On the Effects of Data-Aware Allocation on Fully Distributed Storage Systems for Exascale

Jose A. Pascual(✉), Caroline Concatto, Joshua Lant, and Javier Navaridas

Computer Science School, The University of Manchester, Manchester, UK
{jose.pascual,caroline.concatto,joshua.lant,
javier.navaridas}@manchester.ac.uk

Abstract. The convergence between computing- and data-centric workloads and platforms is imposing new challenges on how to best use the resources of modern computing systems. In this paper we show the need of enhancing system schedulers to differentiate between compute- and data-oriented applications to minimise interferences between storage and application traffic. These interferences can be especially harmful in systems featuring fully distributed storage systems together with unified interconnects, such as our custom-made architecture ExaNeSt. We analyse several data-aware allocation strategies, and found that such strategies are essential to maintain performance in distributed storage systems.

Keywords: Near-data computing · Scheduling · Resource allocation

1 Introduction

Traditional supercomputers have been used to execute large computing-intensive parallel applications such as scientific codes. However, nowadays new types of data-oriented applications are becoming increasingly popular. In contrast with traditional HPC codes, they have to process massive amounts of scientific or business-oriented data and, hence, impose completely different needs to the computing systems.

Indeed, new hardware and software are being developed to suit these necessities, such as our novel, custom-made architecture, ExaNeSt [13]. We are working on the design and construction of a prototype capable of reaching Exascale computation using tens of millions of interconnected low-power-consumption ARM cores [1]. To support such kind of data-intensive applications we are leveraging a unified, low-latency interconnect and a fully distributed storage subsystem with data spread across the nodes. This greatly contrasts with traditional supercomputers and datacentres that rely on Storage Area Networks (SAN) to access the data with separate networks for I/O, system management and application traffic.

A fully distributed file system allows for near-data computation reducing the great overheads of moving data from centralized storage to the compute nodes. A

D. B. Heras and L. Bougé (Eds.): Euro-Par 2017 Workshops, LNCS 10659, pp. 725–736, 2018.
https://doi.org/10.1007/978-3-319-75178-8_58

single, consolidated interconnect offers enormous power-savings when compared with multi-network designs. While these design decisions do, indeed, allow us to cope with power and cost design constraints, they also exacerbate the challenges arising from workload convergence as storage traffic will be distributed all across the system which can interfere negatively with application traffic. We show that job scheduling, in particular the allocation phase where resources are assigned to applications, can have a huge impact on performance.

This is precisely our objective: understanding to what extent the mix of application and storage traffic interfere with each other and how this affects performance. Hence, we conducted an extensive evaluation of data-aware allocation strategies for data-intensive applications which take into consideration the location of both storage devices and data when deciding where application tasks will be allocated. For completeness, we compare these allocation strategies with a baseline HPC SAN-based system. Our evaluation relies on a novel, generic application model that generates synthetic workloads mimicking different types of application, i.e. I/O-, computation- or communication-intensive.

Results show that application performance can be severely degraded when mixing both types of traffic, unless careful allocation of resources is orchestrated, but also that proper resource allocation can outperform traditional storage approaches.

The rest of the paper is organized as follows. In Sect. 2 we discuss some previous works on data-aware allocation for large-scale computing system. Following in Sect. 3 we provide an overview of the architecture of ExaNeSt, specifically the storage and interconnection subsystems. We continue in Sect. 4 explaining the scheduling process and the simple allocation strategies considered in this paper. Then in Sect. 5 we present the experimental framework used to asses the impact of these strategies on the performance of the applications. These results are analysed and discussed in Sect. 6. We close the paper with Sect. 7 which highlights some concluding remarks and sets some future lines of research arising from the findings of this work.

2 Related Work

To the best of our knowledge this is the first time that a fully distributed storage subsystem based on high-performance solid state devices has been leveraged with a unified interconnect that handles both application and storage traffic in the context of high-performance computing system. Hence, there is no previous work tackling resource allocation when such a specific architecture is considered.

