SC22
Dallas, TX | hpc accelerates.

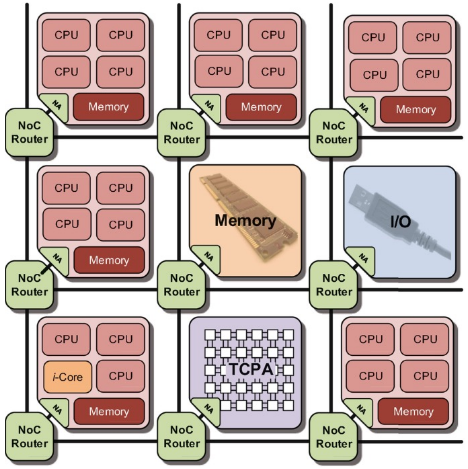
Asynchronous Workload Balancing through Persistent Work-Stealing and Offloading for a Distributed Actor Model Library

Yakup Budanaz, **Mario Wille**, and Michael Bader

Technical University of Munich
School of Computation, Information and Technology – Department of Computer Science

Motivation and Project Overview

- Motivation stems from **Invasive Computing**[1]
  - Dynamic resource allocation and deallocation
  - Provides explicit handles to specify resource requirements desired or required in different phases of execution
  - Usage of actors to facilitate the specification of requirements
- Use **UPC++** to shift from embedded to HPC applications
  - Development of an UPC++ based actor framework[3] [4]
  - Extension of the framework to enable migration of actors for asynchronous workload balancing



Typical tiled invasic architecture – image from (2)

(1) https://www.invasic.de/

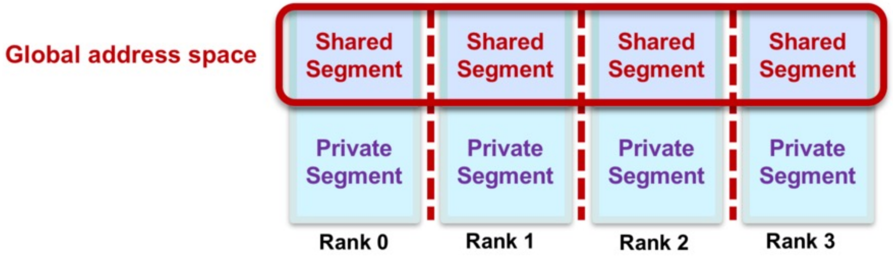
(2) https://link.springer.com/chapter/10.1007/978-3-030-47487-4_9

(3) https://github.com/TUM-I5/Actor-UPCXX

(4) Pöppl, A.; Baden S.; Bader, M.: A UPC++ Actor Library and Its Evaluation On a Shallow Water Proxy Application, PAW-ATM 2019

UPC++[1]

- Asynchronous Partitioned Global Address Space (**APGAS**) Model
- Designed for writing efficient, scalable parallel programs on distributed-memory parallel computers[2]
- Key communication facilities in UPC++ are one-sided **Remote Memory Access** (RMA) and **Remote Procedure Call** (RPC)
- Focused on maximizing scalability
- Communication operations are asynchronous
- Uses GASNet[3] for communication across a wide variety of platforms



PGAS Memory Model – image from (2)

(1) [*https://bitbucket.org/berkeleylab/upcxx/wiki/Home*](https://bitbucket.org/berkeleylab/upcxx/wiki/Home)

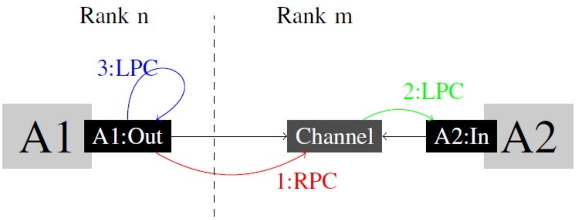
(2) [*https://upcxx.lbl.gov/docs/html/guide.html*](https://upcxx.lbl.gov/docs/html/guide.html)

(3) [*https://gasnet.lbl.gov/*](https://gasnet.lbl.gov/)

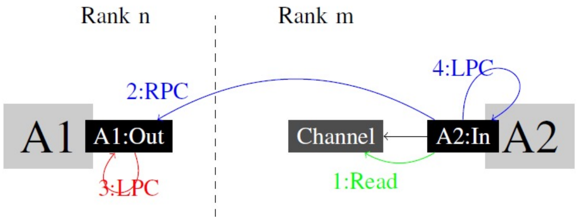
Actor-UPCXX

- Actor encapsulates specific functionality, data and behavior
- Is an object that is identified by its unique name and assigned to a part of the parallel simulation
- Actor and its data can be serialized for sending it to another rank using UPC++
- Connections of actors are saved in a global graph structure that is replicated on every rank in Actor-UPCXX
- Communication through one-sided asynchronous messages

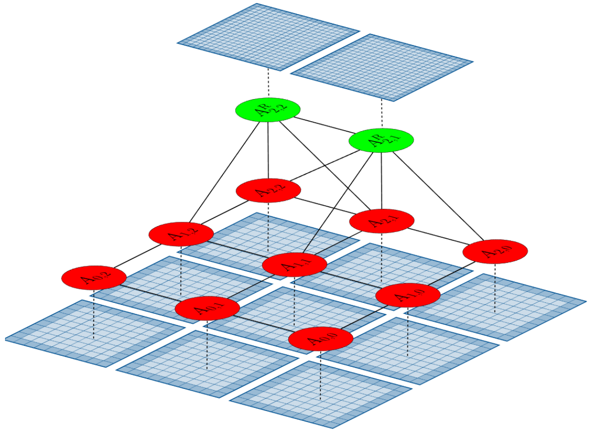
➢ Facilitate actors for dynamic load balancing



(a) A *write* call from actor A1 residing on rank n to actor A2 on rank m.



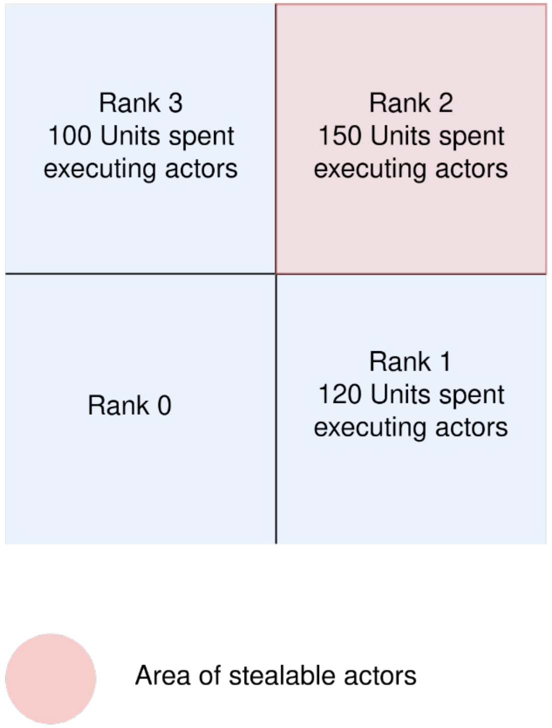
(b) A *read* call from actor A1 residing on rank n to actor A2 on rank m.



