

PROJECT REPORT

Energy Characterization and Optimization in Heartbeat Failure Detection Systems

MO632/MC972 – Energy Efficient Computing - Second Semester, 2021

Pedro Henrique Di Francia Rosso {p233687@dac.unicamp.br}

December 12, 2021

1. Introduction

Fault Tolerance (FT) is a common concern in various scenarios, for example, in High-Performance Computing (HPC), environments containing a large number of nodes, which ultimately leads to an increased number of failures, making the Mean Time Between Failures (MTBF) as low as a few hours [3], and also in Internet of Things (IoT) applications, like the ones using Wireless Sensor Networks (WSNs), which is good to know failures and detected faulty sensors [9].

One important component of FT is the failure detection mechanism. There are a few ways to detect failures, for example, in HPC systems, one can use daemon processes to monitor other processes [5], or use heartbeat monitoring systems, where each process monitors its neighbor, if the process observed stops sending alive beats, it will be considered faulty and the failure will be propagated to the other processes [1]. In WSNs, usually a voting system or a comparing system is employed, where the sensing information of different devices are compared to define if a sensor is faulty or not and other approaches like sending test messages or messages right before battery depletion to take account of permanent failures [9].

This work leverages the library OCFTL [8], an improved heartbeat-based FT library for MPI. Although this library is focused on HPC systems, the same failure detection could be employed in IoT systems to detect permanent failures. The objective of this work is to first characterize the library in terms of energy, and compare with other HPC failure detection implementations and mechanisms, as well as proposing changes to the algorithm and the communication back-end to evaluate if the library can be more energy efficient. Finally we want to discuss whether option would be the better to employ in IoT systems considering their restrictions.

2. Related Work

Failure detection can be divided in two procedures, the detection mechanism and the propagation mechanism.

For the detection mechanism in HPC, some monitoring system is employed. Like using daemon processes (one per node) to monitor the other processes using signals from the Operating System. Further, these daemon processes are monitored either by the root

process [5] or by heartbeat messages exchanged between the daemons [10]. Others rely on the `slurmd` daemon to detect process failures via exit codes [2]. Finally, the heartbeat ring approach works by distributing processes throughout a ring and exchanging messages between neighbors [1]. This paper evaluates a library that is based on the heartbeat approach. For the propagation mechanism, usually a fault-tolerant broadcast is employed. These type of broadcast usually employ redundant messages, for this work, we will be using the broadcast proposed in the evaluated library, a chord-based broadcast that has reduced overload of messages still providing redundant messages [8].

For the mechanisms in IoT applications, a survey on the fault tolerant approaches for WSNs was made [9]. This survey shows different approaches for fault detection, some employed to detect permanent failures (like the OCFTL library) and some employed to detect errors in the sensing system (malfunction of the sensors). The survey shows that the mechanisms are probabilistic based, or employ message exchange with a central devices, or employ a exchange of messages in a group, neighbors or cluster to detect failures in the proximity [9]. Looking at the proposals, a heartbeat failure detection system could be an approach for detection the permanent and transient failures in such systems.

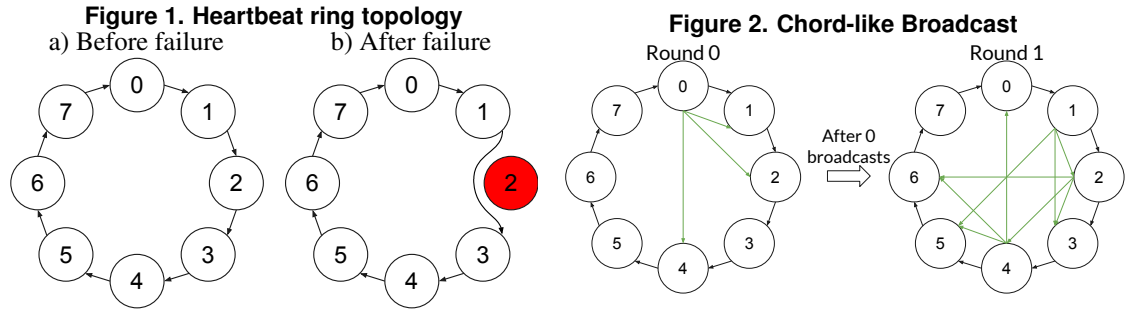
In respect to energy efficiency, the HPC works usually does not evaluate the energy cost of the applications, since they are usually coupled with high performance applications which would consume much more energy than the monitoring system. This works aims to evaluate this proportion by evaluating some monitoring systems in HPC running them with distributed HPC applications. For the WSNs, the survey [9] discusses the energy-efficiency of the approaches, not by measuring them, but by pointing the main focus of energy consumption (what to send in messages, how to exchange messages, and who receives messages). In this work we also want to compared different proposal of the library algorithm to see whether would be the better option if we would port the library to an IoT system.

