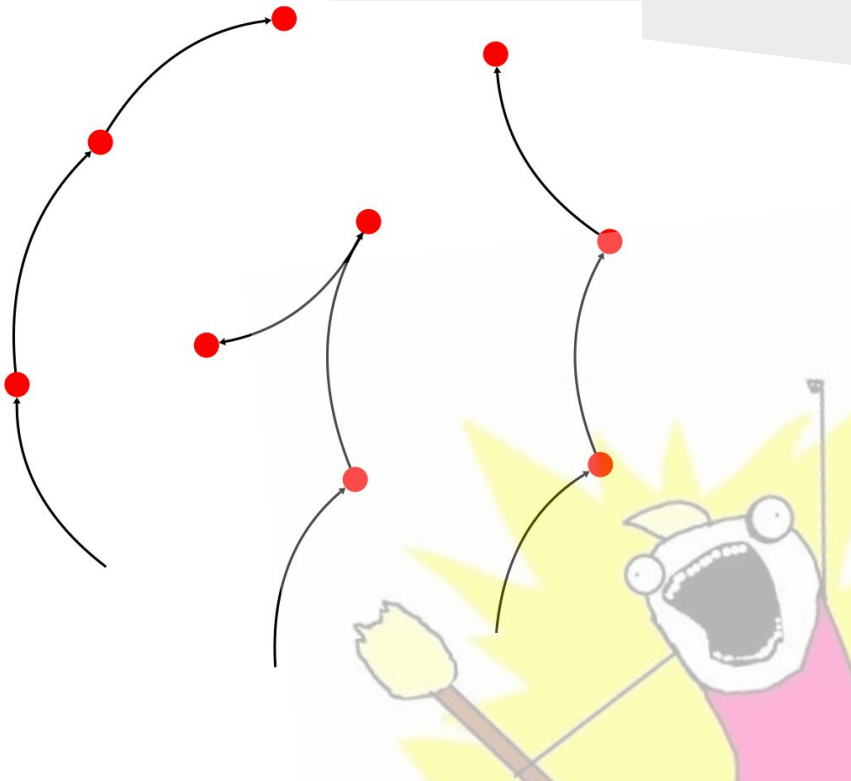
Problem



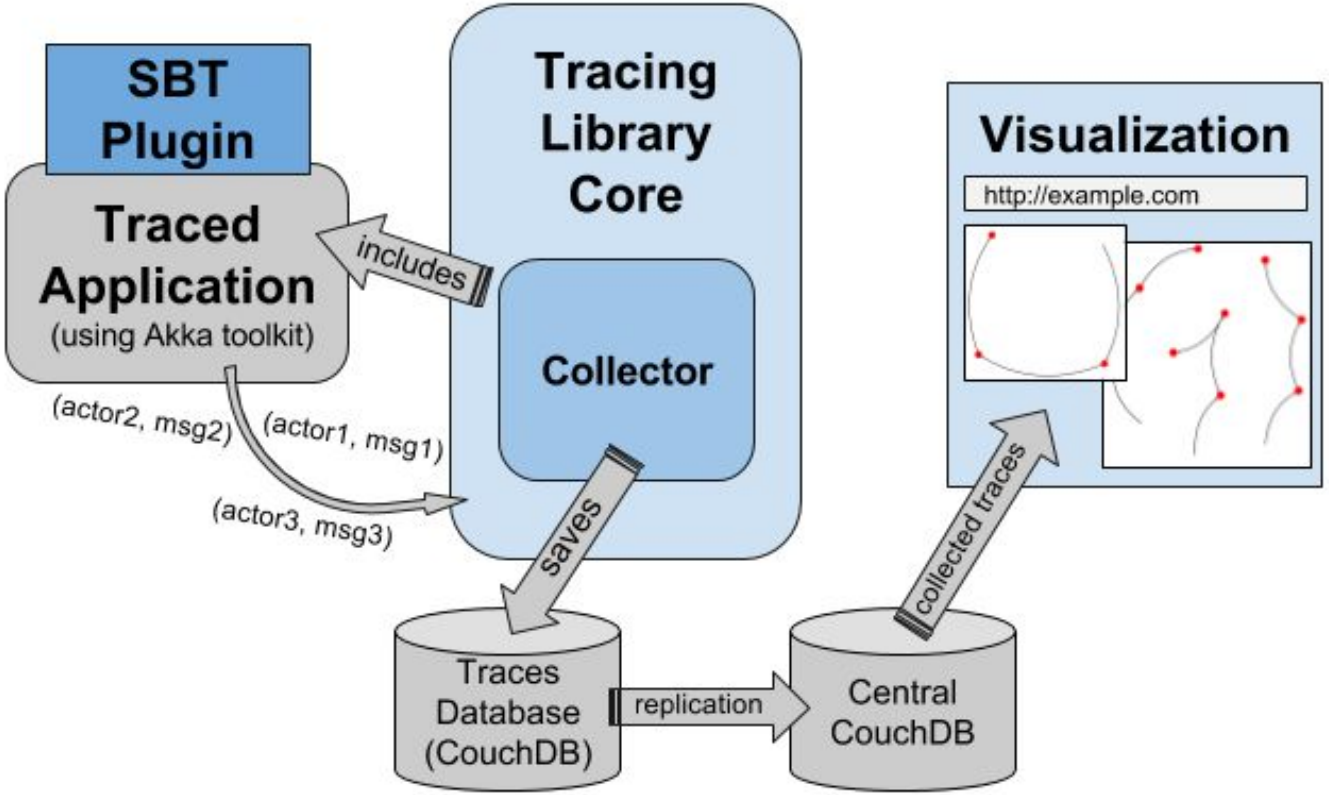
How can we detect the real cause of such errors?

Our solution

