Adding Storage Simulation Capacities to the SimGrid Toolkit: Concepts, Models, and API

Adrien Lebre*, Arnaud Legrand†, Frédéric Suter‡§, Pierre Veyre§

*Inria, Ecole des Mines de Nantes/LINA, Nantes, France
adrien.lebre@inria.fr
†CNRS/Inria/University of Grenoble, Grenoble, France
arnaud.legrand@imag.fr
‡ Inria, LIP, ENS Lyon. Lyon, France
§ IN2P3 Computing Center, CNRS, Lyon-Villeurbanne, France
firstname.lastname@cc.in2p3.fr

***Abstract*—For each kind of distributed computing infrastructures, i.e., clusters, grids, clouds, data centers or supercomputers, storage is a essential component to cope with the tremendous increase in scientific data production and the ever-growing need for data analysis and preservation. Understanding the performance of a storage subsystem or dimensioning it properly is an important concern for which simulation can help by allowing for fast, fully repeatable, and configurable experiments for arbitrary hypothetical scenarios. However, most simulation frameworks tailored for the study of distributed systems offer no or little abstractions or models of storage resources.**

In this paper, we detail the extension of SimGrid, a versatile toolkit for the simulation of large-scale distributed computing systems, with storage simulation capacities. We first define the required abstractions and propose a new API to handle storage components and their contents in SimGrid-based simulators. Then we characterize the performance of the fundamental storage component that are disks and derive models of these resources. Finally we list several concrete use cases of storage simulations in clusters, grids, clouds, and data centers for which the proposed extension would be beneficial.

I. Introduction

The tremendous increase in scientific data production and the ever-growing need for data analysis and preservation coming from various scientific domains create a great emphasis on storage components. Understanding the performance of a storage subsystem thus becomes an important concern independent of the scale and type of distributed computing infrastructure. Clusters, grids, clouds, data centers, and supercomputers all comprise storage components whose specifics may differ but all need to be well understood.

Data centers usually combine different types of storage components, ranging from tape libraries to disks attached to compute nodes, to form a complex hierarchical mass storage system and store several Petabytes of data. Moreover these systems are generally shared by multiple users. Devising the appropriate storage infrastructure for a given workload and defining sharing policies among users are thus complex tasks. Decisions are usually taken based on years of experience shared by system administrators and users. However, this process lacks of objective data about the performance of a given candidate infrastructure. Expertise is subjective and perceptions might be contradictory. Simulation may help to obtain the expected objective indicators and compare candidate infrastructures on a fair basis without having to deploy them. The issues are different for Supercomputers whose storage nodes typically comprise tens of thousands of individual disks interconnected through a dedicated storage high-speed network, and managed by a parallel file system. Due the scale of such infrastructures and the dramatic decrease of the Mean Time Between Failures (MTBF), a lot of papers consider application checkpointing [1]. Accurately modeling and simulating the impact of reading and writing checkpointed data on disks is thus crucial to design efficient policies. Frameworks, such as Hadoop, that allow users to benefit of distributed computing to process large data sets in a reliable, scalable, and simple way are commonly deployed on Clusters. Such frameworks heavily rely on specific and tunable file systems (e.g., HDFS for Hadoop). There, simulation can help to find an optimal tuning according to both cluster configuration and workload characteristics. Finally, in Computing Grids and Clouds, storage is perceived at a coarser grain, usually as a set of services offered by multiple data centers. The underlying infrastructure is often hidden to users, brokers, or even administrators that manage another administrative region. Despite an evident lack of precise information, the simulation of such storage elements, be they file systems or object stores, can be used to design data management and replication policies without having to disrupt production while controlling the noise induced by external load. In the case of Clouds, integrating the monetary cost associated to data storage in the simulated evaluation of provisioning and scheduling heuristics is a great added value.

This non-exhaustive list of use cases illustrates a clear need for tools to simulate storage operations and data management. For each of the aforementioned types of distributed computing infrastructures, there exists simulation toolkits in the literature that offer some storage simulation capacities [2], [3], [4], [5], [6], [7]. However, their use is often limited to the specific study for which they have been designed. Moreover, a design

driven by a specific use case is likely to lead to crude modeling simplifications that do impact the particular study but prevent an application in another context.

In this paper, we detail the extension of SimGrid [8], a *versatile* toolkit for the simulation of large-scale distributed computing systems, with storage simulation capacities. Versatility implies that the implementation of SimGrid provides the necessary capabilities to run simulations for multiple domains accurately and scalably. Then we designed this extension with these objectives of versatility and accuracy in mind: the proposed API does not have to be specific to a type of infrastructure and the underlying models have to faithfully assess the behavior of the studied systems. The contribution of this work can be decomposed as follows:

- A comprehensive description of the characteristics, contents, location, and access method of storage resources.
- An original API to develop SimGrid-based simulators that manipulate storage resources and files.
- A performance analysis of various types of disks from which we derive models used by the simulation kernel.
- A list of envisioned simulators, that cover different types of distributed infrastructures and can all be developed based on the proposed API and models, but whose implementation is out of the scope of this paper.

This paper is organized as follows. After discussing related work in Section II, we give in Section III a brief overview of the SimGrid toolkit. Then we detail in Section IV our aims and design choices. The proposed implementation is detailed in Section V. We experimentally characterize the performance of disks and derive models in Section VI. In Section VII, we detail multiple storage simulators that could be implemented thanks to the proposed contributions. Finally, Section VIII concludes this paper and presents future work directions.

II. Related Work

Simulation of storage resources can be done at low level, to accurately model and study the behavior of magnetic tapes [9], [10], hard-drive disks [11], solid-state drives [12], and up to storage-area networks [5]. Such simulators rely on a detailed discrete-event simulation of storage resources. For instance, the DiskSim simulator [11] models the operation of the storage hardware at the block level. The simulator in [5] uses it as a basis for implementing a storage area network simulator that models other hardware components (e.g., buses and networks) and software components (e.g., file systems). This approach can be seen as the equivalent of the packet-level simulation of network resources or the cycle-accurate simulations of CPUs. They all are of limited scalability because of unacceptably long simulation times. Moreover, correctly instantiating such complex models is difficult.