3. The Heartbeat

OCFTL's heartbeat system consists in a ring-topology process distribution, like in Figure 1. In this topology, each process will have two neighbors (Figure 1-a)). In this representation, each process will be an `emitter` to the following process, periodically sending alive beats, also being an `observer` to the previous process, receiving its alive beats. If the observer does not receive alive beat in a given `timeout`, it will state the its emitter failed and will rearrange the ring to remove the failed process (Figure 1-b)), then, it will propagate the failure using the FT broadcast. In this broadcast, a process will send the failure message to other processes following the power of two (1, 2, 4, ...) starting at the first neighbor, and each time a process receives the first message of a broadcast it will replicate the same way, like the example of Figure 2.

To implement that mechanism, OCFTL proposes the Algorithm 1, as we will be referring as `OCFTL-std`. Given the neighbors of a process, the `emitter`, the `observer` and the heartbeat parameters `period` and `timeout`, the main loop is described in the Algorithm 1.

It is good to note that this main loop iteration occurs every `timestep`, this time step is a value much lower than the `period` and `timeout`, to guarantee we will receive and parse any message as soon as possible, and, because of that, it will be responsible for



Algorithm 1: Standard OCFTL heartbeat algorithm

```

1  while (!hb_done) do
2      if (timeout expires) then
3          broadcast(emitter);
4          find new emitter ;
5          rearrange the ring;
6      end
7      if (period achieved) then
8          alive_beat(observer);
9      end
10     if (alive msg received) then
11         resets timeout;
12     end
13     if (new_observer msg received) then
14         observer  $\leftarrow$  new_observer;
15     end
16     if (broadcast received) then
17         do related procedure;
18         if (first time this broadcast) then
19             broadcast(bc_message);
20         end
21     end
22     sleep_for(timestep)
23 end

```

determining how much computation is done. Higher values of the `timestep` means we are doing less iterations per second, then less computation, lower value of the `timestep` means we are doing more computation.

4. Proposals

We have two proposals to evaluate for this work, the first proposal to change the way the main loop of the Algorithm 1 works but still uses MPI as back-end, and the second proposes using the NNG library, which is a light-weight messaging library.

4.1. Standard Algorithm Change

For this proposal we will work on the computation made at each step of the loop. The goal is to do less computation as possible. In the Algorithm 1, we check for several conditions and based on each one we will do a procedure. Taking a more closer look, we are requesting for the MPI back-end three times if a message was received, if the `timeout` does not expires, we decrease a counter and if the period is not achieved we also decrease a counter.

For this new proposal (Algorithm 2) which we will refer as OCFTL-new, we propose to check one time per iteration if a message was received and parse it accordingly to the tag of the message received, for the `timeout` checking we kept the same way as before, and for the `alive` beat, we create a new thread specialized for sending the beats, where we can eliminate the associated counter and wake-up only at each defined period.

4.2. Back-end Change Proposal

MPI is widely used in HPC applications, but here we want to see when using another library, such as NNG messaging library, first if the library is capable of what the MPI version of the library is (in terms of heartbeat parameters) and if takes less or more energy. The implementation of this version (which we will refer as OCFTL-nng) follows the Algorithm 3, which is similar to the algorithm presented in Section 4.1. For this proposal, we setup a web-socket between each pair of processes, which would make the process of implementing the library simpler and each process would communicate direct with another. This might not be the best option in terms of scalability, since the number of sockets can increase exponentially. Another factor of this implementation is that each iteration, we need to verify if a message was received in each one of the sockets (line 8 in the algorithm), excluding the own socket, this can increase the computation time, but comparing to the MPI back-end a likely approach has to be done depending on the communication back-end used.