Some similar works are focused on allocating applications close to the data either in memory (Spark, see [20,21]) or in storage (Hadoop, see [4,8,10,24]). In all cases the authors present scheduling techniques to maintain data locality in either Hadoop-like or Spark-like clusters. Regarding traditional clusters, the insufficiency of traditional CPU-oriented batch schedulers was exposed and Stork, a scheduler that uses a job's description language to manage data location, was proposed [14]. Other works try to assign the application to the node where the data is mapped

or at least, as close as possible [22]. Other approaches try to maintain the locality dynamically based on the status of the system and the network [11]. A detailed overview of data-aware scheduling can be found in [7].

There exist also plenty of previous work centred around the allocation and mapping of applications to reduce the overhead of inter-process communications, mainly within the realms of HPC systems and parallel applications (e.g. MPI-based). These disregard data locality as the proportion of storage traffic is negligible and, indeed, dealt with by a separate network, as explained above. Many authors [6,17,18] analyse the extent that inter-application interference has on their performance. In order to minimise this interference, many non-contiguous [12,19] and contiguous [18,19] allocation strategies have been proposed for a range of topologies. Similarly, other works [2,5,17] have tried to reduce intra-application contention using different techniques to map the tasks of the application onto the previously selected nodes. This paper motivates the need for merging these two approaches so to obtain the benefits of minimizing both inter-process and storage interferences.

3 The ExaNeSt Architecture

In this section we describe ExaNeSt's architecture. One of the main novelties of our design is the affordance of non-volatile storage devices [23] (NVM) within compute nodes so to reduce latency and energy by exploiting data-locality. Compute nodes will access the storage subsystem transparently using BeeGFS [3], a high performance parallel filesystem that is in charge of reading and writing data between the local NVMs and the external storage system.

In Fig. 1 we depict an overview of the model we use in this paper for the storage subsystem which is based on the typical datacentre storage architecture. In the right side we can see the computing elements (circles) and the NVM devices connected directly to them (squares). Compute nodes access remote NVMs through our custom-made interconnection network (IN, hereafter) which is also used for

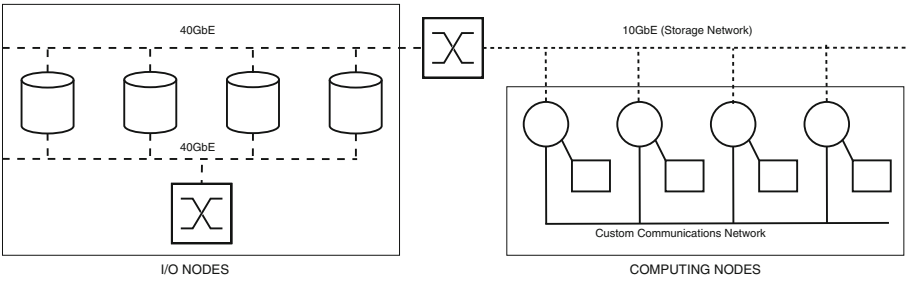


Fig. 1. Visual representation of the ExaNeSt storage architecture. The local NVMs are attached to the computing nodes sharing the main interconnect (solid). An Ethernet network is provided for persistent data storage (dashed).

interprocess-communication (solid lines). NVMs access the external storage back-end, in the left, using an independent storage network (dashed lines). For simplicity, in this work we have modelled the external storage network as a 10 GB Ethernet network which connects the compute nodes to the storage servers using a front-end 10 GB switch (this network is only to be used for I/O operations from/to the persistence storage when applications start/end the execution). As we try to model a realistic high performance system we also model a 40 GB back-end network which is in charge of data replication within the storage servers. The operation of this storage system is the typical in big datacenters [15]: when a computing node writes to disk, one of the servers will be chosen and data-replication to other servers (the number of replicas is configurable) will occur in the background without requiring user-intervention. In case of read operations the computing elements will access several storage servers (the number of replicas) and perform the operation in parallel, in order to improve the throughput. Note that although we have used this standard model for the storage subsystem, the architecture of the persistent storage is still an open question in ExaNeSt; and more efficient solutions are likely to be implemented in the final prototype.

