# Locality-Guided Scheduling in CAF

AGERE 2017, October 23

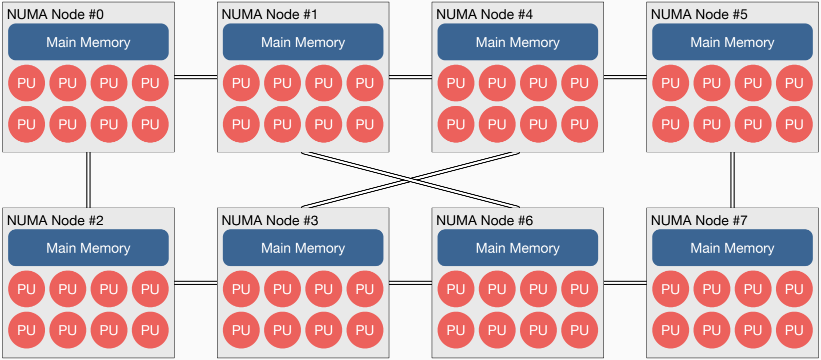
Sebastian Wölke, Raphael Hiesgen, Dominik Charousset
and Thomas C. Schmidt
{sebastian.woelke,raphael.hiesgen,dominik.charousset,t.schmidt}@haw-hamburg.de

iNET RG, Hamburg University of Applied Sciences

# Non-uniform Memory Access (NUMA)

- Memory access is slow compared to processor speed
- Main memory distributed among cores
- Latency for local access is smaller
- This architecture is accessible via a NUMA API

# NUMA Hardware Example




A sample NUMA architecture with 8 nodes and 64 processing units.
(Twisted Ladder Topology)

Does a locality-aware scheduler improve the performance of CAF?

# Outline

Introduction

NUMA-Aware Work Stealing

Locality-Guided Scheduling

Evaluation

Conclusion & Future Work

# Introduction

# CAF

- Actor library written in C++11
  - Low memory footprint
  - Fast, lock-free mailbox implementation
  - Type-safe message passing
- Focus on scalability

# The CAF Scheduler

- Cooperative scheduler running in user space
- Random work-stealing
  - Fixed number of workers with separate job queues
  - Workers process their own queue until it is empty
  - Then, pick a random victim worker to steal one job
- No a priori knowledge of the application behavior required

⇒ **How can we optimize data locality in the scheduler?**

# Data Locality

**Communication Locality (CL)**

- Minimize effort for communication
- Schedule actors near their received messages
- Process cached messages directly when possible

**Execution Locality (EL)**

- Minimize the effort to access state
- Execute actors near their initial worker
- Return actors to the same NUMA node

# NUMA-Aware Work Stealing

# Probabilistic Work Stealing (PWS)

- Increase probability to steal from nearby workers [1]
- Originally designed for computer networks

*"The probability to become a victim is proportional to the inverse of the distance to the thief."*

[1] Quintin et al., 2010, Hierarchical Work-stealing.

# Erlang: Hubs and Affinity

**Colocate actors (hubs) with their communication partners (affinity group).** [2]

- Initial Actor Placement
  - Developer give hints when spawning actors
  - Hubs have their own affinity group and are spread over workers
  - Regular actors inherit affinity and are placed close to their hub
  - Actors store their initial NUMA node as their home node
- Hierarchical Load-Balancing and Work-Stealing
  - A periodic load-balancer migrates actors back home
  - Actors can be move across nodes to balance the system
  - Work stealing prefers workers in the proximity

[2] Francesquini et al., 2013, Actor Scheduling for Multicore Hierarchical Memory Platforms

# Locality-Guided Scheduling

# Weighted Work Stealing

**LGS implements weighted work stealing:**

- Improve probability of stealing based on NUMA architecture
- Interpret processor architecture as a small distributed system
- NUMA-hops between PUs indicate distance

