Received <day> <Month>, <year>; | Revised <day> <Month>, <year>; | Accepted <day> <Month>, <year>

DOI: xxx/xxxx

SPECIAL ISSUE PAPER

On the Effects of Allocation Strategies for Exascale Computing Systems with Distributed Storage and Unified Interconnects

Jose A. Pascual* | Joshua Lant | Caroline Concatto | Andrew Attwood | Javier Navaridas | Mikel Luján | John Goodacre

[1]Computer Science School, The University of Manchester, Manchester, United Kingdom

Correspondence
*Jose A. Pascual Email: jose.pascual@@manchester.ac.uk

Present Address
Present address

Summary

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 investigate alternatives for the storage subsystem of a novel exascale-capable system with special emphasis on how allocation strategies would affect the overall performance. We consider several aspects of data-aware allocation such as the effect of spatial and temporal locality, the affinity of data to storage sources and network-level traffic prioritization for different types of flows. In our experimental set-up, temporal locality can have a substantial effect on application runtime (up to a 10% reduction) whereas spatial locality can be even more significant (up to one order of magnitude faster with perfect locality). The use of structured access patterns to the data and the allocation of bandwidth at the network level can also have a significant impact (up to 20% and 17% reduction of runtime respectively). These results suggest that scheduling policies exposing data-locality information can be essential for the appropriate utilization of future large-scale systems. Finally, we found that the distributed storage system we are implementing can outperform traditional SAN architectures, even with a much smaller (in terms of I/O servers) back-end.

KEYWORDS:
near-data computing, scheduling, resource allocation, storage traffic, inter-processor communications

1 | INTRODUCTION

Traditionally 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 high performance computing (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. One of these systems is our novel, custom-made architecture, ExaNeSt (1). 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 (2). To support such kind of data-intensive applications we are leveraging a unified, low-latency Interconnection Network (hereafter IN) and a fully distributed

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

storage subsystem, BeeGFS, with the data spread across the nodes in local non-volatile storage devices (3) (NVM). This greatly contrasts with traditional supercomputers and datacenters that rely on Storage Area Networks (SAN) to access the data with separate networks for I/O, system management and inter-processor communications (IPC).

A fully distributed file system allows for near-data computation reducing the great overheads of moving data from centralized storage to the compute nodes (4). A single, consolidated IN offers enormous power-savings when compared with multi-network designs. While a unified IN does, indeed, allow us to cope with power and cost design constraints, it also exacerbates the challenges arising from workload convergence as storage traffic will be distributed all across the system which can interfere negatively with inter-process traffic of large, parallel applications.

In ExaNeSt, we are exploring the design space for the storage (and related) subsystems in order to search for a holistic, integrated solution in which the synergy between hardware and software mechanisms is exploited with the objective of reducing the interference between application and storage traffic in our unified IN. More precisely, we look into different options for the storage backend, the effectiveness of traffic-prioritization in the IN, the affinity of application tasks to the data sources and the impact of different data-aware allocation policies for scheduling in a fully distributed storage architecture such as ours. The first step towards this objective is, obviously, understanding to what extent the mix of application- and storage-traffic interferes with each other and how this affects performance.

We acquire this knowledge by conducting an extensive set of experiments in which many aspects of resource allocation are explored (spatial and temporal locality of data, flow-level allocation of network bandwidth). First we look at a data-intensive application occupying a whole system to see only the effects of intra-application interference. Then we look into a multi-application scenario in which a batch system executes a series of applications. For completeness, we compare the allocation strategies suitable for our system with a baseline High-Performance Computing (HPC) SAN-based system, where storage traffic is handled separately from IPC traffic. Our evaluation relies on a novel, flexible generic application model that generates realistic workloads mimicking different types of application, i.e. I/O-, computation- or communication-intensive.

We found that job scheduling, in particular the allocation phase where resources are assigned to applications, can have a huge impact on performance and motivate the inclusion of data and storage information to be considered during the scheduling process.

An extensive simulation study has been carried out using our IN simulator, INRFlow. Results using one application 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. Moreover, we have also evaluated a multi-application scenario, emulating a data center, in which several application are submitted to an execution queue where they are time- and space-multiplexed into the system by means of a system-level scheduler. Results again show the potential of using an unified IN plus a distributed file system instead of two independent networks. A set of preliminary results can be found in (5).

The above allocation strategies are implemented at the software-level. However, as we are also developing a router within ExaNeSt (6), we also want to evaluate to what extent the allocation of network resources to flows impact the performance of the applications when mixed traffic is present. For this reason we also evaluate a number of policies to assign priorities to the traffic which translate into bandwidth reservation. Results show that in the presence of enough storage traffic the use of such policies has a positive impact on the applications runtime and also that they do not affect negatively the performance in the absence of storage traffic .

The rest of the paper is organized as follows. In Section 2 we discuss some previous works on data-aware allocation for large-scale computing system. Following in Section 3 we provide an overview of the architecture of ExaNeSt, specifically the storage and interconnection subsystems. We continue in Section 4 explaining the scheduling process and the simple allocation strategies considered in this paper. Then in Section 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 Section 6. We close the paper with Section 7 which highlights some concluding remarks and sets some future lines of research arising from the findings of this work.

2 | BACKGROUND

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 IN 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. For this reason we focus here on some related aspects of the scheduling process and on previous work related to traffic prioritization. Finally, we will review some existing distributed storage systems and discuss their scalability and suitability for our architecture.

2.1 | Job Scheduling

Given that one of the main aspects of this paper is introducing data-aware strategies into the scheduler, for the sake of completeness, we start the discussion by providing a high-level explanation of application scheduling in the context of large-scale computing systems. The scheduling process in such systems typically involves three different stages after jobs are submitted by the users:

- *Job scheduling*: A scheduling policy selects the next job to be sent to execution based on the state of the system and the queues as well as some other scheduling parameters, such as priorities or quotas.
- *Allocation*: Once the next job is selected, the allocator finds 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.
- *Mapping*: Finally, once the job and the resources are selected, we need to decide how to map tasks into these resources.

Traditionally, the de facto scheduling policy was First Come First Serve (FCFS) (7) in which jobs are executed in the same order as they are submitted. However, this was soon found to be inappropriate for many cases and there was a huge body of research on improving the scheduling policies. The main drawback of this policy is that it severely reduces system utilization. When the job at the head of the queue cannot be put to run because the required resources are not available, all the jobs in the queue must wait due to the sequentially ordered execution of jobs increasing the waiting times. As a result, many processors could remain idle, even when other waiting jobs could be eligible to use them. With the goal of increasing the system utilization, several alternatives have been proposed (7) such as Shortest Job First (SJF) which selects first the fastest jobs (expected), and backfilling (BF) which is a variant of FCFS, based on the idea of advancing jobs through the queue. BF is the most widely used due to its easy implementation and proven benefits.

With the advent of more and larger supercomputers, and the explosion of scientific applications that made use of them, the system administrators realised that the well-behaved, structured communication patterns of these applications relied on spatial locality among the tasks of an application. For this very reason, the scheduling community changed the focus towards communication-aware strategies and, indeed, there exists a huge body of knowledge 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 policies, typically disregarded data locality as the proportion of storage traffic is negligible and, in most cases, dealt with by a separate network, as explained above. Many authors (8, 9, 10) analyse the extent that inter-application interference has on their performance. In order to minimise this interference, many non-contiguous (11, 12) and contiguous (8, 12) allocation strategies have been proposed for a range of topologies. Similarly, other works (9, 13, 14) have tried to reduce intra-application contention using different techniques to map the tasks of the application onto the previously selected nodes.