5. Characterization

Following our methodology, we will be characterizing energy in a distributed system. We will be using the Sorgan mini cluster, composed by three worker nodes. To characterize, we will be using three benchmark applications and one developed application. The first three applications are used to evaluate the library with a high computation load applications while the other one to evaluate to library with a non-intense application, this developed application is called `libonly`, and consists in a sleeping loop application¹.

¹Answering first Q&A question: These applications are commonly used to benchmark MPI works, and are used on other projects like Reinit++[5].

Algorithm 2: New OCFTL heartbeat algorithm

```
1 Main-Thread:
2 while (!hb_done) do
3   if (timeout expires) then
4     find new emitter ;
5     broadcasts the failure;
6     rearrange the ring;
7   end
8   if (any message received) then
9     if (type == alive) then
10      resets timeout;
11    end
12    if (type == new_observer) then
13      observer  $\leftarrow$  new_observer;
14    end
15    if (type == broadcast) then
16      do related procedure;
17      if (first time this broadcast) then
18        replicates the broadcast;
19      end
20    end
21  end
22  sleep_for(timestep)
23 end
24
25 Alive-Thread:
26 while (!hb_done) do
27   alive_beat(observer);
28   sleep_for(period)
29 end
```

Algorithm 3: NNG-OCFTL heartbeat algorithm

```
1 Main-Thread:
2 while (!hb_done) do
3   if (timeout expires) then
4     find new emitter ;
5     broadcasts the failure;
6     rearrange the ring;
7   end
8   for (  $i = 0; i < size; i = i + 1$  ) {
9     if (any message received in socket[i]) then
10      if (type == alive) then
11        resets timeout;
12      end
13      if (type == new_observer) then
14        observer  $\leftarrow$  new_observer;
15      end
16      if (type == broadcast) then
17        do related procedure;
18        if (first time this broadcast) then
19          replicates the broadcast;
20        end
21      end
22    end
23  }
24  sleep_for(timestep)
25 end
26
27 Alive-Thread:
28 while (!hb_done) do
29   alive_beat(observer);
30   sleep_for(period)
31 end
```

Table 1 shows the resume of the applications used to characterize the library until this point of this work. These application were selected from previous works and related works.

Table 1. Applications evaluated

Application	Input	# Processors	# Nodes
LULESH [7]	<i>-i 20 -s 60</i>	27	3
MiniMD [6]	<i>-i mini-input</i>	36	3
HPCCG [6]	150 150 150	36	3
Libonly		36	3

Finally, all the information, scripts, programs, tools used in this project are available in the repository of the project: <https://github.com/PedrooHR/eec-project>.

5.1. Data Collection

To collect the results of each application, we first run each application 5 times in the following approaches: without FT library (openmpi), using OCFTL-std, using OCFTL-new and using OCFTL-nng. All applications are MPI based, so we used the Open MPI version 4.0.0 [4], and for the last case, we used the NNG messaging library in addition. To collect the data about energy, we used the `perf` tool, with certain privileges, the tool permits us to collect information about the counter related to power.

Unfortunately, in the cluster, there are only three counters available, and one of them does not give any result. So, for this work, we will be measuring the counter `power/energy-pkg/`, which measures the energy of the components in the CPU package, and the `power/energy-ram/` counter, which measures the energy related to the RAM. Since we are working with 3 nodes, and the `perf` tool collects the energy for each process, we will be measuring multiple times the counters related to one node. To solve that, in the results we will be showing the average value of the measurements in the same node, for example, if we measure energy from 8 processes in the first node, the average measure of these 8 processes will be our final measure for that node. This way, at the end, we will have the energy consumption for the three nodes for each application, for each scenario.

5.2. Results

Following the methodology described before, we can evaluate the energy consumption of each approach. Here we will be comparing different scenarios (described below) for all the applications.

These results will be divided in three scenarios, changing the parameters of the heartbeat, so we can evaluate the energy efficiency and the efficiency of the heartbeat at the same time. For this report, the metric we adopted is the counter per time, meaning we will divide the total energy by the time spent, to know how much energy is being dissipated per second, this will make the results easier to evaluate. The sheets containing all results is available at: <https://docs.google.com/spreadsheets/d/1kareurl4QXfcWAUnfEIl0twfMEYmleJpPBzNIdfagSg/edit?usp=sharing>. We will be presenting the scenario results and a analysis of those. Following, we will present the final discussions.