A few simulators from the grid computing and cloud computing domains provide simple storage access time models. For instance GridSim [2] and CloudSim [3] model data access times using a simple model based on a (fixed or randomly generated) seek time and a fixed data transfer rate. This model ignores sharing and contention effects, making it not representative of actual storage resource usage. iCanCloud [4] provides the most sophisticated model: it considers individual disk blocks and simulates seek times based on the locality of block accesses, but instantiating such a model in a realistic way is non-trivial. The same authors also proposed SIMCAN [13], a framework for the simulation of distributed architectures that encompasses a complex storage modeling module. It is composed of several layers that respectively simulate the behavior of disk drives, volumes, and file systems (including remote and parallel ones), but stacking components whose behaviors strongly interact is likely to induce a complex and very specific configuration procedure. In the HPC field, the current focus is on scalability, sometimes at the price of accuracy. For instance, I/O times can be randomly generated to increase scalability [6]. However, the ExascaleIO workgroup [14] outlines the need for modeling components built from characterizations of real storage hardware and software components to detect possible side effects that will occur only at a certain scale as soon as possible. Storage modeling and simulation are also crucial to the study of data-intensive applications and systems. There exists several MapReduce simulation toolkits [7], [15] that include a model of HDFS. However, they usually rely on simplistic disk models or on delays coming from execution traces as their focus is on modeling the logic of the file system rather than its performance.

In this work we propose concepts and an API for the simulation of storage resources and data management that is not limited to a specific type of infrastructure or particular case study. This approach is in line with the main objective of versatility targeted by the SimGrid toolkit. We also build tractable models that derive from a thorough characterization of storage resources and take import phenomena such as sharing and contention effects into account. We leverage the SimGrid expertise in network modeling [16] to adapt coarse-grain fluid models to the simulation of storage resources.

III. SimGrid Overview

To support the simulation of storage resources and address the different use cases listed in the introduction, we decided to extend an existing simulation toolkit rather than developing a stand-alone simulator. The rationale is to leverage as much as possible well-tried simulation mechanisms to focus on the addition of the required missing parts that are concepts, models, and an API, specifically designed for storage simulation.

We opted for the SimGrid toolkit that provides core functionalities for the simulation of distributed applications in heterogeneous distributed environments [8]. SimGrid relies on a scalable and extensible simulation engine and offers several user APIs, which we detail hereafter. This tool gained a strong expertise in discrete-event simulation, especially in network modeling [16]. While storage is as much important as compute nodes or interconnection networks in many large scale distributed computing infrastructures, SimGrid did not provide any storage abstraction until the proposed extension implemented in its latest stable release (3.11.1).

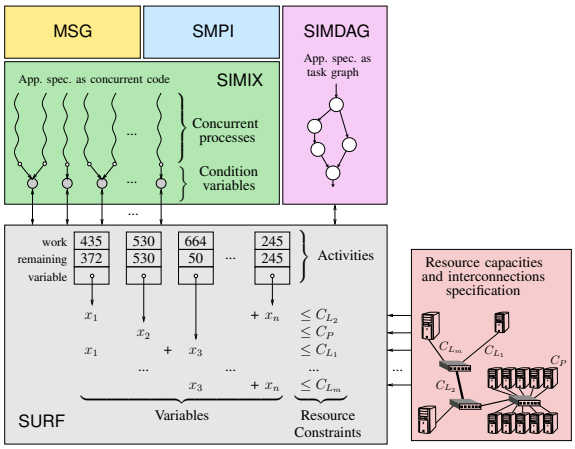

Figure 1. Design and internals of SimGrid.

Figure 1 shows the main components of SimGrid, and depicts some of the key concepts in its design. The top part shows the three APIs through which users can develop simulators. The MSG API allows users to describe an application as a set of concurrent processes. These processes execute user code and place MSG calls to simulate computation and communication activities. The SMPI API makes it possible to simulate unmodified MPI applications. The mechanisms for simulating the concurrent processes for these two APIs are implemented as part of SIMIX, which is a kernel (in the Operating Systems sense of the term) that provides process control, and synchronization abstractions. The set of concurrent processes synchronize on a set of condition variables. Each condition variable corresponds to a simulated activity, computation or communication, and is used to ensure that concurrent processes wait on activity completion to make progress throughout (simulated) time. The third API, SimDAG, does not use concurrent processes but instead specifies an abstract acyclic task graph of communicating computational tasks.

Regardless of the API used, a simulated application consists of a set of *activities* which are to be executed on simulated hardware resources. Compute resources are defined in terms of compute capacities (e.g., CPU cycles per time unit). The compute resources are interconnected via a network topology that comprises network links and routing elements, defined by bandwidth capacities and latencies. An example of specification of available resources is depicted in Figure 1. Capacity (C) values are shown for three of the network links (L_1, L_2, and L_m), and one of the compute resources (P).

The simulation core, i.e., the component that simulates the execution of activities on resources, is called SURF. Each activity is defined by an amount of work to accomplish (e.g., number of CPU cycles to execute, number of bytes to transfer) and a remaining amount of work. When its remaining amount of work reaches zero the activity completes, signaling the corresponding SIMIX condition variable or resolving a task dependency in SimDAG. Each activity corresponds to a variable x_i, which represents a resource share used by the activity. A set of constraints over these variables describes how the activities compete for resources, allowing SURF to compute resource allocations and make the simulation progress.

Adding storage simulation capacities to the SimGrid toolkit then consists in extending the lower layers, i.e., SURF and SIMIX, with models and sharing policies, and exposing specific functions to handle storage resources and their contents within each of the APIs. Thanks to the modularity of SimGrid, such an extension can be easily implemented while leveraging many of the advanced features of SimGrid's internals.

IV. Defining Storage Related Concepts

Before detailing how we do add storage abstractions to the SimGrid toolkit, we specify the aims and scope of the proposed extension and define its underlying concepts. From a user point of view, storage corresponds to two basic concepts: a *storage volume* in which *files* are stored.

In our context, a file can be abstracted by its complete name, i.e., the absolute path from the mount point in the file system tree, its size, and the storage volume it is stored on. The size is the only file-related information that has to be handled by the simulation kernel. Indeed the file size allows the kernel to determine the time needed to complete a read or write operation and to manage the filling of storage volume in a dynamic way. If a user requires other file-related information, e.g., access rights, creation or last modification dates,s/he has to manage it in the code of his/her simulator. To that extent, SimGrid provides a mechanism of *properties* based on key/value pairs that can be associated to any resource and an API to retrieve these properties during the simulation. We extend this mechanism to files. For similar reasons, the proposed extension does not consider the data in files, nor does it allow users to navigate in the file tree. The operations on files that are exposed to users are: *opening* and *closing* a file, *seeking* into a file up to a given offset, *reading* or *writing* a certain amount of data, regardless of its meaning, from or to a file, and *moving*, *copying*, either locally or remotely, or *deleting* a file.

