Energy Characterization and Optimization in Heartbeat Failure Detection Systems

Pedro Henrique Rosso
p233687@dac.unicamp.br

University of Campinas (Unicamp)
MO632/MC972 – Energy Efficient Computing

Final Project Presentation • November 29th 2021

Overview

- Introduction
  - Problem
  - Objectives
- Heartbeat Systems
- Methodology
  - Proposals
  - Experiments
- Results and Discussions

Introduction

Introduction

- High-Performance Computing
- Fault Tolerance
  - Failure detection mechanisms
    - Heartbeat (ULFM, **OCFTL**)
    - Daemon processes (Reinit++)
- Fault Tolerance in IoT

Problem

- During my Master
  - OCFTL
  - MPI-Based FT library for OmpCluster
- No concern about energy efficiency
- Characterize and think about it focusing energy efficiency

Objectives

- Energy Characterization
  - High resource usage - Parallel Applications
  - Low resource usage
- Impact of heartbeat parameters
- Different communication backends
- Optimizations in current algorithm
- Relationship between Energy Efficiency and Heartbeat Efficiency

Heartbeat Systems

Heartbeat

- Inspired by ULFM’s heartbeat
- Broadcast Propagation

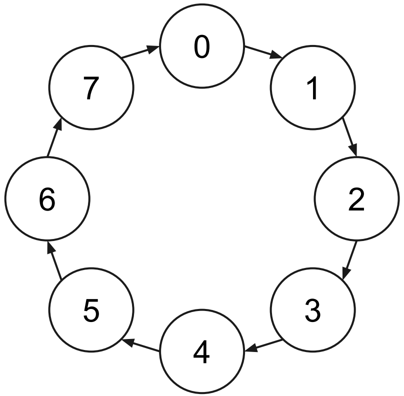
Properties

- Period
- Timeout

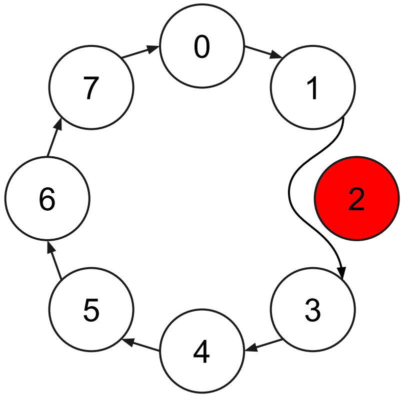
Definitions

- Emitter
- Observer

Initial Ring Distribution



Ring Distribution After Failure



Heartbeat - Implementation (Main Loop)

Algorithm 1: Standard OCFTL heartbeat algorithm

```
while (!hb_done) do
    if (timeout expires) then
        broadcast(emitter);
        find new emitter ;
        rearrange the ring;
    end
    if (period achieved) then
        alive_beat(observer);
    end
    if (alive msg received) then
        resets timeout;
    end
    if (new_observer msg received) then
        observer ← new_observer;
    end
    if (broadcast received) then
        do related procedure;
        if (first time this broadcast) then
            broadcast(bc_message;
        end
    end
    sleep_for(timestep)
end
```

Methodology

Methodology

- Evaluate the standard algorithm
- Proposes modifications to improve the main loop
- Test with other backend (Not MPI)
- Characterize all options for different HB parameters
- Evaluate relationship between HB efficiency and energy

Proposals

NEW - OCFTL

Algorithm 2: New OCFTL heartbeat algorithm

```
Main-Thread:
while (!hb_done) do
    if (timeout expires) then
        find new emitter ;
        broadcasts the failure;
        rearrange the ring;
    end
    if (any message received) then
        if (type == alive) then
            resets timeout;
        end
        if (type == new_observer) then
            observer ← new_observer;
        end
        if (type == broadcast) then
            do related procedure;
            if (first time this broadcast) then
                replicates the broadcast;
            end
        end
    end
    sleep_for(timestep)
end
----------------------------------------------------------------
Alive-Thread:
while (!hb_done) do
    alive_beat(observer);
    sleep_for(period)
end
```