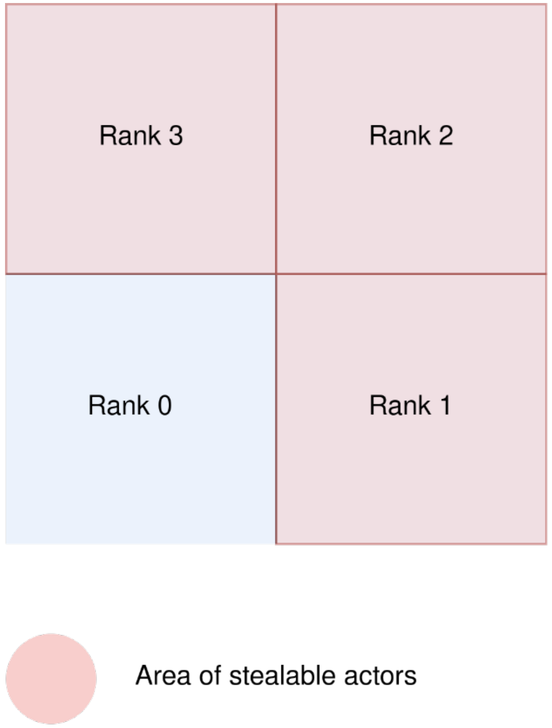
Load Balancing Strategies by Migration of Actors

- Two strategies: actor stealing and actor offloading
- Actor stealing: underloaded rank chooses rank to steal an actor
- Actor offloading: rank that detects an imbalance may decide to offload an actor to a rank that is underloaded
- Both migration strategies perform every action through asynchronous calls (RPCs)
- Actor stealing strategy is further divided into
  - *Global vs. Local:* specifies the set of remote ranks which can be stolen
  - *Random vs. Busy:* specifies the type of polling which is applied to the set of remote ranks

Global Actor Stealing Strategies

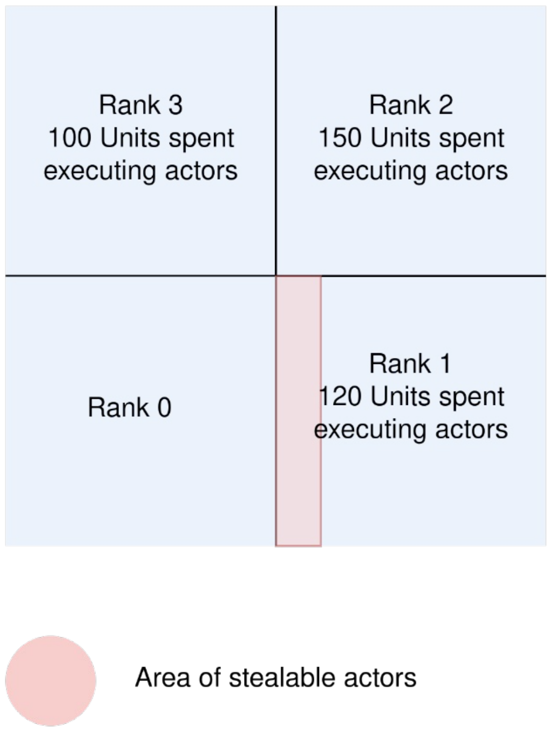


Global-busy: steals an actor from the rank that has spent the most time executing actors

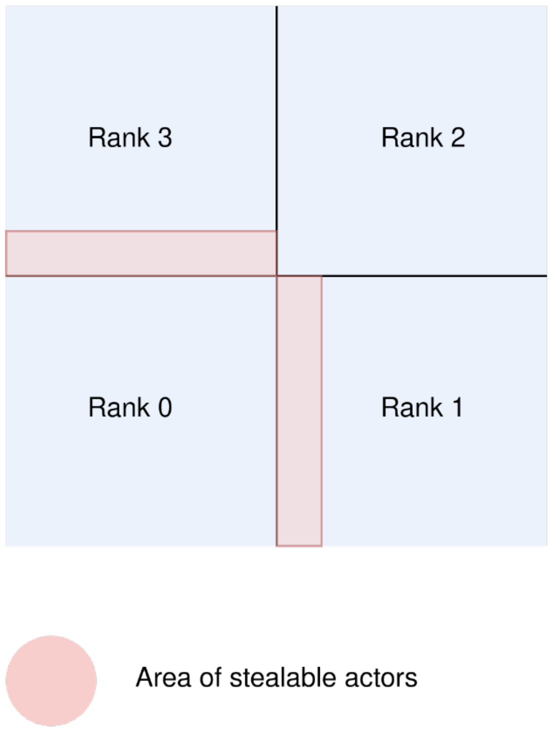


Global-random: steals an actor from any rank without limitations

Local Actor Stealing Strategies



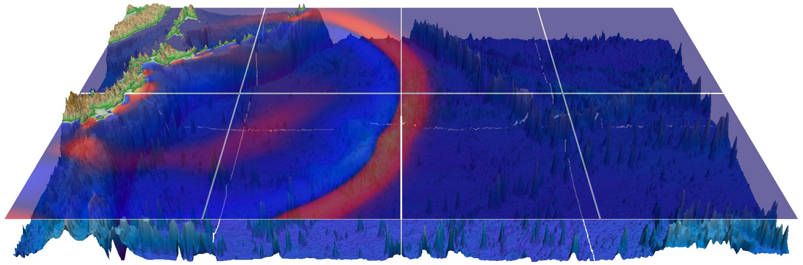
Local-busy: steals an actor from a neighboring rank that has spent the most time executing actors



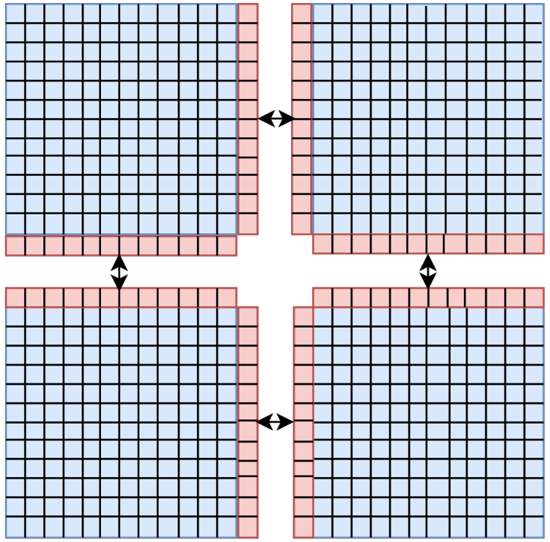
Local-random: steals any neighboring actor from one of the neighboring ranks

Pond – A Shallow Water Proxy Application

- Uses Actor-UPCXX as actor library
- Implements finite volume solvers which solves the shallow water equations

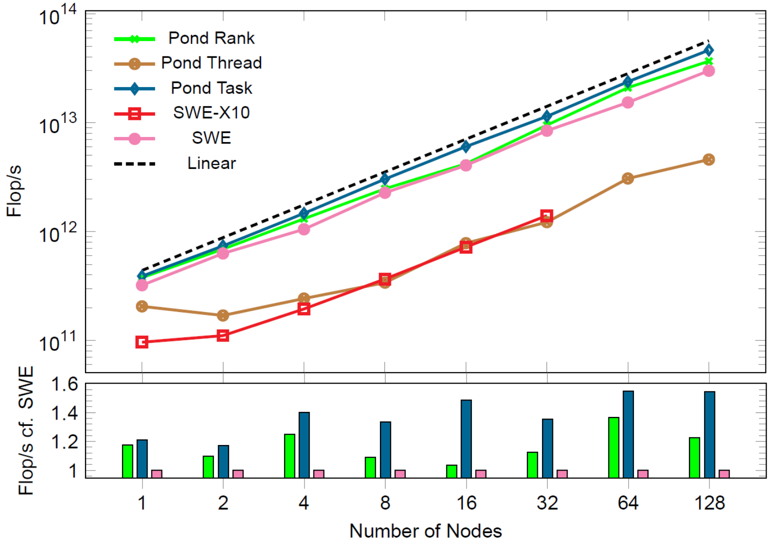


- Pond organizes the 2D Cartesian discretization grid as patches
- Every patch is assigned to an actor permanently
- Example grid decomposed into four patches
- Cells of the patches are marked in blue
- Ghost layers (in red) are used to synchronize data between patches



Pond – A Shallow Water Proxy Application

- Update the cells in the ghost layer according to boundary conditions or with values communicated by neighbor patches
- For each edge compute approximate fluxes between the adjacent cells
- Accumulate the fluxes as net updates
- Update the cell quantities using the net updates
- Ghost cell values are updated by sending one-sided messages between Pond’s actors
- **Lazy activation**: patches are only activated when a propagating wave enters the patch



Weak scaling of Pond – image from (1)

