

Enabling High-Level Graph Processing via Dynamic Tasking

Maurizio Drocco, Vito Giovanni Castellana, Marco Minutoli, Antonino Tumeo, John Feo

MOTIVATION

- Data-intensive computing induces **irregular** and **unbalanced** workloads
- e.g., **graph processing** yields fine-grained unpredictable access patterns
- **large-scale** problems require distributed systems (e.g., multi-core clusters)
- managing dynamic workloads while dealing with **load unbalance** on HPC clusters is an open problem

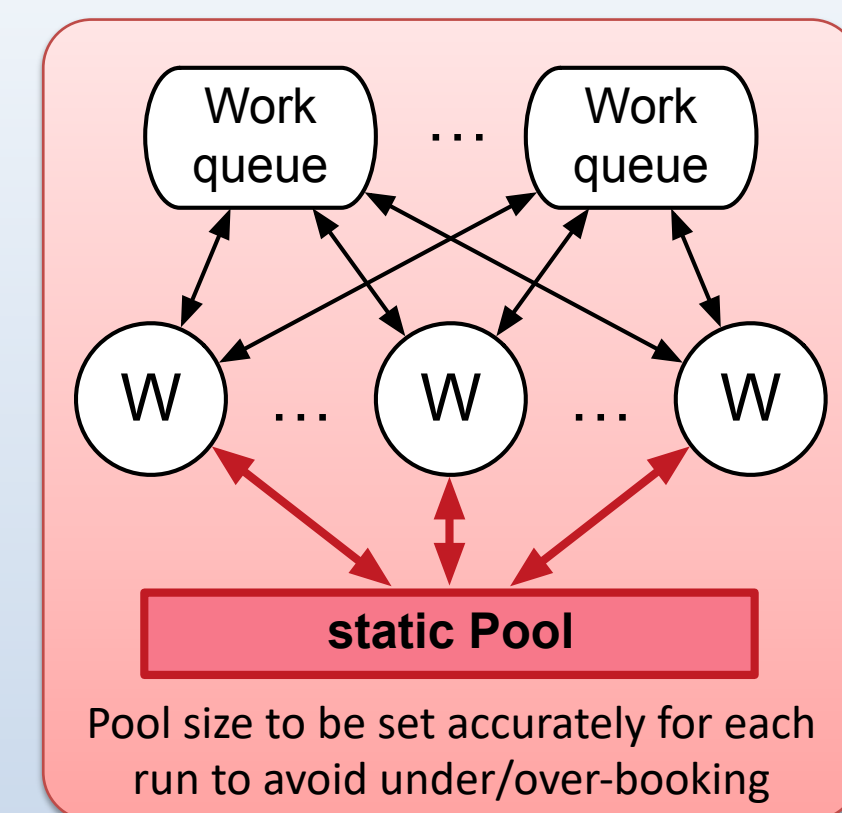
CHALLENGE

- Several *task-oriented runtime systems* chase high performance by **multithreading** and **asynchronous coordination**
- queueing systems play a key role to provide efficient dispatching of tasks, both within and across nodes
- the distribution of tasks across the nodes varies broadly over time, triggering the pitfalls of statically managed resources in the system
- **the goal**: designing strategies for task queueing (on-node) and spawning (across nodes) to deal with dynamic and unbalanced workloads

CONTRIBUTION

- A novel **task allocator** providing optimal on-node memory usage with no need for configuration tuning
- elastic **off-node spawning** based on executor-side buffering and backpressure sensing
- Proof-of-concept implementation within the **GMT** tasking system
- performance evaluation on **high-level graph processing** with SHAD