# Picking Victims

**Preparation during startup**

- Each worker sorts other workers into groups $g_0 \subseteq \ldots \subseteq g_k$
- $g_0$ contains direct neighbors, $g_k$ contains all other workers
- Index correlates with distance (lower is better)

**Strategy at runtime**

- Try to steal from each group in order of increasing index
- Pick workers from each group at random
- Start at the beginning after a successful steal

$\Rightarrow$ **Minimal runtime overhead with configurable granularity.**

# Actor Pinning

**LGS implements automatic static soft actor pinning:**

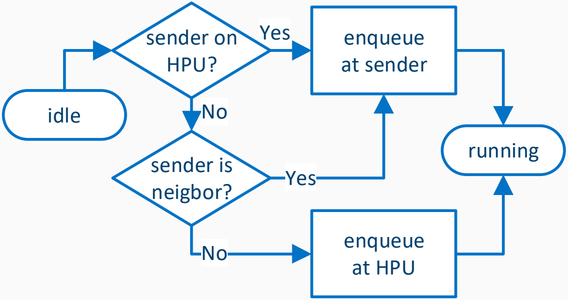
- **Pinning:** actors have a "favored" worker for execution
- **Soft:** actors can be stolen for balancing reasons
- **Static:** the "favored" worker is set only once
- **Automatic:** no developer interaction required

# Initial Actor Placement

- Store worker as home processing unit (HPU) on first execution
- Stolen actors choose a worker on their original NUMA node

# Scheduling

- Actors can be stolen by arbitrary workers
- Idle actors are scheduled on their HPU or a direct neighbor



# Evaluation

# Test Setup

- Server running SUSE Linux (kernel 3.16.7)
  - Four AMD Opteron 6376 processors at 2.3 GHz
  - Eight NUMA nodes, each with eight cores and 64 GB memory
  - NUMA access via hwloc [3]
  - GCC 4.8.3, Java 1.8.0_40 (OpenJDK), hwloc 1.11.4rc2-git
- Activated cores scale from 4 to 64 (in steps of 4)
- Mean runtime in secs over 10 runs with 95% conf. interval

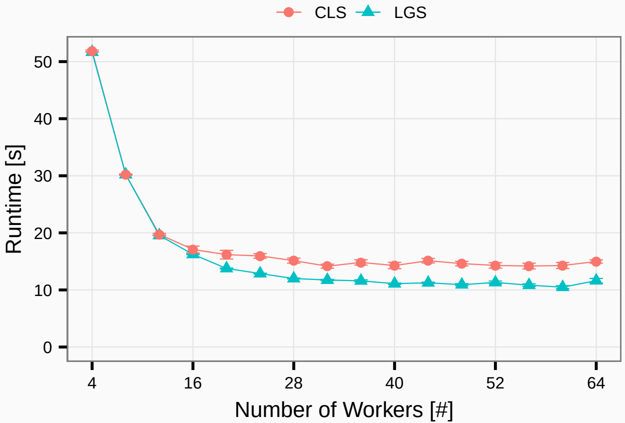
[3] `http://www.open-mpi.de/projects/hwloc/`

# Benchmark: Matrix Search

**Self written benchmark with heavy memory access**

- Solve word-finding puzzles distributed by a coordinator
- Actors hosts their own word-grid, aligned row-wise in memory
- Only matches along columns are valid to bypass prefetching
- Uneven puzzle complexity for irregular rescheduling

# Results: Matrix Search



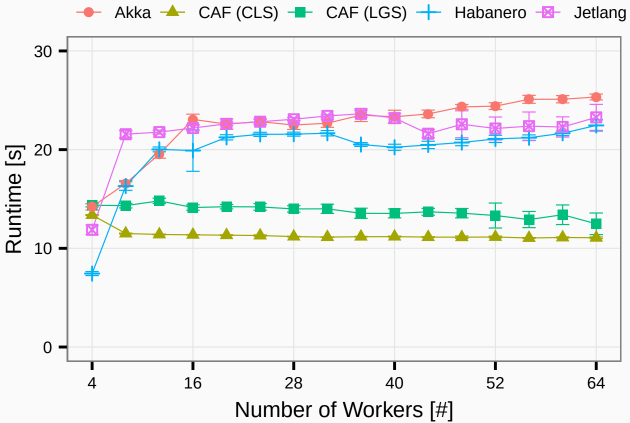

- Similar performance for up to 12 workers, then LGS pulls ahead
- Difference increases until 28 workers (bus capacity?)
- Overall LGS outperforms CLS by up to 26.6%