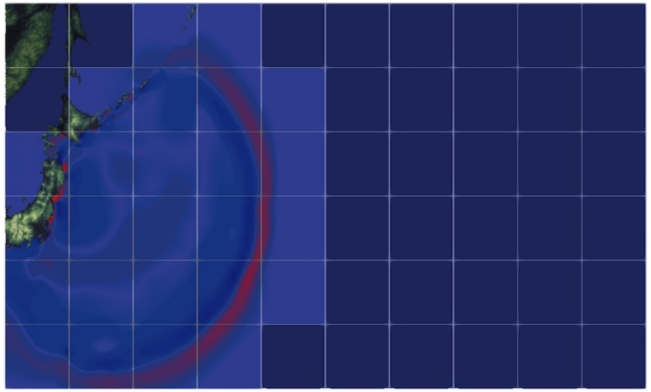


Illustration of lazy activation – image from (2)

(1) *Pöppl, A.; Baden S.; Bader, M.: A UPC++ Actor Library and Its Evaluation On a Shallow Water Proxy Application*

(2) *https://invasic.informatik.uni-erlangen.de/en/tp_a4_PhIII.php*

Evaluation: Comparison of Load Balancing Strategies

- Load balancing strategies have been compared using the scenarios
  1. Static workload: 18000 × 18000 grid divided into 250 × 250 patches with 1 patch per actor
  2. Node slowdown: 18000 × 18000 grid divided into 250 × 250 patches with 1 patch per actor
  3. Lazy activation: 36000 × 36000 grid divided into 250 × 250 patches with 1 patch per actor
- All scenarios are strong scaling tests
- All scenarios use the same solver

- Performed on CoolMUC-2 Cluster hosted by the Leibniz Supercomputing Center (LRZ)[1]
- Equipped with 28-way Intel Xeon E5-2690 v3 compute nodes and FDR14 Infiniband interconnect
- UPC++ version used is 2022.03.0.
- UPC++, and Actor-UPCXX were compiled with the Intel oneAPI compilers[2]
- OpenMPI v4.1.2 and HWLoc 2.6.0 are used by the communication backend of UPC++
- GASNet-EX for the job launch

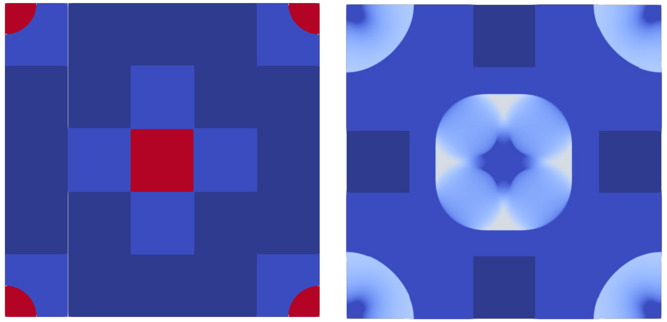
---

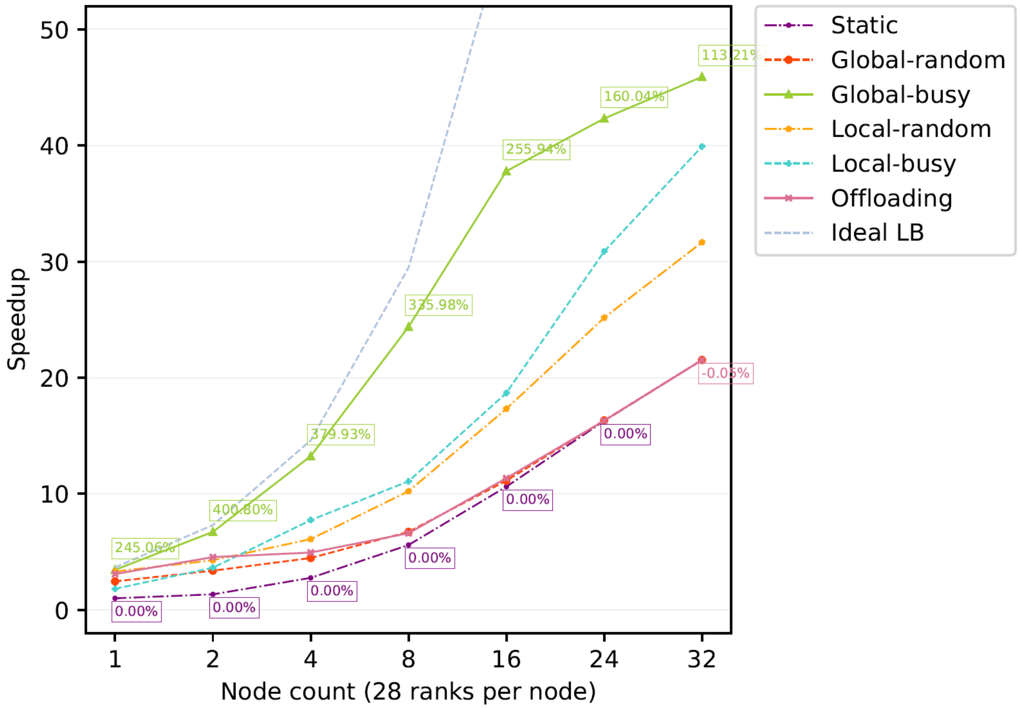
(1) *https://doku.lrz.de/display/PUBLIC/Linux+Cluster*

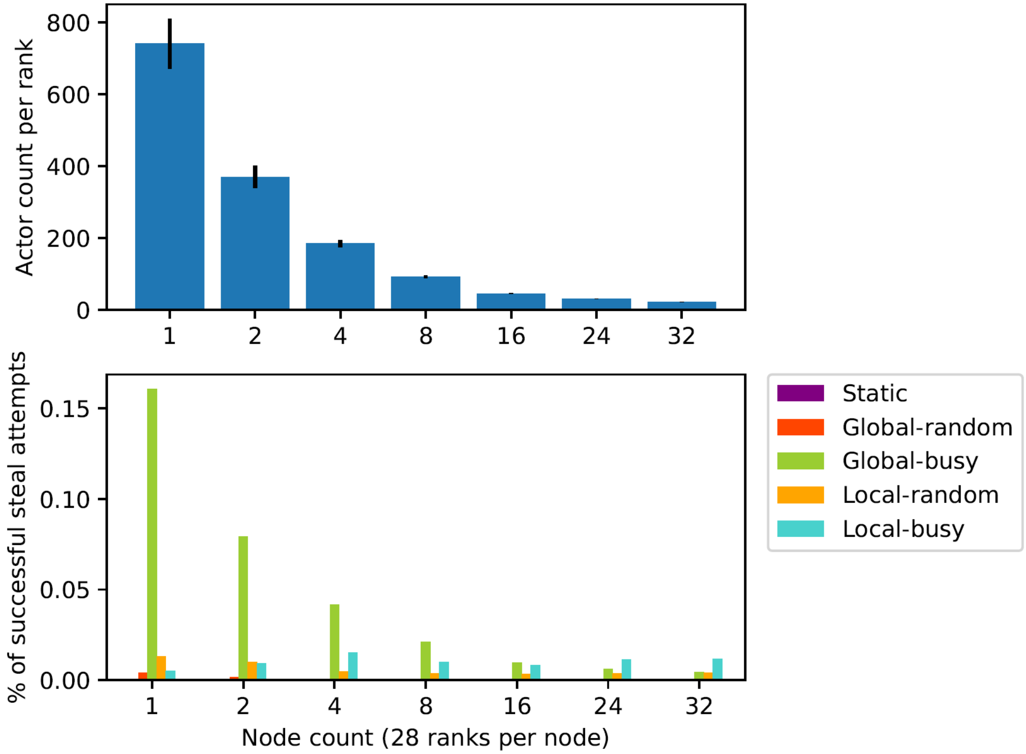
(2) *https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html*