More recently, with the exponential growth of Data-oriented computation coming from both scientific and business domains, the focus has moved towards allocating applications close to the data either in memory, e.g., Spark (15, 16), or in storage, e.g., Hadoop (17, 18, 19, 20). 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 (21). Other works try to assign the application to the node where the data is mapped or at least, keep it as close as possible (22). More advanced solutions try to reduce gradually the job completion time by tuning the initial task allocation adjusting data locality dynamically based on the status of the system and the network (23) or to reduce the system power consumption guiding the scheduling decisions (24). A detailed overview of data-aware scheduling can be found in (25). In any case, there is no work that we know of in which both data- and communication-aware scheduling policies are leveraged. This paper motivates the need for merging these two approaches so to obtain the benefits of minimizing both inter-process and storage interferences.

2.2 | Traffic Prioritization

All the above resource allocation strategies focus on a coarse level of granularity in which the minimal unit is an application (or the tasks therein). However, there also exist many research works that study resource allocation at the network-level with a much finer level of granularity; either flow- or even, packet-level. The idea here is to be able to either prioritize certain types of traffic or to provide Quality of Service (QoS) for specific critical communications, e.g., ensure a certain performance level, either in terms of bandwidth for data-intensive applications or of latency for real-time applications. Typically these mechanisms are implemented on hardware, but to our knowledge mostly all of them are specifically designed for Network on Chips (NoC). While NoCs and INs share many similarities, there are also some critical differences both in terms of workloads and architectures. NoC communications tend to be very short, related to the coherence mechanisms, and latency-critical (somewhat akin to the inter-process traffic considered here). However, these are mixed with bursty DMA transfers which are typically much longer and whose behaviour more or less resembles our storage traffic (although a few orders of magnitude smaller). In this context, avoiding that DMA transfers saturate the network and slow down coherence traffic is essential to keep the processing speed at appropriate levels. In terms of architecture, the main difference is that the latency in NoCs is dominated by router crossing time, where as in larger networks, traversing the link is much slower than performing routing decisions, which means more time could be spent to make allocation decisions.

In (26) the authors explore the necessity of such QoS mechanisms for future NoCs. They developed three mechanisms based on circuit switching, virtual channels and virtual channels combined with aging scheduling showing that the implemented techniques improve the ability of the network in meeting QoS requirements. A similar work (27) studied the assignment of higher priorities to critical messages. They propose a flow control mechanism for input buffers with increasing priority for blocked messages and dropping of old packets. The combination of these strategies with a core placement based on message bandwidth requirements and on message priorities can reduce the number of missed deadlines. More recently in (28) the authors proposed a mechanism to improve the QoS in which the packets include the maximum delay that they can suffer without any adverse effects. This enables routers to service late packets by trading the expendable time associated with the high priority packets hence improving overall quality of service. More focused on the IN is the work developed in (29) where they describe an efficient switch architecture with priorities support suitable for any interconnect technology implementing deterministic source-based routing. However, none of them has studied the impact of assigning priorities when applications and storage traffic share an unified IN and, the hardware requirements to implement such mechanisms.

2.3 | Distributed Storage Systems

The massive amounts of data used in Data-oriented computation require of efficient and safe ways of accessing them. This has been traditionally performed using parallel distributed file systems which allow for *parallel* access to data stored in *distributed* resources using a single, coherent *file system* view. There are many of such file systems and the decision should not be taken lightly as different implementations compromise upon different tradeoffs, which will have a substantial effect on many aspects of system performance (30, 31). For instance, in the context of high performance computing Lustre (32) and IBM's General Parallel File System (GPFS) (33) are the most common mainly thanks to their scalability and high performance characteristics. Other file systems are more focused on using commodity hardware, for instance in the context of datacenters without RAID disks or a Storage Area Network (SAN). The most prominent examples of these are Hadoop Distributed File System (HDFS) (34), Google File System (GFS) (35), Global File System 2 (GFS2) (36) and Ceph (37) which focus on fault tolerance, high throughput and scalability. Within ExaNeSt we are using and enhancing BeeGFS (38), a modern file system developed and optimized for high-performance computing and with a strong focus on I/O intensive workloads.

3 | THE EXANEST ARCHITECTURE

In this section we describe the most relevant aspects of ExaNeSt's architecture and how they affect and motivate the research carried out in this paper. One of the main novelties of our design is the affordance of NVMs within compute nodes so to reduce the latency and energy of data-access and which we plan to leverage for exploiting data-locality. Compute nodes will access the storage subsystem transparently using BeeGFS (38), a high performance parallel filesystem that is in charge of reading and writing data between the local NVMs and the central storage system. In ExaNeSt, BeeGFS access the central storage when applications start or finish execution, transferring data to the NVMs, so that applications always access data from there. This allocation will consider which nodes access the data and will try to map the data to those nodes, so to increase data locality. In case this perfect allocation is not possible, it will try to reduce the distance between the node and the NVM containing the data in order to reduce the utilization of network resources. Regarding the communication infrastructure, we are building our

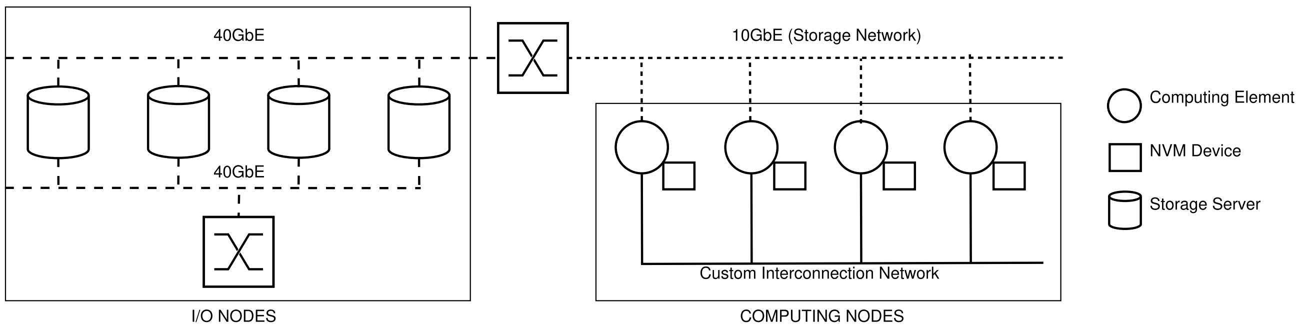


FIGURE 1 Visual representation of the ExaNeSt storage architecture. The local NVMs are attached to the computing nodes sharing the main IN (solid). An Ethernet network is provided for central data storage (dashed).

own multi-tier, custom-made FPGA-based interconnect within ExaNeSt(1) and, indeed, in Manchester we are in charge of the Top-of-Rack (ToR) routers (6). Given the high flexibility provided by FPGA devices we are exploring the adequacy of a range of mechanisms to be incorporated into our design, out of which traffic prioritization is one of the most promising and so, studied here.

In Figure 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 attached directly to them (squares). Compute nodes access remote NVMs through our custom-made IN which is also used for interprocess-communication (solid lines). NVMs access the central storage backend, in the left, using an independent storage network (dashed lines).

For simplicity, in this work we have modelled the central storage network as a 10Gb Ethernet network which is connected the compute nodes using a front-end 10Gb switch (this network is only to be used for I/O operations when applications start/end the execution). As we try to model a realistic high performance system we also model a 40Gb back-end network which is in charge of data replication within the central storage. The operation of this storage system is the typical in big datacenters (39): 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 I/O nodes (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 central storage is still an open question in ExaNeSt; and more power-efficient solutions are likely to be implemented in the final prototype as much lower utilization is expected than in a SAN-centric solution.

