SONAR: Automated Communication Characterization for HPC Applications

Steffen Lammel, Felix Zahn, and Holger Fröning

Computer Engineering Group, Ruprecht-Karls University of Heidelberg, Germany
s.lammel@stud.uni-heidelberg.de
{felix.zahn,holger.froening}@ziti.uni-heidelberg.de
http://www.ziti.uni-heidelberg.de/compeng

Abstract. Future computing systems will need to operate within hard power and energy constraints, this is particularly true for Exascale-class systems. These constraints are hard for technical, economical and ecological reasons, thus, such systems have to operate within given power and energy budgets. Therefore, we anticipate the need for modeling tools that help to predict power and energy consumption. In particular, such modeling tools would allow for detailed explorations of various alternatives when designing systems. While processing and memory already receives a large amount of interest from the research community, power modeling of scalable interconnection networks is rather neglected. However, analyses show that the network contributes about 20% to the overall power consumption of HPC systems. Considering the increasing energy efficiency of other components, this fraction is likely to increase. While models for processing and memory typically rely on performance counters to model power and energy, we observe that the distributed nature of networks leads to significantly more complex metrics. Selecting the right set of abstract metrics, which will be used as input for such a prediction, is crucial for prediction performance.
In this work we introduce our tool called Simple Offline Network Analyzer (SONAR) to derive complex metrics from communication traces of HPC applications. We explain the motivation behind choosing this concept, the implementation, and the ability of the tool to easily support the integration of new metrics. We also show exemplary explorations using an initial set of metrics for a representative range of HPC applications, including contemporary as well as emerging Exascale workloads. In particular, we use SONAR to characterize the communication of applications in terms of verbosity and network utilization, as we believe both to be important metrics for power prediction.

Keywords: Exascale, HPC, MPI, communication, characterization, automated tooling

1 Introduction

Following the end of Dennard scaling, power and energy consumption turned into hard constraints that limit a computing system's computational power. Key to

a continuing performance scaling is improving energy efficiency, which means more computations per Joule can be performed.

This is particularly true for Exascale systems, which will be severely limited by power consumption. Presently, it is estimated that a power dissipation between 20MW and 100MW is anticipated for those systems. Since processors make up a large fraction of the overall power consumption, the majority of current work focuses on understanding and optimizing their power efficiency. Additionally, we believe the network component has historically received too little attention. Analyses show that the network consumes up to 30% of the overall power [1]. This fraction increases if processors become more energy-proportional, i.e. the actually consumed power is linked linearly to the component's load. Therefore, there is an urgent need to understand power consumption in scalable interconnection networks in order to design optimizations.

In order to achieve an understanding of power consumption, we believe the most suitable option is using power-aware network simulations and power models. The first method excels in an accurate prediction, the latter allows for much faster predictions. Therefore, it enables explorations of topology, link configuration and other abstract aspects. While our initial version of a power-aware network simulator already exists [2], we are currently working on a network power model. For such models it is crucial to select the right set of metrics that describe the traffic in a way that is suitable for an abstracted power prediction.

Understanding the communication behavior of large scale HPC applications is essential for our research regarding power consumption and possible optimizations. Therefore, it is mandatory to provide realistic input for simulators and models. Traces of real HPC applications meet this demand and allow simulations of different hardware configurations under realistic conditions. Additionally, traces are examined post-run, which enables analysis for different metrics without running the application again.

There are a variety of tools that generate traces for post-run examination. Some popular examples are VampirTrace[1], TAU[2] and Score-P[3]. Once the traces have been created, tools like Vampir or Jumpshot-4 take a look at the inner workings of the applications. These tools are designed to discover and fix programming weaknesses of applications such as waiting phases, bottlenecks, contention, etc. to maximize the performance.

In this work we introduce our communication characterization tool called SONAR (Simple Offline Network Analyzer), which allows us to easily derive complex metrics from communication traces. Furthermore, it is simple to introduce new metrics based on these traces. In particular, we make the following contributions:

- Introduce SONAR as an open-source tool to automatically generate complex metrics of HPC communication traces

[1] `https://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/forschung/projekte/vampirtrace/`
[2] `https://www.cs.uoregon.edu/research/tau/home.php`
[3] `https://www.vi-hps.org/Tools/Score-P.html`

- Provide reasoning behind the methodology for our approach
- Exemplary characterization of a representative set of HPC workloads, including metrics like verbosity and network load

The remainder of this work is structured as follows: We continue with providing background information including a brief review of related work. Then, we detail our methodology, our analyzed metrics, and how we generate and examine our traces. This is followed by short overview of SONAR's tool-design. Finally, the results of our first analysis of six different HPC application are shown, followed by a conclusion in which we summerize our results and outline possible future directions.

2 Background

Today, power consumption is one of the most important aspects when designing and operating HPC systems. Analyses have shown that a large fraction of the power consumption originates from data movements, and that associated costs significantly increase with distance [3]. Predictions indicate that the gap between energy costs for computation and data movement will actually widen in the future [4].

