Data Structures for Task-based Priority Scheduling

Distributed Computing and Systems
Chalmers university of technology
CHALMERS

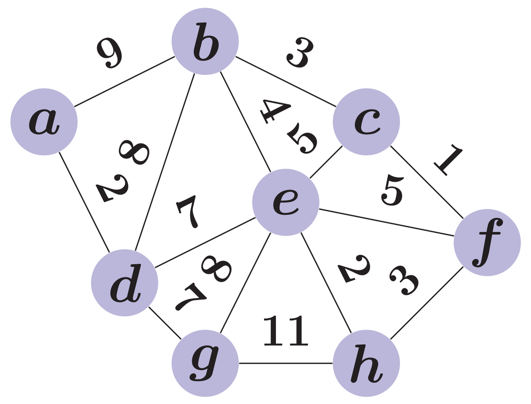
Martin Wimmer [1] Daniel Cederman [2] Francesco Versaci [1] Jesper Larsson Träff [1] Philippas Tsigas [2]

[1] Vienna University of Technology, Institute of Information Systems
[2] Chalmers University of Technology, Computer Science and Engineering

Parallel Computing
TU WIEN
FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Single-Source Shortest Path

This work discusses trade-offs for three types of relaxed priority queues using a simple parallelization of Dijkstra's algorithm for the single-source shortest path problem:



In this label-correcting algorithm, each node is assigned a distance label d_t marking its distance to the source. Nodes are greedily relaxed in parallel, thereby atomically updating their neighbour's distance label if a shorter path is found. Nodes with smaller distance labels are prioritized.

Priority Work-Stealing

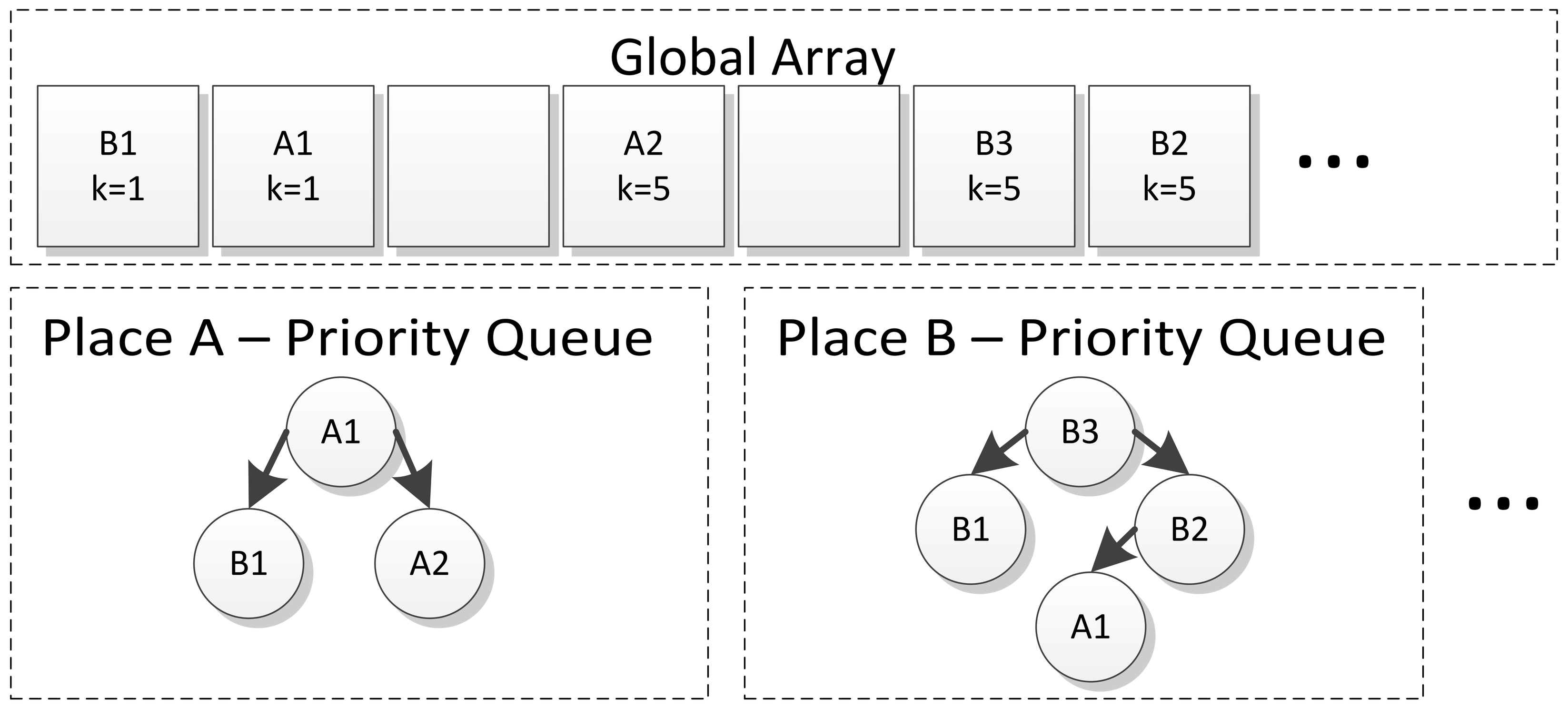
We extend work-stealing to support priorities by using a priority queue per thread instead of a deque. Good scalability due to localized nature of algorithm, but no global guarantees on the priority of tasks.

