SherlockFog: Finding Opportunities for MPI Applications in Fog and Edge Computing

Maximiliano Geier[1] and Esteban Mocskos[1,2(✉)]

[1] Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, C1428EGA Buenos Aires, Argentina
{mgeier,emocskos}@dc.uba.ar
[2] Centro de Simulación Computacional p/Aplic. Tecnológicas, CSC-CONICET, Godoy Cruz 2390, C1425FQD Buenos Aires, Argentina

Abstract. The Fog and Edge Computing paradigms have emerged as a solution to limitations of the Cloud Computing model to serve a huge amount of connected devices efficiently. These devices have unused computing power that could be exploited to execute parallel applications.

In this work we present SherlockFog, a tool to experiment with parallel applications in Fog and Edge network setups, specially focused on the MPI based applications. We propose a methodology to study feasibility of running parallel applications in Fog or Edge environments. We validate this tool contrasting experimental results with theoretical predictions reaching remarkable agreement between both.

We analyze the effect of worsening network conditions for several benchmarks of the MPI version of NAS Parallel Benchmarks on fog-like network topologies. Our results show that this impact is sublinear in some cases, opening up opportunities to use distributed, increasingly ubiquitous computational resources.

Keywords: Distributed systems · Fog and Edge Computing
Parallel applications · Benchmarks

1 Introduction

In the last years, the Cloud Computing model has emerged as an alternative to owning and managing computing infrastructure, thus reducing operation costs. These operations provide an efficient solution to serve several types of applications, such as web servers, databases, storage and batch processing. However, it is not without limitations. As the Cloud is usually farther away from the clients, latency-sensitive applications could suffer from performance degradation in this setup. The Fog Computing model is defined by Bonomi *et al.* [6] as a highly virtualized platform that provides compute, storage and networking services between end devices and the Cloud. This model allows certain services to be offloaded to end nodes, thus enabling services with lower latency requirements. Going further, the proliferation of IoT devices with increasing computing

E. Mocskos and S. Nesmachnow (Eds.): CARLA 2017, CCIS 796, pp. 185–199, 2018.
https://doi.org/10.1007/978-3-319-73353-1_13

power has given an identity to nodes at the edge of the network, defining the Edge Computing model [17]. In this model, nodes at the edge of the network cooperate among themselves to provide solutions at lower response time or bandwidth requirements, while improving data safety and privacy with respect to handling processing at the Cloud. As the number of nodes at the edge increase, it is also becoming increasingly difficult to serve all clients at the cloud with this infrastructure in terms of server load and required bandwidth. End nodes are usually slower and have worse connectivity than the main infrastructure, but both aspects have been steadily improving, enabling execution of more and more applications.

Scientific computing has gone down another path. The standard infrastructure to support this type of applications is cluster computing. It is commonplace that a cluster is homogeneous in computing power and topology (i.e. every node is connected to each other directly and runs at the same speed). This homogeneity allows developers not to worry about process placement. A variation that has appeared during the last decade is multi-core clusters. In this case, processes running on the same computer can communicate with each other faster than with processes on different nodes. Multi-cores are Symmetric Multiprocessing (SMP) machines: each node acts as an homogeneous cluster of CPU cores, which is connected to the outer world homogeneously as well. This paradigm is ubiquitous in scientific computing, driving research and improvements in simulation models and techniques.

The standard API used in scientific applications is MPI [11,12]. It has been designed to provide an abstraction to handle data exchange and synchronization among processes in the same node or in different ones. Developers of MPI implementations target their code to be used on clusters, often requiring full node connectivity and implementing optimizations for collective operations that work more efficiently in homogeneous environments. MPI is also very sensitive to network changes, bringing the computation to a halt if just one node disconnects or changes its network location.

This begets the question of whether it is possible to do scientific computing in the Edge Computing paradigm, taking advantage of unused computing power of nodes at the edge of network. The Edge/Fog paradigm poses new challenges in this regard, such as the variability on the processing power of nodes and the volatility of nodes due to the dynamic nature of the network in terms of churn, latency and bandwidth. Even though Edge Computing could provide better latency than having to offload data to do computing on the Cloud, specialized clusters are connected using a faster dedicated network. This difference could impair application performance greatly, making it unfeasible to use such an infrastructure. We approach this question by introducing a methodology to analyze distributed scientific applications in an heterogeneous environment, using different emulated network settings. It is important to note that assessing performance degradation is not easy from a theoretical standpoint. Factors such as link bandwidth and latency, communication patterns, message sizes, traffic congestion issues and implementation details of the underlying communication framework make it complex to model accurately.