For the purpose of this work, we define data to be **cached** if it is in main memory which allows very fast access to the data (we assume an average bandwidth of 10 GB/s) and **non-cached** if it is in an NVM. Also we define data as being **local** if it is located in the node where it is needed or **remote** if it is located in a nearby node where it can be retrieved from using the IN (performed transparently by BeeGFS). Finally data available only in central storage is denoted as **Central**. Therefore there are 5 possibilities when applications access data:

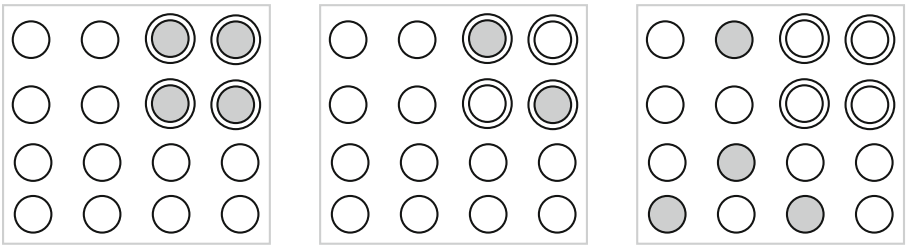
- **Local access, cached data:** This is the fastest access mode. As data are local and cached in main memory, the only limiting factors will be the latency (very low) and bandwidth (very high) of the memory.
- **Local access, non-cached data:** In this case the data are local but not in RAM. Therefore access to the NVM device is required. The limiting factors are the latency (low) and bandwidth (high) of the NVM.
- **Remote access, cached data:** In this case data is not available locally requiring access through the IN, so the limiting factor in this case will be the latency and bandwidth of the main IN, which is highly affected by external factors that could degrade its performance such as traffic interference.
- **Remote access, non-cached data:** This is the worst possible situation. The access to the IN is required because the data are not local but, in this case, both the remote NVM and the IN can become the limiting factor.
- **Central access:** We differentiate two different scenarios here. In ExaNeSt, BeeGFS access the external storage when applications start or finish execution, transferring data between persistent storage and the NVMs, so that applications always access data from the NVMs. The baseline configuration (SAN), represents an scenario where the applications do not use the NVMs so all accesses are done against the external storage; i.e., the SAN will be accessed whenever applications require to read or write data.

4 Scheduling and Resource Allocation Strategies

The scheduling process in a supercomputer involves, at least, three different stages. Applications are submitted to scheduling queues where, following some scheduling policy [9] such as FCFS, Backfilling or Shortest Job First (SJF), they are selected to be executed. After this stage, the allocator must find a set of suitable resources (physical nodes) usually fulfilling some constraints imposed by the application such as available memory, number of cores, type of architecture, etc. Finally the tasks (instances) of the application are mapped to those resources. In this Section we focus on the allocation stage in order to analyse the impact of data location on the performance of the applications.

Once an application has been selected to be run, the allocator will select a set of computing nodes to place the tasks of the application. In that moment, the application will request access to the required data and BeeGFS will load it from persistent storage into local storage. Ideally all the data will be local to each application, meaning that all accesses will be performed within local NVMs. However in a real system with many applications running concurrently and data-oriented applications demanding immense storage space, local-only accesses could be impossible to accomplish. Figure 2 represents the three possible types of storage assignment based on the interference they create in the interconnect:

- **Local:** All the local storage devices are available to load the data for the application. This is the ideal scenario where all the storage traffic remains local and, hence, there is no traffic interference.
- **Internal:** In this case only some of the local storage devices are available. This situation could happen if other applications have requested some of these storage devices previously. This will impose some intra-application interference, but will not generate inter-application interference if the applications are allocated consecutively.



(a) Local. (b) Internal. (c) External.

Fig. 2. Examples of application allocation. Double-circled nodes represent the nodes assigned for compute whilst grey circles represent these for storage. Note that the 2D mesh is used for illustration purposes only.