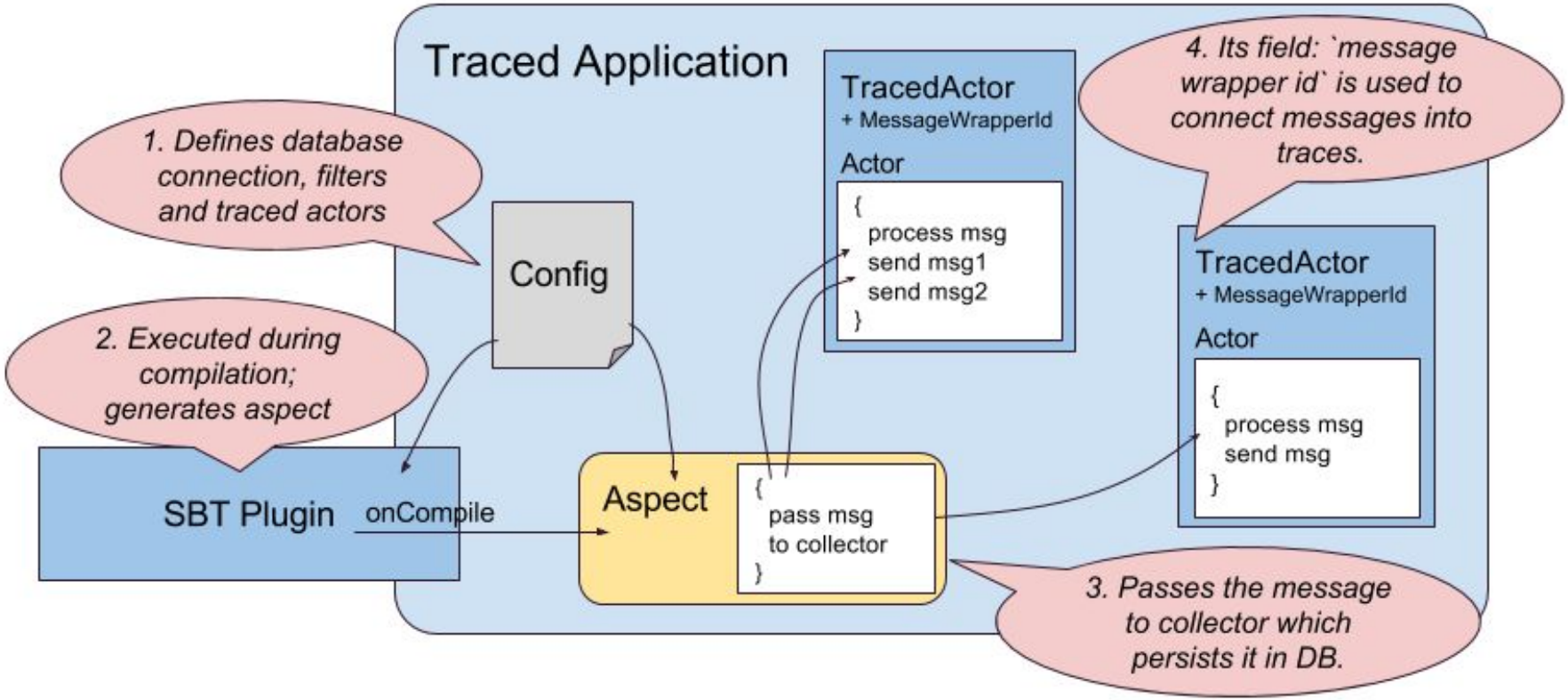
- collect traces!
- what are these?
  - vertices depict actors
  - arrows show messages passed between actors