ρ-relaxation

For improved scalability we adopt ρ-relaxation, a temporal property that allows certain items in a data structure to be ignored. We say an item is ignored whenever an item of lower priority is returned by a pop operation.

Centralized k-priority Queue

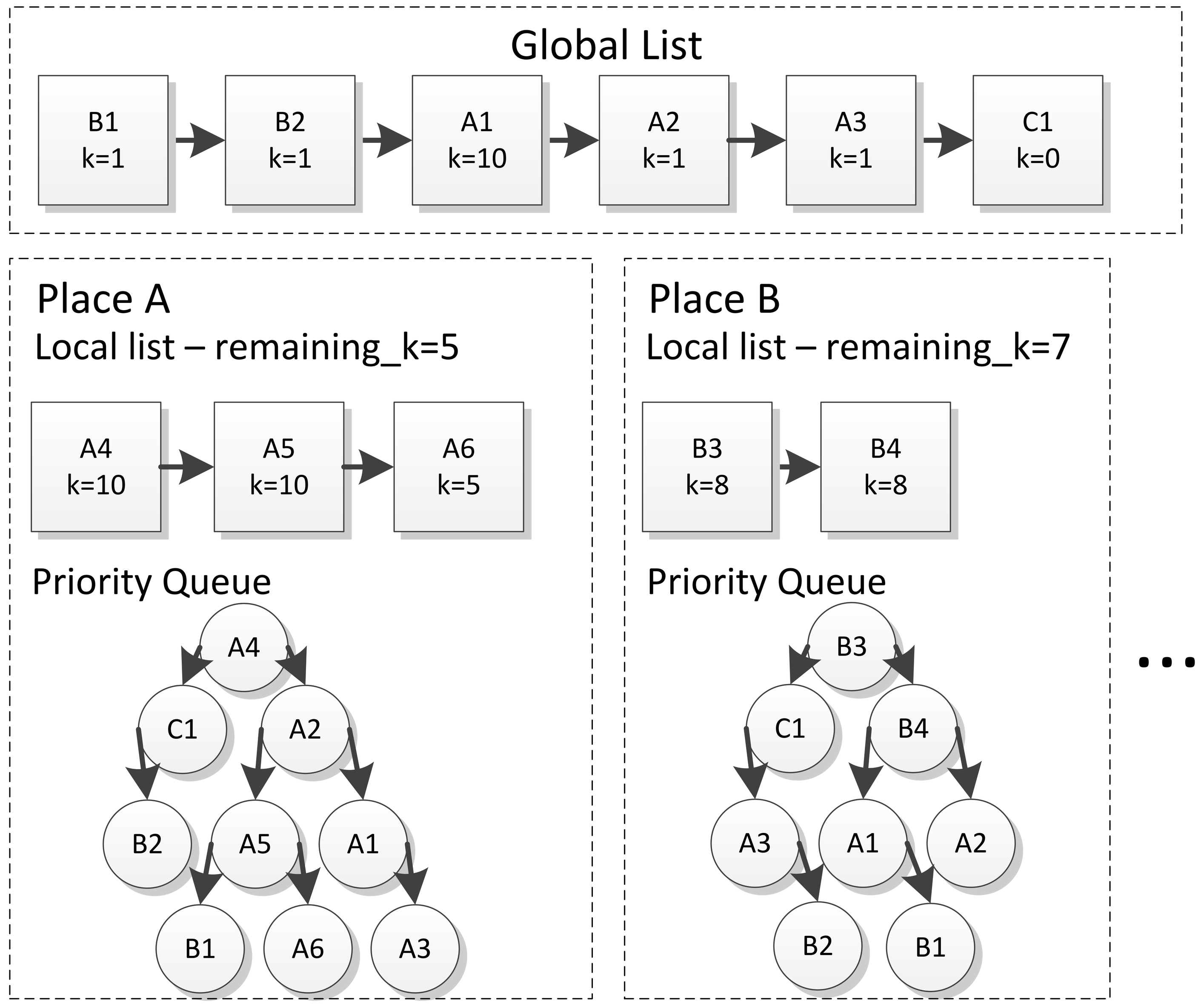
Provides strong guarantees on priorities due to semantics of a centralized, global priority queue. To reduce congestion, we rely on ρ-relaxation as follows: (i) a pop operation is allowed to ignore the items added by the latest $\rho = k$ push operations (which, in the worst case, might be the top k by priority), (ii) each task is visible to at least one thread.



