Optimizing Distributed Load Balancing for Workloads with Time-Varying Imbalance

Jonathan Lifflander (SNL)
Nicole Slattengren (SNL)
Philippe P. Pébaÿ (NGA)
Phil B. Miller (IC)
Francesco Rizzi (NGA)
Matt T. Bettencourt* (NVI)

Based on a paper published at CLUSTER 2021

* Work performed while at SNL

NGA = NexGen Analytics, Inc
SNL = Sandia National Labs
IC = Intense Computing
NVI = NVidia

SAND2021-6350

Sandia National Laboratories

Exceptional service in the national interest

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

What is DARMA?

A toolkit of libraries to support incremental AMT (Asynchronous Many-Task) adoption in production scientific applications

Module	Name	Description
DARMA/**vt**	Virtual Transport	MPI-oriented AMT HPC runtime
DARMA/**checkpoint**	Checkpoint	Serialization & checkpointing library
DARMA/**detector**	C++ trait detection	Optional C++14 trait detection library
DARMA/**LBAF**	Load Balancing Analysis Framework	Python framework for simulating LBs and experimenting with load balancing strategies
DARMA/**checkpoint-analyzer**	Serialization Sanitizer	Clang AST frontend pass that generates serialization sanitization at runtime

DARMA Documentation: *https://darma-tasking.github.io/docs/html/index.html*

Background

➤ Context of AMT development

- MPI has dominated as a distributed-memory programming model (SPMD-style)
  - Deep technical and intellectual ecosystem

- Production Sandia applications are developed atop large MPI libraries/toolkits
  - e.g., Trilinos (linear solvers, etc.); STK (Sierra ToolKit) for meshing
  - There's little chance that the litany of MPI libraries used by production apps at Sandia will be rewritten to target an AMT runtime

- Conclusion
  - We must coexist and provide transitional AMT runtimes to **demonstrate incremental value**

Motivation

➤ Philosophy

- Our philosophy:
  - AMT runtimes must be highly interoperable allowing parts of applications to be incrementally overdecomposed
  - Transition between MPI/AMT must be inexpensive; expect frequent context switches from MPI to AMT runtime (many times, every timestep!)

- For domain developers:
  - Provide SPMD constructs in AMT runtimes for a natural transition while retaining asynchrony
  - Coexist with existing diversity of on-node techniques
    - CUDA, OpenMP, Kokkos, etc.
  - Allow MPI operations to be safely interwoven with AMT execution
  - We've found that serialization and checkpointing is a backdoor into introducing AMT libraries

- Paper reference
  - J. Lifflander, P. Miller, N. L. Slattengren, N. Morales, P. Stickney and P. P. Pébaÿ, ***Design and Implementation Techniques for an MPI-Oriented AMT Runtime***, *2020 SC Workshop on Exascale MPI (ExaMPI)*, 2020, pp. 31-40, doi: 10.1109/ExaMPI52011.2020.00009

Premises

- Types of LB strategies
  - Centralized
    - Send all task graph to a single node and then scatter results
    - They don't scale (might work for 100s of processes)
    - Cost thus limits the value of running (must run infrequently)
  - Hierarchical
    - Form groups of nodes, spanning trees, etc.
    - log(P) scalable, but still limited as system sizes increase
  - Fully Distributed
    - Very inexpensive and scalable
    - Historically difficult to get a good load distribution due to limited information
- We improve upon an fully distributed strategy inspired from epidemic algorithms
  - *H. Menon and L. Kalé, "A distributed dynamic load balancer for iterative applications," in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ser. SC '13.*

LBAF – Load Balancing Analysis Framework

- Simulate load balancers to test new distributed LB algorithms sequentially in Python
- Research Workflow
  - Run application in VT and output LB data (1 per rank)
    - Phases, subphases, communication
  - Feed LB data into LBAF to test new load balancer algorithms
    - Explore new strategies
  - Output new mapping from LBAF based on strategy's determination
  - Run application in VT with the generated mapping from LBAF
    - We have a special LB that follows what it reads from a set of mapping files


```
{
  "phases": [
    {
      "id": 0,
      "tasks": [
        {
          "entity": {
            "collection_id": 7,
            "home": 0,
            "id": 0,
            "index": [
              0
            ],
            "type": "object"
          },
          "node": 0,
          "resource": "cpu",
          "subphases": [
            {
              "id": 0,
              "time": 0.026394367218017578
            }
          ],
          "time": 0.026394367218017578
        },
        {
          "entity": {
            "collection_id": 7,
            "home": 0,
            "id": 4294967296,
            "index": [
              1
            ],
            "type": "object"
          },
          "node": 0,
          "resource": "cpu",
          "subphases": [
            {
              "id": 0,
              "time": 0.027690887451171875
            }
          ],
          "time": 0.027690887451171875
        }
      ]
    },
```