From a user perspective, a storage component mainly corresponds to a volume of a given *capacity* on which files can be stored in a persistent way. A storage volume contains a *set of files* on which the aforementioned operations can be performed. It has a *name* and a *type*, ranging from a single disk to a shared file system or a tape library, is physically *attached* to a machine, and is accessed from a *mount point*. The same storage volume can be mounted by several compute nodes, allowing for the sharing of the data stored on it. A compute node can also mount several storage elements, to simulate different partitions for instance. The main operation related to a storage volume in user space is to *list* the files stored on it. The operations that impact the list of files associated to a storage volume, i.e., creation, modification, or deletion of files, dynamically modify its *available* and *used capacities*.

Files are associated to storage components at the beginning of a simulation when the description of the whole platform is loaded. Typical data centers usually host several millions of files on their storage infrastructure. Then creating a simulated

entity, similar to a file descriptor, for every single file stored on a large scale distributed platform would lead to a prohibitive memory footprint and is thus hardly possible. In our proposal, the data stored on a storage volume is then considered as an inert list of files described by their complete names and sizes. A simulated entity is only created when a given file is opened and destroyed upon closing. We consider at least two description and access methods depending on the number of files that are stored in a storage volume. For small to medium amounts of files, using a *text description* is a simple and reasonable approach. For larger lists of files, relying on a *database* whose entries store the same information as in a text file would constitute an easier and more scalable approach.

V. Proposed Implementation

In this section we detail the proposed extension of the SimGrid toolkit to handle files and storage components. We added the necessary support in the SURF and SIMIX layers, and focused on the most used API (MSG). We present hereafter how to declare storage components and the proposed API.

A. Description of Storage Components

We illustrate the declaration of storage components in the XML description format provided by SimGrid with a simple example depicted in Figure 2. This platform comprises two machines, named `bob` and `alice`, interconnected through a direct network connection. One disk is attached and mounted by each machine. `bob` also *remotely* mounts the disk attached to `alice`, which will imply a network communication in further I/O operations. However, the impact on the CPU of the remote machine is not modeled as explained in Section VI-A.

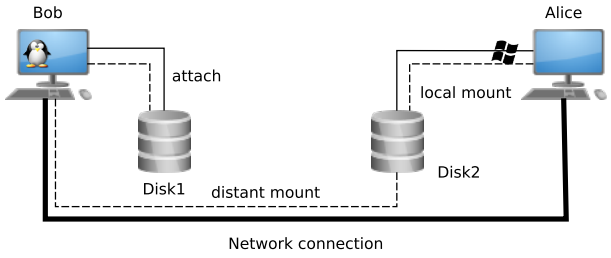

Figure 2. Example of a simple platform with storage elements.

Figure 3 presents a possible XML description of the platform of Figure 2. First, we describe a *storage type* (lines 6-12) that corresponds to a single SATA-II hard drive disk, whose capacity is of 500GB. Storage instances of this type will be simulated according to a simplistic `linear` model, whose parameters, i.e., a read bandwidth `r_bw` and a write bandwidth `w_bw`, are given in lines 10-11. A list of files is associated to this storage type, thanks to the `content` attribute, in a format given by the `content_type` attibute, here in a text file that follows a UNIX syntax for paths. Associating such a list of files to a storage type means that it will be inherited by all the declared instances of that type. Then, the first instance, named `Disk1` (lines 14-15), which is attached to `bob` does not specify any specific list of files and thus inherits of the generic list of its type. Conversely the second instance, named `Disk2`

```
1  <?xml version='1.0'?>
2  <!DOCTYPE platform SYSTEM
3   "http://simgrid.gforge.inria.fr/simgrid.dtd">
4  <platform version="3">
5    <AS id="AS0" routing="Full">
6      <storage_type id="SATA-II_HDD" size="500GB"
7                    content_type="txt_unix"
8                    content="unix_content.txt"
9                    model="linear">
10       <model_prop id="r_bw" value="92MBps"/>
11       <model_prop id="w_bw" value="62MBps"/>
12     </storage_type>
13
14     <storage id="Disk1" typeId="SATA-II_HDD"
15              attach="bob"/>
16
17     <storage id="Disk2" typeId="SATA-II_HDD"
18              attach="alice"
19              content_type="txt_windows"
20              content="windows_content.txt" />
21
22     <host id="bob" power="1Gf">
23       <mount id="Disk1" name="/home"/>
24       <mount id="Disk2" name="/windows"/>
25     </host>
26
27     <host id="alice" power="1Gf">
28       <mount id="Disk2" name="c:"/>
29     </host>
30
31     <link id="link1" bandwidth="125MBps"
32           latency="50us"/>
33
34     <route src="bob" dst="alice"
35            symmetrical="YES">
36       <link_ctn id="link1"/>
37     </route>
38   </AS>
39 </platform>
```

Figure 3. Example of storage description in the SimGrid format.

and attached to `alice` (lines 17-20), is given a specific list of file, described in a different format, that of Windows™. The method used to parse file lists derives from the value of the `content_type` attribute.

The second part of the XML file describes `bob` and `alice`. The former mounts the two disks as in Figure 2 (lines 22-25) while the latter only mounts `Disk2` (lines 27-29). The last part describes the network interconnection between the two machines, i.e., a simple symmetrical route that comprises a single network link (lines 34-37).

B. Proposed API

A subset of the functions added to the MSG C API to handle storage components and files is given in Table I. Java bindings are also available. We omit to present some minor utility functions to focus on the main I/O operations. All the mandatory functions that were mentioned in the previous section have been implemented. We defined two new structures, `msg_file_t` and `msg_storage_t` that represent the user view of files and storage spaces respectively. File and storage sizes are expressed as `sg_size_t` which corresponds to a 64-bit unsigned integer. The `MSG_storages_as_dynar` function allows the user to get a dynamic array that comprises all the storage elements declared in the simulated platform while the `MSG_host_get_attached_storage_list` and `MSG_host_get_mounted_storage_list` functions return a dictionary whose keys are respectively machine

names and mount points and the values are the corresponding `msg_storage_t` objects.

When a storage element is *mounted* by a machine but physically *attached* to another one, e.g., `Disk2` is mounted by `bob` and attached to `alice` in Figure 2, reading or writing a file does not only imply an operation on the storage element itself, but also a network communication between the two machines. We illustrate such a situation in Figure 4 that details the sequence involved in a `MSG_file_read` on a remotely-attached disk (depicted by green circles). The read operation is initiated by `bob` on `Disk 1` that is physically attached to `carol`. The first step consists in delegating the operation to `carol` (1). Then the file is read on the disk (2). Finally, the file is sent back to `bob` over the network (3). Symmetrically, a remote write operation consists in sending the file to the right machine, delegating the write operation, and actually write the file on the disk.