5.2.1. First Scenario

In this first scenario, we evaluate the applications with relaxed parameters. The heartbeat period was set to 1 second, the timeout was set to 10 seconds and timestep was set to 20 milliseconds. Figures 3, 4, 5 and 6 shows the results for this scenario. Missing values for `nng` approach in Figure 3 means that the application showed false-positive failures². OpenMPI approach means the application without a heartbeat library.

Figure 3. First scenario results for HPCCG application.

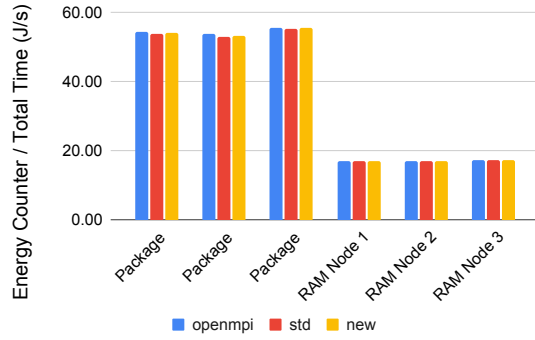


Figure 4. First scenario results for Lulesh application.

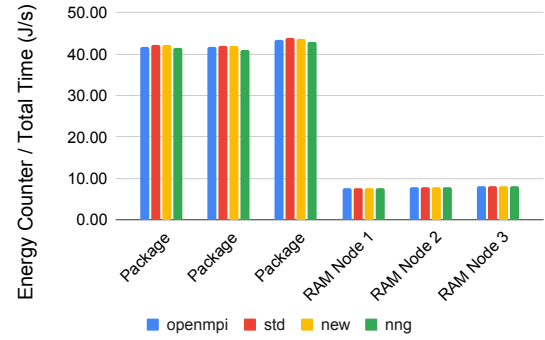


Figure 5. First scenario results for Min-IMD application.

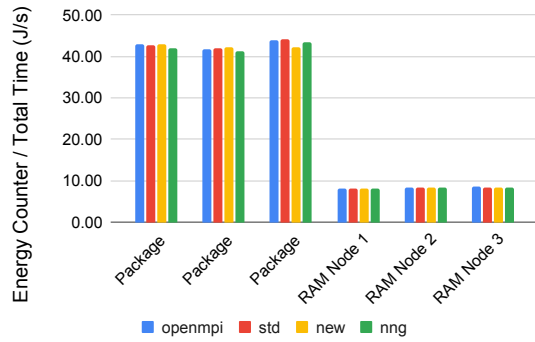
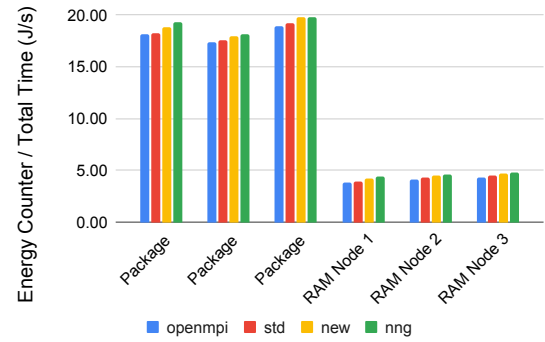


Figure 6. First scenario results for Li-bonly application.



For this scenario, we can observe that the results for the PACKAGE are very close for all applications, where the `new` approach is showing marginally improvement than the others. In the Figure 6, it is possible to see that the `std` approach performs better in terms of energy than the other two approaches. The RAM results are about the same for every application in all nodes.

²Answering second Q&A question: The false-positive failures are an indication of low efficiency of the heartbeat. The ideal scenario is to not find any false-positive failure. In this report, if a results misses one or more approach (between the 4 approaches tested) means that the experiment showed false-positive failures.