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 10GB/s) and **non-cached** if it is in an NVM with an average bandwidth of 2GB/s for reads and 1GB/s for writes. 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**. There are, therefore, 5 possibilities when applications access data (as represented in Figure 2):

- **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 as depicted in Figure 2 (green line from LOCAL to MEM).
- **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 (yellow line in Figure 2 from LOCAL to 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 the distance to the remote node, or interferences with other flows being sent through the network (blue line in Figure 2 from LOCAL to remote MEM).

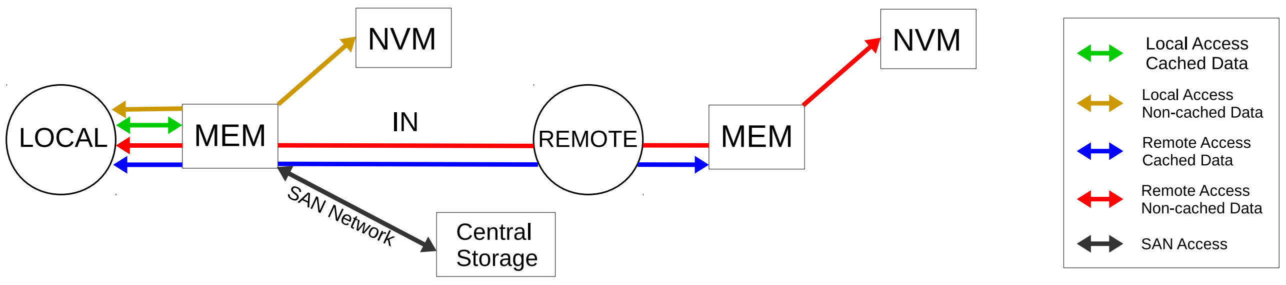
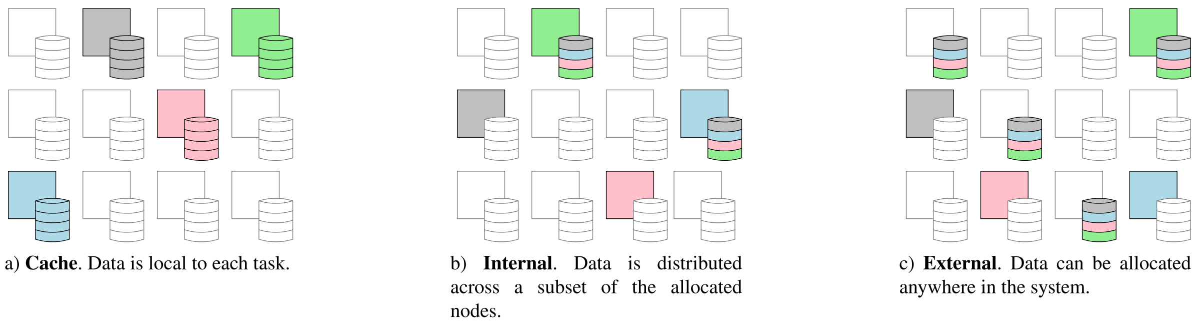


FIGURE 2 Representation of the data access modes used in this work. Colors represent the cost of performing read and write accesses to local and remote cached and non-cached data.



a) **Cache**. Data is local to each task.

b) **Internal**. Data is distributed across a subset of the allocated nodes.

c) **External**. Data can be allocated anywhere in the system.

FIGURE 3 Examples of application allocation. Squares represent processing nodes and cylinders represent NVMs storage. Each color represents the resources allocated to each of the tasks of an application.

- **Remote access, non-cached data**: This is the worst possible situation when fetching data that has been replicated within the compute nodes. 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 (red line in Figure 2 from LOCAL to remote NVM).
- **Central access**: This access mode represents an scenario where the applications do not use the NVMs so all accesses are done against the central storage; i.e., the SAN will be accessed whenever applications require to read or write data (black line in Figure 2 from MEM to Central Storage).

4 | SCHEDULING AND RESOURCE ALLOCATION STRATEGIES

In this Section we focus on the allocation of resources with different levels of granularity in order to analyse the impact on the performance of the applications. We start analysing storage and computing-nodes allocation strategies concluding with more fine-grained flow-level allocation strategies.

4.1 | Storage Assignment Scenarios

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 central storage into local storage. Ideally all the data will be in the same node as the accessing task, meaning that all accesses are Local, see above. 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 3 represents the three possible types of storage assignment based on the interference they create in the interconnect:

- **Cache**: Tasks will always have data ready in their local NVM. This is the ideal scenario where all the storage activity remains local and, hence, storage operations do not leave the node or create interference in the interconnection network (see Figure 3 a).
- **Internal**: Application data is stored in a subset of the allocated compute nodes (could be the full set) so to improve data locality. In this case, tasks may access data in any of the available NVMs, so remote accesses (within that subset) are necessary. This may create some intra-application interference for accessing the NMVs and, given that we do not guarantee contiguous allocation, some level of IPC interference is also likely. We can see an example of Internal assignment in Figure 3 b.
- **External**: This policy allows for some (or all) storage devices to be outside of the partition assigned to the application. Now, remote accesses to the data will generate higher levels of intra- and inter-application interference, see Figure 3 c.

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.

4.2 | Compute-node Allocation strategies

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 evaluate several typical HPC topologies: torus, fat-tree and dragonfly. Given that compute node allocation is not the main focus of this paper, we will focus on two simple, representative allocation strategies: *Sequential allocation*, in which the application tasks maintain certain level of locality, see (8, 12, 40), and *Random allocation*, that mimics the behaviour of a datacenter not using any locality-aware allocation. We do not use contiguous partitioning because of the extra overheads, mainly low utilization and very high waiting times, that these allocation policies cause to the scheduling process (8, 12). 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 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., (9). This selection will consider several application metrics with the goal of reducing the interference between inter- and intra-application storage and communication traffic.

4.3 | Flow Allocation

As explained, we are also looking into tackling the traffic interference problem at the hardware-level through flow-level allocation (i.e. traffic prioritization). For the purpose of this paper, we consider two priority levels (inter-process traffic and storage traffic) and evaluate the following policies (depicted in Figure 4).

- **No priorities (NP):** This is the baseline policy in which all the flows have the same priority. As we can see in Figure 4 a, in this case the available link bandwidth is shared fairly between all the flows.
- **Bandwidth Apportion (BA):** This policy establishes the proportion of the link bandwidth that will be used for each type of traffic. The number of flow types can vary as shown in Figure 4 b where we have assigned 50% of bandwidth to IPC and 50% for storage traffic. In this case half of the bandwidth will be shared by application flows and the remaining bandwidth by storage flows. When only one kind of flows are present, they occupy the full bandwidth.
- **Full Priority (FP):** In this strategy IPC has higher priority than storage traffic so that inter-process traffic is not affected by storage traffic. In practice, this means that storage traffic will only use the network resources that are not employed by the application. An example of this policy is depicted in Figure 4 c where all the bandwidth is used to transmit application flows. When these finish, storage flows are transmitted.