Implementation: Tasks stored in global array in order of creation, but may be shifted by up to k positions to reduce congestion. Priorities maintained locally using a serial priority queue per thread storing references to the global array. Each thread is allowed to ignore the latest k tasks in the global array created by other threads.

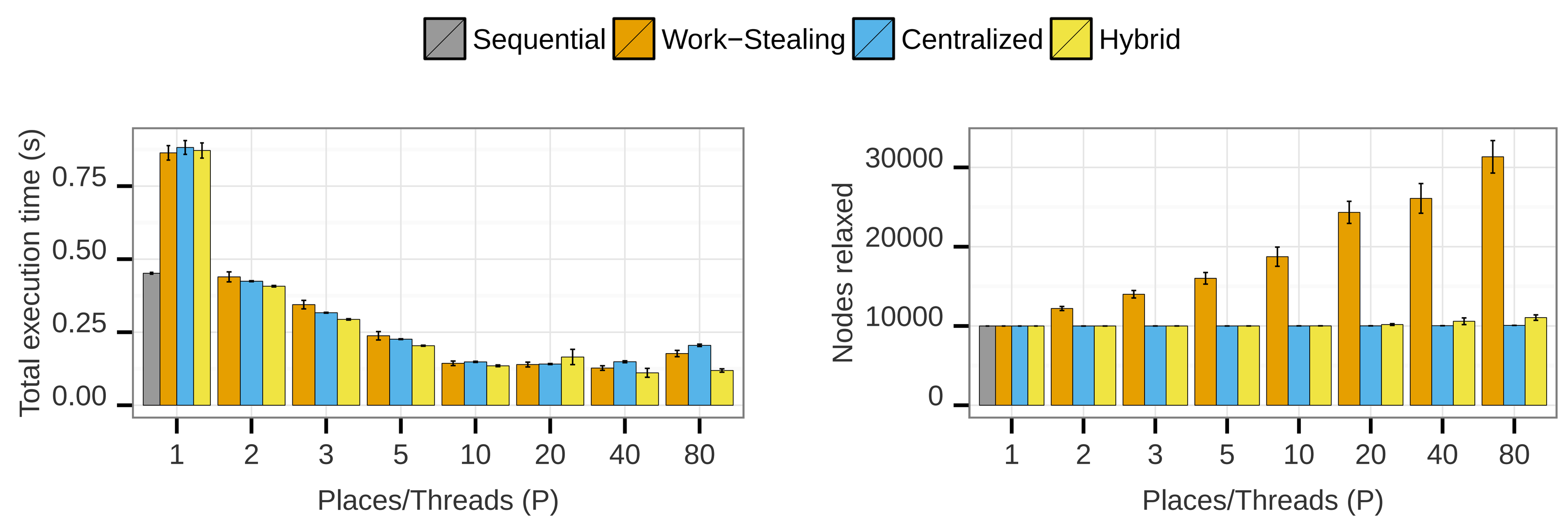
Hybrid k-priority Queue

We combine the priority work-stealing and k-priority ideas, which yields a data structure with scalability close to work-stealing, but providing ρ-relaxation guarantees. Guarantees that at most the latest k items added by each thread to be