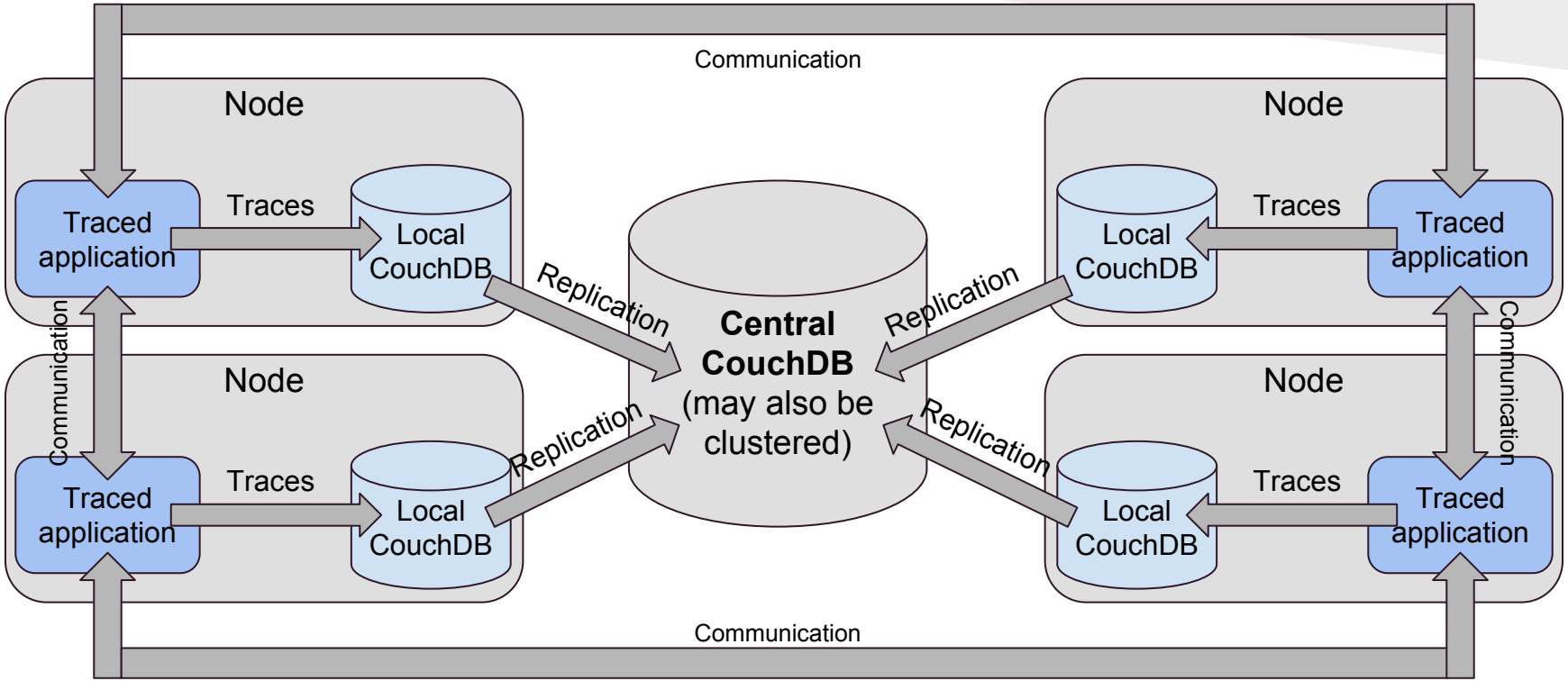
Library architecture



How it works?


Traced Application
1. Defines database connection, filters and traced actors
Config
2. Executed during compilation; generates aspect
SBT Plugin
onCompile
Aspect
{
pass msg
to collector
}
TracedActor
+ MessageWrapperId
Actor
{
process msg
send msg1
send msg2
}
4. Its field: `message wrapper id` is used to connect messages into traces.
TracedActor
+ MessageWrapperId
Actor