Evaluation: Lazy Activation Scenario – Speedup

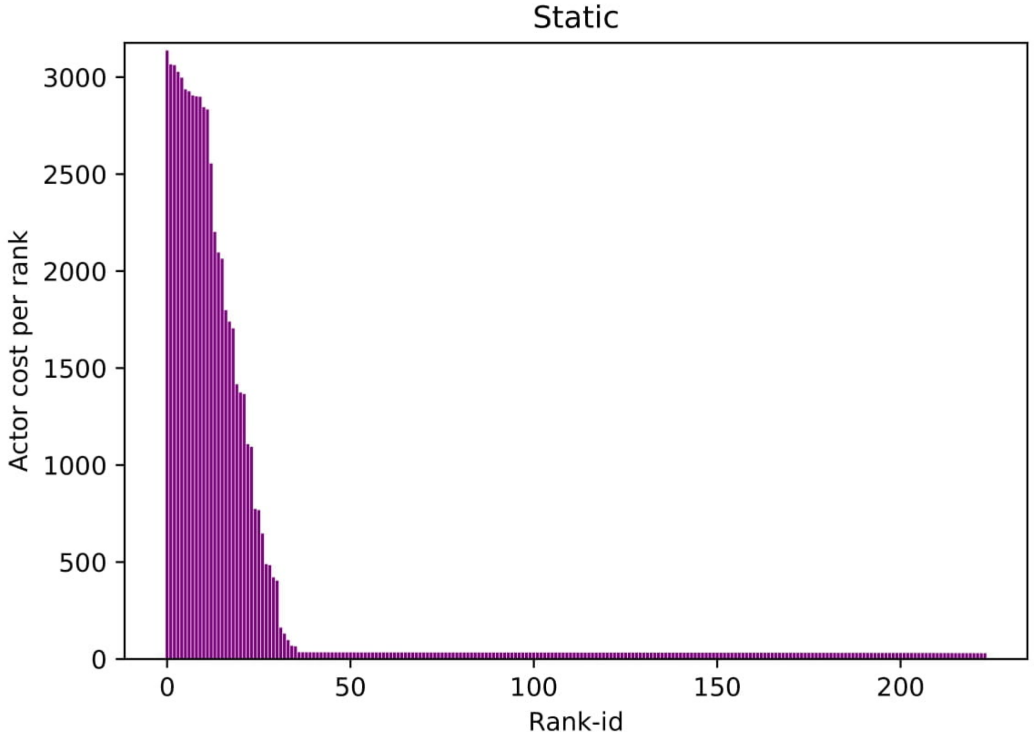
- Workload is determined by the evolving solution
- Actors are activated as waves enter their patch
- Static: no actor migrations