5.2.2. Second Scenario

For this second scenario, we evaluate the applications with less relaxed parameters. The heartbeat period was set to 100 milliseconds, the timeout was set to 2 seconds and timestep was set to 10 milliseconds. Figures 7, 8, 9 and 10 shows the results for this scenario. Missing values for nng approach in Figure 7 means that the application showed false-positive failures. OpenMPI approach means the application without a heartbeat library.

Figure 7. Second scenario results for HPCCG application.

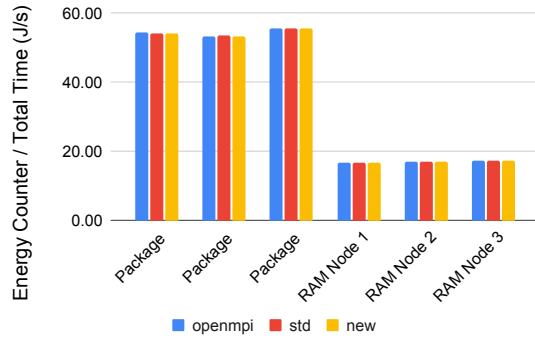


Figure 8. Second scenario results for Lulesh application.

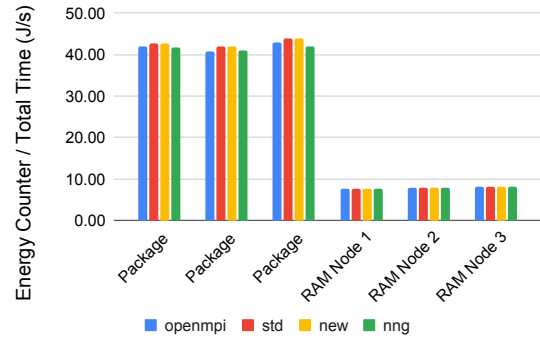


Figure 9. Second scenario results for MiniMD application.

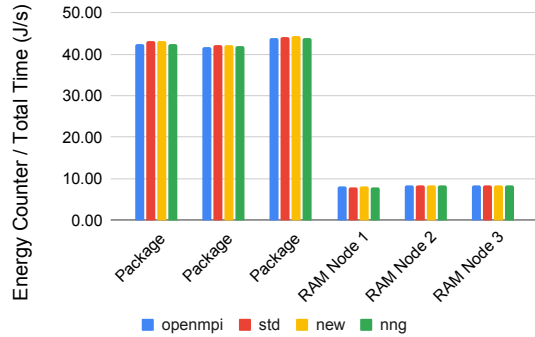
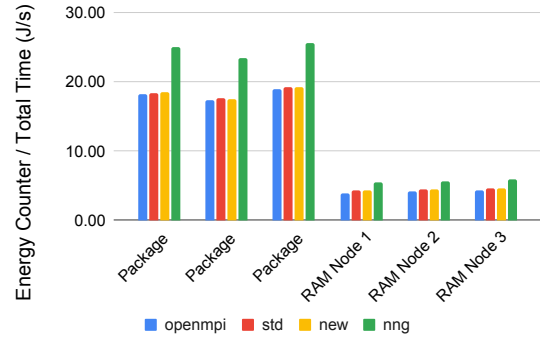


Figure 10. Second scenario results for Libonly application.



For this scenario, we can observe that the results for the PACKAGE are very close for all applications, where the nng approach is showing marginally improvement than the others for the HPCCG, Lulesh and MiniMD applications. In the Figure 10, it is possible to see that the std and new approaches performs better in terms of energy than the nng, this means that when we are running applications together with the heartbeat, the energy spent is more likely to be from the application than from the heartbeat library, but when we look at the Libonly application, the result says that the nng approach show less energy efficiency, spending more energy to do the same work as the other approaches. The RAM results also show the same behavior of the PACKAGE results.

5.2.3. Third Scenario

For the third scenario, we evaluate the applications with more restricted parameters. The heartbeat period was set to 10 milliseconds, the timeout was set to 1 seconds and timestep was set to 1 milliseconds. Figures 11, 12, 13 and 14 shows the results for this scenario. Missing values for `nng` approach in Figures 11, 13 and 14 means that the application showed false-positive failures. OpenMPI approach means the application without a heartbeat library.

Figure 11. Third scenario results for HPCCG application.