Most work on leveraging MPI in heterogeneous environments has been focused on multi-core cluster architectures. This has been a natural step from traditional homogeneous clusters since the introduction of this type of processors into the market.

In this work, we present SherlockFog, a tool that aims to bridge the gap between existing scientific computing applications and Edge/Fog Computing. Our approach explores feasibility of running MPI applications in Edge Computing environments. This kind of architecture is inherently dynamic and has different topologies, performance characteristics and exploitable computing power per node than typical multi-core clusters. MPI on Edge Computing is still mostly unexplored, since computing power and communication capabilities of fog and edge nodes have only been made possible to compute High Performance Computing (HPC) applications very recently. Since implementations of existing MPI libraries have not been designed for this type of environments, providing a framework to test the library itself is also a key feature we have considered in our approach.

This work is focused on MPI as it is widely used by the scientific applications community, aiming to reuse existing code. However, our proposal provides a framework that allows the full software stack to be used unmodifed and it is therefore not limited to MPI applications.

2 Related Work

We discuss some representative examples of tools and metholodogies that can support the use of MPI applications in heterogeneous environments.

Brandfass *et al.* [7] propose a rank-reordering scheme to increase performance of unstructured Computer Fluid Dynamics (CFD) code on parallel multi-cores. This optimization produces a mapping of MPI processes to CPU cores such that main communication happens within compute nodes, exploiting the fact that intra-node communication is much faster than inter-node in this kind of architectures, using characteristics of the target application. Since load per process is not uniform in unstructured code, it makes sense to reorder processes to reduce frontier communication. Dichev *et al.* [10] show two novel algorithms for the scatter and gather MPI primitives that improve performance in multi-core heterogeneous platforms. This work focuses on optimizing broadcast trees used by most MPI implementations using a weight function that depends on topology information. However, the user can not experiment using virtual topologies, thus difficulting the study of MPI applications in edge-like environments. Mercier and Clet-Ortega [15] study a more sophisticated process placement policy that takes into account the architecture's memory structure to improve multi-core performance. This proposal is also not suitable for our purposes since the target platform is potentially dynamic and virtual topologies cannot be analyzed. Navaridas *et al.* [16] study process placement in torus and fat-tree topologies through scheduling simulation using synthetic workloads. This work relies on an execution model which would have to be adapted to study our target platform.

Simulation tools are also widely used to analyze distributed systems. This approach allows the user to explore environments that are difficult to set up in real life. In this case, the application is executed completely in a simulated environment, be it online or offline. This approach usually requires the user to modify application code in order to use the simulator. In the offline case, execution traces must be generated for a defined set of input parameters that are then fed to the simulator. A few representative examples thereof follow:

NS-3 [3] is a widely-used full-stack detailed discrete event simulator designed for network applications. However, since the simulator is not a parallel application itself, it is not possible to scale simulation of MPI applications beyond tens of nodes. It does not provide mechanisms to transform MPI applications directly into NS-3 simulations. An extension called DCE (Direct Code Execution) [2] wraps C library calls to be simulated by NS-3. However, this virtual C library is limited and does not implement all system calls required to run MPI applications.

SimGrid [8] is another widely used discrete event simulator. It is aimed at simulating large-scale distributed systems efficiently. Moreover, it provides an online mechanism to execute MPI applications by wrapping MPI calls to the simulator engine, called SMPI [9]. This API allows MPI applications to be ported to SimGrid simulations easily if the source code is available. In order to scale simulations on a single node, allowing up to thousands of simulated nodes, SimGrid implements a simplified communication model. While this simulator allows to experiment using heterogeneous network topologies, its communication models have only been validated for performance prediction accuracy on fine-tuned homogeneous environments. Moreover, this approach doesn't allow the user to experiment with different MPI libraries. The behavior of the HPC application itself, at the MPI primitives level, relies on implementation details of the simulation engine.

Dimemas [1] is a performance analysis tool for MPI applications, which is able to simulate execution offline on a given target architecture, ranging from single- or multi-core networked clusters to heterogeneous systems. Performance is calculated by means of replaying the execution trace of an application on a built-in simulator. Similarly to SimGrid, this approach relies on a particular implementation of MPI primitives and the communication model itself. We aim towards building a complete framework for analyzing distributed applications on Edge Computing, whereas simulation forces a particular model which depends deeply on the tool.