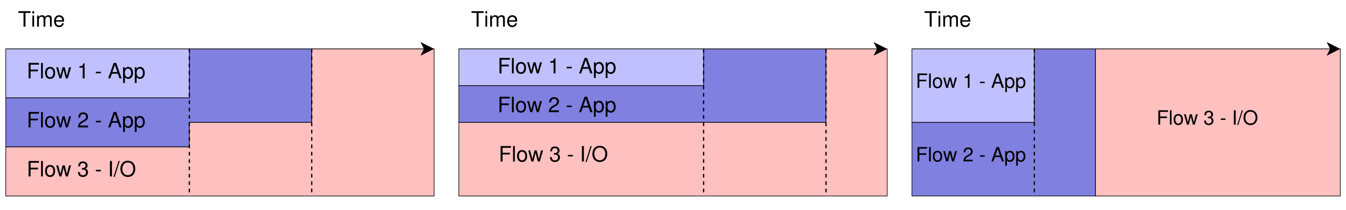


FIGURE 4 Examples of traffic prioritization policies. Time flows left-to-right and bandwidth is represented vertically.

We expect the impact of these policies to be greatly conditioned by the amount of storage traffic present in the network and not to affect the performance negatively in the its absence.

5 | EXPERIMENTAL SET-UP

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

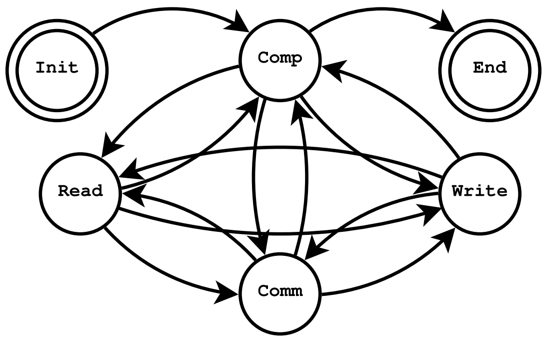
5.1 | Simulation Environment and Application Model

The evaluation has been carried out using INRFlow, our in-house developed simulator, which has been used to evaluate INs in many works (41, 42, 43). INRFlow models the behaviour of parallel systems, including the network 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. INRFlow is able to simulate a datacenter infrastructure, see again Fig. 1 , where multiple applications arriving to the system share the resources (i.e. computational, networking and storage). During execution, the links of the network have capacities and each flow is specified with a bandwidth reflecting the data that must be routed. In addition, INRFlow maintains the causal relationships between flows, so that some flows must finish before others begin. As a result, it provides realistics, flow-level simulation of general real-world workloads, as well as a good estimation of the completion times of a collection of application-inspired workloads.

Given that future Exascale systems will need to support a wide variety of applications (HPC from several scientific domains, big data analytics, etc..) with different needs in terms of computation, 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 5 a 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 I/O needs (e.g. read- or write-intensive, or more balanced access to storage). In particular, we use this model in two different ways. First, for the single-application evaluation, we model a read-write balanced I/O-intensive application (75% storage *versus* 25% of computation and communications, 12.5% each). Then, the multi-application evaluation exploits the flexibility of our model and generates a broad mixture of applications where different ratios of computation, communication and I/O access are employed. For this paper, these ratios are generated randomly at run-time, but are constant across all experiments.

The first four rows of Table 1 represent the parameters that we can configure to generate one application. Prob(C/C) and Prob(I/O) are the probability of performing Computation/Communication and Input/Output operations respectively. The parameter Tasks indicates the maximum number of tasks and I/O Tasks is the number of tasks doing I/O operations, that is, reading and writing data to the I/O devices (central storage or NVMs) and distributing the data to the rest of the computing tasks. The rest of the parameters in the Table correspond to the parameters of the execution model implemented in INRFlow.

We consider four storage allocation strategies: *Internal*, *External*, *Cache* (see Section 4.1) and *SAN* where all the I/O operations are done against the central storage. The number of storage devices is defined by **STG-***k* in which *k* NVMs have been allocated for the application and the required data are spread among them. Notice that when *Cache* is used as allocation strategy, *k* is equal to the size of the application. The next parameters **Mem access** indicates the percentage of cached data access; 0 indicates



a) Transition states.

	Single-application	Multi-application
Prob(C/C)	0.25	[0, 1]
Prob(I/O)	0.75	1-Prob(C/C)
Tasks	64/72	[1, Exp]
I/O tasks	64	[1, Tasks]
STG allocation	Internal,Cache,SAN	Internal,External,Cache,SAN
STG-k	1,2,4,8,16,32,64	[1, Tasks]
Mem access (%)	0,25,50,75,100	[0, 100]
Remote Stg (%)	0,10,20,...,80,90,100	[0, 100]
Data access type	D,A,L	D,A,L
SAN-k	4	1,2,4,8,16,32,64

b) Simulation parameters.

FIGURE 5 & TABLE 1 Representation of the Markov chain used to generate synthetic applications and parameters used to generate traffic in our experiments.

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. Similarly, **Remote Stg** indicates the percentage of accesses to remote nodes from 0% to 100%. Finally, **Data access type** represents how storage devices are accessed: Disperse *(D)* represents that all the computational nodes access any storage device, Affinity *(A)* represents that, for example, if there are two storage devices, half the computational nodes will access to one storage device and the remaining half will access the second device and Local *(L)* represents that all the data is local so no accesses to the network are required. Finally, **-**k represents the number of I/O servers used when SAN allocation is selected.

5.2 | Case Studies

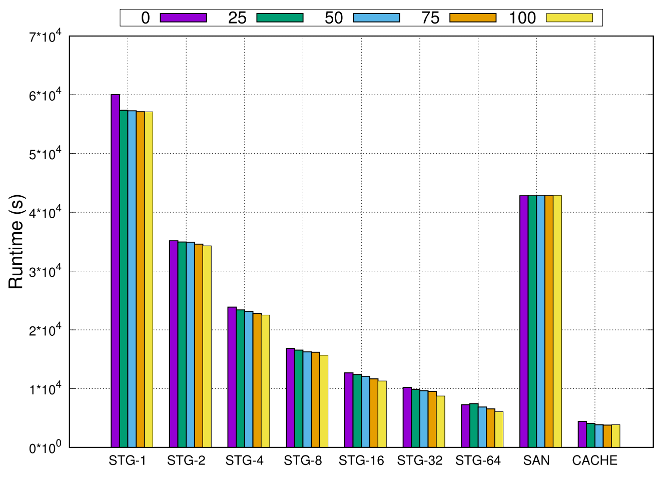
We evaluate two different types of scenarios. First we measure the runtime of a single application when multiple access modes are used. In particular, we measure the impact of accessing cached and non-cached data, of having a varying number of remote NVMs with structured and unstructured access patterns and of hitting in RAM with different frequency. In this scenario the applications run in isolation without any external interference. Specific values of the simulation parameters are given in the first column of the Table 1 .

The effect of interferences is evaluated in the second set of experiments in which we run several applications concurrently. This set of experiments simulates a datacenter in which applications arrive to be executed and depart as soon as they finish. In order to generate this scenario we have generated several workloads composed of 50 applications generated using the previously explained model. The applications within each workload are generated randomly considering the parameters shown in the multi-application column of Table 1 . In this scenario we report metrics related to the scheduling process such as the *total time* composed of the *waiting time* which is the time that the applications spent since they are submitted until they are put to execution and the *running time* which is the actual execution time of the application.