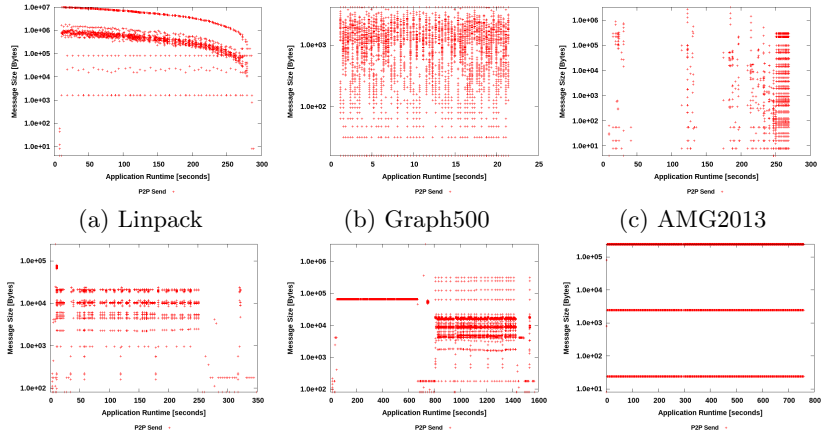
Power saving strategies in the area of networks are based on reducing link frequency or link width, both of which result in a decreased bandwidth. Since transition times of several microseconds are common, it is important to know when a link can operate with lower bandwidth without reducing performance. Therefore it is essential to analyze and understand applications regarding their communication behavior.

In order to examine the impact of different network configurations on power consumption and performance simulations and models are essential. Synthetic traffic is a commonly used for such simulators. This approach is a good first-order approximation, but some peculiarities of real application are neglected. Therefore, real application traces are mandatory for a deep understanding of certain communication patterns. Furthermore, traces can be used for post execution analyses. This allows to examine new metrics without running applications again.

2.1 Communication Pattern

Applications in high performance computing exhibit various different communication patterns. Fig. 1 depicts the injection pattern of one particular node over the normalized run time. It is apparent that these different communication pattern have differing suitability for various power saving strategies. While the Graph500 workload (b) has a very irregular and dense pattern, the network is idling periodically for the AMG2013 (c) and NAMD (d,e) workload.

Analyzing communication patterns and other metrics is mandatory for future interconnection network power models. These models are much faster than



(a) Linpack (b) Graph500 (c) AMG2013

(d) NAMD ApoA1 (e) NAMD STMV (f) LULESH

Fig. 1: P2P injection plots of exemplary workloads

accurate simulations of a complete cluster. This enables the possibility to test a variety of different parameters in a reasonable time. Since these models are not cycle accurate, they are not taking full event-based traces as input data but simpler metrics. Therefore, tools such as SONAR are used to derive different metrics that affect power consumption directly from the traces.

2.2 Related Work

There is a large set of existing tools that gear to profile MPI applications. The most important ones include TAU [5], HPCToolKit[4], Intel VTune[5], IPM[6], mpiP[7], INAM [6] and INAM2 [7]. However, they rather focus on reporting and visualizing MPI communication behavior for profiling purposes instead of generating aggregated metrics like SONAR. In fact, they can be seen as being one level below SONAR, i.e. SONAR builds on top of such tools.

Tools that monitor and analyze MPI jobs also have a large history, such as Lightweight Distributed Metric Service (LDMS) [8] by Sandia National Labs, the HOlistic Performance System Analysis (HOPSA) [9], and TACC STATS [10]. Compared to SONAR, they tend to be more abstract and are geared towards monitoring the behavior of complete MPI jobs.

[4] `http://hpctoolkit.org/`
[5] `https://software.intel.com/en-us/intel-vtune-amplifier-xe`
[6] `http://ipm-hpc.sourceforge.net/`
[7] http://www.llnl.gov/CASC/mpip/

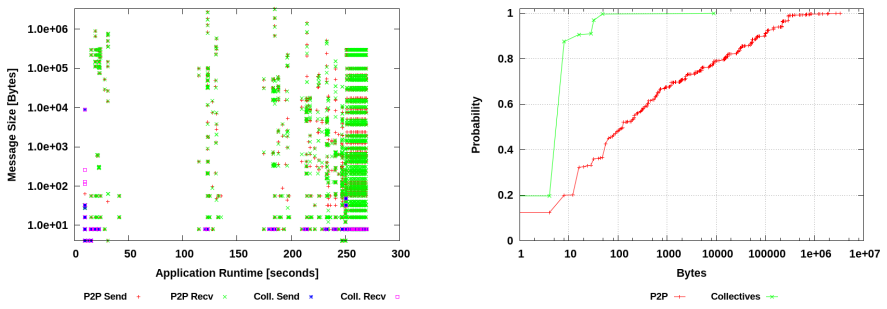
The MPI forum recently proposed a new extension called MPIT[8], which allows profiling tools to access the internal states of MPI, enabling more detailed profiling information. This feature is already used in recent work to provide tuning hints to the user [11]. According to our knowledge MPI does not contribute significantly to the overall power consumption, as most of the energy is spent for core (floating point) computations and not memory-intensive tasks such as queue management, tag matching, and similar. If this assumption turn out out to be wrong we would opt to extend SONAR with the possibilities offered by MPIT.