– **External:** Some (or all) the storage devices are outside of the partition assigned to the application. Now, remote accesses to the data will generate intra- and inter-application interference.

As discussed, the assignment of the storage devices to applications depends on both internal and external factors. Internal factors are the storage space required by the application which could be larger than that available within the local NVMs and the way data is partitioned which could impose access to remote NVMs. External factors are caused by other applications using NVMs outside of their local nodes. This will lead to fragmentation making new applications to allocate storage in remote NVMs instead of in local ones (already busy). In the long run this may end up with no application being able to use local NVMs.

These factors motivate the need of resource allocation policies in order to minimize both fragmentation and interference among traffic of different applications. To this end, the scheduler (allocator) should be enhanced to incorporate knowledge about the data access patterns of the applications and about the physical topology of the network. In this work, we consider two very simple allocation strategies for typical HPC topologies: fat-tree and 3D torus. These strategies will use contiguous partitions, in which the communications of the applications remain internal within the assigned nodes. Strategies to look for contiguous partitions can be found in [16,18,19]. The second strategy is random, that mimics the behaviour of a datacenter not using any locality-aware allocation. At any rate, neither of these strategies considers the actual communication patterns or the data access patterns of the applications and so there is no attempt to reduce internal contention. Of course, we envision both reducing both external and internal contention through optimised allocation essential to take advantage of the colossal raw computing power of Exascale systems. Indeed, part of our current work is the design of strategies that take into account specific information of the applications in order to select the best set of nodes to allocate them, see e.g., [17]. This selection will consider several application metrics with the goal of reducing the interference between inter- and intra-application storage and communication traffic.

5 Experimental Set-Up

In this section we present the simulation environment used to evaluate the effects of the allocation policies. First we describe the experimental environment which is composed of the INRFlow simulator and our data-intensive application models. We conclude the section describing the set of experiments performed.

The evaluation has been carried out using INRFlow, our in-house developed simulator. INRFlow models the behaviour of parallel systems, including the topology (link arrangement), the applications and workload generation and the scheduling policies (selection, allocation and mapping) and measures several static (application-independent) and dynamic (with applications) properties.

Given the wide variety of applications that we need to consider (HPC from several scientific domains, big data analytics, etc.) and their different needs in

terms of communication and storage, we have constructed a generic application model based on Markov chains which can be fine-tuned to model different application types by changing transition probabilities. Figure 3 shows the model we constructed based on an analysis of ExaNeSt's applications.

The model is composed of 6 states each of them representing the different types of operations that can go on during the execution of an application in the ExaNeSt platform. Note that storage traffic has been split into two different states in order to be able to model applications with varying IO needs (e.g. read- or write-intensive, or more balanced access to storage). In particular for this work we use read-write balanced I/O-intensive applications (75% storage *versus* 25% of computation and communications, 12.5% each), leaving other types of applications, in particular actual applications, as future work.

We evaluate two different types of scenarios. First we measure the runtime of a single application when multiple access modes are used. In particular, we measure the impact of accessing cached and non-cached data, of having a varying number of remote NVMs and of hitting in RAM with different frequency. In this scenario the applications run in isolation without any interference. The effect of interferences is evaluated in the second set of experiments in which we run several applications concurrently using two simple allocation strategies.

All the experiments have been carried out using two different INs. The first set uses a 4:3-fat-tree and a (4×4×4) torus both with 64 nodes each. The second uses a 8:3-fat-tree and a (8×8×8) torus with 512 nodes. In this case we use a larger network to execute four 128-node applications concurrently.

We consider three storage strategies: **CACHE** is the optimal case in which all the data is available in the local device, **SAN** where all I/O operations are done against the SAN and, finally, **STG-**k in which k NVMs have been allocated for the application and the required data are spread among them. If k is equal to the number of nodes it represents Local allocation, otherwise it represents Internal allocation (as discussed above). External allocation is not considered in this paper for the sake of brevity.