ignored, which implies that, being P the number of threads, up to $\rho = Pk$ items might be ignored in total.



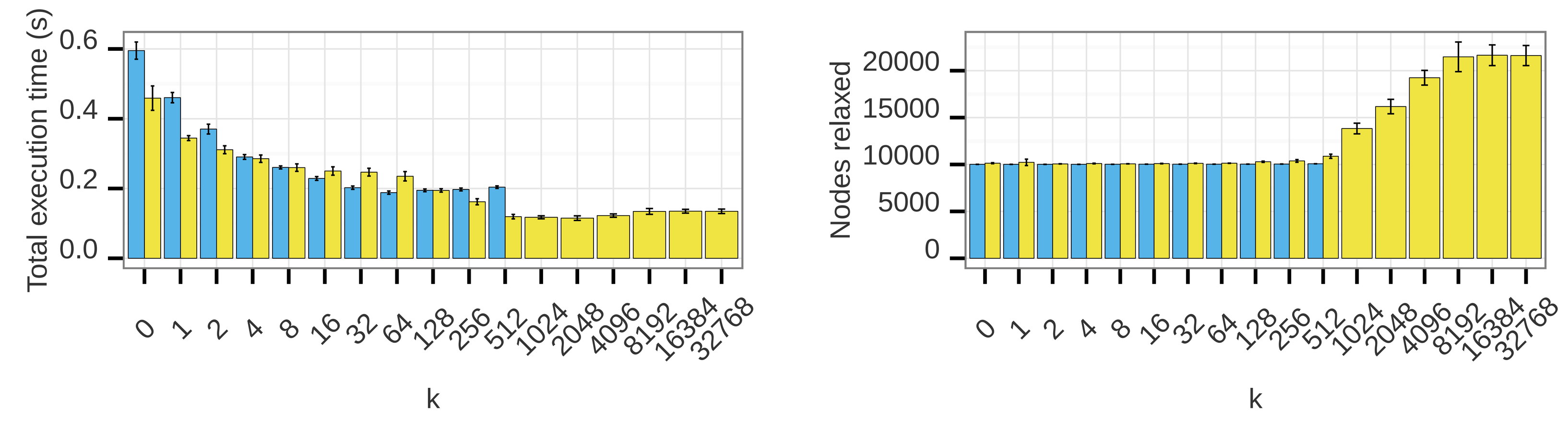
Each thread maintains its own list of tasks of length at most k. After k tasks were added to the list, the list is linked to a global list, thus making it public. Each thread maintains a serial priority queue with references to all tasks in the global list and its local list. If both lists are empty, a thread will *spy* tasks from another thread. Spying is similar to stealing in work-stealing systems, with the main difference being that the original owner still has access to a spied task.