In all aforementioned simulation tools, traffic is simulated using a model that depends on the particular simulator implementation and on user-provided input parameters. Our work focuses not just on the applications themselves, but also on building a framework for analyzing and developing distributed applications and support libraries on the edge of the network.

Another approach is network emulation, which consists of building the execution environment using real nodes on a virtual network topology. Emulation allows using the same software environment as it would be used in a real system,

while providing different network conditions. This solves modelling the application and the communication framework. Several tools make use of traffic shaping facilities in modern operating systems to emulate the network and run distributed applications on it, but none of them focus on MPI applications in heterogeneous architectures. We cite a few examples:

Lantz *et al.* [14] present Mininet, a tool to emulate Software-Defined Networks (SDNs) in a single host. It leverages network namespaces and traffic shaping technologies of the Linux operating system to instantiate arbitrary network topologies. This tool requires virtual nodes to execute on a single host, impairing scalability. Moreover, as isolation occurs only at the network level, intelligent MPI implementations can determine that virtual hosts reside on the same real host and thus communicate more efficiently than expected.

Wette *et al.* [18] extend Mininet in order to span an emulated network over several real hosts. However, it does not address design limitations in the original tool that prevent it from running MPI code, such as hostname isolation on namespaces that share a filesystem.

White *et al.* [19] propose Emulab, a shared testbed that allows running MPI code on it, but it has dedicated hardware requirements that make it expensive to self-deploy. While it is possible for researchers to use the shared testbed at University of Utah, there is very little control of job allocation and bandwidth usage, and is therefore not suitable for performance analysis of CPU and network-intensive applications.

3 Methodology

We propose a novel methodology to support the analysis and porting of distributed computing applications to be executed following the paradigm of Edge Computing. Our proposal focus on the impact of different traffic patterns in applications. We have focused our work on MPI applications, as it is the most widely used API for message-passing distributed computing, but our approach is valid for other distributed programming models.

Figure 1 shows the process schematically.

1. The user selects an application and a topology and creates an experiment script for SherlockFog to deploy it on a set of physical nodes.
2. SherlockFog connects to every node and initializes network namespaces for each virtual node.
3. Virtual links are generated to match input topology.
4. Static routing is used to allow applications on each namespace to communicate to each other.

The tool allows the user to change network parameters during the run. This process can be repeated, comparing different topologies or input parameters. Application output is then analyzed, comparing behavior on different scenarios.

In the next paragraphs, we present the tool further and show the topologies we have used in our experiments. Moreover, we describe the experimental methology in detail.

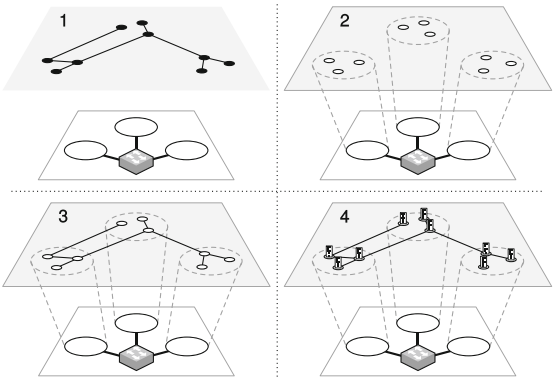

Fig. 1. SherlockFog allows the user to analyze application behavior while varying network topologies and their properties in a reproducible procedure.

3.1 SherlockFog: A Distributed Experimental Framework to Enable Fog and Edge Computing

SherlockFog is a tool that takes care of automating the deployment of a given topology and running the experiments. It makes extensive use of the `ip` tool–found on most GNU/Linux installations–to set up virtual Ethernet interfaces on Linux Network Namespaces. These interfaces are cloned using the `macvlan` feature[1]. A pair is connected by assigning IP addresses in the same point-to-point subnet[2] to both endpoints. Traffic flows through the carrier of the host network interface. This new interface has different MAC address and configuration (eg. name resolution dictionary, firewall, ARP and routing tables). Routing is also configured statically on every namespace in order to match the input topology. All virtual nodes act as routers, forwarding packets to its neighbors. Moreover, ARP is disabled on all virtual interfaces to prevent virtual nodes which are not neighbors in the input topology to find each other by sending ARP requests, thus bypassing our configuration. An SSH server is brought up on every container automatically to be able to run MPI applications. It runs on a different UTS namespace[3], whose hostname matches that of the virtual host. This feature is required for MPI applications to work on the virtual infrastructure, as some implementations check the hostname to define whether shared memory or a network transport should be used for communication. To the best of our knowledge, no other network experimentation tool takes this into account. It also allows MPI hostfiles or rankfiles to be set up more easily using consistent names, the choice of real nodes notwithstanding. Name resolution is handled by generating appropriate `/etc/hosts` files for each namespace automatically.