Open source: `https://github.com/DARMA-tasking/LB-analysis-framework`

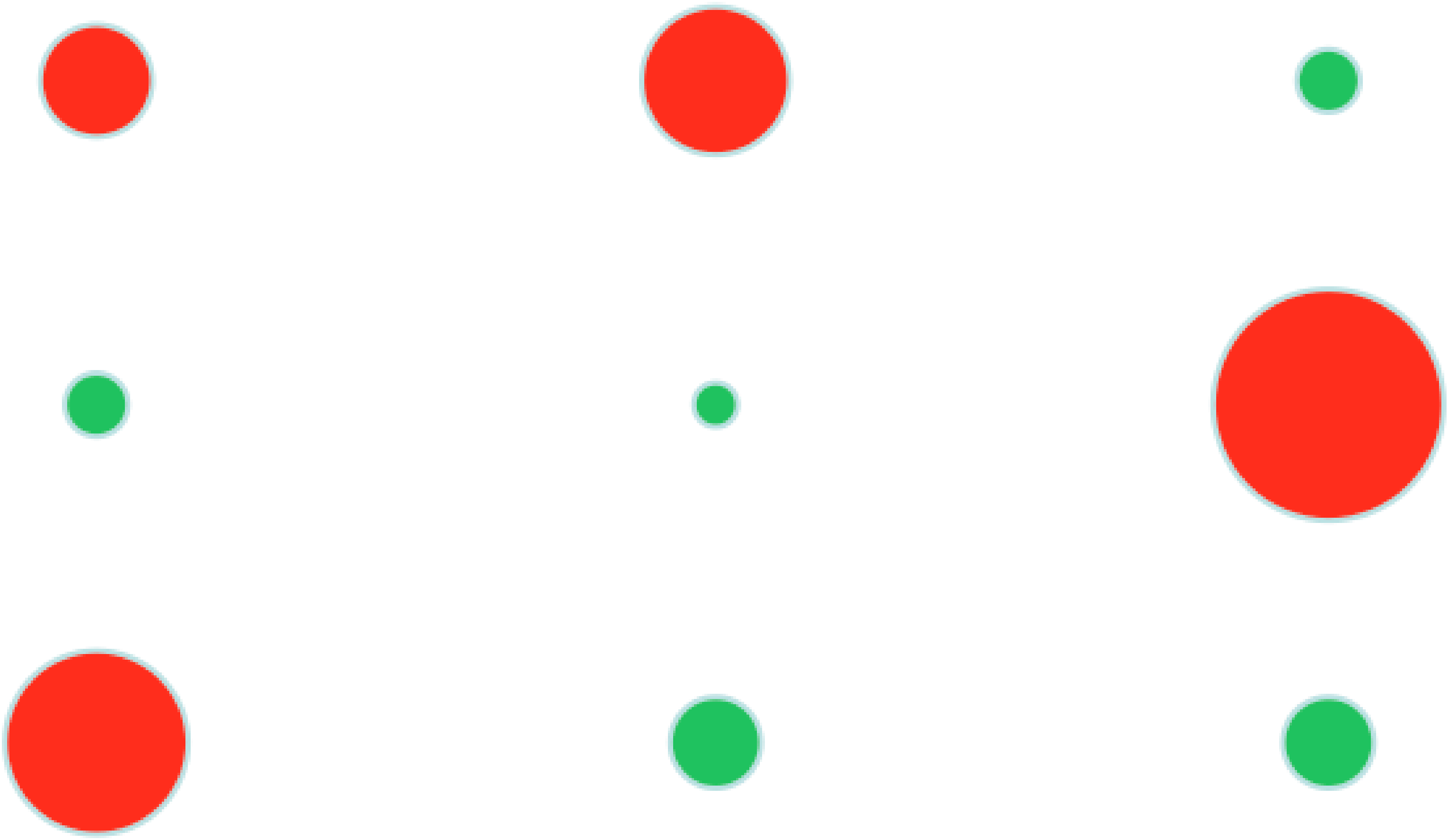
Base Algorithm

- Fully distributed
  - Inspired from epidemic algorithms
  - No central coordination or tree/group building
- Operates with two distinct stages
  - Gossip --- spread information by randomly selecting ranks to send load data
  - Transfer --- use information gained to make transfers from overloaded to underloaded to reduce imbalance