3 Methodology

Modern HPC network infrastructures are not energy proportional [2]. A first step to improve energy-proportionality for HPC interconnects is understanding how the interconnect behaves during runtime and what is a suitable approach to minimize network power consumption.

To determine how an application utilizes the provided network resources, we have postulated a set of metrics which give us a qualitative and quantitative insight to the communication behavior of an HPC application. The following metrics have been found to be important:

Network Activity Map: This metric visualizes all point-to-point and collective messages by size and relation to the application runtime in a graph. Each data-point in the plot indicates a particular event. Fig. 2 (a) depicts the network activity map of of an exemplary workload (AMG2013).



(a) Network Activity Map (b) Message Size Distribution Map

Fig. 2: Examples of the visual metrics SONAR derives from an application trace

MPI idle time: We determine the minimum, maximum and average times during which a node does not have to handle any messages to or from the

[8] http://cscads.rice.edu/workshops/summer-2010/slides/performance-tools/2010-08-cscads-mpit.pdf

interconnect. This values represent the white spots in the message activity map in Fig. 2 (a). A good example in the plot can be seen in Fig. 2 (a) from second 50 to 100. In this region, the network is completely idle on this node.

Message Distribution: We use a cumulative distribution function graph (CDF) to visualize the message sizes, which occur in a trace. This metric shows the probability of a message having a size X or smaller. Figure 2 (b) shows the CDF-graph of an exemplary workload (AMG2013).

Verbosity: This represents the ratio of work and the total amount of data which has been consumed by the application. The work is quantified by the number of floating point operations (Flops) issued, as many HPC applications are depending on floating point arithmetic. The Flops are determined via the hardware performance counters of the CPUs. For integer based workloads (such as graph algorithms), the verbosity is defined by their number of integer operations instead of Flops.

Message Rate: The message rate indicates how many messages are sent by one node in a given time period. This metric can be interpreted as a single-value approximation of the network activity map.

All the described metrics are MPI process based. This means we get $N \times P$ results for each metric, where N is the number of nodes and P is the number of MPI processes launched per node. Derived from these metrics, we are able to estimate how the application utilizes the cluster. Largely differing numbers indicate an over- or under utilization of specific nodes in the cluster. Metrics which can be quantified by a single number, such as the verbosity, will also be reported as a global average value.

3.1 The Open Trace Format (OTF)

The open trace format (OTF) stores application activities as events. Each event is associated with a time stamp and additional event-specific information. For example, the *MessageSent*-event contains information such as source, destination and length of the message, whereas the *FunctionEnter*-event requires the function's signature and the ID of the concerning node. All non-numeric values, e.g. the function signatures, are encoded as integer values to minimize the trace's size. To refer these encodings to their respective values, the OTF trace contains a section with definitions. These definition provide the mapping of the encodings to their actual meaning, e.g. the function with the ID 42 belongs to the function with the signature `void foo(int bar)`.

3.2 Trace Generation

The development of SONAR is motivated by the need to efficiently gather information from application traces. Depending on the number of nodes, the complexity of the underlying problem, the communication characteristics, and other factors, these traces can become very large. The traces we have generated for this paper require between 5 and 50 GBs of disk space each. In order to be able to store such traces efficiently, an appropriate format is necessary.

We have tested the Tuning and Analysis Utilities (TAU) and the VampirTrace frameworks. In both cases, an application can be compiled with code instrumentation to examine specific parts. They provide also the possibility to run existing binaries without any modifications. With this black-box approach, the traces contain only a reduced amount of information. Application-specific functions are opaque in this case. The only data that can be gathered with this method is the entry and exit point of the application's `main()`-function as well as the entry and exit points of the MPI library functions used by the application.

As mentioned before, instead of looking at specific parts of an application, we want to examine the behavior of the whole application with special attention to the communication aspects. Since all information regarding MPI and communication are retained when running an existing application with the run-wrappers, this method is sufficient for our experiments.

For our purposes, the VampirTrace framework proved to be the better fit. It stores traces directly in the compressed Open-Trace-Format (OTF), whereas TAU requires an intermediate format. The collective records are stored in a more convenient way with VampirTrace. TAU uses OTF counters to encode the collective's payload. The involved nodes have to be recovered with the functions's entry and exit points. VampirTrace utilizes the OTF collective handlers, so that all information regarding collective events are available and accessible from a single location.

Fig. 3 shows the abstract steps needed to acquire metrics from an application trace with SONAR.

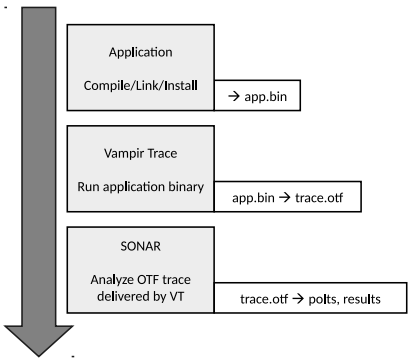


Fig. 3: Schematic view of the workflow of acquiring metrics with SONAR

3.3 Trace post-processing & exploration

Once the trace has been obtained with VampirTrace, it contains, among others, the following events:

1. Definitions: Nodes, Functions, Communicators
2. Function Enter/Leave Events
3. Point-to-point Messages: Send/Receive Events
4. Collective Messages: Begin/End Events

All these events are associated with a time stamp and additional information regarding this event, such as, the message length or type. VampirTrace stores the time stamp as ticks and provides a trace-specific ticks-per-second value to convert the time stamps into a reasonable unit. This should be considered when handling the raw OTF data.