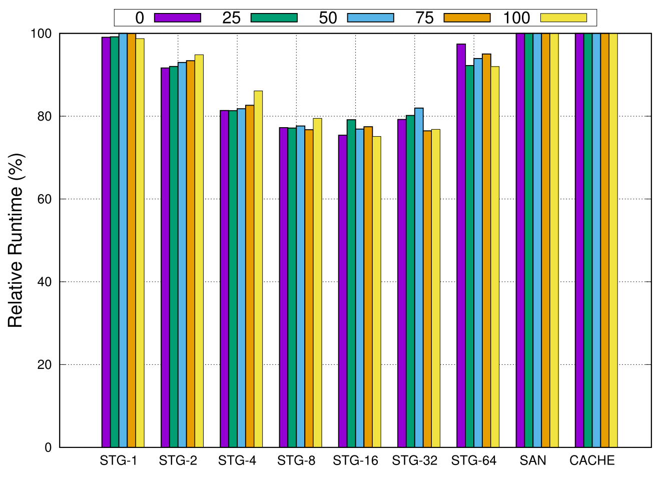
All the experiments have been carried out using three different INs. The first set, used in the single-application scenario, uses a (2,4,2)-dragonfly with 72 nodes, a 4:3-fat-tree and a $(4 \times 4 \times 4)$ torus both with 64 nodes. The second set, used in the multi-application scenario, uses a (4,6,4)-dragonfly with 600 nodes and a 8:3-fat-tree and a $(8 \times 8 \times 8)$ torus both with 512 nodes. In this case we use larger networks to execute multiple variable-size applications concurrently. We use Valiant routing (44) with the dragonfly topologies, static routing (45) with the fat-tree topologies and DOR routing (46) with the torus topologies. The purpose of having different topologies is not to compare them directly but to show that the trends we found here are similar regardless of the actual network architecture.

6 | ANALYSIS OF THE PERFORMANCE

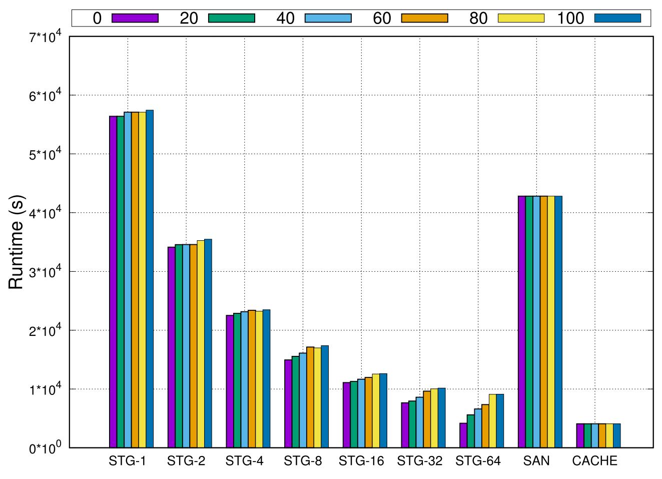
In this section we analyse the results of our experimental set-up and extract some insights from the obtained performance metrics.



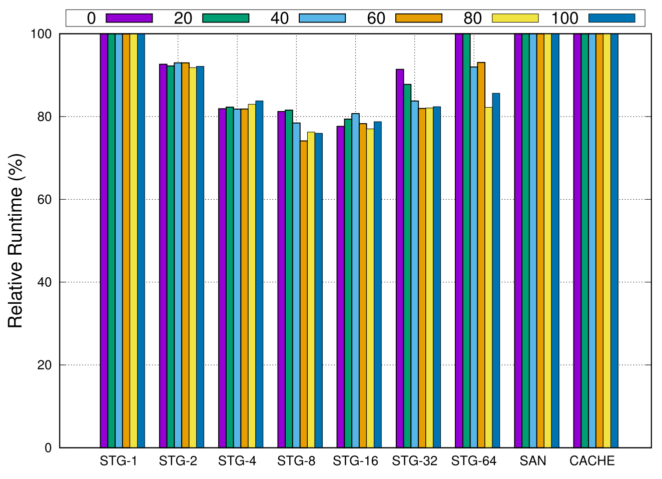
a) Dispersed access pattern



b) Affinity access pattern



c) Dispersed access pattern



d) Affinity access pattern

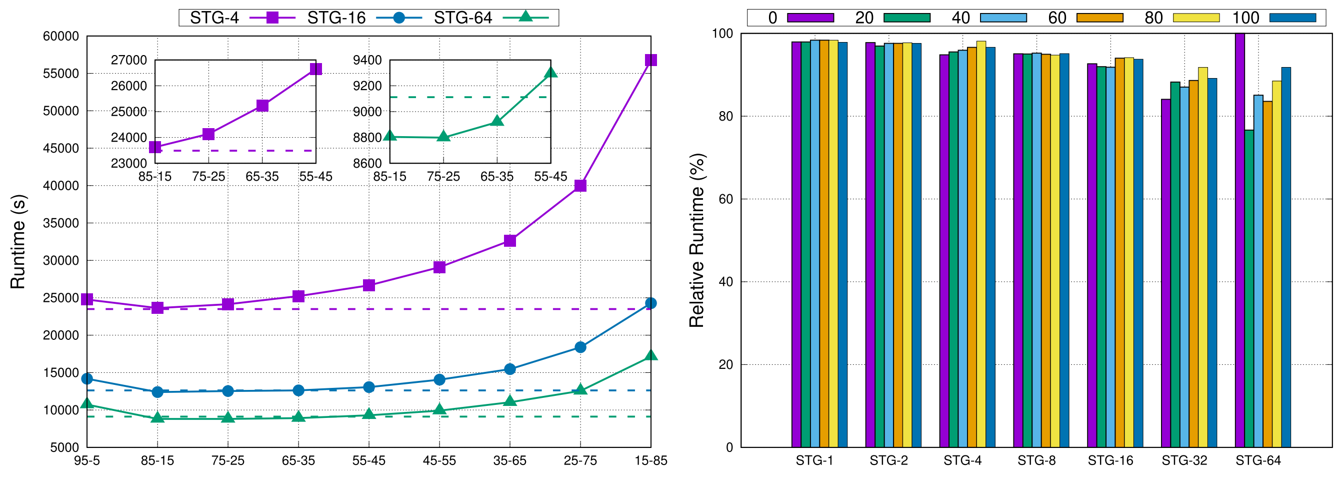
FIGURE 6 Runtime (and Relative Runtime) of one application running in a 72 nodes Dragonfly network using the Disperse (Left) and Affinity access pattern (Right) using a number of storage devices (STG-*k*). Top part shows the results for 50% of accesses to remote nodes and several ratios of accesses to main memory (0, 25, 50, 75, 100) while the bottom part shows the results for 50% of accesses to main memory and several ratios, from 0 to 100, of accesses to remote nodes.

6.1 | Single-application scenario

First, we show our single-application experiments in terms of runtime (time required to process all the events in the trace) and the effects of applying traffic prioritization policies. For the sake of brevity we only present results obtained with dragonfly, but all other results (fat-tree, 3D torus) are consistent with the ones shown here.

Let us start analysing the impact of accessing the NVM device when the required data is not mapped in main memory using both disperse and affinity access patterns. In Figure 6 a 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 always have to access the storage subsystem; in contrast 100% means all the data is available in main memory.

Results clearly show that when misses occur, that is, when 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 NVMs. From the results we can also notice the effects on the performance of remote accesses comparing the STG-64 and CACHE strategies. Although both



a) BA priority policy

b) FP priority policy

FIGURE 7 Effects of traffic prioritization on the runtime. For the sake of space we only show the results using the Disperse access pattern using 50% of accesses to memory. BA traffic priority policies for a range of assigned bandwidths. dotted lines represent the case without BA (left) and relative runtime with the FP policy (right).

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.

Now let us analyse the impact of accessing remote storage devices. Figure 6 c 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.