Base Algorithm

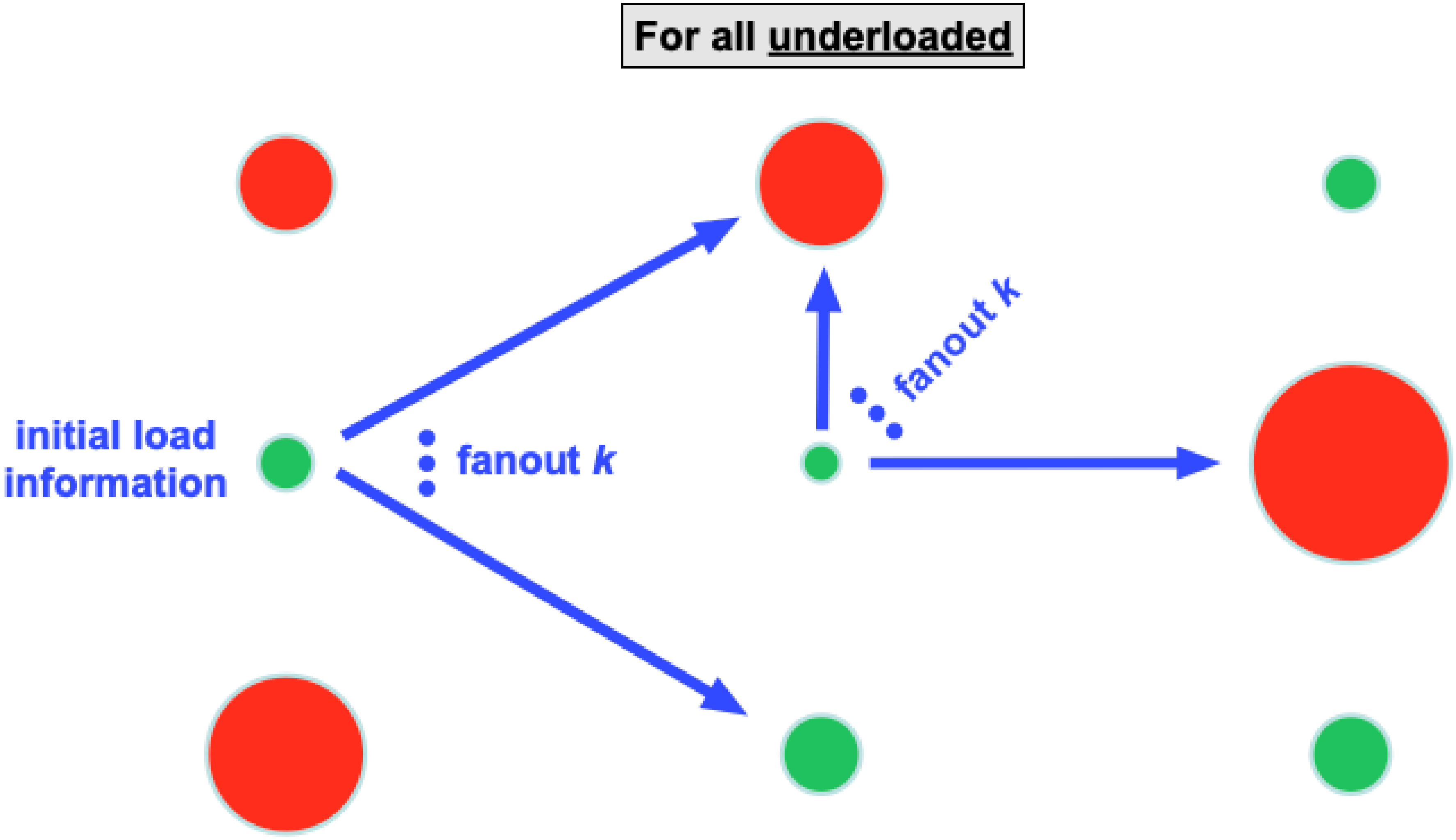
- ➤Initialization

$$L_{underloaded} < L_{average} < L_{overloaded}$$



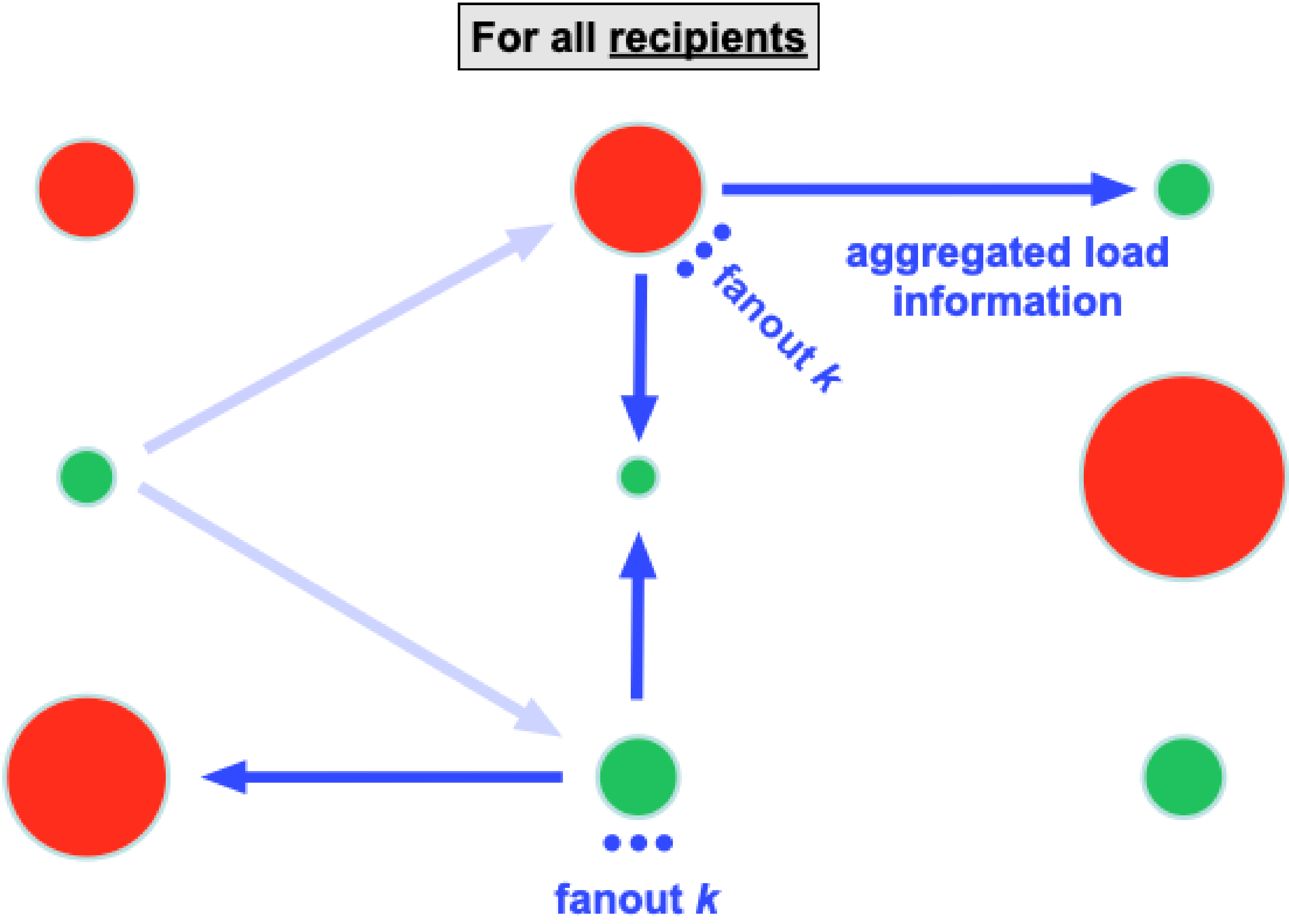
Base Algorithm

➤Gossiping Phase – Round 1



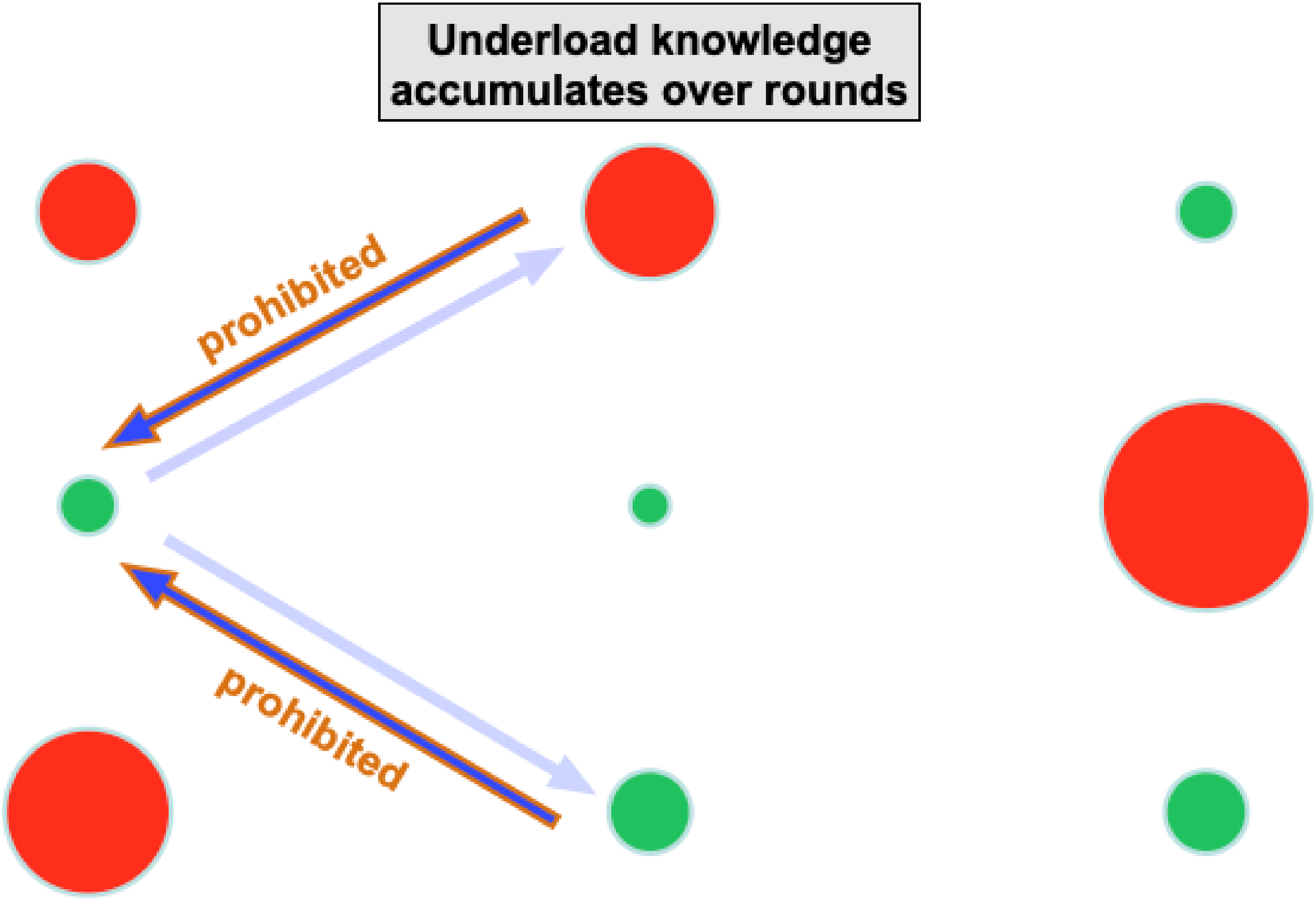
Base Algorithm

- ➤Gossiping Phase – Rounds 2,…n



Base Algorithm

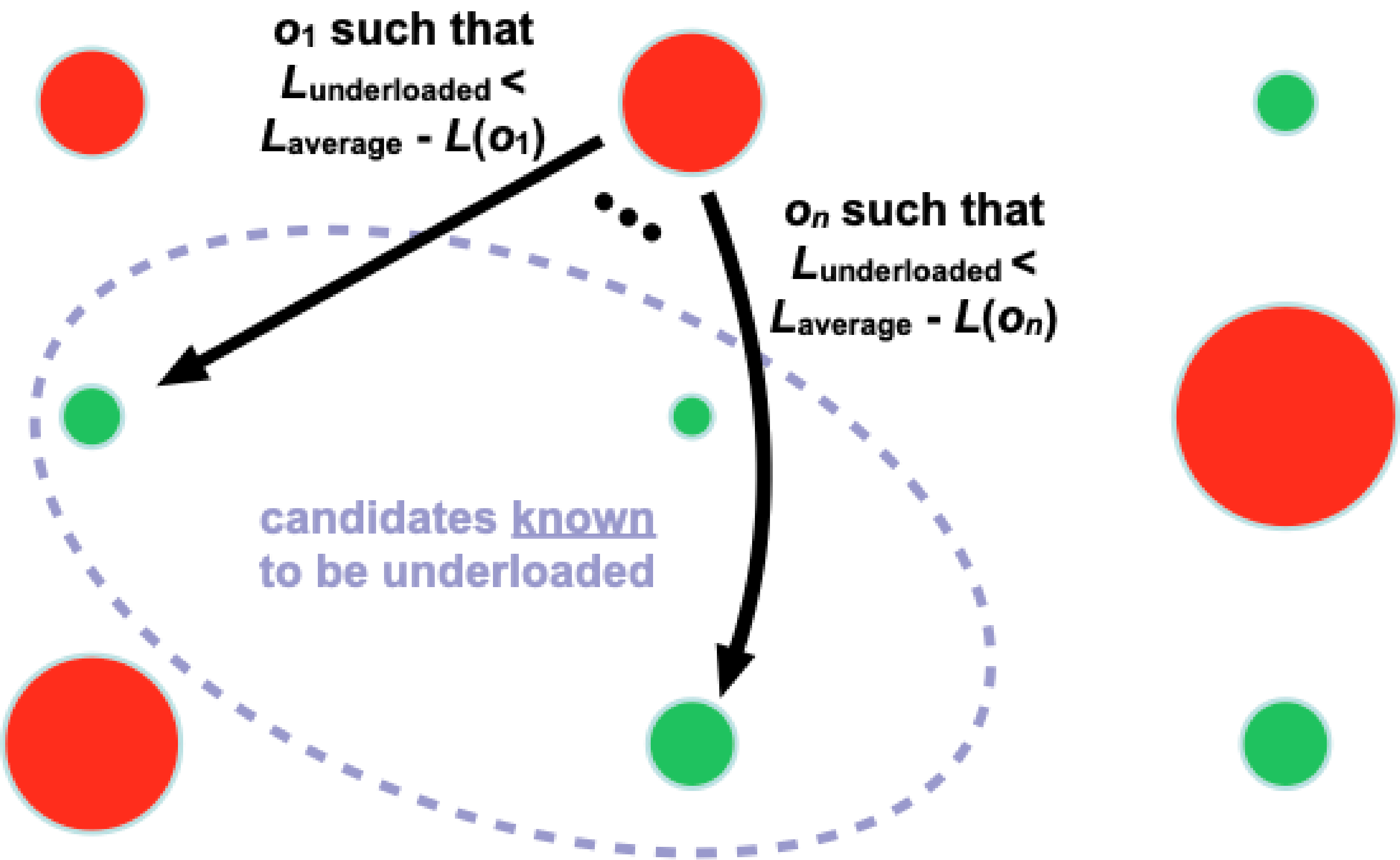
➤Gossiping Phase – Informed Selection



Base Algorithm

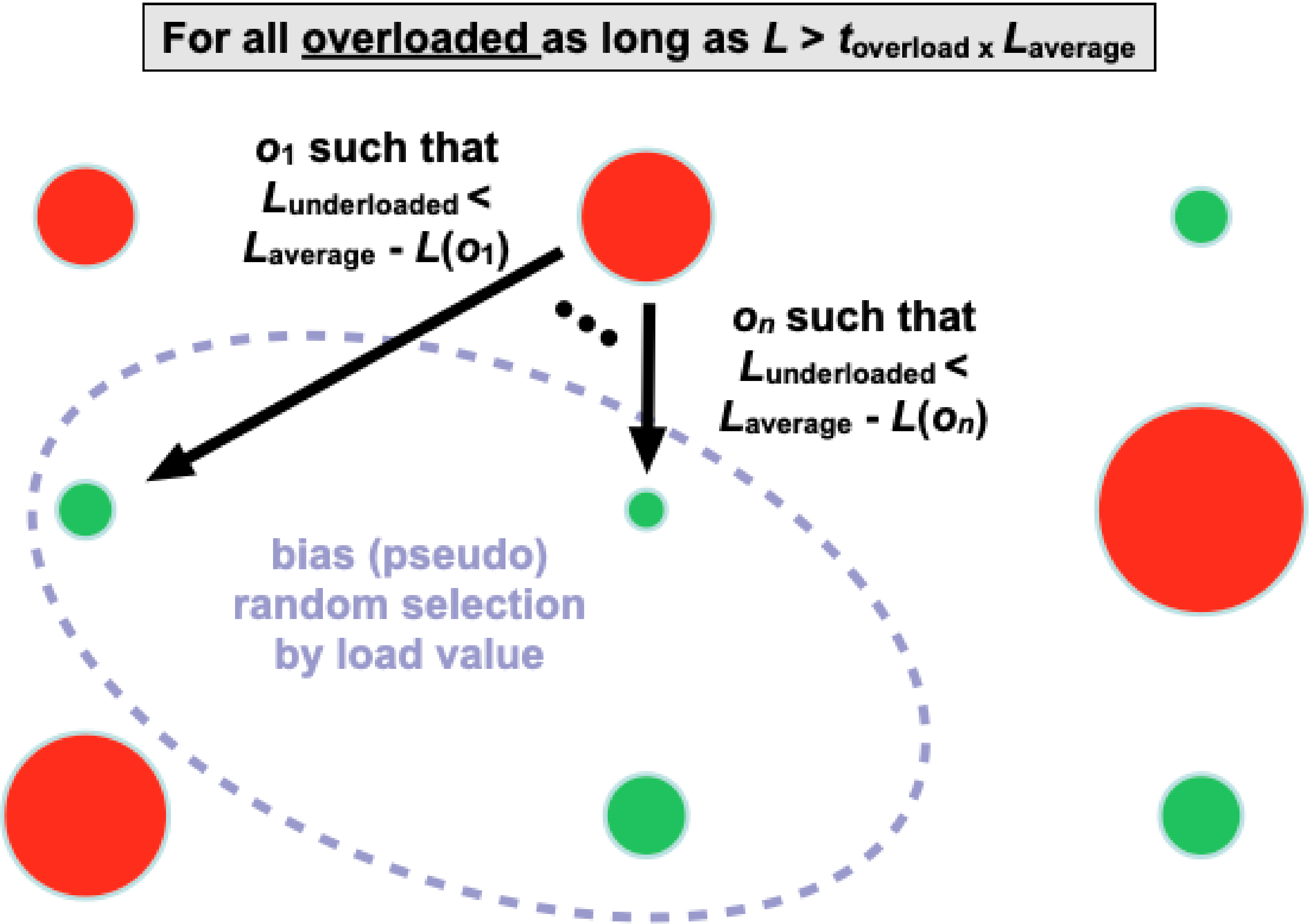
➤Transfer Phase

For all <u>overloaded</u> as long as $L > t_{overload} \times L_{average}$



Base Algorithm

➤Transfer Phase



Improvements

➤Iteration and Trials

- Apply the algorithm iteratively to keep improving imbalances before performing transfers
- Perform multiple trials of the iteration process to increase the odds of avoiding local minima

Algorithm 3 Iterative refinement of task-rank mapping.

$T^p_{\text{orig}} \leftarrow T^p$
for $t \leftarrow 1, n_{\text{trials}}$ **do**
    $T^p \leftarrow T^p_{\text{orig}}$ ▷ Reset for each trial
    $M^p \leftarrow \emptyset$
    $\textsc{Target}^p() \leftarrow \emptyset$
    for $i \leftarrow 1, n_{\text{iters}}$ **do**
        $\textsc{Inform}(\ell_{\text{ave}}, \ell^p, 0)$
        $\textsc{Transfer}(\ell_{\text{ave}}, \ell^p)$
        Evaluate $\mathcal{I}_{\text{proposed}}$ using Eqn. 1
        Save $T^p_{\text{best}}, M^p_{\text{best}}, \textsc{Target}^p_{\text{best}}$ for lowest $\mathcal{I}_{\text{proposed}}$