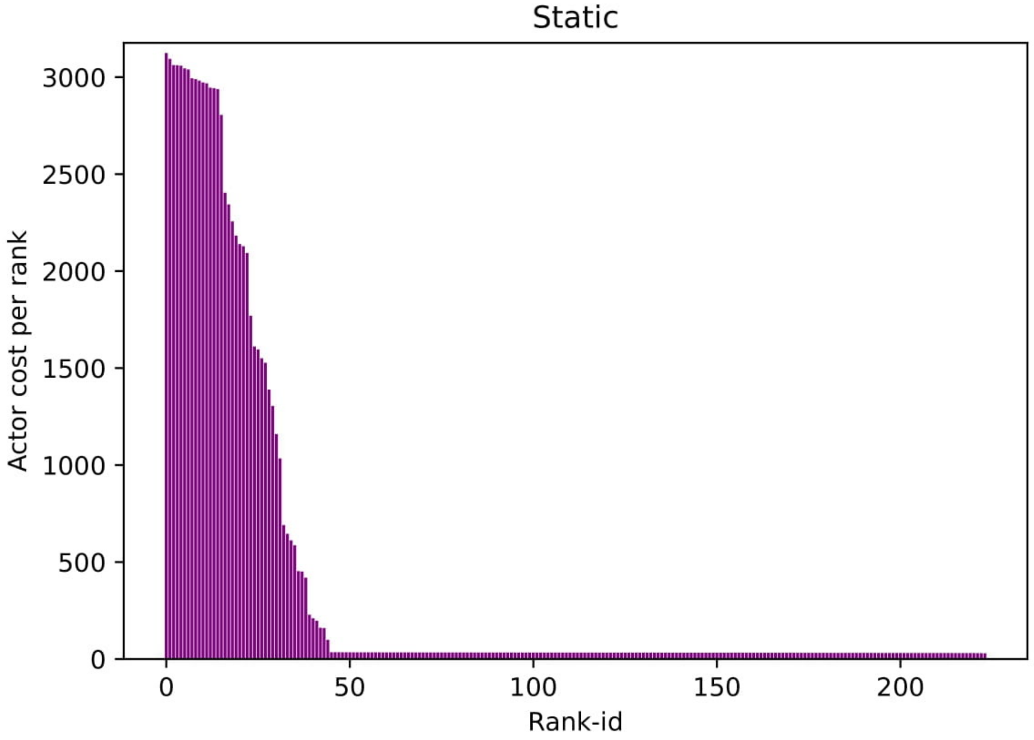




Evaluation: Lazy Activation Scenario – Load Distribution

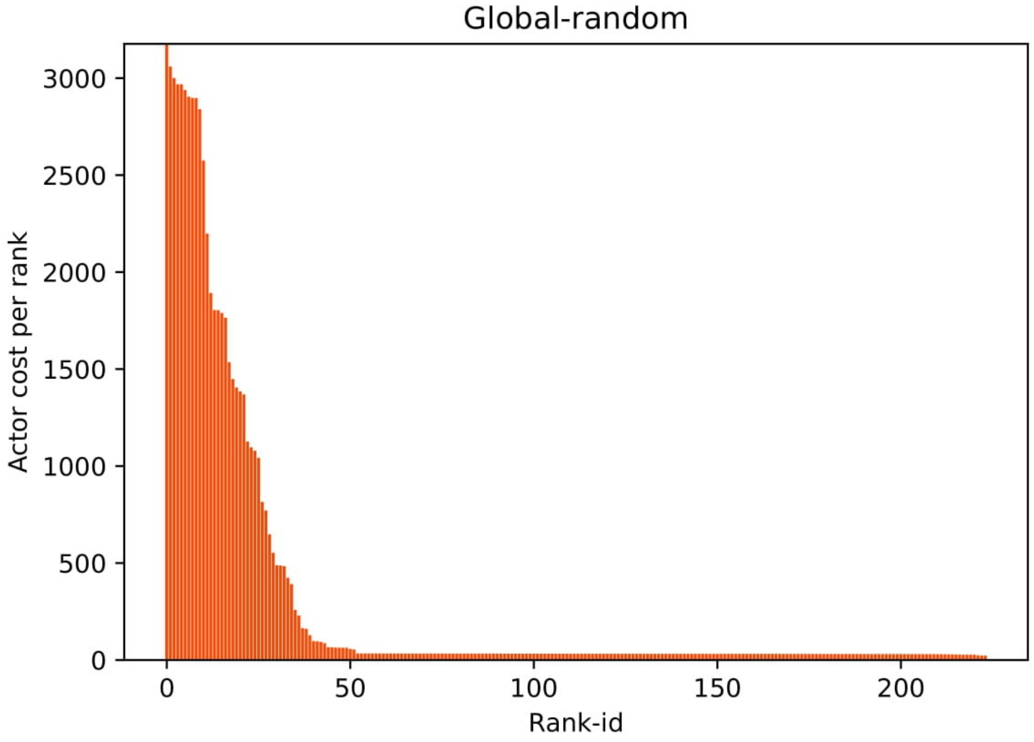


At 180 seconds

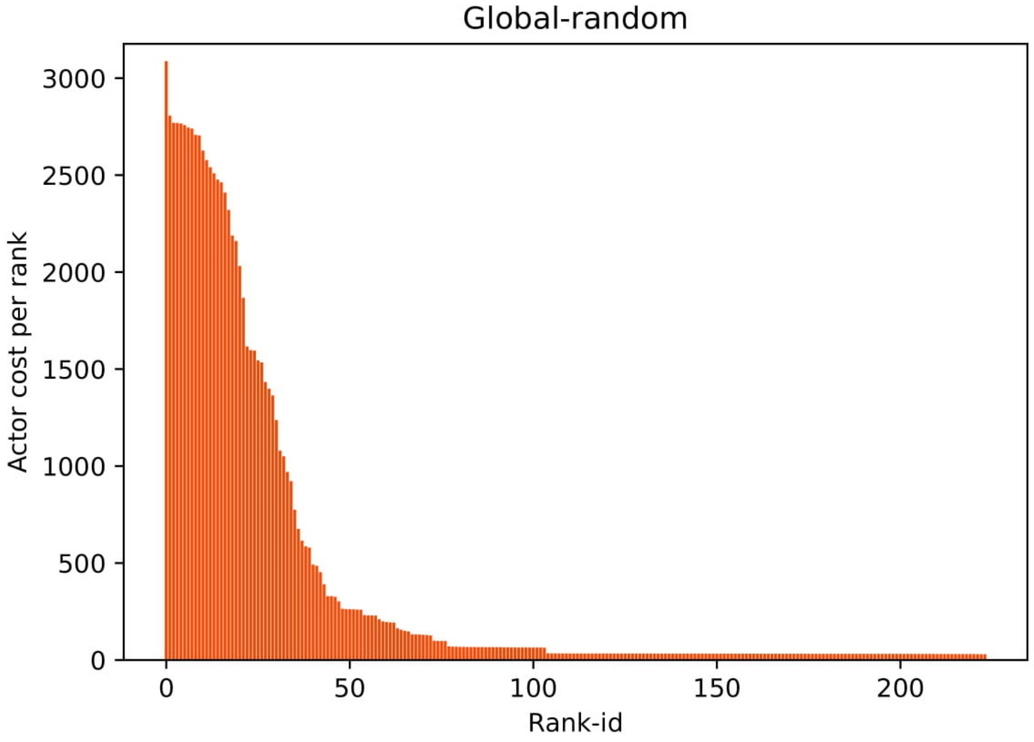


At 720 seconds

Evaluation: Lazy Activation Scenario – Load Distribution

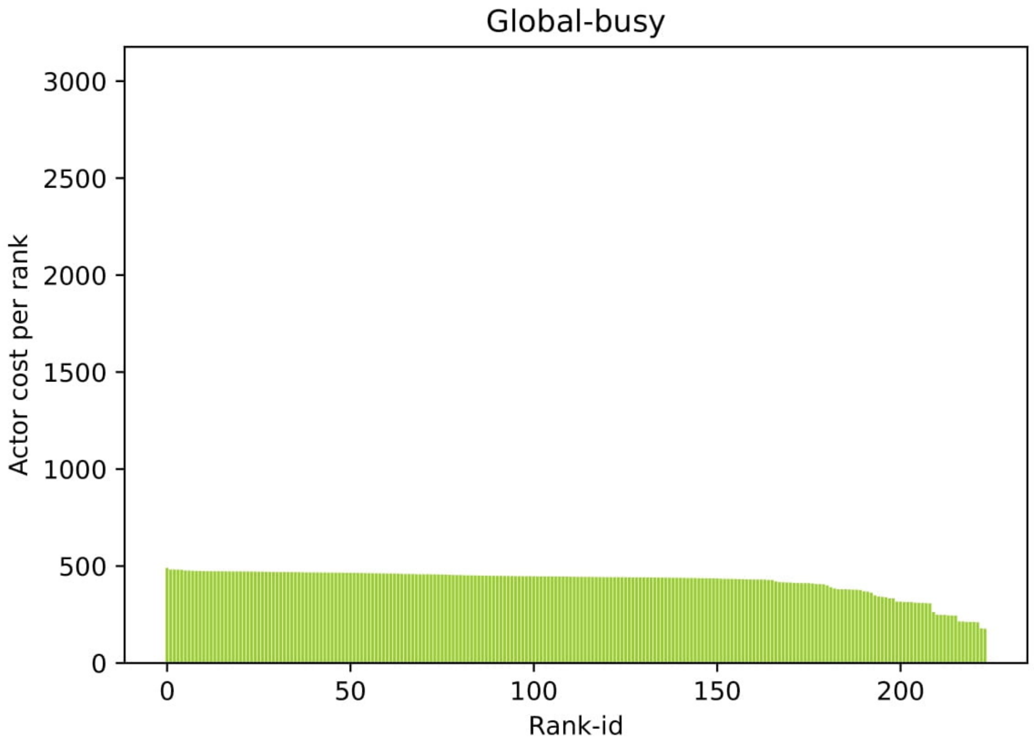


At 180 seconds

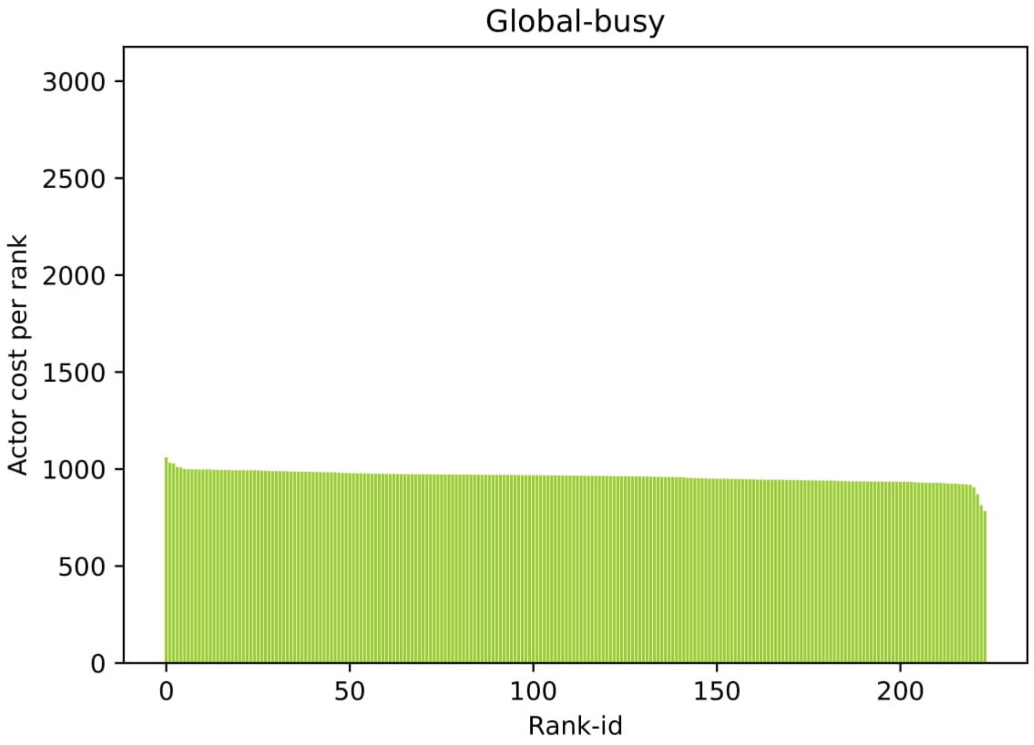


At 720 seconds

Evaluation: Lazy Activation Scenario – Load Distribution