APPROACH



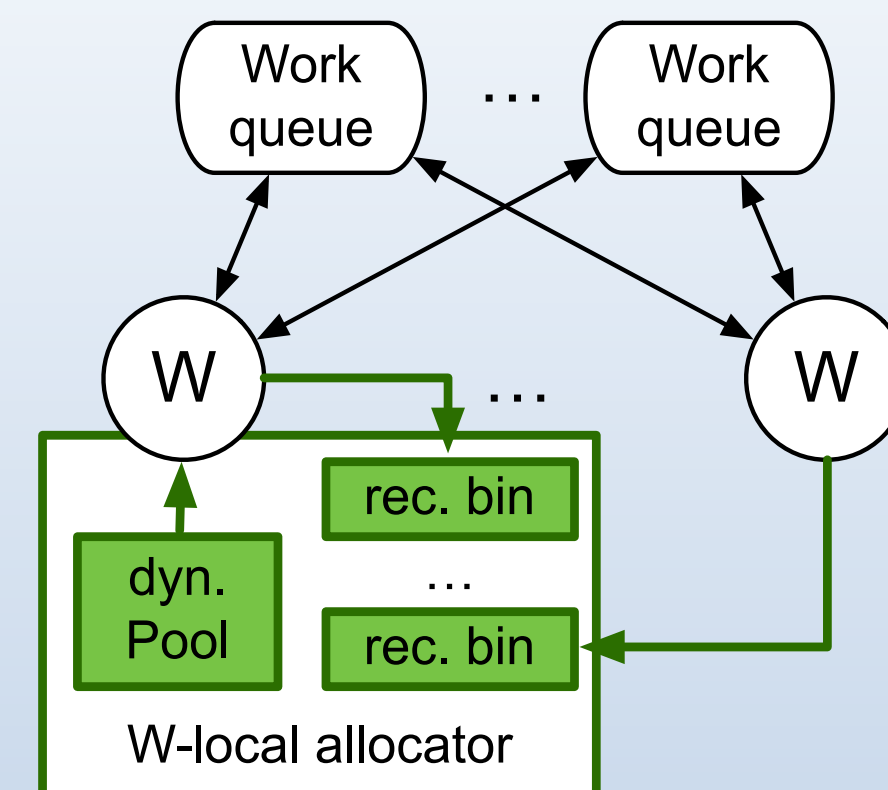
OFF-NODE EXECUTION

COMMON CONFIGURATION

A **reservation protocol** is used to allocate remote execution slots, avoiding overflows at the price of extra communication traffic.

DYNAMIC SCHEMA

Tasks are spawned and possibly buffered only at the executor side, relying on **backpressure feedback** and **sensing** to avoid overflows.



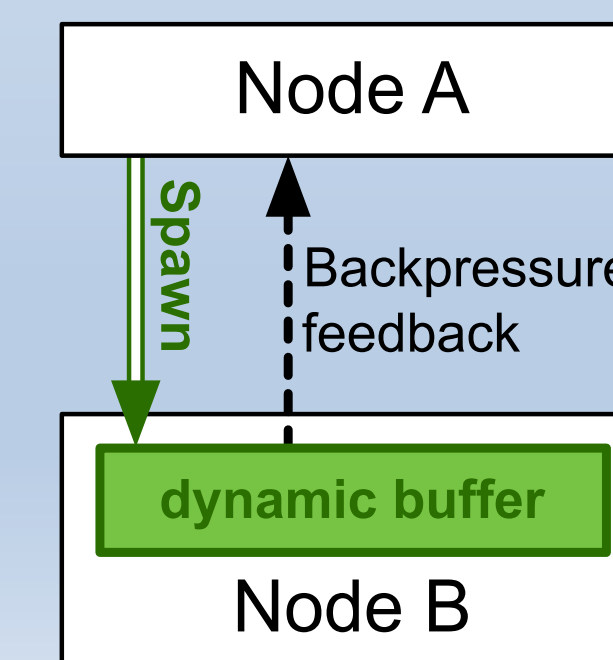
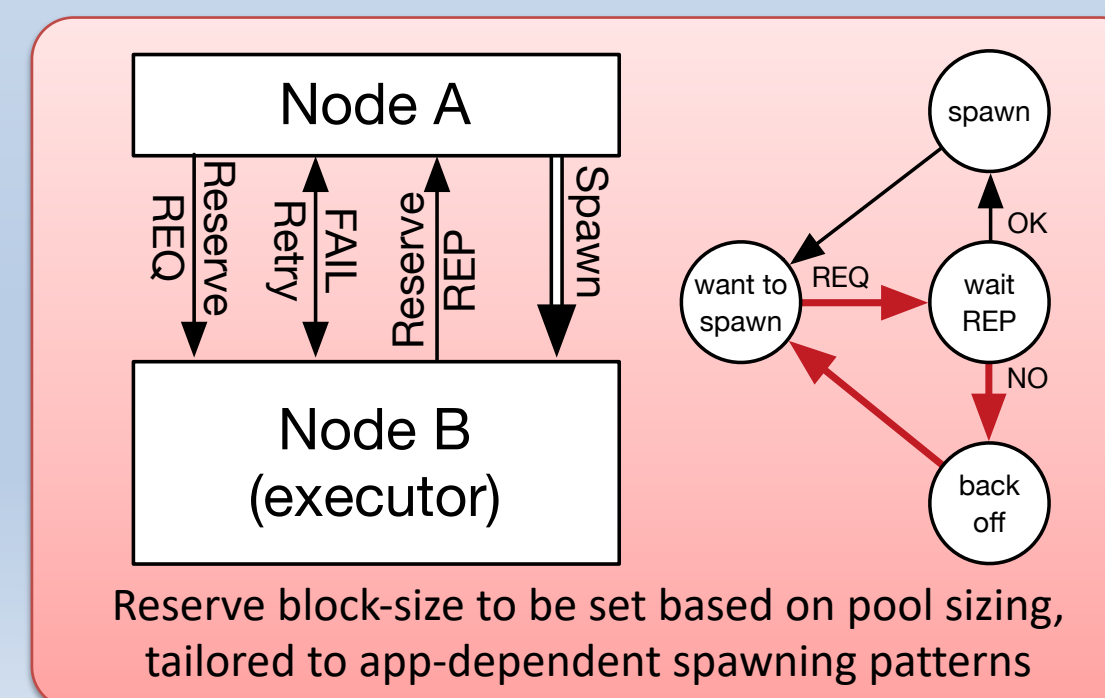
ON-NODE ALLOCATION

COMMON CONFIGURATION

A single fixed-size MPMC pool serves as centralized task allocator, suffering from **high MPMC contention**.