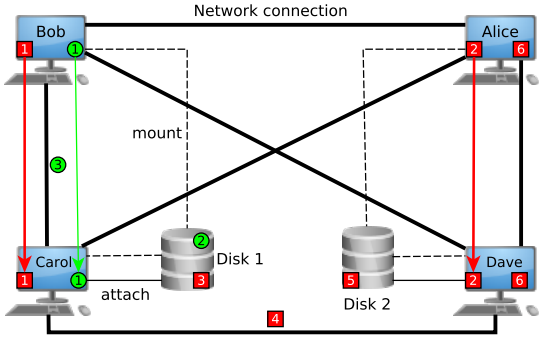

Figure 4. Decomposition of the `MSG_file_read` (green circles) and `MSG_file_rcopy` (red squares) operations.

For similar reasons a remote operation between two machines, i.e., `MSG_file_rcopy` or `MSG_file_rmove` might imply a network communication between a completely different pair of machines. This is illustrated (with red squares) by Figure 4. To remotely copy a file stored on `Disk1` onto `Disk2` that is mounted by `alice`, `bob` first has to delegate a *read* operation to the machine to which `Disk1` is attached, that is `carol` (1). Conversely, the machine to which the destination disk is attached has to be identified as being `dave` by `alice` (2). Then the file is read by `carol` on `Disk1` (3), sent to `dave` over the network (4) and written on `Disk2` by `dave` (5). Finally, the modification of the contents of `Disk2` is notified to `dave` and `alice` (6). The proposed implementation takes such a delegation of operations into account in the aforementioned functions.

VI. Modeling Storage Resources

A. Implementing a Disk Model in SimGrid

From the simulation toolkit standpoint, a storage space is a *resource* as CPU or network links are. Its usage is then defined by a *model* that describes the evolution of an action, e.g., reading or writing a file, on this resource under certain constraints, e.g., access throughput or replication algorithms. As mentioned earlier, we place ourselves at the granularity of a file when we simulate such storage resources. The amount of data involved in one I/O operation is thus assumed to be rather large and we expect a sequential read or write of data. In this context, natural assumptions underlying a model of a storage resource such as disks are: (i) *linearity*, i.e., the execution linearly increases along with the size of the file; (ii) *negligible latency*, i.e., there is no specific and incompressible significant overhead associated to I/O operations; and (iii) *perfectly fair and efficient bandwidth sharing*, i.e., no given I/O operation is given extra priority nor has a "better" access to the resource than other concurrent operations, and that increasing the number of concurrent operations does not degrade the aggregate bandwidth.

These common sense expectations about the behavior of storage resources let us believe that implementing a storage model in SimGrid is rather straightforward. Indeed, SimGrid uses a unified model to simulate the execution of activities on simulated resources. This model is purely analytical so as to afford scalability by avoiding cycle-, block-, and packet-level simulation of compute, storage, and network resource usage. Formally, given a resource r, and a set of simulated activities, $\mathcal{A}$, the model specifies the following constrained Max-Min optimization problem:

$$\begin{aligned} &\text{MAXIMIZE } \min_{a \in \mathcal{A}} \ \rho_a \\ &\text{UNDER CONSTRAINTS} \\ &\left\{ \sum_{a \in \mathcal{A} \text{ using resource } r} \ \rho_a \leq C_r, \right. \end{aligned} \tag{1}$$

where C_r denotes the capacity of resource r, and ρ_a denotes the resource share allocated to activity a. Solving this optimization problem, which boils down to solving a linear system, yields instantaneous resource shares given which resources are used by which activities. Given these computed resource shares at simulated time t_0, for all simulated resources, the SURF component of SimGrid computes the first activity that will complete, advances the simulated clock to that time, say t_1, removes the completed activity from consideration, accounts for the progress of each activity given its resource shares and the simulated elapsed time $t_1 - t_0$, and possibly adds newly created activities. The key aspect of this model is that it is general and can be used to simulate key aspects of CPU and network. The validity of this type of models in the context of networking has been thoroughly assessed [16].

We propose to follow a similar approach to model storage resources. Indeed, modeling the concurrent execution of basic I/O operations, i.e., read and write, amounts to determining how bandwidth is shared among I/O "flows". This flow-level vision is not the only similarity shared with what has been proposed to model the network in SimGrid. First, there exists a highly-detailed alternative for storage modeling, i.e., block-level models as offered by DiskSim [11], as there are packet-level models in the network domain. Both might lead to more accurate simulated times, but at the cost of unacceptably long simulation times. Second, one could object, as some do for network models, that ignoring the fine details captured by block- or packet-level simulators and the different software layers between an application-level operation and the modeled

`msg_file_t`	`MSG_file_open (char* fullpath, void* data)`
	Open a file and create the corresponding `msg_file_t` object. Arbitrary user data can be attached to the object.
`msg_error_t`	`MSG_file_close (msg_file_t fd)`
	Close the file and destroy the `msg_file_t` object
`sg_size_t`	`MSG_file_read (msg_file_t fd, sg_size_t size)`
	Read `size` bytes from a file. Return the number of actually read bytes.
`sg_size_t`	`MSG_file_write (msg_file_t fd, sg_size_t size)`
	Write `size` bytes to a file. Return the number of actually written bytes.
`msg_error_t`	`MSG_file_seek (msg_file_t fd, sg_offset_t offset, int origin)`
	Set the file position indicator in the `msg_file_t` by adding `offset` bytes to the position specified by `origin` (`SEEK_SET`, `SEEK_CUR`, or `SEEK_END`)
`sg_size_t`	`MSG_file_tell (msg_file_t fd)`
	Return the current position indicator in the `msg_file_t`.
`msg_error_t`	`MSG_file_unlink (msg_file_t fd)`
	Remove the file from the contents of its associated storage. Also destroy the `msg_file_t` object
`msg_error_t`	`MSG_file_move (msg_file_t fd, char* new_name)`
	Move the file within the contents of the associated mounted storage.
`msg_error_t`	`MSG_file_rcopy (msg_file_t fd, msg_host_t dest, char* new_name)`
	Copy a file to a storage element mounted by a remote host. The file is added to contents of the remote storage element on `dest`.
`msg_error_t`	`MSG_file_rmove (msg_file_t fd, msg_host_t dest, char* new_name)`
	Move a file to a storage element mounted by a remote host. The file is added to contents of the remote storage element on `dest` and removed from the contents of its original location.