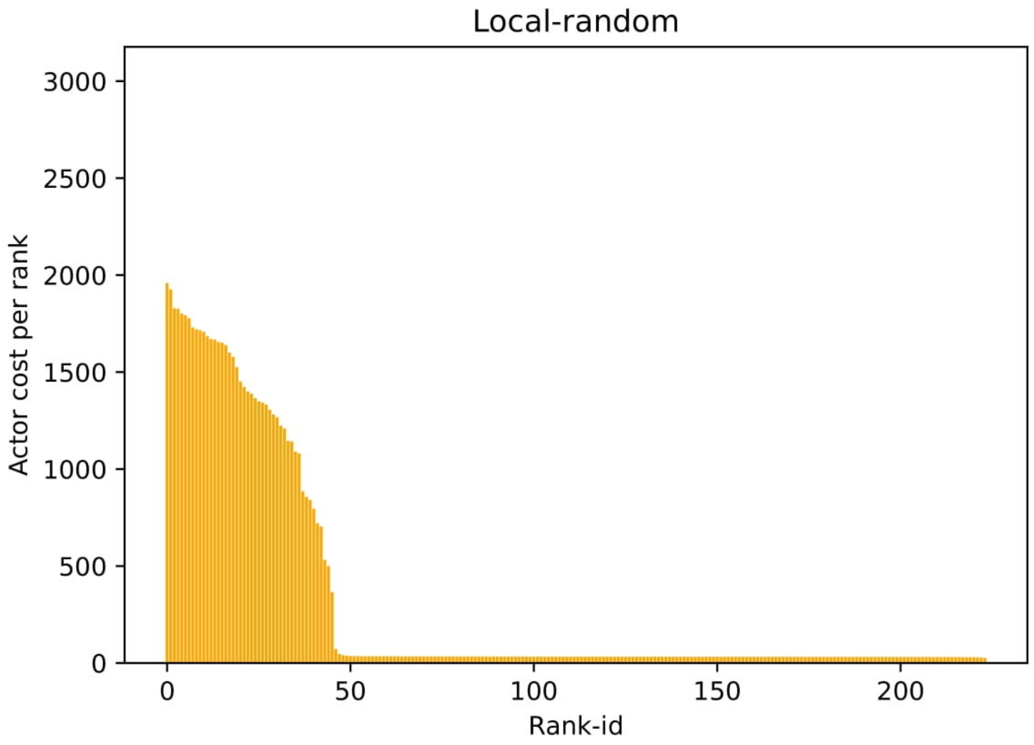


At 180 seconds

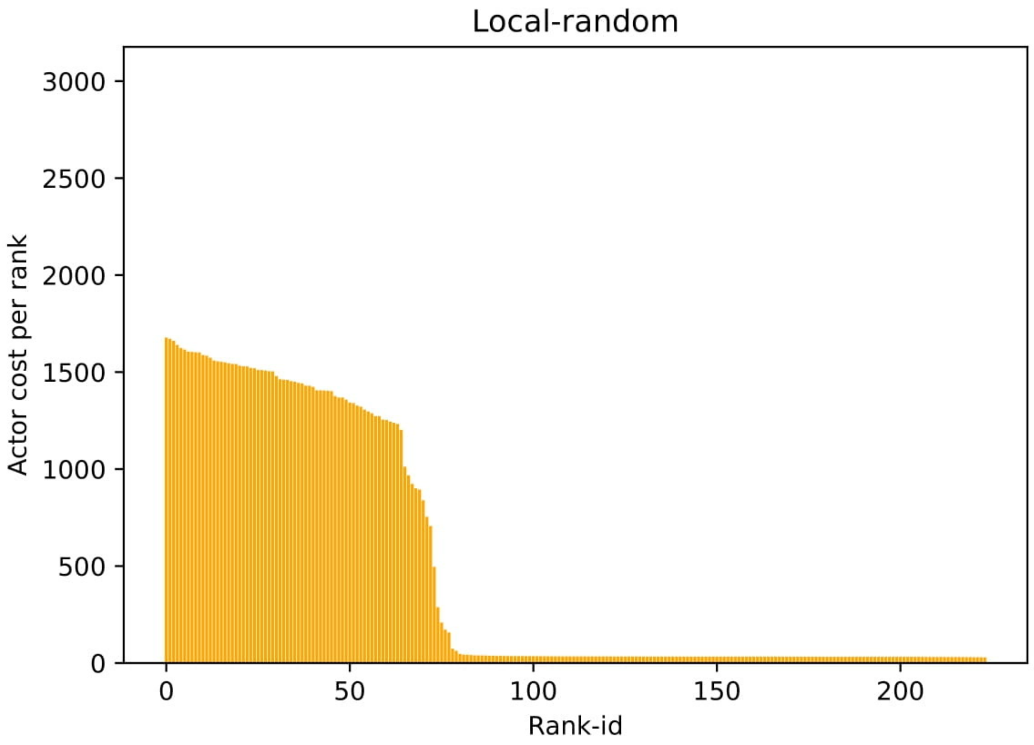


At 720 seconds

Evaluation: Lazy Activation Scenario – Load Distribution

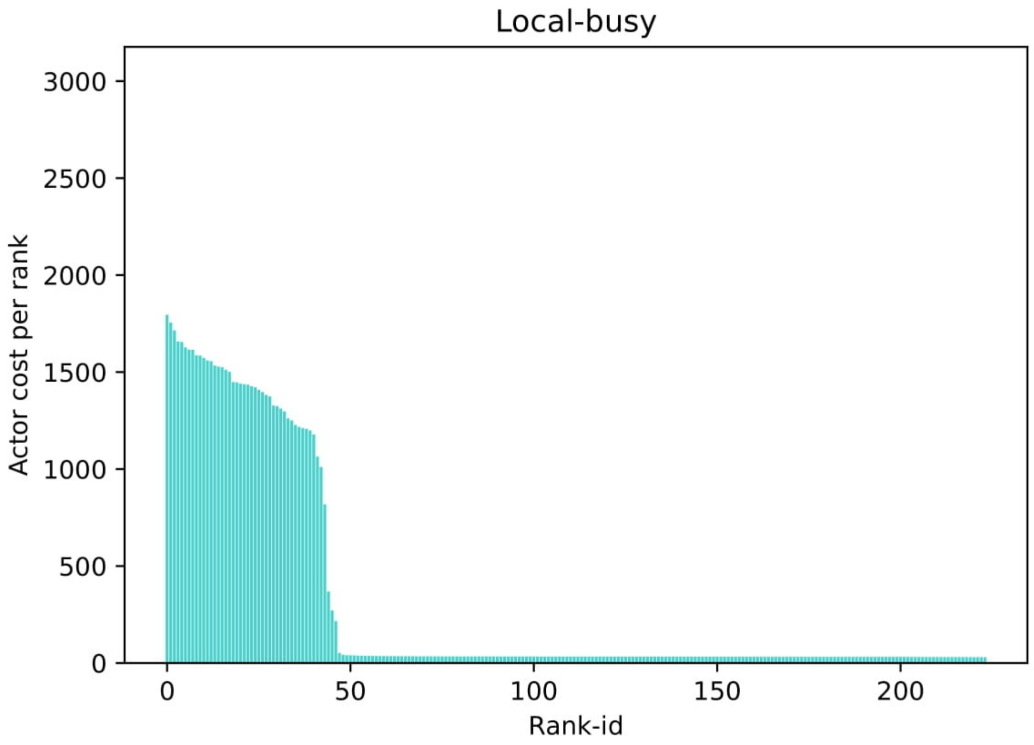


At 180 seconds

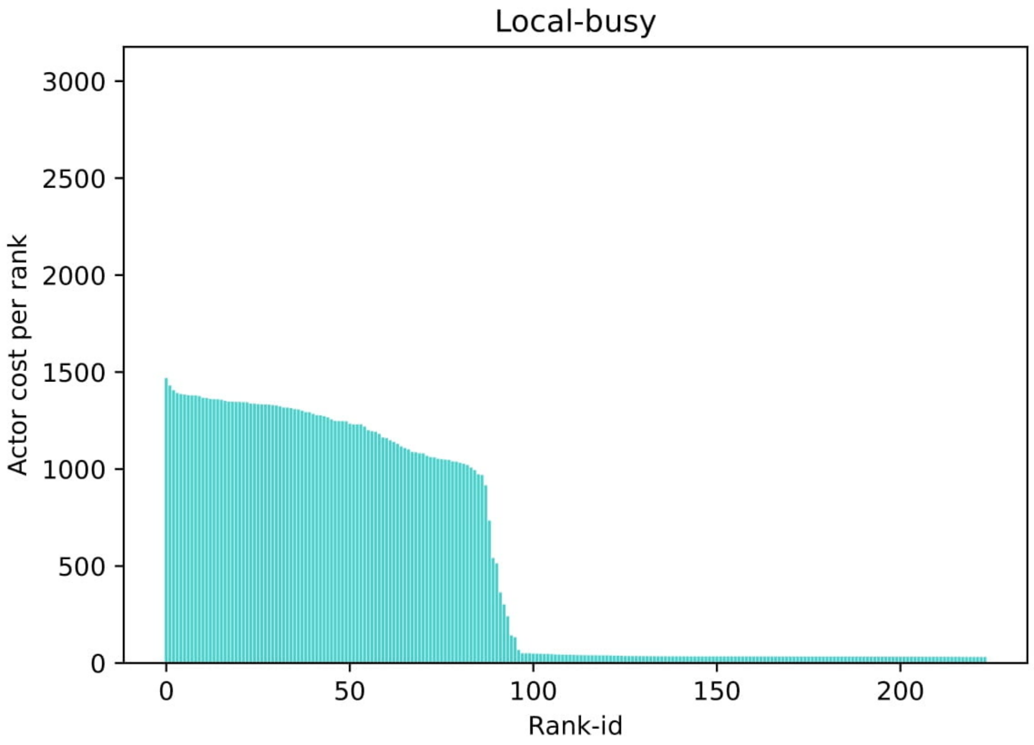


At 720 seconds

Evaluation: Lazy Activation Scenario – Load Distribution

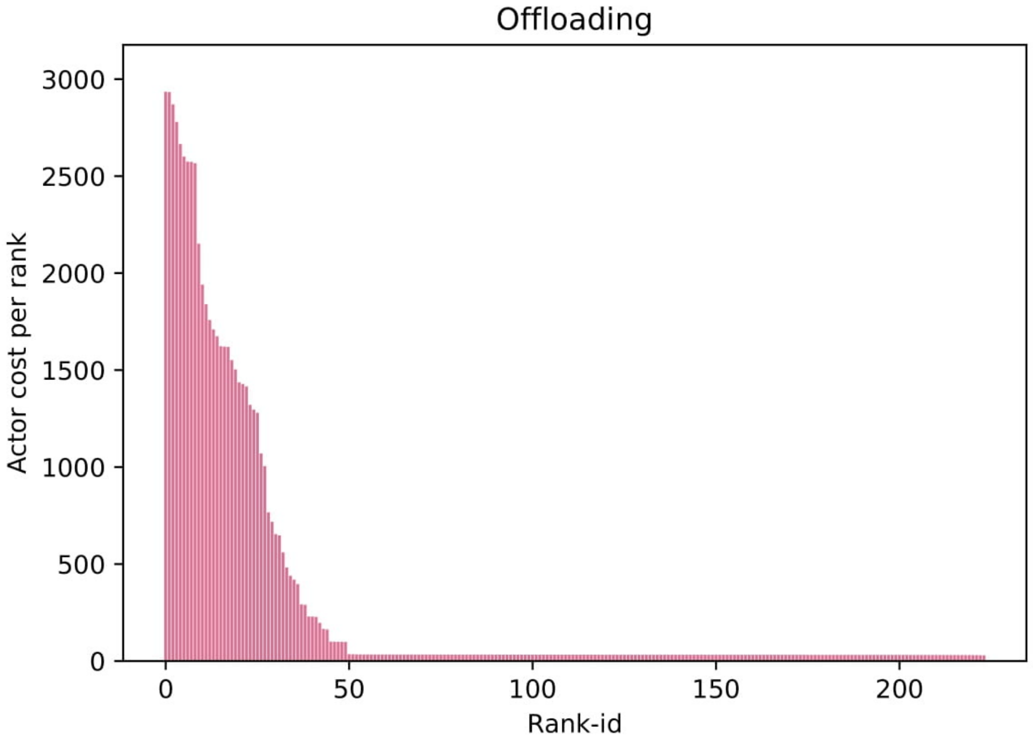


At 180 seconds

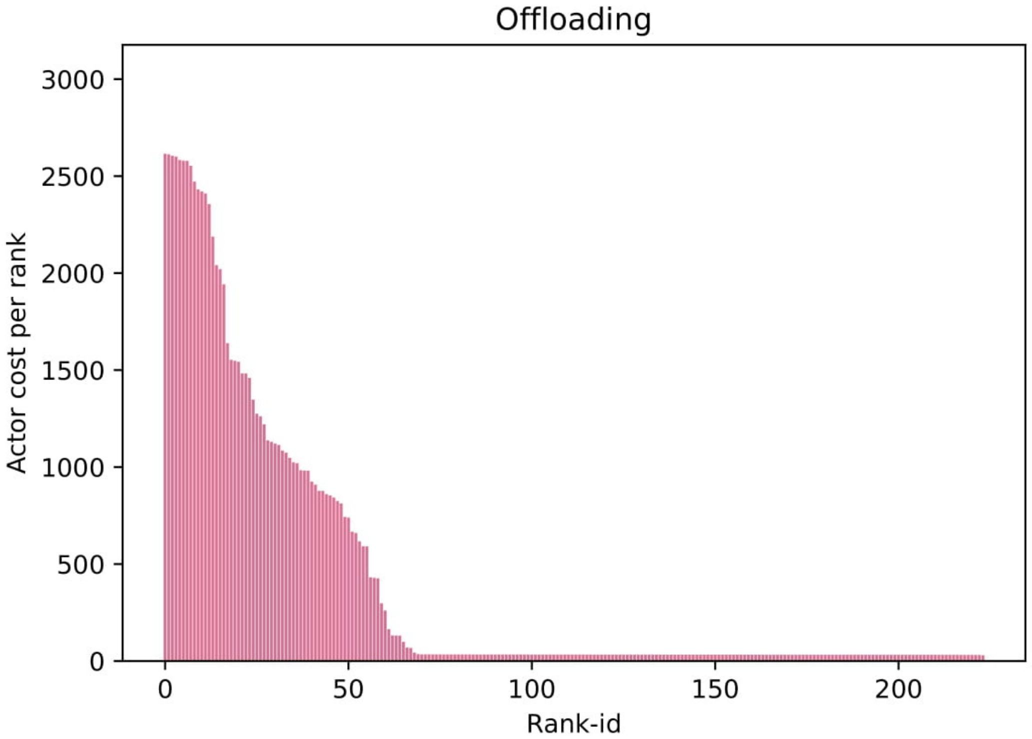


At 720 seconds