- Checks only one time for a message
- Standard calls 4 times MPI calls
- This approaches calls 1 time
- Extra thread to send beats

Algorithm 3: NNG-OCFTL heartbeat algorithm

```
Main-Thread:
while (!hb_done) do
    if (timeout expires) then
        find new emitter ;
        broadcasts the failure;
        rearrange the ring;
    end
    for ( i = 0; i < size; i = i + 1 ) {
        if (any message received in socket[i]) then
            if (type == alive) then
                resets timeout;
            end
            if (type == new_observer) then
                observer ← new_observer;
            end
            if (type == broadcast) then
                do related procedure;
                if (first time this broadcast) then
                    replicates the broadcast;
                end
            end
        end
    }
    sleep_for(timestep)
end
______________________________________________
Alive-Thread:
while (!hb_done) do
    alive_beat(observer);
    sleep_for(period)
end
```

NNG - OCFTL

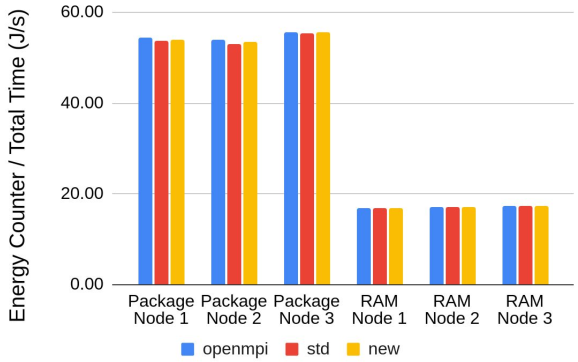
- Uses TPC based communication
- Using NNG backend
- TCP Socket between each pair of processes
- Extra thread to send beats

Experiments

- Execution on Sorgan (OmpCluster's small cluster)
- 4 Applications (5 samples)
  - LULESH (https://github.com/LLNL/LULESH) - Hydrodynamics
  - HPCCG (https://github.com/Mantevo/HPCCG) - Conjugate Gradients
  - MINIMD (https://github.com/Mantevo/miniMD) - Molecular Dynamics
  - LIBONLY (Custom made) - Low resource usage program
- Based on OpenMPI V4.0
- Range from 27 to 36 processes divided in 3 nodes
- Evaluation
  - Package (Total Energy / Time Spent)
  - RAM (Total Energy / Time Spent)