[1] https://hicu.be/bridge-vs-macvlan.
[2] A/30 network prefix.
[3] http://windsock.io/uts-namespace/.

These files are bound by the `ip netns exec` command. Finally, using the NetEm traffic control extension [13] via the `tc` tool, link parameters can be modified on a given virtual network interface's outbound port.

3.2 Features of SherlockFog

SherlockFog runs on commodity hardware, such as interconnected desktop computers in an university campus. No special interconnection technology or programmable switch is required, lowering the cost of ownership significantly with respect to similar solutions.

The scripting language allows to set up experiment parameters and runs, enabling for reproducible experimentation.

The tool can connect namespaces in the same physical computer or in different hosts, provided that the traffic that every host generates is reachable from the rest. This allows to grow experiment scale by using hosts on different interconnected switches.

Application code can be executed unmodified. The user can execute open- or closed-source programs in the same software environment as they would in a real environment.

It is also possible to experiment with changes to the MPI library, such as broadcast implementations for edge environments or features that make it more resilient to churn or changes in latency or bandwidth. As we are exploring MPI on non-standard network settings, our tool could be used as a testing framework for these use cases.

Modeling mobility is also an important aspect in Fog or Edge Computing environments. Our tool provides a mechanism to do so by changing bandwidth and packet loss for a link.

We show a sample experiment script in Fig. 2. In this example, 4 nodes named *n0* to *n3* (line 2) are initialized and connected sequentially, generating a

```
1  ### node def
2  for n in 0..4 do def n{n} {nextRealHost}
3  ### connect nodes
4  connect n0 n1
5  connect n1 n2 5ms
6  connect n2 n3
7  ### build
8  build-network
9  ### run exp
10 for m in 10..101..10 do
11   runas n0 netns n0 myuser mpirun -f h.txt ./p {m} > {m}.log
12   set-delay n1 n2 {m}ms
13 end for
```

Fig. 2. Example SherlockFog experiment script to launch a virtual topology of 4 nodes and execute an MPI application in different network conditions.

"linked-list", while setting up a 5 ms delay between nodes *n1* and *n2* (lines 4–6). Line 8 configures IP addresses for all nodes and sets up static routing tables accordingly. Finally, lines 10–13 run the actual experiment: an MPI application is repeatedly executed with argument m ranging from 10 to 100 in steps of 10, increasing latency between *n1* and *n2* on each step while saving its output for offline analysis.

3.3 Considerations When Using SherlockFog

We discuss a few usage considerations for SherlockFog:

Due to the usage of static routing, two different paths from one node to another are not allowed. Minimum Spanning Tree (MST) can be calculated on any topology to define unique paths for every pair of nodes. Real (dynamic) routing protocols on the network could give us one such path configuration.

As traffic is routed from the host carrier to the right namespace by looking up its destination MAC address, it is not possible to experiment with applications that make use of multicast messages. However, as our main focus is MPI applications and most implementations handle global communication using multiple unicast messages on some sort of virtual tree, this is not a limitation for our experiments.

Total bandwidth is shared among nodes. The user must be careful not to overflow the actual carrier. It is possible avoid this by limiting maximum bandwidth in each virtual network interface.

Finally, real link latency must be taken into account when designing the experiment. Since SherlockFog can scale on nodes on different switches, it is likely that pairwise latencies differ. They must be taken into account, as latency is increased on top of the actual link's. As it is the case for all network emulation tools, this could lead to inaccurate results if latency increments are closer to the underlying link's values.

3.4 Underlying Topology

All experiments in this work were run on an 8-node cluster of AMD Opteron 6276 processors with 128 GB of RAM. Each node has 64 cores and runs Debian GNU/Linux 8.7 amd64 with kernel version 4.9.18. MPI applications were compiled and executed using MPICH version 3.2.