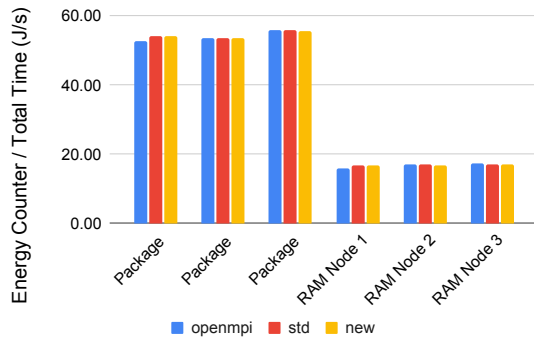


Figure 12. Third scenario results for Lulesh application.

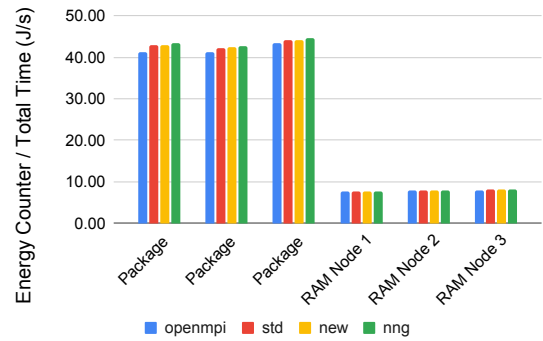


Figure 13. Third scenario results for MiniMD application.

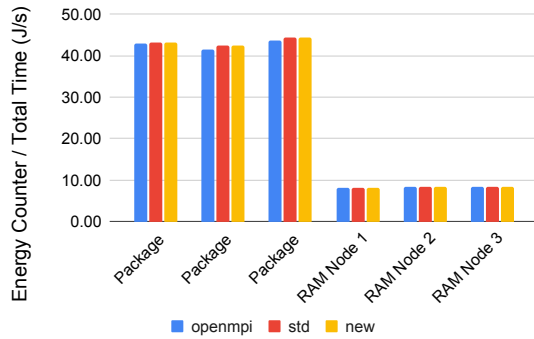
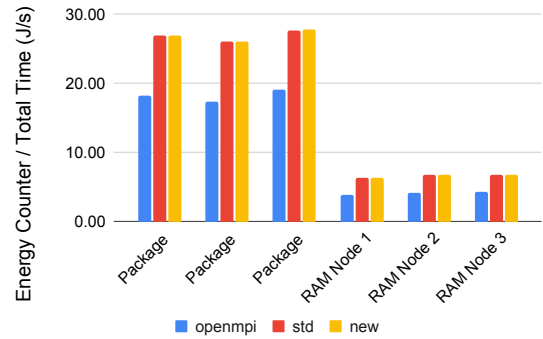


Figure 14. Third scenario results for Li-bonly application.



In the third scenario, `nng` showed worse performance in terms of energy for the Lulesh application, and worse performance in terms of heartbeat efficiency for the other applications. Comparing `new` and `std` approaches, the results are about the same, showing that the two approaches have about the same energy use. In Figure 14 it is possible to see that for the low parameters of the heartbeat, the energy use is increased, comparing to the results of the other two scenarios for this application, the energy spent for the heartbeat approach is much higher than the energy spent by the application without the heartbeat library (`openmpi`).

5.3. Discussions

Looking at the displayed results before, we can observe few things. First, the `new` and `std` approaches have a very similar behavior in all tests, while `nng` approach showed a lot of problems with false-positive failure detection. Since the results are very similar for the approaches, specially with the heavy resource usage applications (excluding `libonly`), we can observe that the energy consumption of the application is from the application itself and not from the heartbeat library, but, as soon as we reduce the heartbeat parameters values, the differences become more observable, specially for the `libonly` application. With the non-relaxed parameters (third scenario) the results of energy consumption in both `PACKAGE` and `RAM`, is much higher than the consumption of `openmpi` only run, a behavior that is not observed in the first two scenarios.

Looking at the results, looks like that are no much more room to improve the energy consumption by software optimizations. Instead, although results shows worse efficiency (both in energy and performance) from the `nng`, a pure TCP library, this indicates that a different communication backend could lead to improvements in the energy consumption.