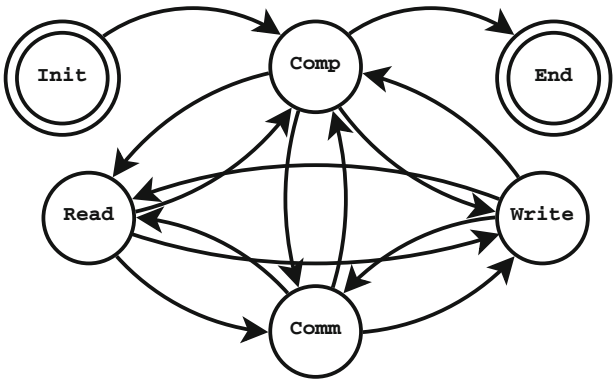


Fig. 3. Representation of the Markov chain used to generate synthetic applications. For the sake of clarity transition probabilities between states are omitted.

6 Analysis of the Results

In this section we analyse the results in terms of runtime (time required to process all the events in the trace). For the sake of brevity we only present results obtained with the fat-tree using consecutive placement, but all other results (tori and random allocation) are consistent with the ones discussed here.

6.1 Single Application Scenario

Let us start analysing the impact of accessing the NVM device where the required data is not mapped in main memory. In Fig. 4 we have represented the runtime with varying percentages of cached data access; 0 indicates that 0% of the operations are in memory, i.e., we have to always access the storage subsystem, to 100% in which all the data is accessed using main memory.

Results clearly show that when misses occur, that is, when the data must be loaded from disk, the performance is degraded. This effect is more evident in remote nodes due to the use of the unified network but it also occurs when the storage device is local. However in that case the effect is less evident due to the low latency and high bandwidth of the devices. From the results we can also notice the effects on the performance of remote accesses comparing the STG-64 and CACHE strategies. Although both strategies use 64 NVMs devices, the use of the interconnect to access 50% of the data has severely degraded the performance of the application increasing the runtime, in average, one order of magnitude.

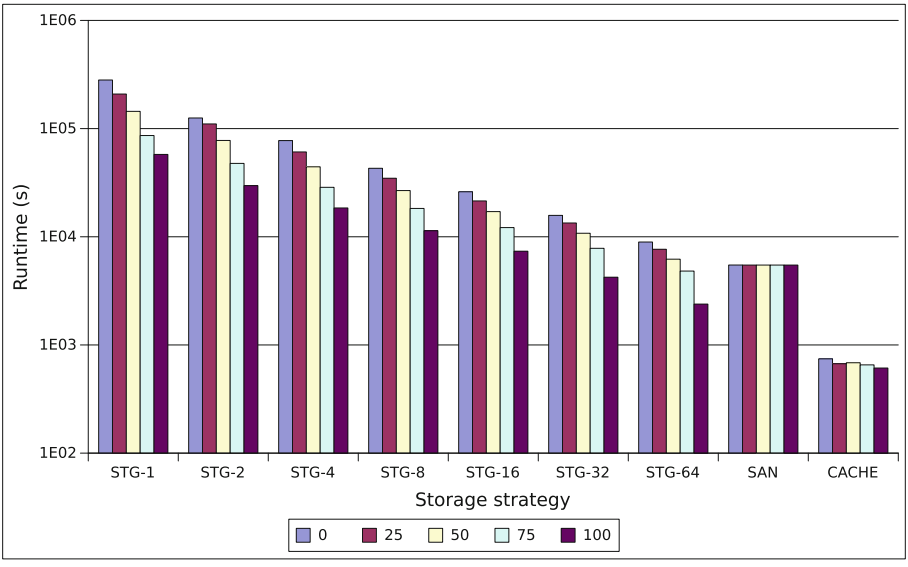


Fig. 4. Runtime of one applications running in a 64 nodes network for 50% of accesses to remote nodes and several ratios of accesses to main memory (0, 25, 50, 75 and 100%).