Initial tests using this hardware show that no more than 48 cores can be used at the same time without incurring in performance hiccups. This is consistent with behavior of other applications used in this hardware and is related to a bottleneck of the memory bus.

4 Validation

In this section, we propose experiments to study the accuracy of our methodology in representing different network scenarios, which are defined in Table 1.

Table 1. Network topologies.

Topology name	Sizes	Description	Reference in text
Barabási-Albert	100 nodes	Random graph generated using the Barabási-Albert model for scale-free networks with preferential attachment	barabasi
Isles	16, 64 and 256 nodes	Two clusters of nodes (star topology) connected through a single path	isles

The isles topologies represent two interconnected clusters of computational resources. These clusters are connected to each other through a single distinguished link. The latency of this link indicates the distance in terms of communication. This scenario represents, for example, two sets of nodes in the edge of the network which are connected to a common infrastructure such as the Internet.

Let n be the size of the network, the process placement rules are:

1. The distinguished link connects the first node (node 0) to the last one (node $n-1$).
2. The nodes are partitioned evenly on each cluster.
   - Nodes 0 to $\lfloor \frac{n-1}{2} \rfloor$ go to the first cluster.
   - Nodes $\lceil \frac{n-1}{2} \rceil$ to $n-1$ go to the second cluster.
3. The nodes connected by the distinguished link become the exit nodes for each cluster.
4. Every other node is connected to its respective exit node.

The barabasi topology is a random graph generated using the the Barábasi-Albert model for scale-free networks with preferential attachment. It represents a connectivity model which is found on the Internet [5]. This topology was generated using model parameter $m_0 = 2$. In this case, processes are assigned randomly.

We will show that SherlockFog can emulate different network conditions by analyzing prediction output compared to the expected theoretical results.

4.1 Latency Emulation

In order to show how latency emulation works, we need to use an application with a traffic pattern for which we can obtain an analytical expression for the total communication time. By doing so, we can then compare the expected theoretical time to the output of our tool.

In particular, we have used an implementation of a passing token through a ring logical topology. Each node knows its neighbors and its order in the ring. Token size is configured to be a single integer (4 bytes) throughout this work.

The number of times the token is received by the initiator (rounds) is also a parameter of the application.

We have analyzed total number of messages on the network and execution time for this application, using two different implementations:

- **Token Ring:** implements communication using TCP sockets. This version allows us to have fine grain control of message generation and protocol.
- **MPI Token Ring:** same application, but using MPI for communications. In this case, we can test if the use of the MPI library could also be managed by our tool.

Since we know the traffic pattern, if we were to keep the topology unchanged, but increased latency of one or more links, it would be easy to estimate how much longer the application would take to complete with respect to the original network settings. This increment is calculated as follows: let N be the number of nodes in the topology, t_0 the original execution time, $c_{i,j}$ the total send count from node i to node j and $w_{i,j}$ the shortest path weight[4] from node i to node j, the expected execution time t_e is defined by:

$$t_e = t_0 + \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} c_{i,j} \cdot w_{i,j} \tag{1}$$

It is important to emphasize that Eq. 1 represents the expected execution time accurately only since **Token Ring**'s traffic pattern is sequential. Otherwise, we would have to take into account communication overlapping.

We calculated the original execution time, with a fixed latency value on all links, the estimated times for different latency settings and the actual execution times when using SherlockFog with those settings. The full description of the runs is shown in Table 2.

Table 2. Parameter configuration for validation experiments

Application	Topologies	Argument range	Latency	Comments
Token Ring	barabasi and isles	100–1000 rounds	10, 90, 170 ms	Latency increased on all edges
MPI Token Ring		100–1000 rounds	5–25 ms	Latency increased on a single edge

4.2 Token Ring

In Table 3, a partial view of the results is shown. We can observe that the predicted time differs from the measured time by less than 1% in all cases. This is also consistent with the rest of the results for all round counts, latencies and topologies in our experiment set.

[4] Each link's weight is set to how much its delay is increased with respect to the underlying platform.

Table 3. Excerpt of validation results for Token Ring, 300–500 rounds, on barabasi (100 nodes). Latency added uniformly to all edges.