Experiments



Large run-time system overhead in comparison with sequential Dijkstra implementation, but good scalability. Hybrid k-priority queue can scale up to 40 threads. Algorithm becomes memory bandwidth bound for more threads. Amount of node relaxations close to optimal for all data structures except work stealing. (Intel Xeon E7-8850, $n = 10000$, $p = 50\%$)

Influence of Parameter k



High overhead with small k, but close to optimal useless work with k up to 512. Best compromise between scalability and priority guarantees at $k = 512$ in this case. (Intel Xeon E7-8850, $n = 10000$, $p = 50\%$)

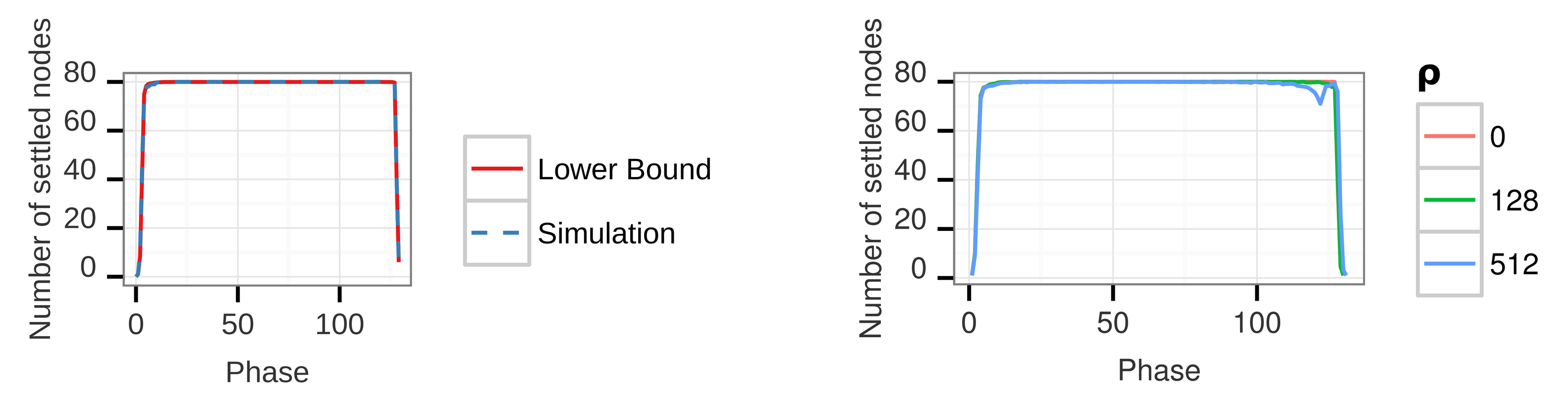
Upper Bounds on Useless Work in SSSP

Analysis for Erdős-Rényi random graphs with parameters n and p: Let W_t be the useless work performed at time t by our algorithm, using an ideal priority queue, and let $h_t(i,j) = d_t(j) - d_t(i)$. We can bound W_t from above as:

$$W_t \leq \sum_{j=1}^{P} \left[1 - \prod_{i=1}^{j-1} \prod_{L=1}^{n-1} \left(1 - \frac{(p\, h_t(i,j))^L}{L!} \right)^{\frac{(n-2)!}{(n-1-L)!}} \right] .$$

We adapt this bound to ρ-relaxed priority queues by appropriately changing the range of the sum.

Simulation



A simulation based on our theoretical model confirms the obtained upper bounds on useless work. Throughout most of the execution the expected useless work is very small. ($n = 10000$, $P = 80$, $p = 50\%$)

Results

- Efficient parallel implementations of Dijkstra's algorithm with ρ-relaxed priority queues.
- Hybrid k-priority queue provides the best compromise between scalability and priority guarantees for SSSP.
- Theoretical model relies on a weaker formulation of ρ-relaxation, thus allowing for more relaxed priority queues in future work.
- Code is available as part of the open source task-scheduling framework *Pheet*. (`www.pheet.org`)