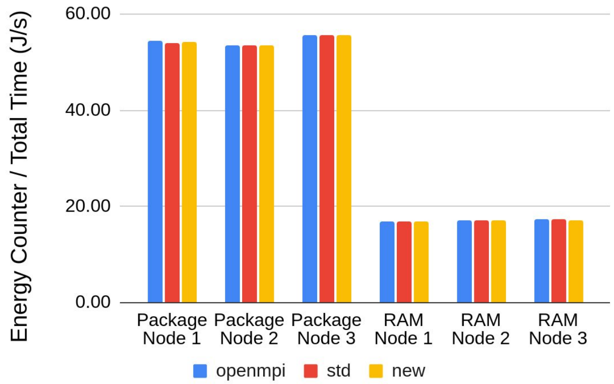
Results and Discussions

Analysis of HPCCG

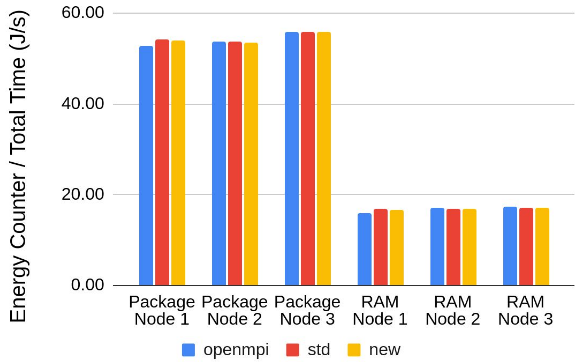


Timeout = 10s
Period = 1s
Timestep = 20ms

Keynotes:
- NNG showed false-positives in every test
- Results are very close

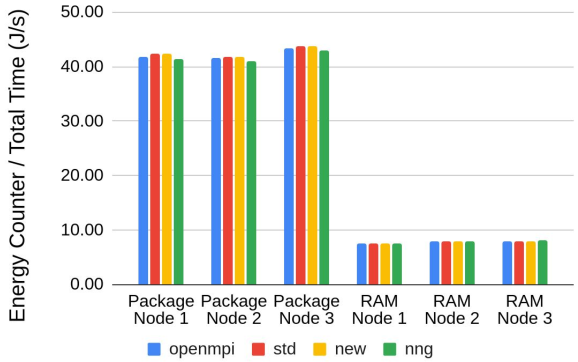


Timeout = 2s
Period = 100ms
Timestep = 10ms



Timeout = 1s
Period = 10ms
Timestep = 1ms

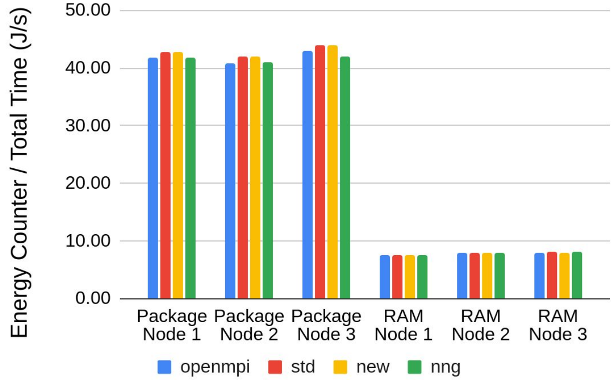
Analysis of LULESH



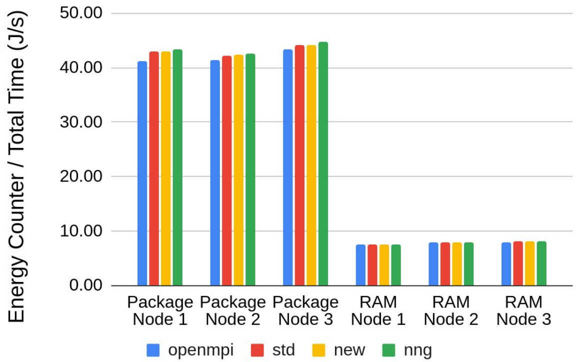
Timeout = 10s
Period = 1s
Timestep = 20ms

Keynotes:

- NNG performs a bit better for relaxed HB parameters. Lose in low value parameters
- STD and NEW keep very close results