Evaluation: Lazy Activation Scenario – Load Distribution



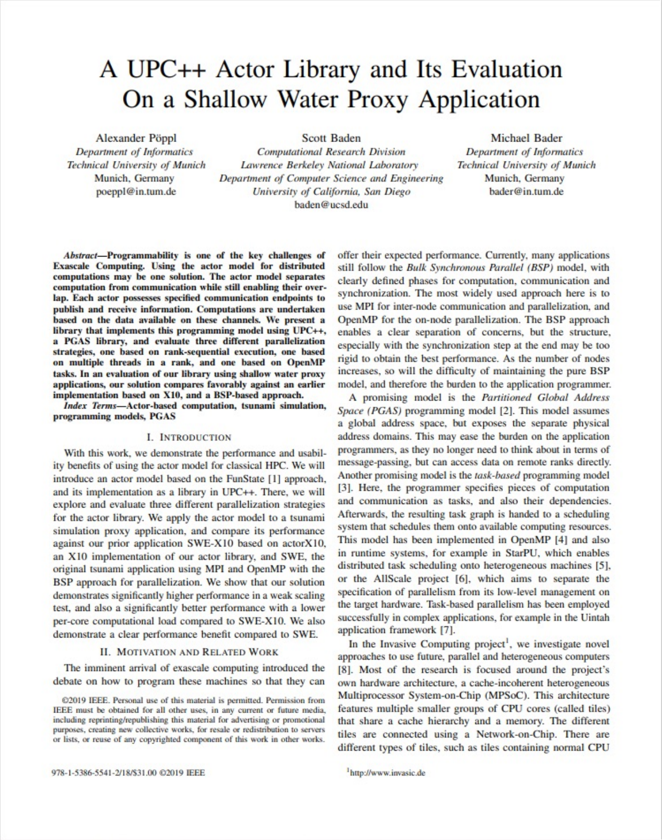
At 180 seconds



At 720 seconds

Acknowledgements

- This research was funded by the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft), Project number 14671743, TRR 89 Invasive Computing
- We would like to thank
  - Philipp Samfass for his advice, which has been of great help during this research and
  - Alexander Pöppl for laying the foundations with Actor-UPCXX and Pond