We also can establish a relation between energy and performance. To obtain a better heartbeat performance, the ideal scenario is to have low values for the heartbeat parameters, but this can overload the application and degrade the performance. On previous results [8], we saw that we could achieve a `1ms` timestep value without causing significant application overhead. But, the results in this report shows that this lower values impact much more on energy consumption, as we can observe more in Figure 14. This gives us a trade-off, higher values of heartbeat parameters reduces the performance efficiency of the heartbeat (more time to detect failures) but reduces the energy consumption, while lower values of the parameters (detecting failures faster) increases the energy consumption. So, to define the parameters will depend on the energy and performance constraints.

Finally, we tried to test and compare this results with other heartbeat approaches, like the heartbeat implemented in the source code of OpenMPI [1] by ULFM, and Reinit++, which uses daemon processes to monitor and detect failures [5]. Unfortunately, we had a lot of problems running those OpenMPI forks in our distributed system and could not achieve reliable results. This could be an analysis for future energy efficiency works on this type of systems.

6. Conclusions

We have characterized the use of the library and the proposals with different applications, intensive and non-intensive which was our first objective of this work. We also evaluated the impact of the heartbeat parameters on energy consumption for the different applications, composing a three scenario experiment varying all the three heartbeat parameters, which was our second objective.

We also implemented the heartbeat with another communication backend, using NNG library, which was the third objective of this work. Although the results were worse, this give an insight that different backends could lead to a better energy efficiency. We also evaluated a different algorithm (the fourth objective), sticking to the use o MPI, the results were about the same of the standard algorithm, showing that probably are no much more room to improve the energy consumption by software when using MPI.

As our last objective, the idea was to establish a relationship between energy consumption and heartbeat performance. With the results, we can say that exists a trade-off between energy consumption and heartbeat efficiency. And the parameters values choice will depend on both heartbeat and energy efficiency constraints.

We also tried to compare the energy consumption of the OCFTL with the ULFM's proposal, a likely heartbeat implemented in the source code of OpenMPI [1] and Reinit++, which uses daemon processes to monitor and detect failures [5]. But, due to internal problems, we could not execute the experiments in our cluster, so this could be a potential future work of this project.

References

- [1] George Bosilca et al. "A failure detector for HPC platforms". In: *The International Journal of High Performance Computing Applications* 32.1 (2018), pp. 139–158.
- [2] Sourav Chakraborty et al. "EReinit: Scalable and efficient fault-tolerance for bulk-synchronous MPI applications". In: *Concurrency and Computation: Practice and Experience* 32.3 (2020), e4863.
- [3] Ifeanyi P Egwutuoha et al. "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems". In: *The Journal of Supercomputing* 65.3 (2013), pp. 1302–1326.
- [4] Edgar Gabriel et al. "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation". In: *Proceedings, 11th European PVM/MPI Users' Group Meeting*. Budapest, Hungary, 2004, pp. 97–104.
- [5] Giorgis Georgakoudis, Luanzheng Guo, and Ignacio Laguna. "Reinit⁺⁺: Evaluating the Performance of Global-Restart Recovery Methods for MPI Fault Tolerance". In: *International Conference on High Performance Computing*. Springer. 2020, pp. 536–554.
- [6] Michael A Heroux et al. "Improving performance via mini-applications". In: *Sandia National Laboratories, Tech. Rep. SAND2009-5574* 3 (2009).
- [7] Ian Karlin, Jeff Keasler, and JR Neely. *Lulesh 2.0 updates and changes*. Tech. rep. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2013.
- [8] Pedro HDF Rosso and Emilio Franceschini. "Improved Failure Detection and Propagation Mechanisms for MPI". In: *Anais Estendidos da XII Escola Regional de Alto Desempenho de São Paulo. Aceito para publicação*. SBC. 2021. URL: <http://cradsp.sbc.org.br/eradsp/2021/artigos/sl.2.pdf>.
- [9] Yue Zhang, Nicola Dragoni, and Jiangtao Wang. "A framework and classification for fault detection approaches in wireless sensor networks with an energy efficiency perspective". In: *International Journal of Distributed Sensor Networks* 11.11 (2015), p. 678029.
- [10] Dong Zhong et al. "Runtime level failure detection and propagation in HPC systems". In: *Proceedings of the 26th European MPI Users' Group Meeting*. 2019, pp. 1–11.