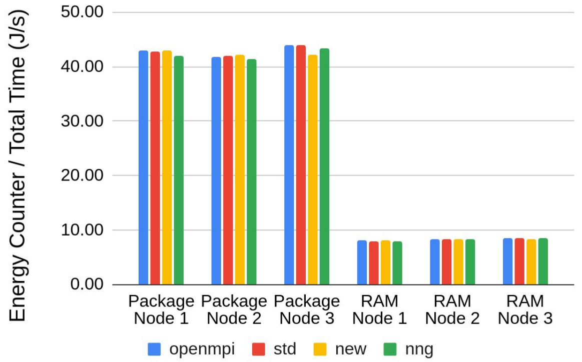


Timeout = 2s
Period = 100ms
Timestep = 10ms



Timeout = 1s
Period = 10ms
Timestep = 1ms

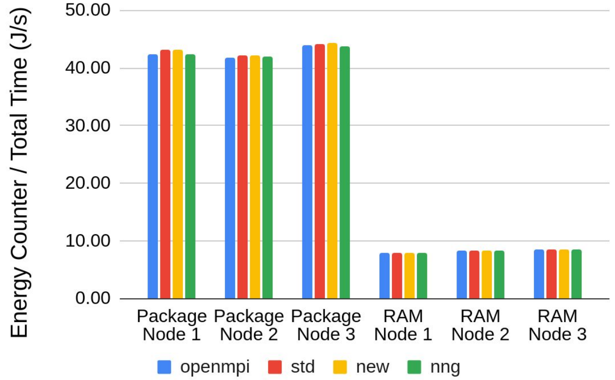
Analysis of MINIMD



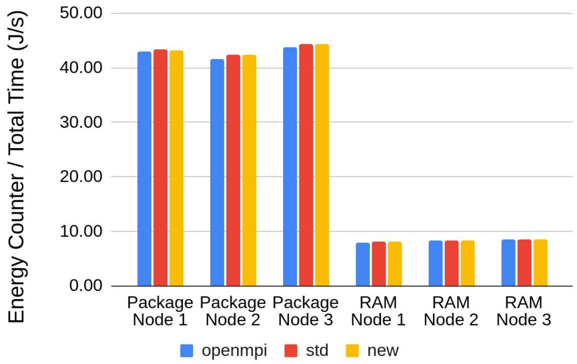
Timeout = 10s
Period = 1s
Timestep = 20ms

Keynotes:

- NNG can not perform low HB parameters
- NEW performs marginally better than STD

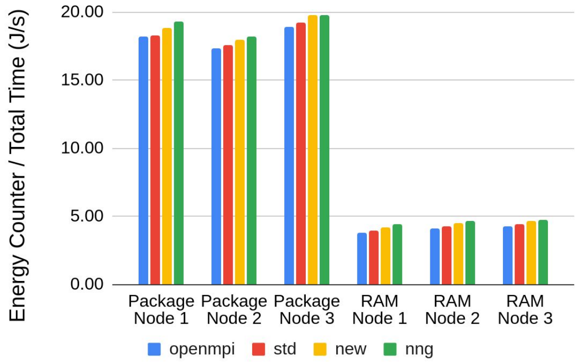


Timeout = 2s
Period = 100ms
Timestep = 10ms



Timeout = 1s
Period = 10ms
Timestep = 1ms

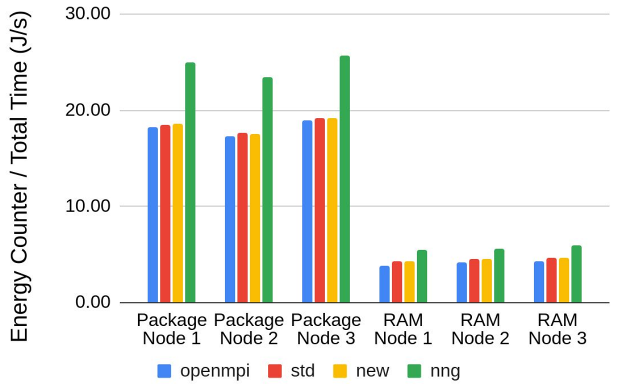
Analysis of LIBONLY



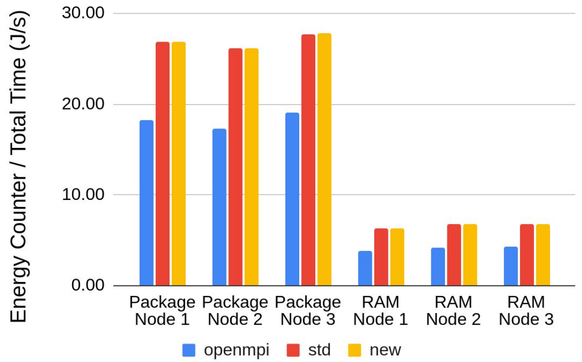
Timeout = 10s
Period = 1s
Timestep = 20ms

Keynotes:

- Pure FT lib overhead
- NNG worst in all cases. Can not perform low HB parameters
- STD and NEW very close results



Timeout = 2s
Period = 100ms
Timestep = 10ms



Timeout = 1s
Period = 10ms
Timestep = 1ms

Conclusions

- Characterized the HB system
- Resource Intensive
- Pure lib overhead
- Energy vs HB efficiency
- Different Backends
  - NNG <u>is not</u> a good option
- What about other FD systems?
- Algorithm

Thank you!

UNICAMP