When viewing the trace with the *otfprint* tool of the OTF library, the information looks like this:

```
$ otfprint mytrace.otf

// Functions:
(#38271) 5719742052 Enter: function 82, process 10, source 0
(#38272) 5719742504 Leave: function 0, process 9, source 0

// Point-to-point communication:
(#23768) 5708630907 SendMessage: sender 4, receiver 14, group
     1000000004, type 1375, length 80, source 0, KeyValue:
     1:5709047177
(#23812) 5708664706 ReceiveMessage: receiver 14, sender 4, group
     1000000004, type 1375, length 80, source 0

// Collective communication:
(#2536) 5689898534 BeginCollective: process 13, collective 6, group
     1000000004, matchingId 5, root 0, sent 0, received 0, source 0
(#2567) 5691141019 EndCollective: process 4, matchingId 5
```

Listing 1.1: OTF Events

This textual representation of the trace can be used to get an overview of the trace. Specific informations can be found and processed with the GNU tools *grep*, *sed*, *awk* and alike. To characterize traces automatically with a set of multiple metrics, this approach is not feasible, as the GNU tools tend to be very slow on large amounts of data and cumbersome to use when implementing new metrics. To be more efficient, we need to be able to process the trace's data as-it-is instead of the detour with the textual representation. SONAR uses the OTF library functions to access the numerical values shown in Listing 1.1 directly as their respective data type, e.g. integers.

4 Tool-Design

For SONAR, we used the C based OTF library[9]. It provides fast and convenient interfaces to access OTF traces from C/C++ and Python applications. SONAR itself has been implemented in C++.

[9] `https://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/forschung/projekte/otf/`

4.1 Implementation and prerequisites

In order to access traces, the high-level OTF library functions expect one handler function for each event. SONAR uses these handler functions as an interface to acquire data from the traces. Since a trace usually contains more events than needed, a selection of chosen events can be set up in the reader of the OTF library.

We provide a script that downloads the OTF library, configures, builds, and installs it to the current working directory. The OTF library has some dependencies itself, such as the *zlib* for the trace's compression. If this build process fails, these dependencies have to be resolved manually. The same holds true for the Boost C++ Libraries, which are used to handle the program options.

Gnuplot is used to generate the graphs. If gnuplot is not present on the system, this part will be skipped and SONAR generates only data files. These files are encoded as comma-separated values (CSV) to be (re-)used by any other data processing tool.

4.2 Custom metrics

SONAR can easily be extended with new metrics. To do so, we need to identify the data which is required for the metric. For example, we want to count all messages which are exactly 42 bytes in size. The data for this new metric is located in the handler function which is called on every outgoing messages.

The class *OTF_Handler* in the file "otf_handler.h "contains stub implementations of all available OTF handlers. In his function the location of the data of interest is handled. For a new metric, a new class must be derived from this base class and re-implement the relevant functions.

```
static int
handleSendMsg(void* userData, uint64_t time, uint32_t sender,
    uint32_t receiver, uint32_t group, uint32_t type,
    uint32_t length, uint32_t source, OTF_KeyValueList *list)
{
    if (length == 42)
        *((int*)userData)++;
    return OTF_RETURN_OK;
}
```

Listing 1.2: Example implementation to derive a new metric from the OTF-trace

The code listing 1.2 show the implementation of the new metric. The *user-Data*-pointer is defined as an integer to be incremented at ever message which is exactly 42 bytes in size. Metrics, which are more complex than this simple example, are likely to depend on several values. In this case, it is advisable to use a structure or class to organize the data.

5 Results

In this section, we present results that are generated by SONAR after an analysis of MPI traces of the following applications: HPL, Graph500, NAMD, LULESH, and AMG2013.

5.1 Test System

We use an HPC system that consists of 8 nodes. Each node hosts two six-core *Intel Xeon E5-2630 v2* CPUs and 64 GBs of RAM and runs a standard Linux distribution. The nodes are connected to each other with Gigabit Ethernet. All the traces and measurements were acquired with this system.

5.2 Benchmarks and Workloads

We used a set of benchmarks to evaluate the output of SONAR. The workloads represent the typical demands of an HPC environment. We chose the configuration of these applications in a way to keep the total runtime low and the resulting traces small.

HPL (High-Performance Linpack) is the benchmark used to determine the Top500-List[10]. It solves a dense $N \times N$ system of linear equations. The performance is reported in GFlops/s. In our test we used a dimension of $N = 50000$.

The **Graph500 Benchmark** is used to determine the Graph500-List. The workload is a breadth-first search (BFS) graph traversal. Unlike HPL, the Graph500 Benchmark relies more on the communication abilities of the cluster. For our tests, we used the reference implementation[11] with emulated one-sided communication and a scale factor of 12.