    end for
end for
Execute transfers defined by $T^p_{\text{best}}, M^p_{\text{best}}, \textsc{Target}^p_{\text{best}}()$

Improvements

➤Recomputing the CMF during Transfer

- CMF -- cumulative mass function
- Probability distribution built during transfer stage to determine which rank to try to transfer work
- Sampled for each task to select a possible candidate for transfer
- As we assign new tasks to underloaded processors, we rebuild the CMF
  - As tasks are moved, other underloaded processors may be more profitable to select

Algorithm 2 The transfer stage to choose tasks for migration based on partial knowledge gathered in the inform stage.

$h \leftarrow$ threshold ▷ Constant value
function TRANSFER(ℓ_{ave}, ℓ^p)
Require: $\sum_{n=1}^{|T^p|}(\text{LOAD}(T_n^p)) \equiv \ell^p$
  $O^p \leftarrow$ORDERTASKS(T^p, ℓ_{ave}, ℓ^p) ▷ Traversal order
  $n \leftarrow 0$ ▷ Index of task to try
  if CMF is original **then** $F \leftarrow$ BUILDCMF(ℓ_{ave})
  while $\ell^p > h \times \ell_{\text{ave}} \wedge n < |O^p|$ **do** ▷ Overloaded
    if CMF is modified **then** $F \leftarrow$ BUILDCMF(ℓ_{ave})
    $o_x \leftarrow O_n^p$
    $p_x \in S^p$ using F ▷ Pick rank sampling CMF
    $\ell_x \leftarrow \text{LOAD}_j^p \mid p_j \equiv p_x$ ▷ Known load of rank
    if EVALUATECRITERION(ℓ_x, o_x, ℓ_{ave}, ℓ^p) **then**
      $\ell_x \leftarrow \ell_x + \text{LOAD}(o_x)$