`sg_size_t`	`MSG_storage_get_size (msg_storage_t st)`
	Return the full capacity of `st`.
`sg_size_t`	`MSG_storage_get_free_size (msg_storage_t st)`
	Return the currently available capacity of `st`.
`sg_size_t`	`MSG_storage_get_used_size (msg_storage_t st)`
	Return the currently used capacity of `st`.

`xbt_dict_t`	`MSG_host_get_attached_storage_list (msg_host_t host)`
	Return the list of storage components attached to `host`.
`xbt_dict_t`	`MSG_host_get_mounted_storage_list (msg_host_t host)`
	Return the list of storage components mounted by `host`.

`xbt_dynar_t`	`MSG_storages_as_dynar (void)`
	Return the list of all the storage elements in the platform.

Table I
SUBSET OF THE MSG API RELATED TO STORAGE AND FILE MANAGEMENT.

resource is doomed to be inaccurate. However, this popular wisdom was rebutted for network resources by the flow-level models implemented within the SimGrid toolkit. Not only they can be as accurate as their packet-level contenders, but they are also orders of magnitude faster (see [16] for details). We thus believe that flow-level models is a sound approach to the coarse-grain simulation of storage resources.

Since all SimGrid fluid models rely on the same kind of equations, one may wonder whether it would be possible to come up with a single model unifying CPU, network, and storage resources. While this would be possible in theory, some of the adaptations we came up with for correctly modeling some of the peculiarities of the TCP bandwidth sharing mechanisms (see [16] for more details) are not compatible with a standard model for CPU or with a simple fluid model for I/O like the one we propose in this article. Furthermore, such model would allows for a prediction of the impact of a remote I/O operation on the CPU of the host server, but this kind of interferences is difficult to model and may be non-linear. Then these different models are kept separate for now. The SimGrid toolkit rather focuses on correctly predicting interferences between actions of similar nature (computations with computations, communications with communications, I/Os with I/Os).

B. Checking the Model Assumptions

In this section, we assess the previous modeling assumptions in the context of single hard drive disks attached to compute nodes. To that extent we perform experiments to characterize the performance behavior of the resource when basic operations, i.e., unitary read and write operations, are performed, first, independently and second, concurrently. We take advantage of the Grid'5000 experimental testbed to perform this analysis [17]. More specifically, we ran experiments on three different clusters that comprise different kinds of hard drives. Table II summarizes the disk specifics for each cluster. Other information regarding CPU, RAM size, ... is available directly on the Grid'5000 website (https://www.grid5000.fr).

It is worth mentioning that this whole work was done in the spirit of open science and reproducible research. All ex-

Name	Model	Interface	Size (in GiB)	Max. Bandwidth (in MiB/sec)$^{(*)}$
edel	C400-MTFDDAA	SATA/SSD	128	244.8
griffon	Hitachi HDP72503	SATA-II	320	79.01
granduc	Seagate ST9146802SS	SAS	146	84.7

Table II
CHARACTERISTICS OF THE STORAGE DEVICES USED FOR ANALYSIS.
$^{(*)}$ Values observed with the `hdparm -t` unix command.

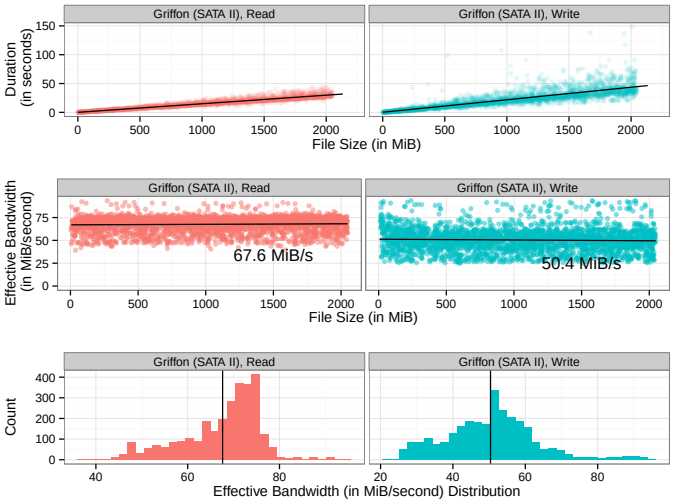

Figure 5. Characterization of read (left) and write (right) operation vs. file size on the SATA-II disks of the Griffon cluster.

periment results presented in this paper are publicly available on *figshare* [18]. Supplementary data, which is not presented in this paper due to space limitation, is also available at the same location along with all the scripts and raw data files which allow anyone to regenerate this document.

1) Independent Access: The following benchmarking procedure has been applied. We rely on the Flexible I/O (FIO) [19] tool (version 2.0.8) to perform unitary read or write operations. FIO is a widely used and highly configurable tool to benchmark and stress/hardware verification of storage resources. It provides all sorts of direct and indirect performance information, e.g., duration, bandwidth, operations per seconds, or latencies. We configure the tool in a way that limits the potential measure biases introduced either by the Operating System or the file system by using synchronous, non-buffered I/O operations executed in an exclusive way. We perform the read and write operations over a range from 32 kiB up to 2 GiB using a fixed block size of 32 kiB. I/O operations on files smaller than 32 kiB are likely to trigger specific phenomena, which would require a different experimental protocol to be captured. We decide to focus on files larger than 32 kiB to characterize the general behavior of disk resources. Then we randomize our sequence of unitary experiments to avoid any undesired effect due to the consecutive execution of operations on monotonically increasing or decreasing file sizes. We highlight that for write operations on the Granduc cluster, we had to use an additional micro-benchmark based on `dd` invocations. Indeed, the driver of the raid controller deployed on these nodes, prevents us to correctly setup the SAS hard drive leading to measure nonsensical values (due to a wrong configuration of the controller, it was not possible to correctly propagate data to the disk as expected by using the FIO `direct-io` flag). By measuring the time to perform and synchronize the read/write operations through `dd`, we were able to reproduce behaviors similar to those obtained with FIO.

a) Modeling SATA-II Disks: Figure 5 depicts the results obtained on the SATA-II disks of the Griffon cluster. The top graph shows the evolution of the duration in seconds of an I/O operation (read on the left column and write on the right column) when the size of the file (in MiB) increases.