NAMD (Nanoscale Molecular Dynamics program) is a molecular dynamics simulation program[12]. Two widely known workloads are the *Apolipoprotein A1 (ApoA1)* and the *Satellite Tobacco Mosaic Virus (STMV)*. These workloads are publicly available and are commonly used to compare different systems against each other. The number of computational steps was limited to 100 for each workload to keep the traces small.

LULESH (Livermore Unstructured Lagrange Explicit Shock Hydrodynamics) represents the field of hydrodynamic simulations[13]. LULESH uses a stencil code to calculate the physical forces. The problem size parameter was set to 100, which results in one million elements per node. The number of iterations was limited to 500.

AMG2013 (Algebraic Multigrid Solver) solves linear systems of unstructured grids with the algebraic multigrid method[14]. For our tests we used the

[10] `http://www.netlib.org/benchmark/hpl/`
[11] `http://www.graph500.org/referencecode`
[12] `http://www.ks.uiuc.edu/Research/namd/`
[13] `https://codesign.llnl.gov/lulesh.php`
[14] `https://codesign.llnl.gov/amg2013.php`

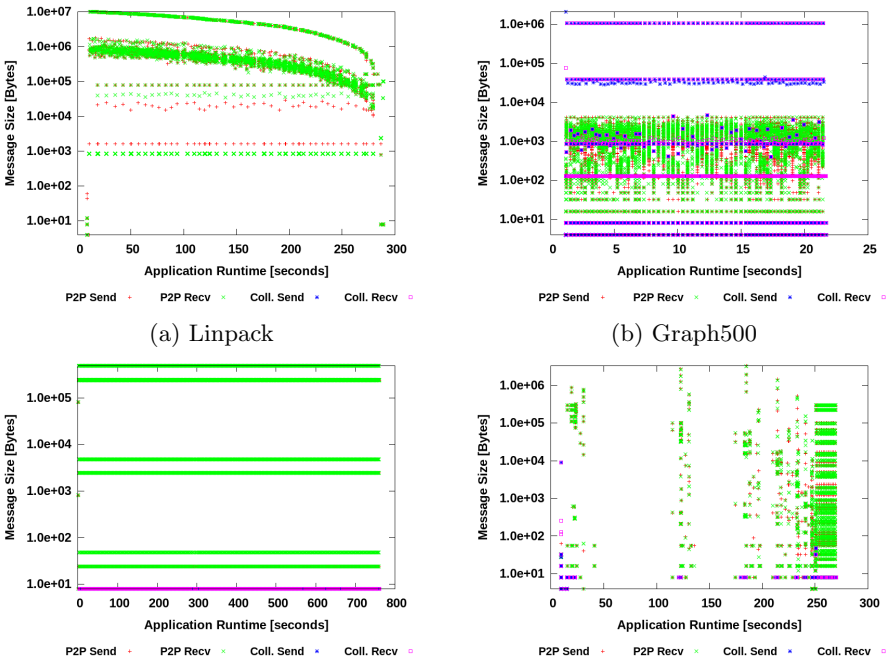
default problem and adjusted the problem scaling parameter to prolong the runtime of the application.

LULESH and AMG2013 are part of a collection of proxy applications[15] which represent current and future HPC workloads.

5.3 SONAR Measurements

In this section, we present the insights we have gathered with SONAR. We ran the workloads described in 5.2 on our cluster. The traces were acquired using the *vtrun* wrapper of VampirTrace. The following metrics were selected for first analyses.

Network activity: The graphics in Fig. 4 depict the network activity footprints of the High-Performance Linpack, Graph500, LULESH and AMG2013 benchmarks.



(a) Linpack (b) Graph500

(c) LULESH (d) AMG2013

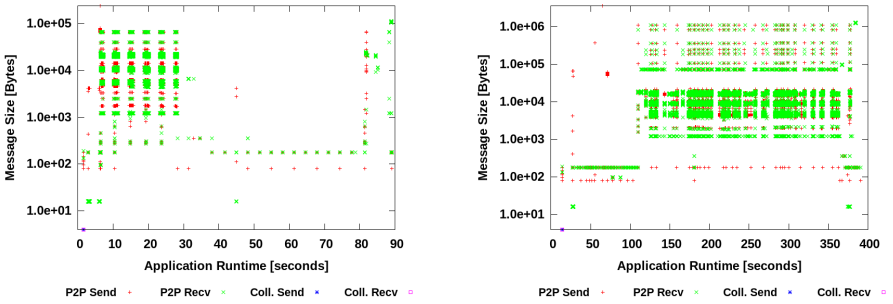
Fig. 4: Network activity of exemplary workloads

Each data point in the graphs represents a distinct event in the network. The position on the X and Y axis indicates the time of occurrence and respectively

[15] `https://codesign.llnl.gov/proxy-apps.php`

the size of the message. The colors visualize point-to-point (red, green) and collectives messages (purple, blue). SONAR produces such one graph for each node, recorded in the trace. Here, we selected only one graph per node, which we think is representative.

It is apparent that the communication characteristics vary widely between the different applications. This is not surprising for workloads, which are inherently different from each other such as the ones shown in Fig. 4. Fig. 5 depicts the NAMD application with two different sets of input data. Although both show periodic communication behavior, their network activity maps differ a lot.