A UPC++ Actor Library and Its Evaluation On a Shallow Water Proxy Application

Alexander Pöppl, Department of Informatics, Technical University of Munich, Munich, Germany, poeppl@in.tum.de
Scott Baden, Computational Research Division, Lawrence Berkeley National Laboratory, Department of Computer Science and Engineering, University of California, San Diego, baden@ucsd.edu
Michael Bader, Department of Informatics, Technical University of Munich, Munich, Germany, bader@in.tum.de

Asynchronous Workload Balancing through Persistent Work-Stealing and Offloading for a Distributed Actor Model Library

Yakup Budanaz, Department of Informatics, Technical University of Munich, Garching, Germany, yakup.budanaz@tum.de
Mario Wille, Department of Informatics, Technical University of Munich, Garching, Germany, mario.wille@tum.de
Michael Bader, Department of Informatics, Technical University of Munich, Garching, Germany, bader@in.tum.de

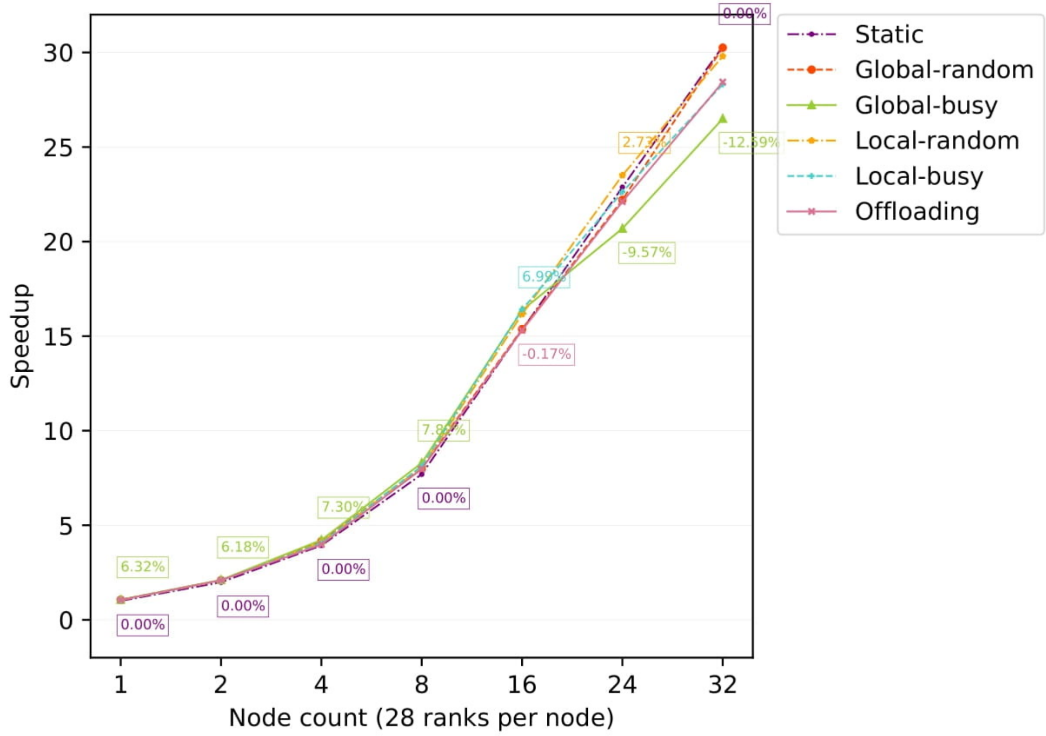
Conclusion

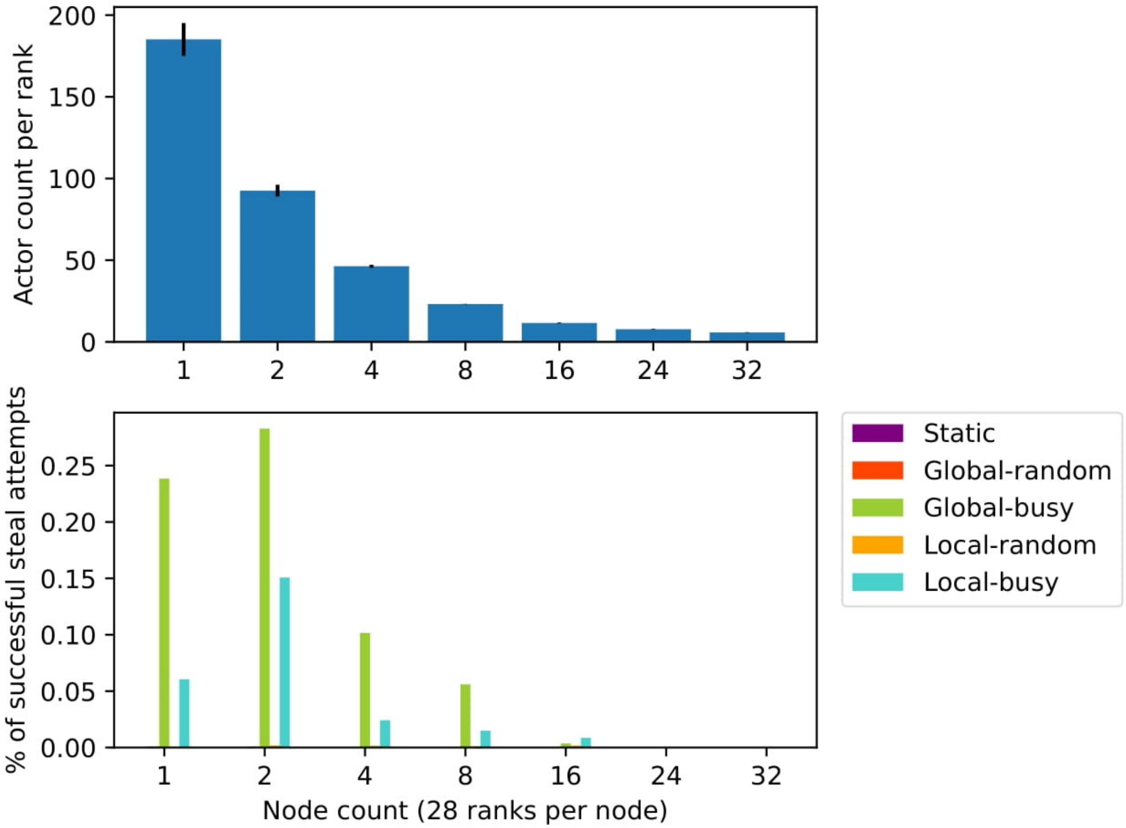
- Fully asynchronous and decentralized dynamic workload balancing scheme
- UPC++ accelerates the implementation of the migration strategies
- UPC++ allows easy implementation of serialization to migrate actors
- Asynchronous nature of UPC++ enables the implementation of actor migration as a chain of RPCs
- Implemented strategies for dynamic load balancing improve runtime in predictable and unpredictable load imbalances
- Achieve speedup of up to 400% compared to the static base case with no actor migration

Backup Slides

Evaluation: Static Workload Scenario – Speedup

- Initial workload distribution of the actors modeled as a graph partitioning problem
- Static mapping of actors to compute nodes is calculated with METIS[1]





(1) http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

Evaluation: Node Slowdown Scenario – Speedup

- Artificial scenario to test the performance under unpredictable workload imbalance
- A subset of ranks is slowed down artificially