{
process msg
send msg
}
3. Passes the message to collector which persists it in DB.

Storage layer architecture

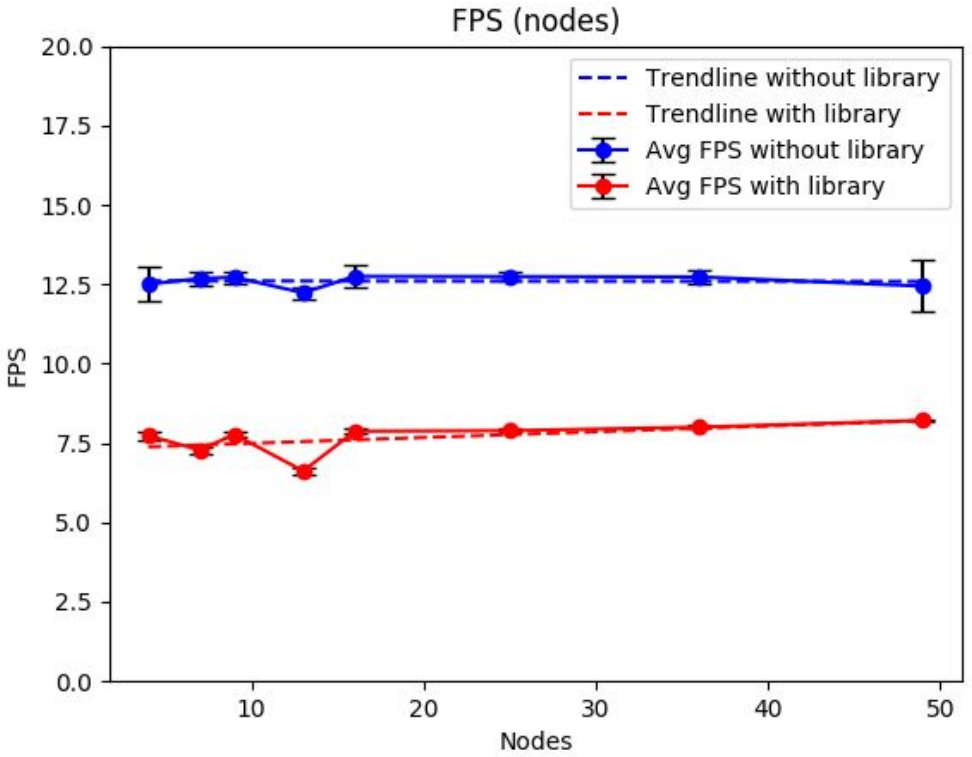


Test scenario - traffic simulation

- city divided into areas, every actor process traffic inside its own area
- communication happens only between neighbours areas - messages about incoming/outcoming cars & free space (many messages!)
- microscopic simulation, every car is simulated on its own