(a) NAMD ApoA1 (b) NAMD STMV

Fig. 5: Network activity of the same application, but with different workloads

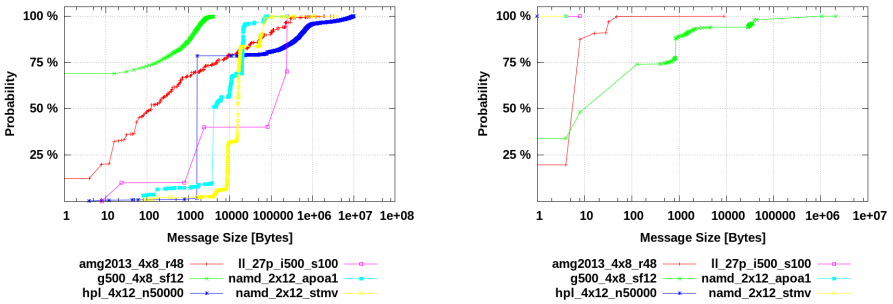
The ApoA1 workload causes dense communication patterns in the first third of the application's runtime. There are also some white spots which indicate no network activity at all. STMV communicates heavily from the middle to the end of the application and has fewer idle gaps.

Message Distribution: Fig. 6 shows the message distribution of our selected workloads as a cumulative distribution function (CDF). SONAR reports one CDF graph for each trace. For a better comparability, we merged them into one graph.

The most important insight is that point-to-point communication is preferred over collective communication. Only AMG2013 and the Graph500 are using collective messages to transfer data, which are larger than 10 Bytes. This suggests that the other workloads use collective operations only for synchronization purposes.

The second observation in Fig. 6 (a) is that about 80% of all point-to-point messages we gathered with our workloads are smaller than 20 kByte. One exception is Graph500, in which most messages are smaller than one kByte. The other one is LULESH with most messages being smaller than 200 kByte.

Aggregated Metrics: The results, which can be represented as numerical values, have been summarized in Table 1. The presented data are averaged values



(a) P2P Message Size Distribution (b) Coll. Message Size Distribution

Fig. 6: Message distribution of different workloads for point-to-point (a) and collective (b) messages

of all nodes. This table provides an overview how an application utilizes the cluster's components such as processors and interconnection network.

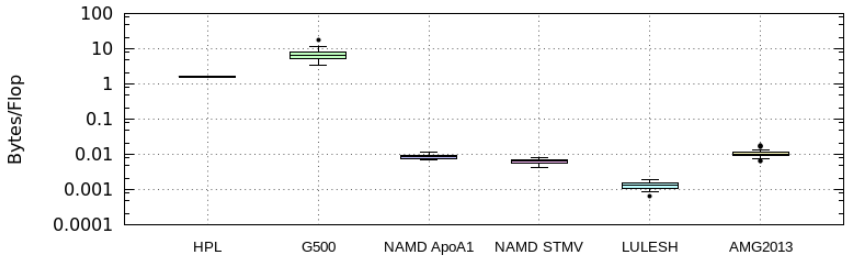
Application	MPI Processes	Verbosity [Bytes/Flop]	Message Rate [Messages/s]	MPI Idle min. [Seconds]	MPI Idle max. [Seconds]	MPI Idle avg. [Seconds]
Graph500	2x8	2.087e+01	4.795e+02	4.096e-06	1.256e+00	8.742e-04
Graph500	4x8	6.180e+00	5.924e+02	4.821e-06	2.849e+00	7.096e-04
Graph500	8x8	1.560e+00	8.697e+02	4.902e-06	3.411e+00	5.161e-04
HPL	2x12	9.597e-01	6.885e+01	2.528e-06	6.629e+00	7.160e-03
HPL	4x12	1.602e+00	3.729e+01	2.609e-06	6.250e+00	1.328e-02
HPL	8x12	1.478e+00	3.032e+01	2.654e-06	7.589e+00	1.638e-02
LULESH	8	9.398e-04	1.027e+01	4.132e-06	1.327e+00	4.695e-02
LULESH	27	1.236e-03	1.417e+01	3.764e-06	1.914e+00	3.825e-02
LULESH	64	1.409e-03	1.712e+01	3.536e-06	2.648e+00	3.210e-02
AMG2013	2x8	3.690e-03	1.598e+01	0.000e+00	8.757e+00	3.004e-02
AMG2013	4x8	1.009e-02	1.508e+01	0.000e+00	6.917e+01	3.290e-02
NAMD ApoA1	2x12	6.020e-03	8.501e+01	4.117e-06	4.317e+00	6.023e-03
NAMD ApoA1	4x12	8.651e-03	2.181e+01	4.453e-06	5.920e+00	2.356e-02
NAMD STMV	2x12	4.543e-03	6.861e+01	4.142e-06	1.375e+01	7.197e-03
NAMD STMV	4x12	6.274e-03	1.262e+01	4.389e-06	1.419e+01	3.951e-02