      $\ell^p \leftarrow \ell^p - \text{LOAD}(o_x)$
      $T^p \leftarrow T^p \setminus \{o_x\}$
      $M^p \leftarrow M^p \cup \{o_x\}$ ▷ Record proposed transfer
      $\text{TARGET}^p() \leftarrow \text{TARGET}^p() \cup \{o_x \mapsto p_x\}$
    end if
    $n \leftarrow n + 1$
  end while
end function
function BUILDCMF(ℓ_{ave})
  if CMF is original **then**
    $\ell_s \leftarrow \ell_{\text{ave}}$
  else if CMF is modified **then** ▷ Described in § V-C
    $\ell_s \leftarrow \max(\ell_{\text{ave}}, \max(\text{LOAD}^p()))$
  end if
  $z \leftarrow \sum_{i=1}^{|S^p|}\left(1 - \frac{\text{LOAD}^p(i)}{\ell_s}\right)$
  $p_i \leftarrow \frac{1}{z}\left(1 - \frac{\text{LOAD}^p(i)}{\ell_s}\right)$
  $\varphi_j \leftarrow \sum_{i=1}^{j} p_i$
  $F \leftarrow \{\varphi_i\}_{i=1}^{|S^p|}$
  return F
end function

Sandia National Laboratories

Improvements

➤Relaxing the objective function during transfer

- Analysis under iteration using the Load Balancing Analysis Framework (LBAF) for a synthetic problem with huge amounts of imbalance
  - Using the original objective function

Iteration (index)	Transfers (count)	Rejected (count)	Rejection Rate (%)	Imbalance ($\mathcal{I}$)
0	-	-	-	280
1	9084	154 931	94.46	187
2	4	1654	99.76	187
3	1	1130	99.91	187
4	7	2682	99.74	185
5	6	2396	99.75	183
6	2	1143	99.83	183
7	1	1041	99.90	183
8	0	882	100.0	183
9	0	882	100.0	182
10	3	1405	99.79	182