This visualization of our randomized experiments globally confirms our first modeling assumption of *linearity*. For both operation types, it is easy to fit a linear model, depicted by the black lines on Figure 5. There is however an important variability and both sequential read and write accesses on SATA-II disks clearly show a *heteroscedastic* behavior. However, the variability appears to be proportional to file size and with such file sizes, latency is negligible, which justifies to solely focus on effective bandwidth of such operations, i.e., on the file size divided by the duration. The middle graph of Figure 5 shows the evolution of the effective bandwidth when the file size increases. As expected, this rate does not depend on file size but exhibits an important variability. It is thus safe to model these bandwidths as random variables whose empirical distributions are given in the bottom graph of Figure 5. These distributions are relatively regular (i.e., with only a single clear more) but do not follow any particular well-know distribution. The easiest way to account for such variability is thus to provide such sample distributions to SimGrid and to draw random variables according to them whenever a new access occurs.

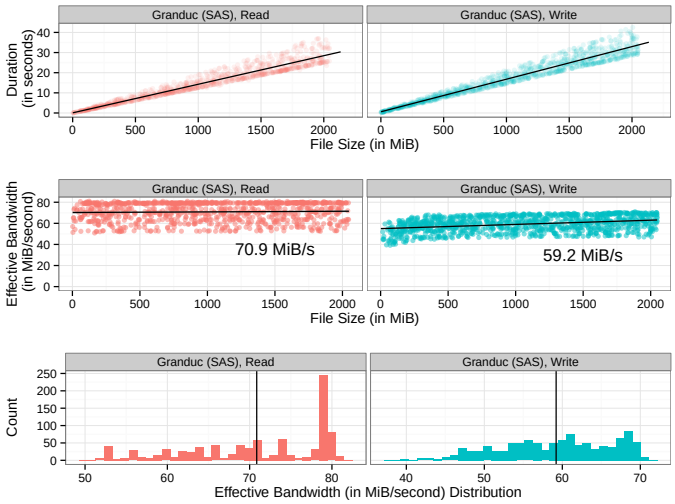

Figure 6. Characterization of read (left) and write (right) operation vs. file size on the SAS disks of the Granduc cluster.

b) Modeling SAS Disks: We follow the same approach to study the performance of SAS Disks (see Figure 6). Again, despite an important variability, the behavior is still linear and heteroscedastic (top graph) and the effective bandwidth is independent on file size (middle graph). The empirical distributions (both for read and write operations) exhibits a uncommon distribution with many evenly spaced local modes. Such a behavior is quite surprising but is uniform across the cluster nodes and cannot be explained by a temporary perturbation of the machines since the data acquisition period spans over several weeks. While simply replaying the sample distribution is enough to account for such variability, it may

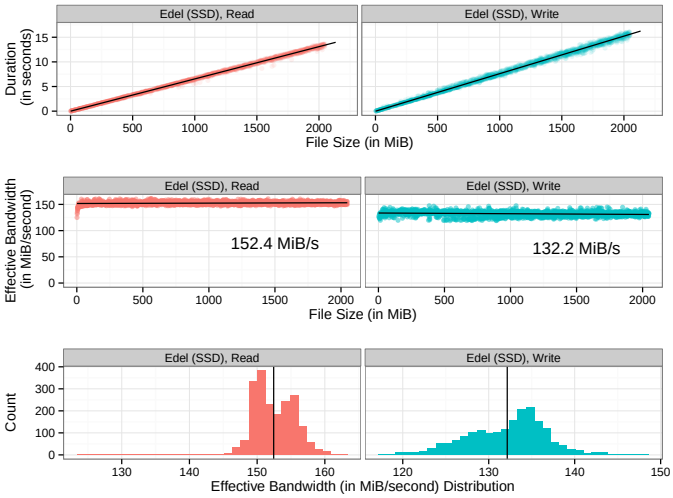

Figure 7. Characterization of read (left) and write (right) operation vs. file size on the SSD disks of the Edel cluster.

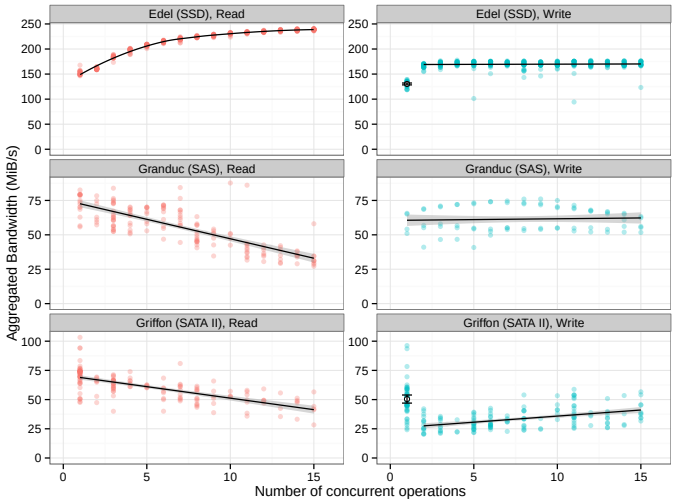

Figure 8. Modeling the impact of concurrent operations on the aggregate bandwidth.

be more meaningful to approximate such a distribution by Markov Chain Monte Carlo (MCMC) methods.

c) Modeling SSD Disks: Here we investigate the performance of SSD disks (on the Edel cluster) using the same experimental protocol. Again, the top graph of Figure 7 depicts the evolution of the duration in seconds of an I/O operation when the size of the file increases. As expected, such duration is linear for both operations and easily fits a linear model (depicted by black lines). Unlike classical hard drives, there is very little variability but we can apply the same modeling approach, i.e., focus on the distribution of the effective bandwidth. These distributions are quite regular (little variability and simple shape) but still do not follow any particular well-know distribution. Again, the easiest way to account for such variability is to provide sample distributions to SimGrid

Finally, it is worth noting that the observed bandwidths are far from the maximum value measured with the `hdparm` command (close to 250 MiB/sec as indicated in Table II). Additional investigations enabled us to determine that the default configuration of the `ext4` file system does not allow to get the maximum performance of SSD drives. As will be discussed in Section VI-B2, it seems that in such conditions, several accesses are mandatory to reach the maximum bandwidth.