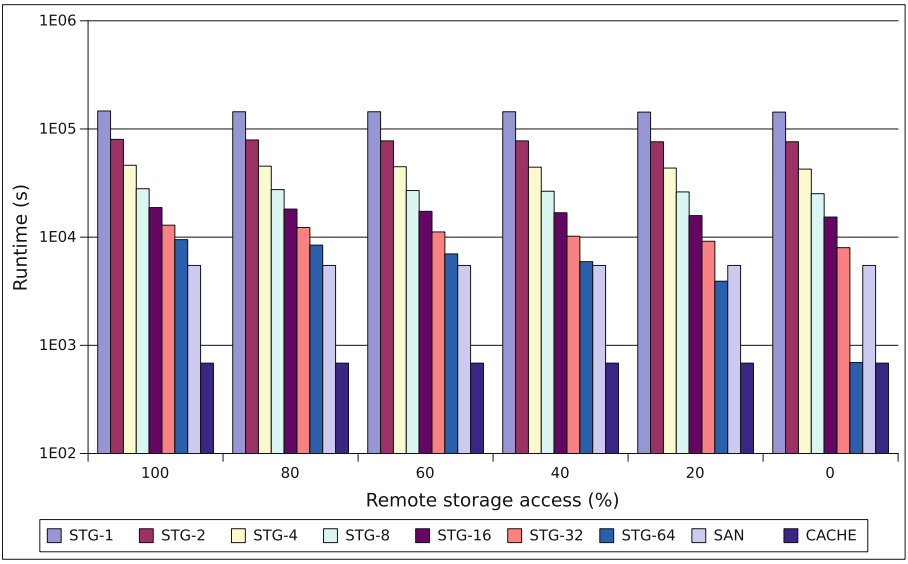


Fig. 5. Runtime of one applications running in a 64 nodes network for 50% of accesses to main memory and several ratios of accesses to remote nodes using 9 storage strategies (STG-{1, 2, 4, 8, 16, 32, 64}, SAN and CACHE).

Now let us analyse the impact of accessing remote storage devices. Figure 5 shows results for a configuration using 50% of accesses to main memory (cached data) and varying the amount of accesses to remote nodes from 0% to 100%. Results are very clear, showing that accessing remote storage devices does not comes without tremendous overheads. The worst case happens when just one NVM is used and all the tasks access it to retrieve the data, with the subsequent contention in the IN and the NVM. Increasing the number of storage devices makes the traffic spread through the IN, therefore reducing contention. In this scenario and in the one shown previously, the CACHE strategy is the best performer showing that locality for the data (both in memory and in the network) is required to take advantage of the distributed storage. Regarding the SAN access strategy, it was expected to perform well because it relies on a completely independent and high performance network and features immense bandwidth to the permanent storage. However, even in this case, STG-64 can outperform it when accessing mostly cached-data.

At any rate, having a single application running in a large parallel system is uncommon. For this reason, in the next section we will explore the effects of multiple applications accessing the storage subsystem concurrently.

6.2 Multi-application Scenario

Figure 6 shows the results for the multi-application scenario using contiguous allocation. Due to space constraints we omit the results for the random

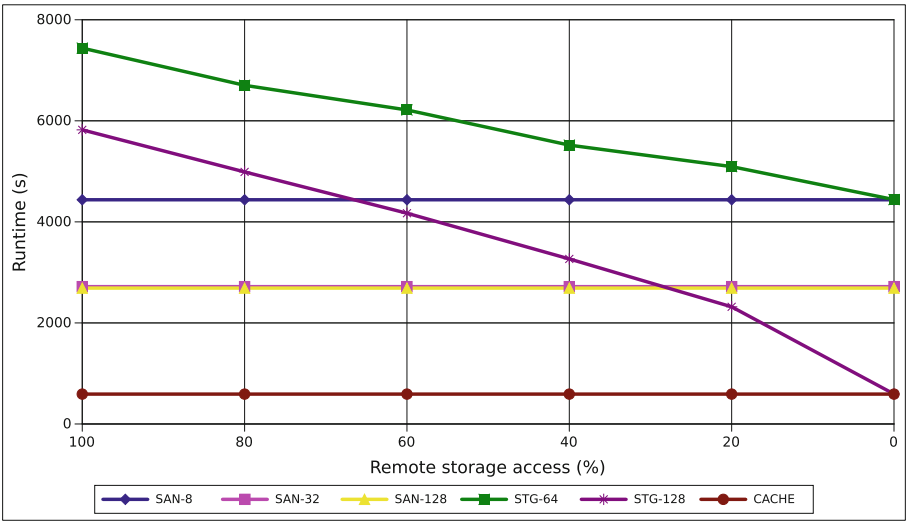


Fig. 6. Runtime of four applications running concurrently in a 512-node fat-tree using 75% of cached data and a varying percent of accesses to remote storage.