Let us analyse now the impact of using the affinity data access pattern. Results are depicted in Figures 6 b and 6 d as the relative runtime over using the disperse data access pattern. Results show that the use of more structured communications can remarkably improve the performance, up to a 20% for STG-4, STG-8, STG-16 and STG-32. This was expected because we impose to certain nodes to only access remotely to a number of NVMs. In this cases 4 sets of 18 tasks accessing each set to 1 NVM (STG-4), 8 sets of 9 tasks to 1 NVM (STG-8) and 16(8/8) sets of 5/4 tasks accessing to 1 NVM (STG-16). If we look at the results using STG-64, we can see that the performance is only improved where the remote accesses are higher than 20% because for those cases the communications will remain mostly local. Regarding STG-1 there are no improvements because as in the disperse access pattern all of nodes access just one NVM. SAN and CACHE are not affected because the storage does not use the interconnect and the communications are local to the nodes respectively.

Now we analyse the impact of prioritizing traffic on the performance. We start measuring the effect of the BA policy on the performance. The results are depicted in Figure 7 a where we assign different priorities, from 5% to 85%, to IPC flows. For the sake of space we only show the results for an application performing 50% of memory and remote accesses and compared them with the runtime achieved without priorities (dashed lines).

Results show that the assignment of higher BW to IPC traffic can slightly improve the performance when the amount of storage traffic is high (look at the zoomed results for STG-4 and STG-64 in Figure 7 a). These results only corroborate that reducing the latency of the IPC can improve the performance when large transfers of data are present in the network.

Let us take a look to the performance achieved using the FP traffic priority policy depicted in Figure 7 b. Results, shown as the relative runtime for 50% of accesses to main memory using a dispersed pattern, clearly shows that this policy is able to decrease the runtime more than 10% when enough storage traffic (STG-16, STG-32 and STG-64) is present in the network. Notice that STG-64 with 0% of remote accesses is not improved because all the traffic is local to the NVMs in the nodes.

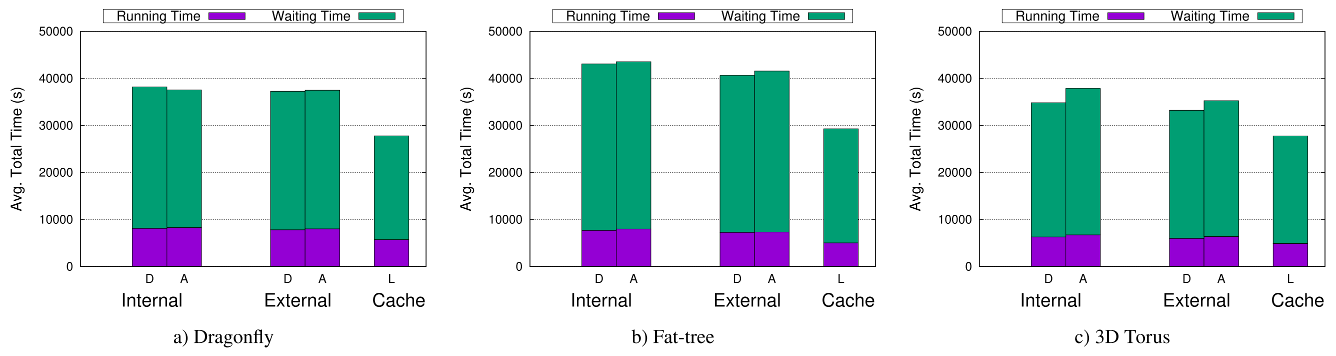


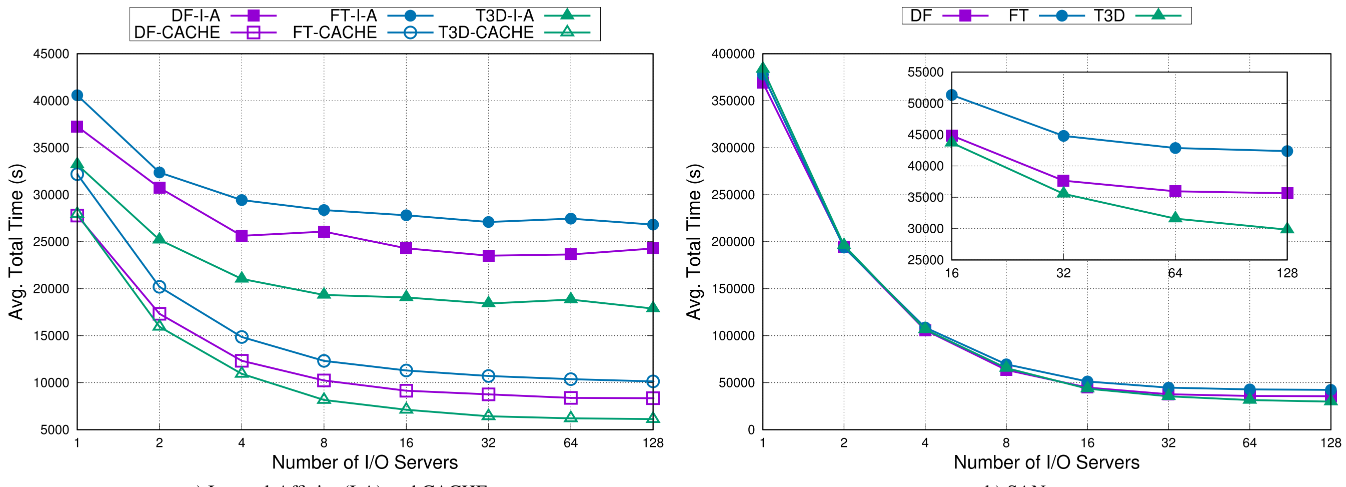
FIGURE 8 Average Total Time (Running Time + Waiting Time) of each application for three storage allocation policies. Each bar indicates the type of access to the data. Disperse (D), Affinity (A) and Local (L).

6.2 | Multi-application scenario

As having a single application running in a large parallel system is rather uncommon we will explore now the effects of multiple applications accessing the storage subsystem concurrently. Figure 8 shows the results for the multi-application scenario in three topologies. For the sake of readability we omit the results with SAN storage due to the order magnitude differences. Obviously the best storage allocation strategy is Cache in which all the accesses are local achieving the fastest runtime which, in turn, reduces the waiting times of the jobs in the queue. However, when multiple applications run concurrently, the penalty of using the Internal and External storage allocation strategies is not exceedingly higher (around 44% slower in the dragonfly) confirming that a unified network can cope with this kind of applications. These results are consistent across the three topologies. The reason is the great variability of application types in the workloads in which many of the applications are more focused on computation and communications without requiring many accesses to the storage subsystem. Furthermore, this is also the reason why the disperse and the affinity data access patterns do not have a great impact on the performance.