Table 1: Summary of aggregated SONAR metrics on the cluster level

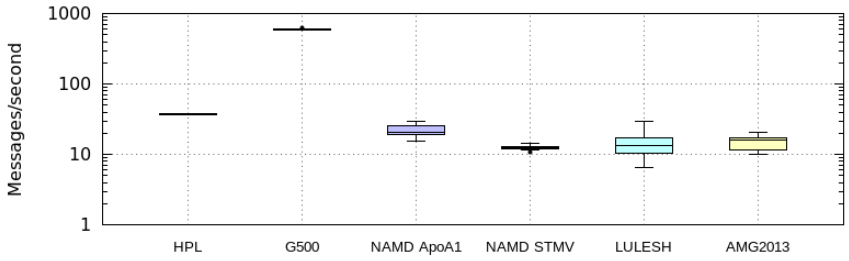
The information about sending and receiving of messages is measured at MPI level. Therefore, the MPI idle time represents the time period between two successive MPI events. As we take the send, as well as the associated receive event into account, these numbers give a reasonably accurate idea of the behavior of the underlying network interface.

Node divergence: To show how evenly an application utilizes the different nodes, SONAR produces also graphics that represent the data from Table 1 at node level. The box plots in Fig. 7 and 8 show the variance of these metrics spread over all nodes. Workloads with a low box and short whiskers indicate an

even utilization of each cluster node. This means the bigger the boxes the bigger the larger variance between all nodes.



(a) Verbosity



(b) Message Rate

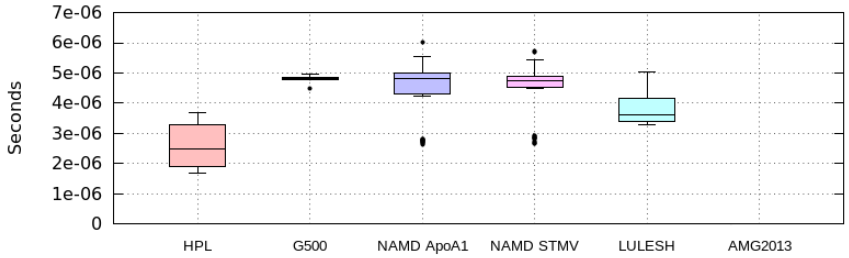
Fig. 7: Verbosity and message rate variance of tested workloads for all nodes

Fig. 7 depicts the variance of the metrics verbosity and message rate. The high verbosity of the Linpack benchmark surprises, respectively reveals the huge amounts of data this workload needs to transfer. Graph500 has the highest verbosity, as this work does not rely on floating point operations. Otherwise, the message rate is the largest of all tested workloads. This is because the Graph500 relies heavily on communication. NAMD, LULESH and AMG2013 show similar behavior regarding the verbosity and the message rate. They seem to be more efficient than Linpack, as they show a lower verbosity. This is in line with the message rate, which for these workloads is lower than on HPL.

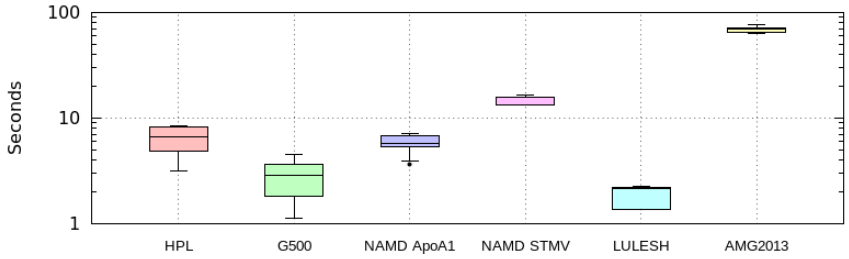
Fig. 8 shows the distribution of idle times on the MPI layer. The minimum idle time depends on the actual interconnect hardware. In our measurements, we see gaps of a few microseconds for successive messages. This is within the expected capabilities of the Gigabit Ethernet network we have used.

The average idle time between successive networks is between $20ms$ and $80ms$. The Graph500 is the outlier with an average gap of about $80\mu s$. Once again, this shows the communication demands of this workload.

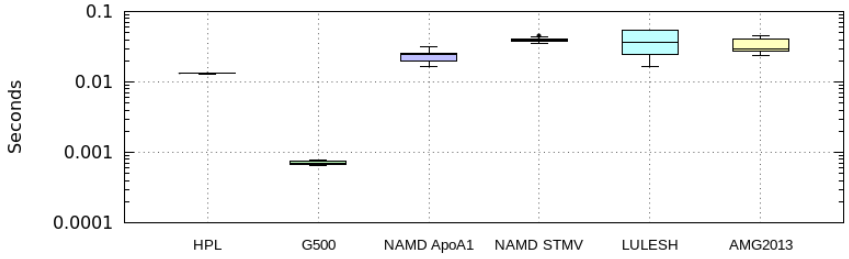
A hypothetical energy-proportional network must be able to switch its power state much faster than the average message gap to save energy and retain the



(a) MPI Idle Time: Minimum



(b) MPI Idle Time: Maximum



(c) MPI Idle Time: Average

Fig. 8: Idle time variance of tested workloads for all nodes

application's performance. The maximum idle time reveals, that many of the tested applications have at least one phase, where the network is not used at all for more than five seconds (HPL, NAMD, AMG2013). The apparently missing box of AMG2013's minimum idle time is in fact zero seconds on each node. This is likely caused by overlapping messages. AMG2013 shows a maximum idle time of even more than one minute for each node. We assume this is the result of an unfavorable configuration of the application. For a energy-proportional network, this means, during this idle time the network can be set to the lowest energy state or even be switched off completely.