allocation, but the conclusions are akin to these presented here. As we can see, when several applications compete for the network, reducing the number of remote accesses, below 20–40% remarkably improves the performance of the applications. Regarding the number of I/O servers for the SAN strategy, we have evaluated several configurations (we only show here the use of 8, 32 and 128 I/O servers). Looking at the results it is clear that the SAN approaches perform well but at the cost of increasing the number of I/O servers. Notice that here we match the number of computing nodes with the size of the applications; this is possible for small networks as the one shown here, but clearly unaffordable for larger networks. In that case the SAN will become a bottleneck for applications.

If we focus on the STG strategies, we can see that if the number of accesses to remote NVMs is reduced below 25% the runtimes are shortened considerably, outperforming even the biggest SAN when the STG-128 strategy is used. The best performer is again the CACHE strategy that minimizes the use of the network for storage traffic. Notice that these results consider 75% of cached-data.

In summary, from all the results, we can conclude that when keeping all the data local is not possible, reducing the number of accesses to remote storage device is critical to maintain the performance. In any case, if good locality is achieved, the ExaNeSt storage subsystem can outperform classic storage systems based on SANs. We want to remark SAN-based systems require the number of I/O servers to scale with the number of compute nodes in order to keep up with the performance levels, which will be unaffordable for Exascale-capable computing systems. Alternatively, the performance of the I/O infrastructure will be degraded as systems grow. This evaluation remains as future work.

7 Conclusions and Future Work

In this work we have presented the storage architecture of ExaNeSt composed of fast NVM devices attached to the computing nodes. These devices provide to the applications low latency and high bandwidth for accessing the data. However, as this system will use a unified interconnect for all types of traffic, we wanted to measure to what extent the performance of the applications could be degraded and if the addition of specific data-aware allocation policies to the scheduling system could help alleviating this effect.

First we have seen how much accessing storage devices instead of *hot* data mapped into main memory affects the performance. Then, we looked at the effects of accessing remote storage devices. Finally, we assessed the effects of inter-application interferences. Our results show the potential benefits that exploiting locality when mapping data would bring when employing data-locality aware allocation functions in fully-distributed storage systems.

This has been just a preliminary study to assess whether specific storage allocation policies can benefit the execution of the applications in ExaNeSt and other systems using unified interconnects. In future works we will evaluate much larger networks executing a mix of applications such as communication- and computation-oriented applications. We also plan to develop specific allocators to optimise the assignment of resources that take into account both the storage and application traffic in order to improve application performance.

Acknowledgement. This work was funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 671553.

References

1. ARM. https://www.arm.com
2. Balzuweit, E., et al.: Local search to improve coordinate-based task mapping. Parallel Comput. **51**, 67–78 (2016)
3. BeeGFS. https://www.beegfs.com
4. Bezerra, A., et al.: Job scheduling for optimizing data locality in Hadoop clusters. In: 20th European MPI Users' Group Meeting, EuroMPI 2013, pp. 271–276. ACM, New York, NY, USA (2013)
5. Bhatele, A., et al.: Mapping applications with collectives over sub-communicators on torus networks. In: International Conference on High Performance Computing Networking, Storage and Analysis, SC 2012, Salt Lake City, UT, p. 97 (2012)
6. Bhatele, A., et al.: There goes the neighborhood: performance degradation due to nearby jobs. In: International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2013, Denver, CO, USA, pp. 41:1–41:12 (2013)
7. Caíno-Lores, S., Carretero, J.: A survey on data-centric and data-aware techniques for large scale infrastructures. Int. J. Comput. Electr. Autom. Control Inf. Eng. **10**(3), 517–523 (2016). http://waset.org/Publications?p=111
8. Chen, T.Y., et al.: LaSA: A locality-aware scheduling algorithm for Hadoop-MapReduce resource assignment. In: 2013 International Conference on Collaboration Technologies and Systems (CTS), pp. 342–346, May 2013