Rounds	Latency (ms)	Predicted (s)	Measured (s)	Error
300	5.00	615.84	619.39	0.0057
300	15.00	1801.78	1810.63	0.0049
300	25.00	2987.72	2983.03	0.0015
300	35.00	4173.66	4177.55	0.0009
300	45.00	5359.60	5347.30	0.0022
400	5.00	820.27	826.30	0.0073
400	15.00	2400.21	2402.16	0.0008
400	25.00	3980.15	3978.04	0.0005
400	35.00	5560.09	5554.26	0.0010
400	45.00	7140.03	7130.86	0.0012
500	5.00	1024.79	1033.60	0.0085
500	15.00	2998.73	3007.53	0.0029
500	25.00	4972.67	4973.75	0.0002
500	35.00	6946.61	6943.70	0.0004
500	45.00	8920.55	8955.64	0.0039

4.3 MPI Token Ring

The MPI version produces similar results. In this case, error ranges are slightly higher, but also remain below 1% in all cases, even on a topology on which the logical order of the nodes produces a complex communication path in this application, such as barabasi. We consider that this is due to the fact that MPICH handles messaging differently than in our plain TCP implementation, though we find this difference not to be significant.

We can conclude that latency is accurately represented in our tool when executing applications that use MPI for communication on different emulated topologies.

5 Results

In this section, we show the effects of latency on different scenarios in the MPI version of NAS Parallel Benchmarks [4]. These benchmarks are derived from Computational Fluid Dynamics applications and have been throughly tested and analyzed by the community.

We have chosen three kernels (IS, CG and MG) and two pseudo applications (BT and LU) and evaluated performance loss on the isles topologies.

All benchmarks were executed using SherlockFog to increase the latency of the distinguished link up to 100 times for three different problem sizes (A, B

and C). These problem sizes are standard for this kind of applications and are defined such that going from one class to the next represents a four-fold increase. All experiments were repeated 5 times.

Our interest lies in finding out how the performance of these benchmarks–which were designed to be executed on a single cluster of nodes with low communication overhead–fares in this use case, comparing total execution time for each latency value to its no-extra-latency counterpart[5].

Each of the plots presented in this section describes the increment in total execution time as a function of the increment in latency for all network sizes. The semi-transparent patches over the curves show the standard deviation for each data series.

The results for all benchmarks are shown in Fig. 3. We can observe similar patterns for each network size.

On 16 nodes, total execution time for all network sizes grows linearly as latency is increased. In the worst case, a 100-fold latency increase results in 14 times slower total execution time. The slope of the curve is usually lower as the problem size grows: problem size C has less of an impact than the smaller sizes on most cases. We believe this to be related to the fact that each process has more work to do, reducing the impact of the overhead in communication.

On 64 nodes, the difference between problem sizes A and C is more significant. Moreover, we can also observe that the maximum increment is much lower than in the smaller network, up to 3.5 times for a 100-fold latency increase. For BT, CG and MG (problem size A), the incidence is also much more significant for smaller latency values. For example, increasing latency 10 times in CG, size A, results in the application taking 2.5 times more to complete. However, increasing latency 100 times results in it only taking 3.4 times more. LU, on the other hand, doesn't show a significant performance loss for all latency values.

On 256 nodes, we can observe similar results to 64 nodes. However, in this case, the scale is much smaller: the worst case is shown in MG, size A, which takes twice as much time to complete when subject to a 100-fold latency increment. The case of CG is also interesting, as going from no latency to 1 ms results in the benchmark taking 1.6 times more to complete. However, increasing the latency further doesn't produce a noticeable effect. This is similar to the results for 64 nodes, but the effect is more pronounced. We believe this to be related to the communication that goes through the distinguished link representing a much smaller ratio with respect to smaller networks.

Finally, in the case of IS, the increments in total execution time are much less noticeable than in the previous cases. On 16 nodes, the curves for each problem size tend to drift away from each other as the latency goes up. However, as the node count goes up, the effects of changes in latency on this topology are much less noticeable. We can conclude that this application is not greatly impaired, being the most fog-ready of all these benchmarks in this particular scenario.

[5] In the no extra latency case, the topology remains unchanged, but the latency of every link is exactly the same.