# Benchmark: Condict & Concsll

- Central data structure encapsulated by central actor
- Others access structure by sending get & put requests
- **Condict**: dictionary with read/write complexity $O(1)$
- **Concsll**: sorted liked list with read/write complexity $O(N)$
- Part of the Savina Benchmark Suite [4]

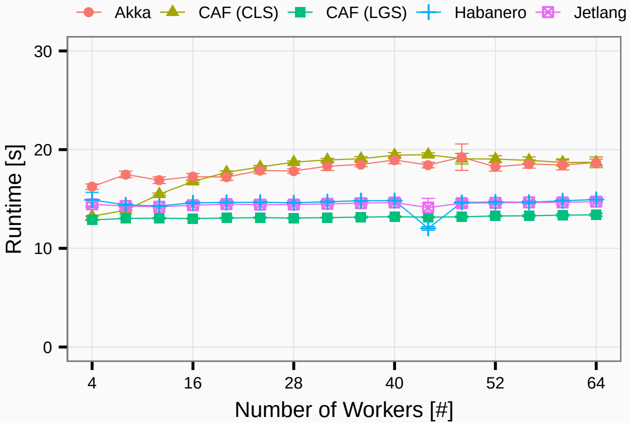
[4] Imam et al., 2014, Savina – An Actor Benchmark

# Results: Condict – Concurrent Dictionary



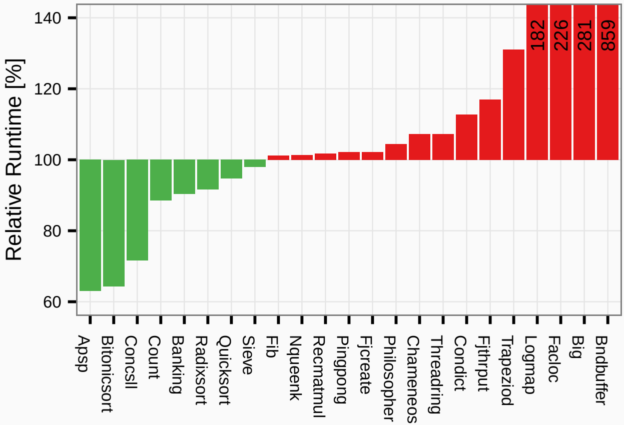

- Both CAF implementations perform well
- CLS up to 17.6% faster than LGS
- Remaining impls. show a strong increase in the beginning

# Results: Concsll – Concurrent Sorted Linked List




- Similar performance except for CLS and Akka
- LGS shows the best performance (32.5% better than CLS)
- Data structure access explains performance differences

# Savina Summary




- Relative runtime of LGS compared to CLS in CAF
- A few benchmarks show excellent results
- Performance degrades significantly for others

# Conclusion & Future Work

# Conclusion

- NUMA architectures exposed through software APIs
- LGS exploits this knowledge to focus on execution locality
  - Weighted Work Stealing
  - Soft Actor Pinning
- Extensive evaluation
  - Memory intensive benchmarks perform excellently
  - Significantly impacts other benchmarks
  - Maybe suitable as an optional strategy
- Future Work
  - Examine the possibility for scheduling hints
  - Comparison with other NUMA-aware actor systems

Thank you for your attention.

Questions?

Developer blog: http://actor-framework.org

Sources: https://github.com/actor-framework/

iNET: https://inet.haw-hamburg.de