6 Conclusion

With SONAR, we introduced a tool to derive advanced communication characteristics from traces of common HPC applications. These traces were obtained with VampirTrace, a well-known MPI trace generator.

Using these trace, we demonstrated the capabilities of SONAR by extracting various metrics, which we believe are crucial to develop a power-aware network model. For example, the generated network activity maps show a wide range of different communication patterns. Energy-proportional networks show significant power saving potential on workloads, such as NAMD or AMG2013. Various opportunities exist to save power without any loss in performance by dynamically reducing the link width or even by switching them off completely. Similar potentials are seen with LULESH and its highly regular communication pattern and fixed message sizes. Although links cannot be switched off completely, fine tuning the network is sufficient to save power for this workload.

Our observations of the network activity maps are also proven by other metrics, such as MPI idle times of the single nodes. SONAR revealed that the gaps between MPI events provide a possibility to put less active links into a reduced power state for the duration of these inactivity periods.

Supporting all these scenarios, however, requires an energy-proportional network infrastructure. Therefore, further research in this area is mandatory and we believe SONAR to be a first and important step in this direction.

7 Acknowledgements

We thank the anonymous reviewers for their constructive and detailed reviews. We would also like to express our thanks to Alexander Matz for his support. Furthermore, we want to thank Pedro J. Garcia and Jesus Escudero-Sahuquillo for our insightful technical discussions.

References

1. Saravanan, K.P., Carpenter, P.M., Ramirez, A.: Power/performance evaluation of energy efficient ethernet (eee) for high performance computing. In: Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on. (April 2013) 205–214
2. Zahn, F., Yebenes, P., Lammel, S., Garcia, P.J., Froning, H.: Analyzing the energy (dis-) proportionality of scalable interconnection networks. In: HiPINEB, 2016, 2016 2nd IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB). (2016) 25–32
3. Borkar, S.: Exascale computing - a fact or a fiction? (invited talk). In: IPDPS, IEEE Computer Society (2013) 3

4. Shalf, J., Dosanjh, S., Morrison, J.: Exascale computing technology challenges. In: Proceedings of the 9th International Conference on High Performance Computing for Computational Science. VECPAR'10, Berlin, Heidelberg, Springer-Verlag (2011) 1–25
5. Malony, A.D., Shende, S.: Performance Technology for Complex Parallel and Distributed Systems. In: Distributed and Parallel Systems: From Instruction Parallelism to Cluster Computing. Springer US, Boston, MA (2000) 37–46
6. Dandapanthula, N., Subramoni, H., Vienne, J., Kandalla, K., Sur, S., Panda, D.K., Brightwell, R.: INAM - A Scalable InfiniBand Network Analysis and Monitoring Tool. In: Euro-Par 2011: Parallel Processing Workshops: CCPI, CGWS, HeteroPar, HiBB, HPCVirt, HPPC, HPSS, MDGS, ProPer, Resilience, UCHPC, VHPC, Bordeaux, France, August 29 – September 2, 2011, Revised Selected Papers, Part II. Springer Berlin Heidelberg, Berlin, Heidelberg (2012) 166–177
7. Subramoni, H., Augustine, A.M., Arnold, M., Perkins, J., Lu, X., Hamidouche, K., Panda, D.K.: Inam2̂: Infiniband network analysis & monitoring with mpi. (2016)
8. Agelastos, A., Allan, B., Brandt, J., Cassella, P., Enos, J., Fullop, J., Gentile, A., Monk, S., Naksinehaboon, N., Ogden, J., Rajan, M., Showerman, M., Stevenson, J., Taerat, N., Tucker, T.: The lightweight distributed metric service: A scalable infrastructure for continuous monitoring of large scale computing systems and applications. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC '14, Piscataway, NJ, USA, IEEE Press (2014) 154–165
9. Mohr, B., Voevodin, V., Giménez, J., Hagersten, E., Knüpfer, A., Nikitenko, D.A., Nilsson, M., Servat, H., Shah, A., Winkler, F., Wolf, F., Zhukov, I.: The HOPSA Workflow and Tools. In: Tools for High Performance Computing 2012. Springer Berlin Heidelberg, Berlin, Heidelberg (2013) 127–146
10. Evans, T., Barth, W.L., Browne, J.C., DeLeon, R.L., Furlani, T.R., Gallo, S.M., Jones, M.D., Patra, A.K.: Comprehensive resource use monitoring for hpc systems with tacc stats. In: Proceedings of the First International Workshop on HPC User Support Tools. HUST '14, Piscataway, NJ, USA, IEEE Press (2014) 13–21
11. Gallardo, E., Vienne, J., Fialho, L., Teller, P., Browne, J.: Mpi advisor: A minimal overhead tool for mpi library performance tuning. In: Proceedings of the 22Nd European MPI Users' Group Meeting. EuroMPI '15, New York, NY, USA, ACM (2015) 6:1–6:10