DYNAMIC SCHEMA

In our decentralized schema, tasks are allocated by each worker from **dynamic private pools** and recycled through one-to-one, fast, cache-friendly **SPSC recycle bins**.

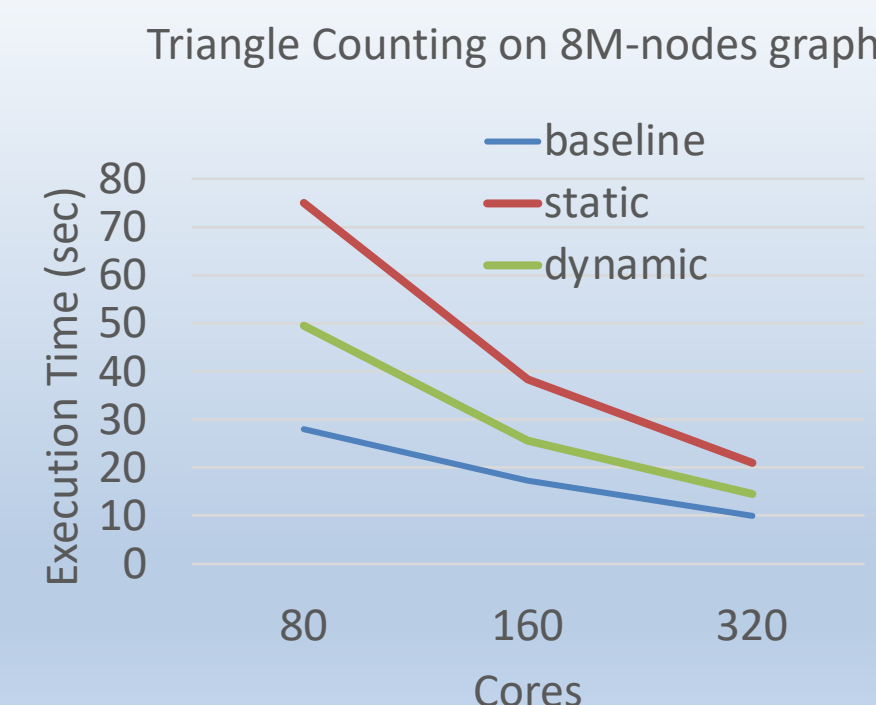


EVALUATION

- graph **triangle counting** on a 16-node cluster of 20-core Intel(R) Xeon(R) E5-2680 v2 @2.80GHz

CONFIGURATIONS

- **static**: high-level SHAD + GMT backend
- **dynamic**: high-level SHAD + GMT backend, with the novel dynamic on-node allocation and remote execution schemas
- **baseline**: low-level custom GMT implementation



STRONG SCALING (dynamic)

Cores	16M nodes	32M nodes
80	108.16 s	226.91 s
160	55.45 s	122.25 s
320	32.16 s	68.46 s

- average **1.5x speedup** vs static GMT
- less memory waste
- less configuration parameters

ABOUT GMT

Global Memory and Threading

FEATURES

- Originally Designed for Partitioned Global Address Space (PGAS)
- API: fork/join control model with constructs for task and data parallelism
- Key performance aspects: fine-grained **multithreading** combined with **data aggregation** for latency tolerance

IMPLEMENTATION

- Three types of threads on each GMT node: communication servers for MPI networking, workers for task execution, and helpers for serving remote tasking
- Tasking system based on multiple task queues shared by worker threads + an off-node reservation/execution protocol

LIMITATIONS

- Since the tasking system is **all-static**, GMT hardly responds to variations in time of the task distribution across nodes
- Without accurate **tuning** of configuration parameters (e.g., queue sizing), the performance may degrade up to 10x

github.com/pnnl/GMT

ABOUT SHAD



SCALABLE
HIGH-PERFORMANCE
ALGORITHMS &
DATA-STRUCTURES

- The PNNL library of HPC containers and algorithms
- Inspired by C++17 STL
- Supports applications on commodity HPC clusters with a NUMA abstraction
- Modular design with multiple backends: plain C++, Intel TBB, **GMT**

github.com/pnnl/SHAD