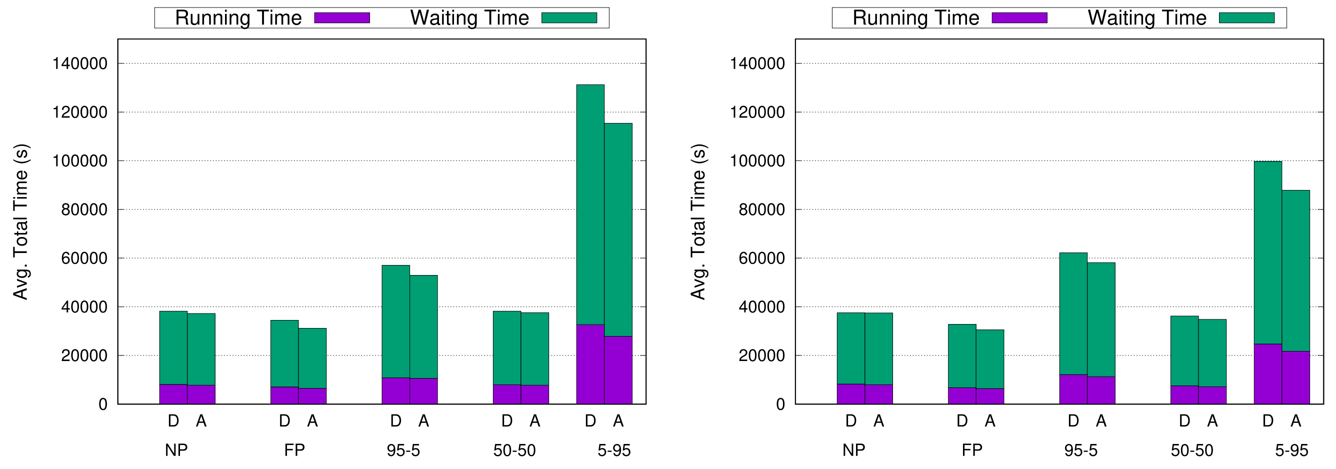
We compare now the effects of changing the number of I/O servers in the central storage back-end. For simplicity, and given than results are pretty similar, we restrict our discussion to the Internal storage strategy together with the Affinity access pattern (I-A) and the Cache strategy. We have depicted in Figure 9 the average total time achieved using from 1 to 128 I/O servers. Let us start looking at the effect of increasing the number of I/O servers for the I-A and the Cache strategy (see Figure 9 a) for the three topologies. The results show that the higher the number of I/O servers, the lower the average time time. However, at around 8 to 16 servers this improvement stabilises and adding more servers barely provides any benefit. If we look at the performance using the NAS strategy (see Figure 9 b, we can see that when just one I/O server is used, the performance is about one order of magnitude worse regardless of the topology. As we increase the number of I/O servers up to 16 the performance improves remarkably. The highly limited performance was expected as the independent SAN network becomes highly saturated and can not cope with all the I/O operations. Afterwards, adding more I/O servers keeps improving the performance, although at a much lower rate (see the zoomed results for 32, 64 and 128 in Figure 9 b). It is worth noticing that using a SAN is unable of outperforming Local even if Local uses a single I/O server in the back-end and SAN uses 128. This suggest that the near-data computation can be very useful if proper data-locality can be achieved. Even if data-locality is not maximized, e.g., A-I strategy, we are still greatly outperforming the SAN even if the later is using many more I/O servers in the backend. This is an important result because it shows we should be able to maintain or even improve the performance of traditional systems using much fewer I/O servers.

Finally, let us analyse the impact of using traffic prioritization policies for multiple applications in the dragonfly topology (see Figure 10). As in the single-application scenario the FP policy is the best performer (up to 17% faster than NP) showing that increasing the priority of IPC has a positive impact on the overall performance. If we look at the BA policy, surprisingly the lowest total times are achieved assigning a 50% of the bandwidth to the IPC. The rationale behind this behaviour is that in this scenario traffic from different applications is present in the network and giving more priority to a certain type of traffic can penalize other applications using other kind of traffic.



a) Internal-Affinity (I-A) and CACHE b) SAN

FIGURE 9 Average Total Time (Running Time + Waiting Time) of each application for three storage allocation strategies: Internal-Affinity and CACHE (Left) and SAN (Left). The results are shown for a vaying number of I/O servers (1,2,4,8,16,32,64,128).



a) Internal b) External

FIGURE 10 Effects on the average runtime of applications when applying the FP policy and the BA policy for three bandwidth assignments (95-5, 50-50 and 5-95) in the Dragonfly topology.

6.3 | Discussion

To conclude, we can highlight 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. More importantly, we found out that the ExaNeSt storage subsystem can outperform classic storage systems based on SANs. Indeed, 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. We have also seen the potential that traffic prioritization could have in systems where application and storage traffic coexist in the same interconnection network. We want also to remark that even small improvements of the average total time could have a great positive impact on other performance metrics such as job throughput and system utilization.

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 low latency and high bandwidth for applications to access 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 the performance is affected by accessing storage devices instead of *hot* data mapped into main memory. Then, we looked at the effects of accessing remote storage devices and the effects of inter-application interferences. Our results show the huge benefits that exploiting locality when mapping data would bring when employing data-locality aware allocation functions in fully-distributed storage systems. When compared with traditional storage approaches, our model requires fewer resources to achieve the same or even better performance. Finally, we have assessed the impact that traffic prioritization policies can have on the performance. Results have shown the potential of this approach showing high reduction (up to 17%) in terms of waiting and running times.

This paper has assessed whether specific storage allocation policies can speed up the execution of the applications in ExaNeSt and other systems using unified interconnects. In future works we will evaluate in more detail the impact of policies to assign priorities to the traffic and will look into architectural implementations to accompany our ToR router. 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. Finally, as the ExaNeSt project targets Exascale performance, we will evaluate all the policies and strategies using larger applications and networks.

References

[1] Katevenis M., al . The ExaNeSt Project: Interconnects, Storage, and Packaging for Exascale Systems. *2016 Euromicro Conf. on Digital System Design.* 2016;00:60-67.

[2] ARM . https://www.arm.com .

[3] Xu Qiumin, al . Performance Analysis of NVMe SSDs and Their Implication on Real World Databases. In: Procs. of the 8th ACM Intl. Systems and Storage Conf.:1–11ACM; 2015; New York, NY, USA.

[4] Zhao D., Liu N., Kimpe D., Ross R., Sun X. H., Raicu I.. Towards Exploring Data-Intensive Scientific Applications at Extreme Scales through Systems and Simulations. *IEEE Transactions on Parallel and Distributed Systems.* 2016;27(6):1824-1837.

[5] Pascual Jose Antonio, Concatto Caroline, Lant Joshua, Navaridas Javier. On the Effects of Data-Aware Allocation on Fully Distributed Storage Systems for Exascale. In: Heras Dora Blanca, Bougé Luc, Mencagli Gabriele, et al. , eds. *Euro-Par 2017: Parallel Processing Workshops - Euro-Par 2017 International Workshops, Santiago de Compostela, Spain, August 28-29, 2017, Revised Selected Papers*, Lecture Notes in Computer Science, vol. 10659: :725–736Springer; 2017.

[6] Concatto Caroline, Pascual Jose A., Navaridas Javier, et al. A CAM-free Exascalable HPC Router. In: Architecture of Computing Systems (ARCS):To appear; 2018.

[7] Feitelson Dror G., Rudolph Larry, Schwiegelshohn Uwe. Parallel Job Scheduling — a Status Report. In: JSSPP'04:1–16Springer-Verlag; 2005; Berlin, Heidelberg.

[8] Pascual Jose Antonio, Miguel-Alonso José, Lozano José Antonio. Locality-aware policies to improve job scheduling on 3D tori. *The Journal of Supercomputing.* 2015;71(3):966–994.

[9] Pascual Jose A., Miguel-Alonso Jose, Lozano Jose A.. Optimization-based mapping framework for parallel applications. *Journal of Parallel and Distributed Computing.* 2011;71(10):1377 - 1387.

[10] Bhatele Abhinav, Mohror Kathryn, Langer Steven H., Isaacs Katherine E.. There Goes the Neighborhood: Performance Degradation Due to Nearby Jobs. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis:1–12ACM; 2013; New York, NY, USA.