Improvements

➤Relaxing the objective function during transfer

- The high rejection rate hints that the objective function is too strict!
- Thus, we relax the objective function to allow transfers as long as the global max load doesn't increase
- We provide a proof of optimality in our paper for our new, relaxed criterion

```
function EvaluateCriterion(ℓ_x, o_x, ℓ_ave, ℓ^p)
    if Criterion is original then
        return ℓ_x + Load(o_x) < ℓ_ave
    else if Criterion is relaxed then ▷ Described in § V-C
        return Load(o_x) < ℓ^p − ℓ_x
    end if
end function
```

Iteration (index)	Transfers (number)	Rejected (number)	Rejection rate (%)	Imbalance ($\mathcal{I}$)
0	-	-	-	280
1	11 292	648	5.43	3.34
2	4044	3603	47.12	1.60
3	2201	3412	60.79	0.873
4	1324	3586	73.03	0.632
5	765	3171	80.56	0.632
6	410	2969	87.87	0.626
7	247	2794	91.88	0.626
8	159	2749	94.53	0.626
9	120	2682	95.72	0.626
10	72	2643	97.35	0.623

Improvements

➤Task ordering

- During the transfer stage, each overloaded process must select tasks to try to transfer
  - Originally, arbitrary task selection was proposed
  - We propose three new mappings
    - Strawman (most load intensive)
    - Fewest migrations (algorithm 5)
      - Pick smallest task from overloaded that will bring load down to average
    - Most Lightweight Tasks (algorithm 6)
      - Find the “marginal” task, the most load intensive of lightweight tasks that must be migrated for a rank to not be overloaded

Algorithm 5 The algorithm for ordering tasks for selection that minimizes the number of migrations during the transfer phase (see line 3 in Algorithm 2).

function ORDERTASKS_FEWESTMIGRATIONS(T^p, ℓ_{ave}, ℓ^p)
  $\ell_{\text{ex}} \leftarrow \ell^p - \ell_{\text{ave}}$ ▷ Excess load on this rank
  if $\max_i T_i^p < \ell_{\text{ex}}$ **then**
    return ORDERTASKS_DESCENDING(T^p, ℓ_{ave}, ℓ^p)
  end if
  $\ell_{\text{cut}} \leftarrow \min_i \{T_i^p \mid T_i^p > \ell_{\text{ex}}\}$ ▷ Cutoff load
  c $\leftarrow$ **lambda** $(a, b) \mapsto$ { ▷ Load sort comparator
    if LOAD$(a) \leq \ell_{\text{cut}} \wedge$ LOAD$(b) \leq \ell_{\text{cut}}$
      then return LOAD$(a) >$ LOAD(b)
      else return LOAD$(a) <$ LOAD(b)
  }
  return SORT(T^p,c)
end function

Algorithm 6 The algorithm for ordering tasks for selection that picks the most lightweight tasks first during the transfer phase (see line 3 in Algorithm 2).

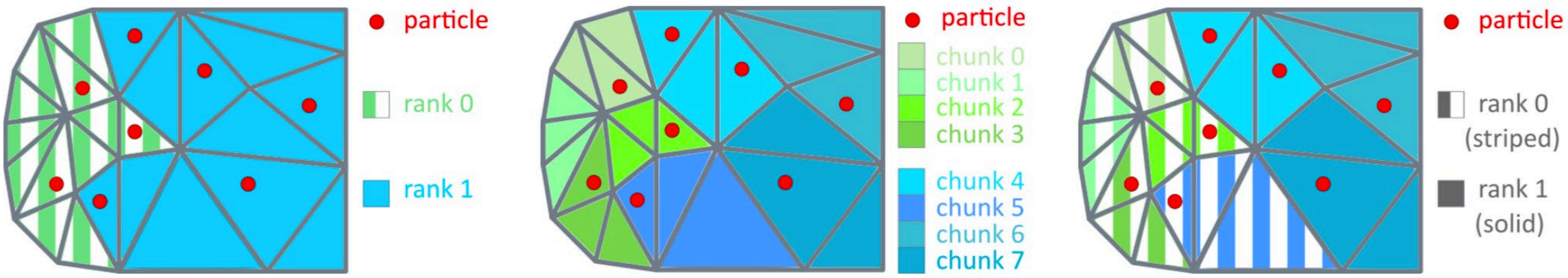
function ORDERTASKS_LIGHTEST(T^p, ℓ_{ave}, ℓ^p)
  $\ell_{\text{ex}} \leftarrow \ell^p - \ell_{\text{ave}}$ ▷ Excess load on this rank
  $c_1 \leftarrow$ **lambda** $(a, b) \mapsto$ ▷ Sort ascending to start
  { **return** LOAD$(a) <$ LOAD(b) } ▷ Ascending load
  $S^p \leftarrow$ SORT(T^p,c_1)
  $\ell_{\text{marg}} \leftarrow \min_j \left\{ S_j^p \,\middle|\, \sum_{i=0}^{j} S_i^p \geq \ell_{\text{ex}} \right\}$ ▷ Partial sum
  $c_2 \leftarrow$ **lambda** $(a, b) \mapsto$ { ▷ Final sort comparator
    return if LOAD$(a) \leq \ell_{\text{marg}} \wedge$ LOAD$(b) \leq \ell_{\text{marg}}$
      then LOAD$(a) >$ LOAD(b)
      else LOAD$(a) <$ LOAD(b)
  }
  return SORT(S^p,c_2)
end function

Implementation in VT

- We have built a production load balancer with all these improvements called *TemperedLB*
  - Implements trials, iterations, old/new CMF, and several transfer criterion
  - Open source
  - Can be found here: `https://github.com/DARMA-tasking/vt`

Application Results

- We evaluate our load balancing algorithm for EMPIRE, an electromagnetic/electrostatic plasma physics next-generation application
  - Initial PIC particle distributions can be spatially concentrated, creating **heavy load imbalance**
  - Particles may move rapidly across the domain, inducing **dynamic workload variation** over time