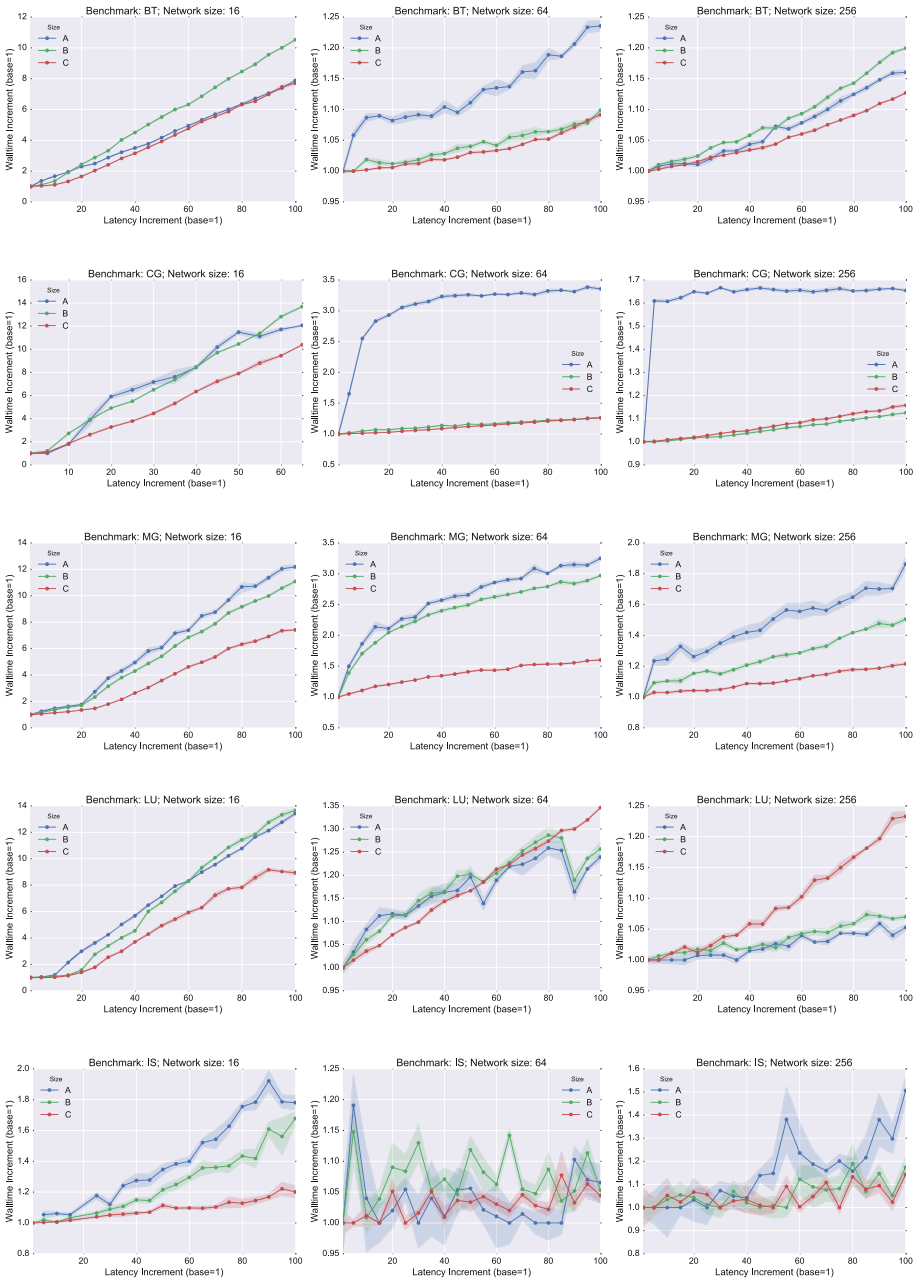

Fig. 3. Slowdown on the isles topologies as a function of the increment in latency of the distinguished link in different NAS parallel benchmarks.

6 Conclusions

In this work we introduced SherlockFog, a tool that enables experimentation with MPI applications in Fog and Edge Computing scenarios. We proposed a methodology to analyze if an MPI application can be deployed on a Fog or Edge scenario without incurring in a big performance loss, given its communication pattern and that particular network setting. Our tool also provides a testing framework to explore MPI applications and library implementations in heterogeneous scenarios.

Latency emulation in SherlockFog was validated by estimating the communication overhead in a custom application that implements a token ring. This application describes a sequential communication pattern and is therefore suitable for estimating the overhead theoretically.

We have analyzed five well-known benchmarks that use MPI to reproduce patterns in computation similar to those of CFD applications. We proposed a network topology in which two clusters are connected to each other through a single distinguished link. Using this topology, we have evaluated the impact of increasing the latency of the distinguished link on the performance of each application.