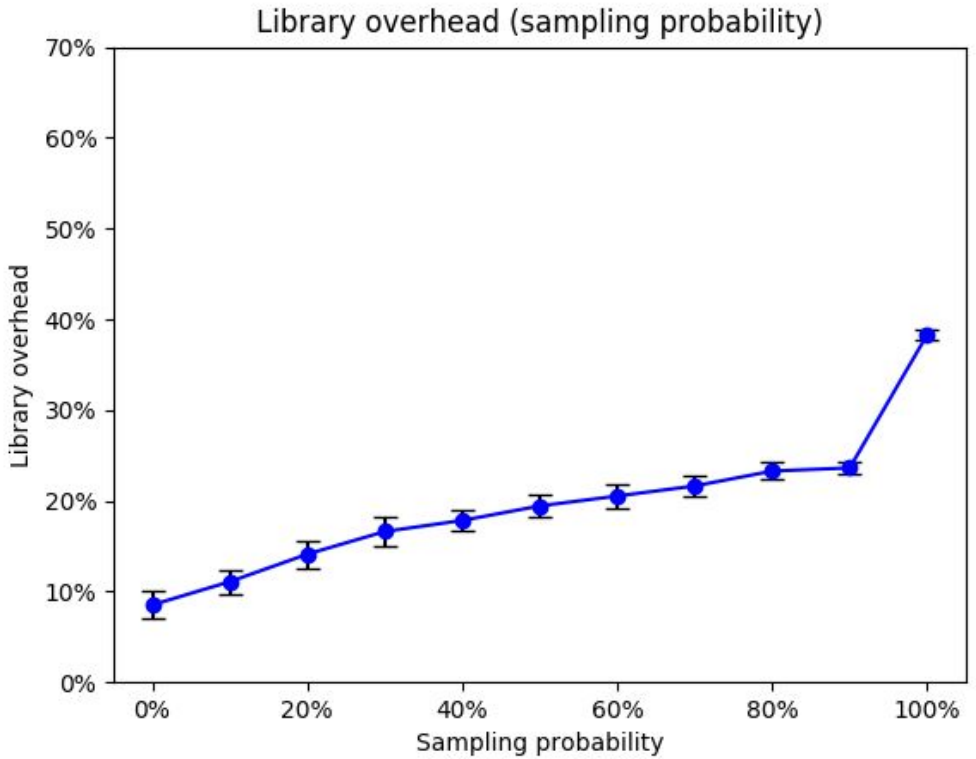
Evaluation

- first set of tests - scalability of the library
- overhead: ~39-45% decrease in FPS
- trendlines are parallel - library doesn't affect scalability of the simulation



Evaluation (sampling)

- second set of tests - scalability with respect to the number of messages
- number of messages changed via sampling
- linear scaling from 0% to 90% sampling probability
- at 100% we seem to reach some limit - we suspect it is connected with CouchDB
- Overheads measured in % of FPS loss



Summary & conclusions

- created Scala library for collecting traces in Akka toolkit
- tested performance of the tool on a real application (traffic simulation):
  - average performance loss ~39-45% in FPS
  - scalability of the solution (nodes & messages)
- proved linear scalability for up to 50 nodes
- no impact on scalability of the test simulation

Future work

future directions:

- create better visualisation tool
- gather statistics, e.g. time spent on communication, average message processing time

how can we use collected traces?

- length of paths
- vertices degree
- histogram on numerical values from messages, distribution of messages type, etc.
- tool for smarter searching through traces (data mining, machine learning)

Thank you!

For more information about **Akka Tracing Tool** visit:

https://github.com/akka-tracing-tool