2) Concurrent Access: To confirm the last assumption on *perfectly fair and efficient bandwidth sharing*, we run another set of FIO experiments. We select a small range of file sizes (10, 50, 100, 500, 1,024, and 2,048 MiB) and make the number of concurrent operations, either read or write, evolve from 1 up to 15. Again, we randomize the whole set of runs to avoid experimental biases. Figure 8 shows the obtained results and a sound model for each setting. Only the write operations performed on Granduc cluster obtain a fixed bandwidth that is not influenced by the number of concurrent operations. For all the other machine/operation couples, the bandwidth is significantly impacted by the number of concurrent operations. For instance, the aggregate bandwidth achieved for concurrent read operations on the Granduc and Griffon clusters decays linearly with concurrency. While it is not surprising to observe such I/O degradation because of arm movements on such hard drives, it is interesting to note that I/O improvements due to an increased concurrency can also be observed for the write operations on the Griffon cluster (SATA-II) and read operations on the Edel cluster (SSD). For the SSD disk The observed aggregate bandwidth improvement is quite significant, which means that the peak performance of the disk is far from being reached by a single operation. It is also interesting to note that the improvement or read operations is clearly non linear but is perfectly approximated by a LOESS regression. Although the write operations of Edel also improve with concurrent operations, there is no reason to resort to a non-linear model. The aggregated throughput improves when writing a second file in parallel but there is no benefit in increasing concurrency further. This can certainly be explained by the fact that the default configuration of the `ext4` file system does not allow to get the maximum performance of SSD drives. Finally, when writing data on the Griffon disks, there is a clear performance drop as soon as there are concurrent transfers but surprisingly, the aggregated throughput neither keeps decreasing (as with read operations on SAS or SATA-II disks) nor remains constant (as with SAS or SSD disks) but instead seems to increase. The exact reason for such behavior is not completely clear but is persistent and seems significant.

All such models can very easily be implemented in a simulator like SimGrid since it simply amounts to perform a slight modification of the total capacity of the resource as the number of concurrent transfers increases. Such modification is obviously reevaluated every time an access starts or ends and immediately reflects on all the the amounts of data to access.

While some of the behaviors encountered in this series of experiments can be imputed to non-optimal machine configuration (either at the hardware or software level) and may thus be considered as abnormal by I/O experts, they are real and

perfectly reproducible. In practice, production environments are far from being perfectly configured and often exhibit surprising performances. In the context of a generic simulator that aims at providing a way to evaluate scheduling heuristics, replication policies or checkpointing mechanisms, we think that it is important to be able to account for non ideal (but possible and realistic) behaviors. Indeed, although very smart proposals may have excellent performance in perfectly well-behaved environments, they may not be able to handle such variability or diversity of behaviors as well as simpler non-clairvoyant solutions.

C. Limitations of this Disk Model

This first storage resource model suffers of some limitations. First, our model is fluid and relies on a steady-state assumption that only makes sense for large files accessed sequentially. While our measurements for single operations reveal that a simple linear model is a good approximation, we expect our model to break on an intensive workload comprising either small files or random accesses. Indeed, if we can expect the behavior to be the same as the sequential one for SSD disks, overheads implied by arm movements on more conventional hard drives should significantly degrade the performance and thus invalidate our proposal.

There are also a few other hypothesis we did not check. We assumed that whenever a flow appears or disappears, bandwidth is instantaneously shared according to the new workload. We did neither evaluate the speed of re-convergence to steady-state nor checked what happens when performing both read and writes at the same time and how they interfere. Such evaluations are part of our planned future work.

VII. Storage Simulation Use Cases

The proposed addition of storage simulation capacities to the SimGrid toolkit is a preliminary but fundamental building block in the design of specialized simulators to conduct studies on clusters, grids, clouds, data centers, or supercomputers. Such studies leveraging SimGrid have been published before [7], [20], [21], but the lack of storage simulation capacities forced their authors to either ignore, circumvent, or implement in user space the storage part of their simulators. In the remaining of this section we present different envisioned uses cases that could benefit of the proposed extension as is, or require alternate models of storage resources. Whenever possible, we also indicate the added value of using SimGrid rather than developing a specific ad-hoc simulator.

A first use case is the simulation of MapReduce applications and HDFS on **clusters**. We mentioned in Section II that storage performance was usually ignored or inaccurately modeled. However, the evaluation of efficient data placement, movement, and replication policies can be badly impacted by a simplistic or unrealistic storage modeling. Moreover, capturing other performance drivers such as the network interconnects or the heterogeneity of compute nodes may allow researchers to evaluate more complex strategies. Thanks to the versatility of SimGrid, such simulators can be easily implemented without having to oversimplify any of the considered abstractions.

In the **grid** and **cloud** computing domains, simulation is a common approach to evaluate the performance of solutions to problems such as scheduling or resource provisioning and brokering. To be valuable, the conclusions drawn by such simulations studies have to remain sound once the developed solutions are deployed in production. The level of realism offered by SimGrid in terms of network simulation has already been leveraged in [20]. Adding the simulation of high-level storage components such as file catalog, storage resource manager, or gridFTP services, will reinforce this realism. However, declaring and modeling every single disk in a large-scale grid or cloud infrastructure would be cumbersome. Moreover, the specifics of the storage elements, e.g., disk bay, parallel file system or even a whole data center, are usually not exposed neither to the middleware nor the user in this context. Then a coarser approach is required to model storage elements as *black-boxes* and describe their performance as *bandwidth (and/or latency) matrices* as done by the authors of [22] for instance. Values in such matrices typically depend on the number of concurrent accesses and can be obtained from explicit measures or collections of data found in the literature. Such models would obviously be less accurate, but reflect what is actually experienced by grid or cloud users.

The hierarchical mass storage infrastructure of large **data centers**, comprises: (i) a magnetic tape library that provides large capacity at a low cost; (ii) a disk storage layer that acts as a cache with better performance of the mass storage subsystem; and (iii) a computing farm of around 1,000 servers that have their own disks. Thanks to the proposed concepts and API, it is possible to follow a incremental approach to develop simulators at each level of the hierarchy. This requires to describe the organization of the different components, i.e., tape library, disk drives, file catalog, and management policies, and develop models for the different types of involved storage resources. Additional models are also needed, typically for magnetic tapes. Performance studies and models of the access time to this kind of storage media have been proposed in the late nineties [9], [10]. The access time depends not only on the time to read the file itself but also on several factors such as the position of the file on the tape, whether the tape is already mounted or not, the position of the tape in the library, or even on the load of the tape readers and the availability of the robotic arms that move the tapes. Integrating all these factors into a single model is already challenging, but made even more complex by the fact that tape libraries are usually managed by systems that hide the location of files on tapes. Then simulating the exact chain of actions that take place in the tape library just from the name of a given file is hardly possible. Instead, we plan to distinguish in our model the time to access a tape, which is highly variable but independent of the file size, from the time to read a file on that tape, which can be predicted according to the model in [10]. A possible approach to model the tape access time could be to derive a distribution law from the analysis of usage logs.