[11] Johnson Christopher R., Bunde David P., Leung Vitus J.. A Tie-Breaking Strategy for Processor Allocation in Meshes. In: 39th Intl. Conf. on Parallel Processing, ICPP Workshops, San Diego, California, USA:331–338; 2010.

[12] Pascual Jose Antonio, Navaridas Javier, Miguel-Alonso José. Effects of Topology-Aware Allocation Policies on Scheduling Performance. In: Job Scheduling Strategies for Parallel Processing,, JSSPP 2009, Rome, Italy, May 29, 2009.:138–156; 2009.

[13] Balzuweit Evan, al . Local search to improve coordinate-based task mapping. *Parallel Computing.* 2016;51:67–78.

[14] Bhatele Abhinav, Gamblin Todd, Langer Steven H., et al. Mapping Applications with Collectives over Sub-communicators on Torus Networks. In: SC '12:1–11IEEE Computer Society Press; 2012; Los Alamitos, CA, USA.

[15] Power Russell, Li Jinyang. Piccolo: Building Fast, Distributed Programs with Partitioned Tables. In: Procs. of the 9th USENIX Conf. on Operating Systems Design and Implementation:293–306; 2010; Berkeley, CA, USA.

[16] Santos-Neto Elizeu, Cirne Walfredo, Brasileiro Francisco, Lima Aliandro. Exploiting Replication and Data Reuse to Efficiently Schedule Data-Intensive Applications on Grids:210–232. Berlin, Heidelberg: Springer 2005.

[17] Hammoud M., Sakr M. F... Locality-Aware Reduce Task Scheduling for MapReduce. In: Intl. Conf. on Cloud Computing Technology and Science:570–576; 2011; Washington, DC, USA.

[18] Chen T. Y., Wei H. W., Wei M. F., Chen Y. J., Hsu T., Shih W. K.. LaSA: A locality-aware scheduling algorithm for Hadoop-MapReduce resource assignment. In: 2013 International Conference on Collaboration Technologies and Systems (CTS):342-346; 2013.

[19] Zhang Xiaohong, al . An Effective Data Locality Aware Task Scheduling Method for MapReduce Framework in Heterogeneous Environments. In: Intl. Conf. on Cloud and Service Computing:235–242; 2011; Washington, DC, USA.

[20] Bezerra Aprigio, al. . Job Scheduling for Optimizing Data Locality in Hadoop Clusters. In: 20th European MPI Users' Group Meeting:271–276ACM; 2013; New York, NY, USA.

[21] Kosar Tevfik, Balman Mehmet. A New Paradigm: Data-aware Scheduling in Grid Computing. *Future Gener. Comput. Syst.*. 2009;25(4):406–413.

[22] Topcuouglu Haluk, Hariri Salim, Wu Min-you. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Trans. Parallel Distrib. Syst.*. 2002;13(3):260–274.

[23] Jin Jiahui, Luo Junzhou, Song Aibo, Dong Fang, Xiong Runqun. BAR: An Efficient Data Locality Driven Task Scheduling Algorithm for Cloud Computing. In: CCGRID '11:295–304IEEE Computer Society; 2011; Washington, DC, USA.

[24] Wallace Sean, Yang Xu, Vishwanath Venkatram, et al. A Data Driven Scheduling Approach for Power Management on HPC Systems. In: SC '16:56:1–56:11IEEE Press; 2016; Piscataway, NJ, USA.

[25] Caíno-Lores Silvina, Carretero Jesús. A Survey on Data-Centric and Data-Aware Techniques for Large Scale Infrastructures. *Intl. Journal of Computer, Electrical, Automation, Control and Information Engineering.* 2016;10(3):517 - 523.

[26] Berejuck Marcelo Daniel, Zeferino Cesar Albenes. Adding Mechanisms for QoS to a Network-on-chip. In: Proceedings of the 22Nd Annual Symposium on Integrated Circuits and System Design: Chip on the Dunes:1–6ACM; 2009; New York, USA.

[27] Corrêa Edgard de Faria, Silva Leonardo Alves de Paula e, Wagner Flávio Rech, Carro Luigi. Fitting the Router Characteristics in NoCs to Meet QoS Requirements. In: SBCCI '07:105–110ACM; 2007; New York, NY, USA.

[28] Sudev B., Indrusiak L. S., Harbin J.. Network-on-Chip packet prioritisation based on instantaneous slack awareness. In: 2015 IEEE 13th International Conference on Industrial Informatics (INDIN):227-232; 2015.

[29] Villar Juan A., García Pedro J., Alfaro Francisco J., Sánchez José L., Quiles Francisco J.. An integrated solution for QoS provision and congestion management in high-performance interconnection networks using deterministic source-based routing. *The Journal of Supercomputing.* 2013;66(1):284–304.

[30] Blomer J. Experiences on File Systems: Which is the best file system for you?. *Journal of Physics: Conference Series.* 2015;664(4):042004.

[31] Vaidya Madhavi, Deshpande Shrinivas. Critical Study of Performance Parameters on Distributed File Systems Using MapReduce. *Procedia Computer Science.* 2016;78:224 - 232. 1st International Conference on Information Security and Privacy 2015.

[32] Lustre . https://www.lustre.org .

[33] GPFS . https://www.ibm.com/us-en/marketplace/scale-out-file-and-object-storage .

[34] HDFS . https://hadoop.apache.org/ .

[35] GFS . https://research.google.com/archive/gfs.html .

[36] GFS2 . https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html-single/global_file_system_2/index .

[37] CEPH . ceph.com .

[38] BeeGFS . https://www.beegfs.com .

[39] Mellanox . https://www.mellanox.com/related-docs/whitepapers/WP_Deploying_ Ceph_over_High_Performance_Networks.pdf .

[40] Pascual Jose A., Miguel-Alonso Jose, Lozano Jose A.. Strategies to Map Parallel Applications onto Meshes:197–204. Distributed Computing and Artificial Intelligence: 7th Intl. SymposiumSpringer Berlin Heidelberg 2010.

[41] Erickson Alejandro, Kiasari Abbas Eslami, Navaridas Javier, Stewart Iain A.. An Optimal Single-Path Routing Algorithm in the Datacenter Network DPillar. *IEEE Trans. Parallel Distrib. Syst.*. 2017;28(3):689–703.

[42] Erickson Alejandro, Stewart Iain A., Pascual Jose Antonio, Navaridas Javier. Improved routing algorithms in the dual-port datacenter networks HCN and BCN. *Future Generation Comp. Syst.*. 2017;75:58–71.

[43] Erickson Alejandro, Stewart Iain A., Navaridas Javier, Kiasari Abbas Eslami. The stellar transformation: From interconnection networks to datacenter networks. *Computer Networks.* 2017;113:29–45.

[44] Valiant L. G.. A Scheme for Fast Parallel Communication. *SIAM Journal on Computing.* 1982;11(2):350–361.

[45] Gomez C., Gilabert F., Gomez M. E., Lopez P., Duato J.. Deterministic versus Adaptive Routing in Fat-Trees. In: :1–8; 2007.

[46] Sullivan Herbert, Bashkow T R. A Large Scale, Homogeneous, Fully Distributed Parallel Machine, I. In: ISCA '77:105–117ACM; 1977; New York, NY, USA.

How cite this article: Jose A. Pascual, Joshua Lant, Caroline Concatto, Andrew Attwood Javier Navaridas Mikel Luján and John Goodacre (2017), On the Effects of Allocation Strategies for Exascale Computing Systems with Distributed Storage and Unified Interconnects, *Concurrency and Computation: Practice and Experience*, *2017;00:1–6*.