All results show a linear or sublinear impact on this particular topology, opening up opportunities to use distributed, increasingly ubiquitous computational resources.

As future work, other aspects of the Edge/Fog paradigm such as the dynamic nature of the network have to be studied. This requires adapting the MPI programming model to handle node churn and changes in logical topology. SherlockFog also models changes in bandwidth. This feature should also effect application performance but have not yet been evaluated.

Acknowledgements. The authors would like to thank D. González Márquez for his assistance with schematic drawings and the Centro de Simulación Computacional para Aplicaciones Científicas/CSC-CONICET and the Centro de Cómputos de Alto Rendimiento (CeCAR, FCEN-UBA) for providing the equipment we haved used in the experimental setup.

References

1. Dimemas. http://tools.bsc.es/dimemas. Accessed 2 Dec 2017
2. ns-3 Direct Code Execution. https://www.nsnam.org/overview/projects/direct-code-execution/. Accessed 2 Dec 2017
3. ns-3 Overview. https://www.nsnam.org/docs/ns-3-overview.pdf. Accessed 2 Dec 2017
4. Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V., Weeratunga, S.: The NAS parallel benchmarks. Report RNR-94-007, Department of Mathematics and Computer Science, Emory University, March 1994
5. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. Science **286**, 509–512 (1999)

6. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC 2012, pp. 13–16. ACM, New York (2012). http://doi.acm.org/10.1145/2342509.2342513
7. Brandfass, B., Alrutz, T., Gerhold, T.: Rank reordering for MPI communication optimization. Comput. Fluids **80**, 372–380 (2013). Selected Contributions of the 23rd International Conference on Parallel Fluid Dynamics ParCFD2011. http://www.sciencedirect.com/science/article/pii/S004579301200028X
8. Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, scalable, and accurate simulation of distributed applications and platforms. J. Parallel Distrib. Comput. **74**(10), 2899–2917 (2014). http://hal.inria.fr/hal-01017319
9. Degomme, A., Legrand, A., Markomanolis, G., Quinson, M., Stillwell, M., Suter, F.: Simulating MPI applications: the SMPI approach. IEEE Trans. Parallel Distrib. Syst. **PP**(99), 1 (2017)
10. Dichev, K., Rychkov, V., Lastovetsky, A.: Two algorithms of irregular Scatter/-Gather operations for heterogeneous platforms. In: Keller, R., Gabriel, E., Resch, M., Dongarra, J. (eds.) EuroMPI 2010. LNCS, vol. 6305, pp. 289–293. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15646-5_31
11. Gropp, W., Lusk, E., Skjellum, A.: Using MPI: Portable Parallel Programming with the Message-Passing Interface, 2nd edn. MIT Press, Cambridge (1999)
12. Gropp, W., Lusk, E., Thakur, R.: Using MPI-2: Advanced Features of the Message-Passing Interface, 2nd edn. MIT Press, Cambridge (1999)
13. Hemminger, S.: Network emulation with NetEm. In: Pool, M. (ed.) LCA 2005, Australia's 6th National Linux Conference (linux.conf.au). Linux Australia, Sydney (2005). http://developer.osdl.org/shemminger/netem/LCA2005_paper.pdf
14. Lantz, B., Heller, B., McKeown, N.: A network in a laptop: rapid prototyping for software-defined networks. In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Hotnets-IX, pp. 19:1–19:6. ACM, New York (2010). http://doi.acm.org/10.1145/1868447.1868466
15. Mercier, G., Clet-Ortega, J.: Towards an efficient process placement policy for MPI applications in multicore environments. In: Ropo, M., Westerholm, J., Dongarra, J. (eds.) EuroPVM/MPI 2009. LNCS, vol. 5759, pp. 104–115. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03770-2_17
16. Navaridas, J., Pascual, J.A., Miguel-Alonso, J.: Effects of job and task placement on parallel scientific applications performance. In: 2009 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, pp. 55–61, February 2009
17. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: vision and challenges. IEEE Int. Things J. **3**(5), 637–646 (2016)
18. Wette, P., Dräxler, M., Schwabe, A.: Maxinet: distributed emulation of software-defined networks. In: 2014 Networking Conference, IFIP, pp. 1–9, June 2014
19. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An integrated experimental environment for distributed systems and networks. In: Proceedings of the Fifth Symposium on Operating Systems Design and Implementation, pp. 255–270. USENIX Association, Boston, December 2002