In its current state, our extension of the SimGrid toolkit to storage simulation does not allow us to target **supercomputers**. Simulation studies in this context require models of high-performance parallel file systems. However we are in line with the requirements expressed by the ExascaleIO workgroup [14] such as the need for modeling components that are built from characterizations of real storage hardware and software components. The flexible yet accurate modeling approach we follow lets us study evolution trends while detecting side effects related to scale as soon as possible. Moreover, we believe that the seamless integration of network communication for some I/O operations (e.g., reading or writing on a remotely mounted disk), which is, to the best of our knowledge, not the case in other simulation toolkits, paves the way to the simulation of parallel file systems deployed over complex storage networks.

VIII. Conclusion

With the recent data deluge, storage is becoming the most important resource to master in modern computing infrastructures. Dimensioning and assessing the performance of storage systems are challenges for which simulation constitutes a sound approach. Unfortunately, only a few existing simulators of large scale distributed computing systems go beyond providing merely a notion of storage capacity. In this paper, we made a first step toward the simulation of such systems by extending the SimGrid toolkit with models, abstractions, and a programming interface, for files and storage components. We exposed the aims and limitations of this extension and characterized the performance behavior of several types of disks to derive a first model of storage resource. Finally, we listed use cases pertaining to different incarnations of distributed computing infrastructures that could benefit of the proposed contributions. All the presented developments are freely distributed as part the SimGrid project under the LGPL license (http://simgrid.org/download.html).

Our short term future work will be to implement, or help users of SimGrid to implement the aforementioned use cases. This implies to define the required models and propose a sound and realistic instantiation of this model. Then we plan to expose storage and file abstractions in the other user APIs offered by SimGrid. First, we aim at extending the SimDAG API to allow for the simulation of data-driven scientific workflows. Second, we will consider implementing the MPI-IO subset of the MPI standard as part of the SMPI API.

Acknowledgments

This work is partially supported by the SONGS ANR project (11-ANR-INFRA-13). Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the Inria ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies.

References

[1] M. Snir, R. Wisniewski, J. Abraham, S. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. Chien, P. Coteus, N. Debardeleben, P. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. Munson, R. Schreiber, J. Stearley, and E. Hensbergen, "Addressing Failures in Exascale Computing," *IJHPCA*, vol. 28, no. 2, pp. 129–173, 2014.

[2] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing," *CCPE*, vol. 14, no. 13-15, pp. 1175–1220, 2002.

[3] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *SPE*, vol. 41, no. 1, pp. 23–50, 2011.

[4] A. Núñez, J. Vázquez-Poletti, A. Caminero, J. Carretero, and I. M. Llorente, "Design of a New Cloud Computing Simulation Platform," in *Proc. of the 11th Intl. Conf. on Computational Science and its Applications*, 2011, pp. 582–593.

[5] A. Núñez, J. Fernández, J. D. García, F. García, and J. Carretero, "New Techniques for Simulating High Performance MPI Applications on Large Storage Networks," *Journal of Supercomputing*, vol. 51, no. 1, pp. 40–57, 2010.

[6] C. Carothers, D. Bauer, and S. Pearce, "ROSS: A High-Performance, Low-Memory, Modular Time Warp System," *Journal of Parallel and Distributed Computing*, vol. 62, no. 11, pp. 1648–1669, 2002.

[7] W. Kolberg, P. D. B. Marcos, J. C. S. Anjos, A. K. S. Miyazaki, C. R. Geyer, and L. B. Arantes, "MRSG - A MapReduce Simulator over SimGrid," *Parallel Computing*, vol. 39, no. 4-5, pp. 233–244, Apr. 2013.

[8] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *JPDC*, vol. 74, no. 10, pp. 2899 – 2917, 2014.

[9] T. Johnson and E. Miller, "Performance Measurements of Tertiary Storage Devices," in *Proc. of the 24th Intl. Conf. on Very Large Data Bases*, 1998, pp. 50–61.

[10] O. Sandstå and R. Midtstraum, "Low-Cost Access Time Model for a Serpentine Tape Drive," in *Proc. of the 16th IEEE Symposium on Mass Storage Systems*, 1999, pp. 116–127.

[11] J. S. Bucy, J. Schindler, S. W. Schlosser, G. R. Ganger, and Contributors, "The DiskSim Simulation Environment Version 4.0 Reference Manual," Carnegie Mellon University, Parallel Data Lab, Tech. Rep. CMU-PDL-08-101, 2008.

[12] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar, "Flashsim: A simulator for NAND Flash-Based Solid-State Drives," in *Proc. of the 1st Intl. Conf. on Advances in System Simulation*, 2009, pp. 125–131.

[13] A. Núñez, J. Fernández, R. Filgueira, F. García, and J. Carretero, "SIMCAN: A Flexible, Scalable and Expandable Simulation Platform for Modelling and Simulating Distributed Architectures and Applications," *Simulation Modelling Practice and Theory*, vol. 20, pp. 12–32, 2012.

[14] "The Exascale IO workgroup," http://www.eiow.org/, Oct. 2014.

[15] G. Wang, A. Butt, P. Pandey, and K. Gupta, "A Simulation Approach to Evaluating Design Decisions in MapReduce Setups," in *Proc. of the 17th IEEE/ACM Intl. Symp. on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, 2009.

[16] P. Velho, L. Schnorr, H. Casanova, and A. Legrand, "On the Validity of Flow-level TCP Network Models for Grid and Cloud Simulations," *ACM TOMACS*, vol. 23, no. 4, Oct. 2013.

[17] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, ser. CCIS. Springer, 2013, vol. 367, pp. 3–20.

[18] A. Lebre, A. Legrand, F. Suter, and P. Veyre, "Companion of the article with experimental data and scripts," 2014. [Online]. Available: http://dx.doi.org/10.6084/m9.figshare.1175156

[19] J. Axboe, "Flexible I/O Tester Synthetic Benchmark," http://freecode.com/projects/fio, accessed on April 28, 2014.

[20] S. Camarasu-Pop, T. Glatard, and H. Benoit-Cattin, "Simulating Application Workflows and Services Deployed on the European Grid Infrastructure," in *Proc. of the 13th IEEE/ACM Intl. Symp. on Cluster, Cloud, and Grid Computing*, 2013, pp. 18–25.

[21] F. Desprez, J.-L. Lucas-Simarro, R. Moreno-Vozmediano, and J. Rouzaud-Cornabas, "Image Transfer and Storage Cost Aware Brokering Strategies for Multiple Clouds," in *Proc. of 7th IEEE International Conference on Cloud Computing*, Anchorage, AK, Jun. 2014.

[22] M. Branco, E. Zaluska, D. De Roure, M. Lassnig, and V. Garonne, "Managing Very Large Distributed Data Sets on a Data Grid," *CCPE*, vol. 22, no. 11, pp. 1338–1364, 2010.