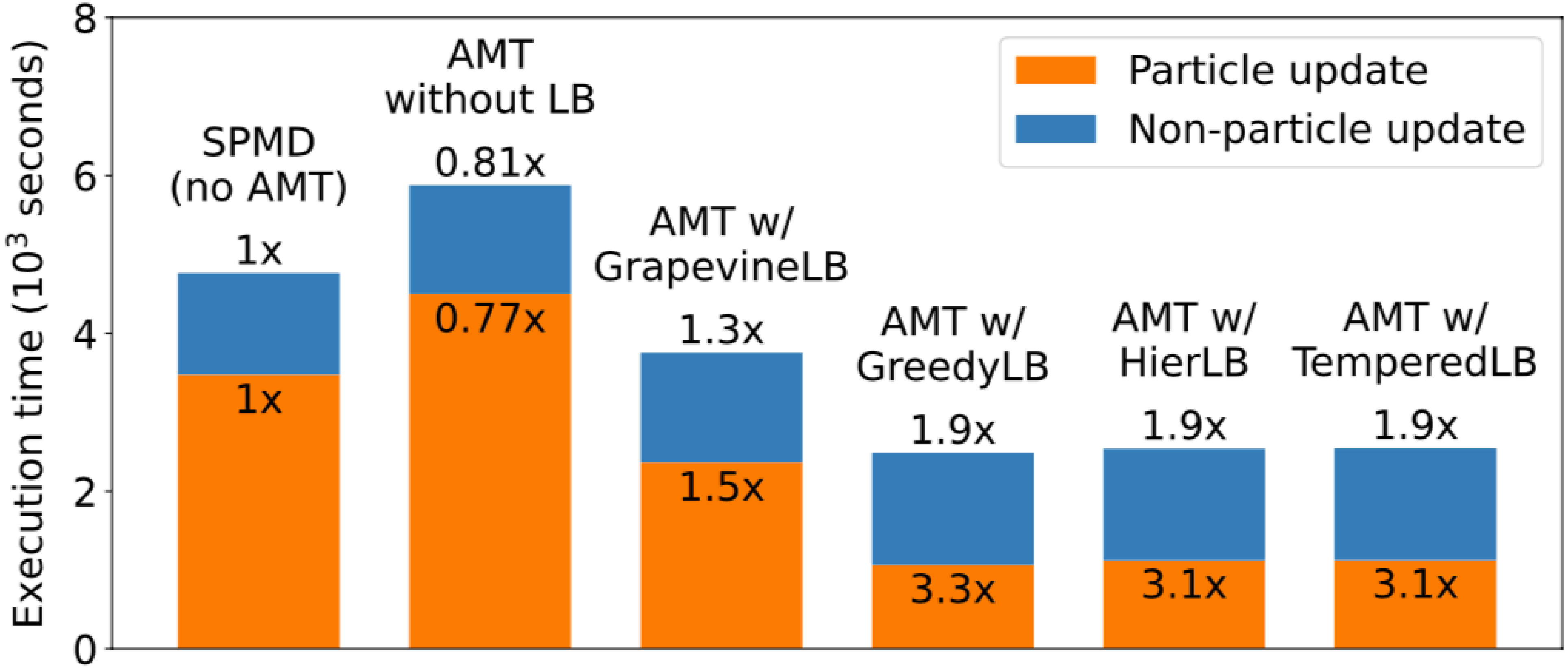


(a) SPMD mesh decomposition (b) Overdecomposition (4 chunks/rank) (c) Overdecomposition with LB

*Actual runs: 24 chunks per MPI rank

Application Results: TemperedLB Performance

➤ B-DOT Problem on ARM cluster

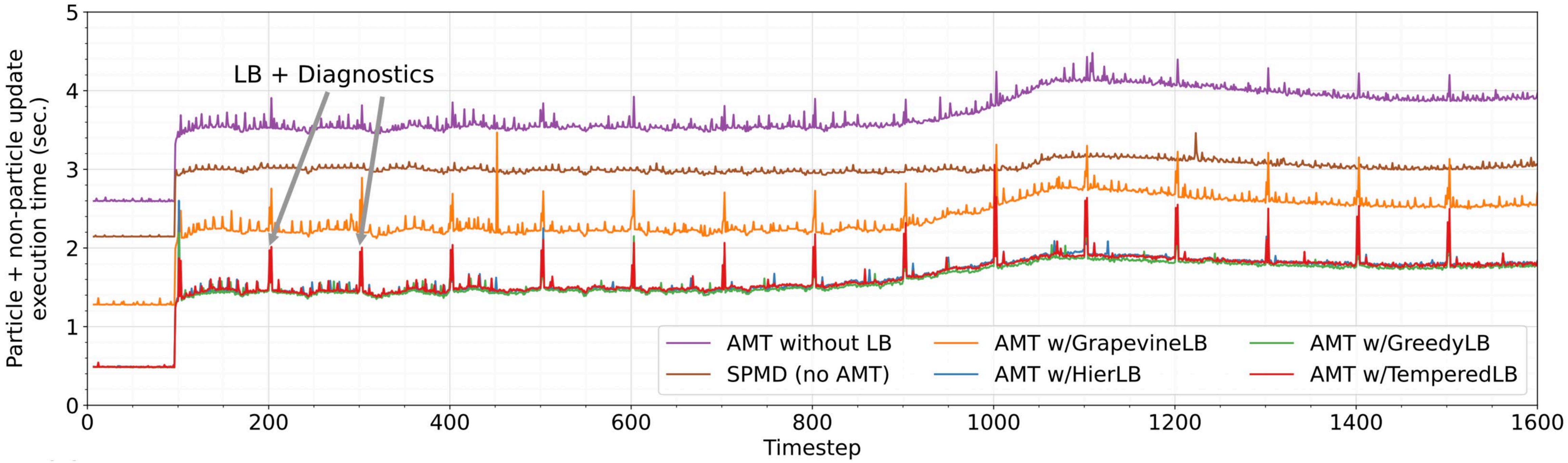


Application Results: TemperedLB Performance

Sandia National Laboratories

- B-DOT Problem on ARM cluster

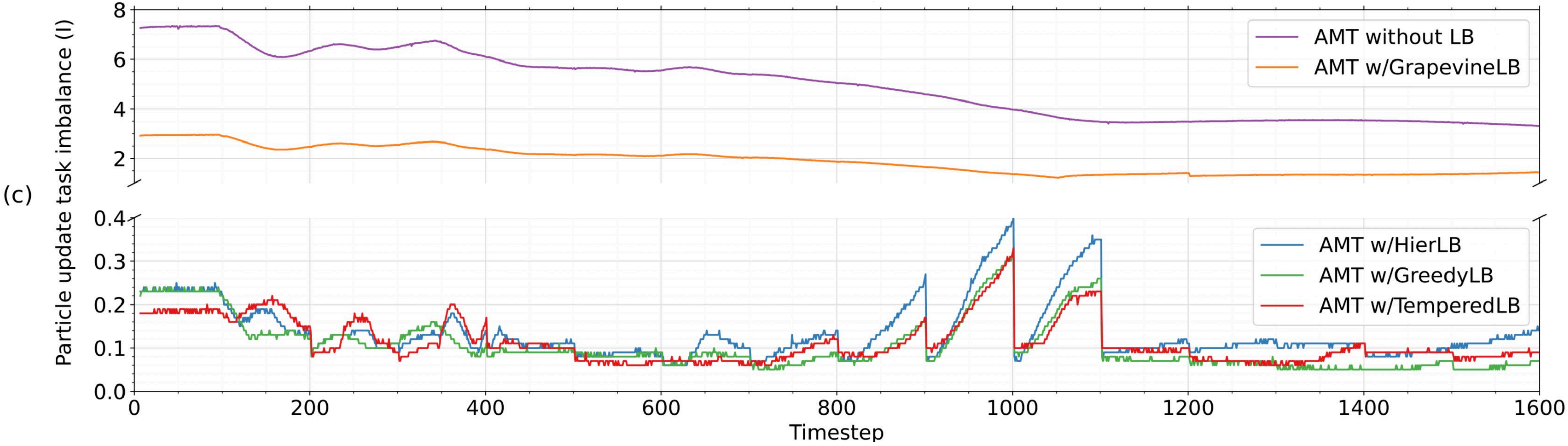
(a)