9. Feitelson, D.G., Rudolph, L., Schwiegelshohn, U.: Parallel job scheduling – a status report. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2004. LNCS, vol. 3277, pp. 1–16. Springer, Heidelberg (2005). https://doi.org/10.1007/11407522_1
10. Hammoud, M., Sakr, M.F.: Locality-aware reduce task scheduling for MapReduce. In: International Conference on Cloud Computing Technology and Science, CLOUDCOM 2011, pp. 570–576, Washington, DC, USA (2011)
11. Jin, J., et al.: Bar: An efficient data locality driven task scheduling algorithm for cloud computing. In: CCGRID, pp. 295–304. IEEE Computer Society (2011). http://dblp.uni-trier.de/db/conf/ccgrid/ccgrid2011.html#JinLSDX11
12. Johnson, C.R., Bunde, D.P., Leung, V.J.: A tie-breaking strategy for processor allocation in meshes. In: 39th International Conference on Parallel Processing, ICPP Workshops, San Diego, California, USA, pp. 331–338 (2010)
13. Katevenis, M., et al.: The ExaNeST project: interconnects, storage, and packaging for exascale systems. In: Euromicro Conferene on Digital System Design (DSD) (2016)
14. Kosar, T., Balman, M.: A new paradigm: data-aware scheduling in grid computing. Future Gener. Comput. Syst. **25**(4), 406–413 (2009)
15. Mellanox. https://www.mellanox.com/related-docs/whitepapers/WP_Deploying_Ceph_over_High_Performance_Networks.pdf
16. Pascual, J.A., Miguel-Alonso, J., Lozano, J.A.: Strategies to map parallel applications onto meshes. In: de Leon F. de Carvalho, A.P., Rodríguez-González, S., De Paz Santana, J.F., Rodríguez, J.M.C. (eds.) Distributed Computing and Artificial Intelligence. Advances in Intelligent and Soft Computing, vol 79, pp. 197–204. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14883-5_26
17. Pascual, J.A., Miguel-Alonso, J., Lozano, J.A.: Optimization-based mapping framework for parallel applications. J. Parallel Distrib. Comput. **71**(10), 1377–1387 (2011)
18. Pascual, J.A., Miguel-Alonso, J., Lozano, J.A.: Locality-aware policies to improve job scheduling on 3D tori. J. Supercomput. **71**(3), 966–994 (2015)
19. Pascual, J.A., Navaridas, J., Miguel-Alonso, J.: Effects of topology-aware allocation policies on scheduling performance. In: Frachtenberg, E., Schwiegelshohn, U. (eds.) JSSPP 2009. LNCS, vol. 5798, pp. 138–156. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04633-9_8
20. Power, R., Li, J.: Piccolo: building fast, distributed programs with partitioned tables. In: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI 2010, pp. 293–306 (2010)
21. Santos-Neto, E., Cirne, W., Brasileiro, F., Lima, A.: Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2004. LNCS, vol. 3277, pp. 210–232. Springer, Heidelberg (2005). https://doi.org/10.1007/11407522_12
22. Topcuouglu, H., Hariri, S., Wu, M.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002)
23. Xu, Q., et al.: Performance analysis of NVMe SSDs and their implication on real world databases. In: Proceedings of the 8th ACM International Systems and Storage Conference, SYSTOR 2015, pp. 6:1–6:11. ACM, New York, NY, USA (2015)
24. Zhang, X., et al.: An effective data locality aware task scheduling method for MapReduce framework in heterogeneous environments. In: International Conference on Cloud and Service Computing, CSC 2011, pp. 235–242. Washington, DC, USA (2011)