Application Results: TemperedLB Performance

Sandia National Laboratories

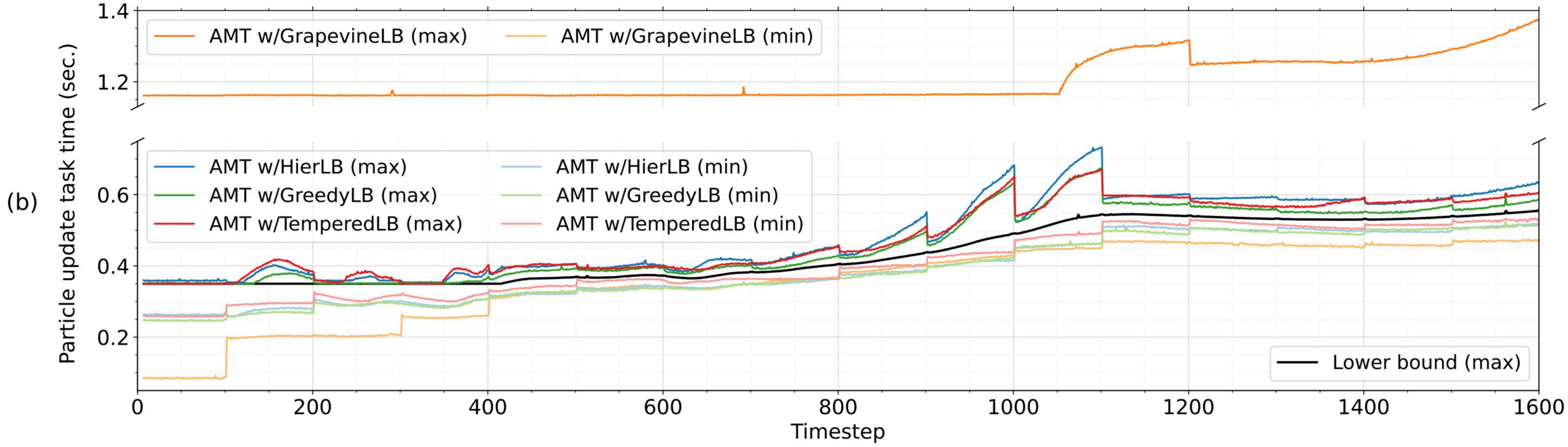
- ➤ B-DOT Problem on ARM cluster



$$\mathcal{I} = \frac{\ell_{\max}}{\ell_{\text{ave}}} - 1$$

Application Results: TemperedLB Performance

➤ B-DOT Problem on ARM cluster

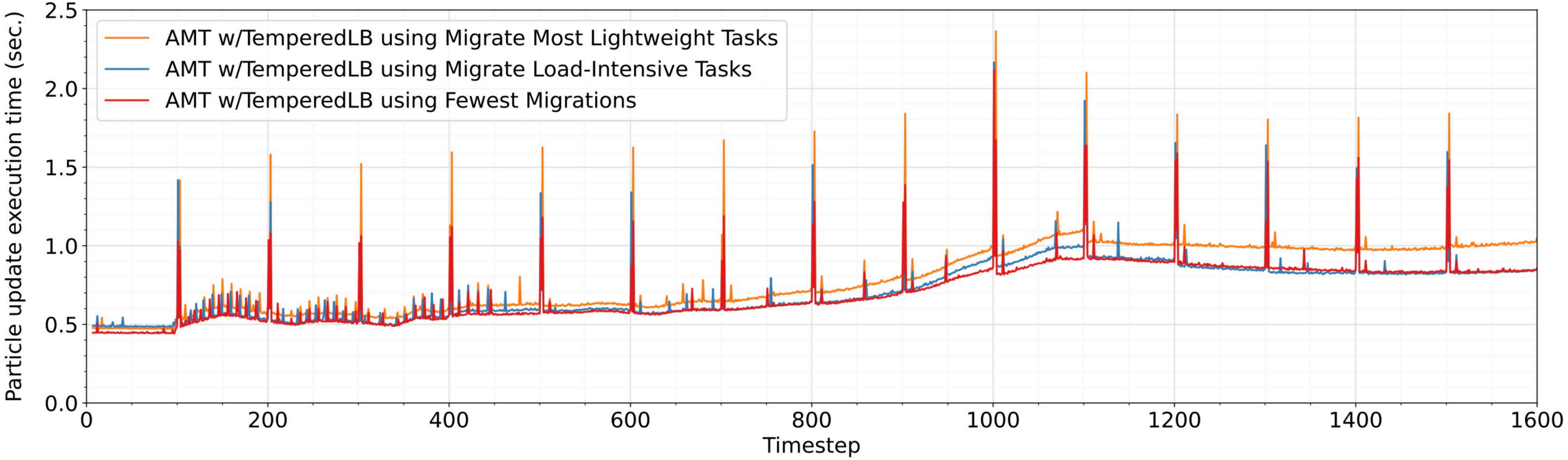


Max: maximum per-rank task load across all ranks;
Min: minimum per-rank task load across all ranks;
Lower bound (max): maximum of ℓ_{ave} and the load of the most load-intensive task.

Application Results: TemperedLB Performance

- B-DOT Problem on ARM cluster

(d)



Concluding Remarks

Sandia National Laboratories

- Main contribution is a set of improvements to seminal work on fully distributed load balancers
  - We have identified some weaknesses in the load transfer phase of the original algorithm
  - We have established some new theoretical results to justify the optimality of our relaxed transfer criterion
- We have demonstrated the real-world benefits in a soon-to-be production application used for PIC computations
- We think that task orderings may improve performance in other contexts
- We are working on further testing our algorithmic improvements on other applications
  - NimbleSM: solid mechanics contact code planned as a pipeline to SierraSM
  - GEMMA: matrix assembly is imbalanced; challenge: not *phase-based* (no timesteps)