Comput Sci Res Dev (2010) 25: 197–205
DOI 10.1007/s00450-010-0121-5

SPECIAL ISSUE PAPER

Collecting energy consumption of scientific data

Energy demands for files during their life cycle

Julian M. Kunkel · Olga Mordvinova · Michael Kuhn · Thomas Ludwig

Published online: 4 August 2010

Abstract In this paper the data life cycle management is extended by accounting for energy consumption during the life cycle of files. Information about the energy consumption of data not only allows to account for the correct costs of its life cycle, but also provides a feedback to the user and administrator, and improves awareness of the energy consumption of file I/O. Ideas to realize a storage landscape which determines the energy consumption for maintaining and accessing each file are discussed. We propose to add new extended attributes to file metadata which enable to compute the energy consumed during the life cycle of each file.

Keywords Storage systems · Energy efficiency · File life cycle · I/O modeling

1 Introduction

In the last years we can see an explosion of the amount of data produced in different scientific fields. Prominent examples are the LHC [2] in nuclear physics, SDSS [1] in astronomy, protein folding simulations [8] in biology or quantum-chemical simulations of large molecular systems [13].

The produced *scientific data* is primarily created either in simulations or it is collected from measurement devices during scientific experiments. Secondarily, it is formed by the analysis results. Additionally, the amount of data is multiplied since experiments are often repeated multiple times in natural sciences to validate the results. Altogether large data sets are written and stored. Thereby one of the distinctive features of scientific data is its versioning, that is, growing with every run, data sets are grouped into releases and stored for further analysis. Similarly, data obtained from scientific equipment—for example, sensors—can be split, saved and analyzed considering the time period of measurements.

Similar to common data the life cycle of scientific data begins with creation and initial storage of the first data set. At the end of its lifetime it becomes obsolete and is deleted. In between,—depending on the specific experiment—data can be read, extended and in some cases be modified. During its life cycle, scientific data can be defined to be *active* or *passive*. Active refers to data which is accessed (read or written) in a given time period.

See Fig. 1 for an example of a file's life cycle. In the example we assume a simulation which iteratively updates the current state and stores the computed results in a file—that is, appends them. The modifications to the file size are shown in Fig. 2. A post-processing (or visualization of the results) requires to read the data and might generate derived results. Later, additional analysis might be conducted with the simulated data. For instance, to compare the old results with newer results or to answer new scientific questions with the recorded simulation results.

Over time, data loses its importance and is accessed less often—gradually losing its value—, and ultimately gets archived (passive data) or disposed of. As a rule, newer data and data that must be accessed more frequently is stored on faster but more expensive storage media, while less critical data is stored on cheaper but slower media. However, the actual usage of data depends on the use case and on the user interaction.

J.M. Kunkel (✉)
Deutsches Klimarechenzentrum GmbH, Hamburg, Germany
e-mail: kunkel@dkrz.de

O. Mordvinova
SAP AG, oCTO TREX, Walldorf, Germany

M. Kuhn · T. Ludwig
Department of Informatics, University of Hamburg, Hamburg, Germany

Table 1 Selection of modern hard disk drives and their energy characteristics. Data is provided by the vendors

Model	RPM	Capacity in GBytes	Power consumption (start) in Watts	(transfer) in Watts	(idle) in Watts	(sleep/standby) in Watts	IOPS	Transfer rate in MBytes/s
Seagate Barracuda 7200.12 (ST31000528AS)	7,200	1,000	24.0	9.4	5.0	N/A	80	125
Western Digital Caviar Green (WD10EARS)	7,200	1,000	20.1	4.9	2.8	0.4	80	111
Western Digital VelociRaptor (WD6000HLHX)	10,000	600	21.6	10.7	8.2	1.3	150	145
Seagate Cheetah 15K.7 (ST3600057SS)	15,000	600	22.9	16.4	11.7	N/A	184	204

Table 2 Selection of modern solid state disk and their energy characteristics. Data is provided by the vendors

Model	Type	Capacity in GBytes	Power consumption (transfer) in Watts	(idle) in Watts	IOPS (read)	(write)	Transfer rate (read) in MBytes/s	(write) in MBytes/s
Intel ® X25-M G2 Postville	MLC	160	0.15	0.08	up to 35,000	8,600	250	100
Intel ® X25-E Extreme	SLC	64	2.6	0.06	35,000	3,300	250	170

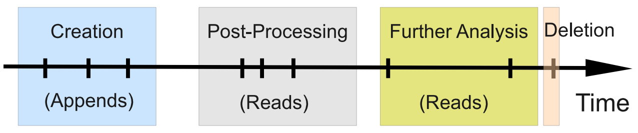

Fig. 1 Example scientific data life cycle

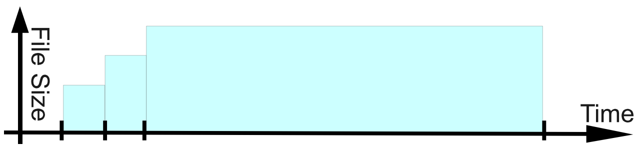

Fig. 2 Example file size during the life cycle

Data life cycle management (DLM) tries to automate and optimize the data life cycle processes according to specified policies and requirements [14]. There are several approaches to handle the challenges associated with DLM of the scientific workflow [4, 5, 14]. However, none of them consider the aspect of energy consumption during the lifetime of data, which is of growing importance since the last years.

File energy consumption depends on several aspects, most significantly storage technology, characteristics of the storage system and access patterns of the applications itself. To show the variability of energy consumption of various storage systems, Tables 1, 2 and 3 show the characteristics of hard disk drives, solid state drives and tapes. The average IOPS for random access are specified. Mount time for tapes is not included, but takes tens of seconds. The MAID[1] architecture evolved into a storage technology to leverage the power consumption of spinning disks, but yet provide a faster response time than tape. Compared to conventional storage systems MAID systems only have about a quarter of disks spinning, reducing the power consumption by the same factor.

A hierarchical storage system migrates data among multiple storage systems—and even technologies—to improve performance and to reduce the storage landscape's total cost of ownership. However, the migration rules are coarse-grained and individual costs of files cannot be accounted for, leading to suboptimal rules and user behavior. Due to the lack of feedback of file handling users are not aware of how to reduce energy consumption in the storage landscape. As a consequence of the only metric being available to the user, the performance, and the coarse-grained accounting, users are not trained to reduce energy consumption of the file life cycle.

A deeper understanding of energy needed during the file life cycle is essential to increase awareness of the energy costs for storing data over a long time. Knowledge of energy costs is also important for the owner of a data center to provide a more accurate accounting model. Ultimately, awareness will lead to improved migration schemes in hierarchical storage systems and save power consumption of the stored data.

[1]Massive Array of Idle Disks.

Table 3 Selection of modern tapes and their energy characteristics. Data is provided by the vendors

Model	Capacity		Power consumption			IOPS	Transfer rate	
	(native) in GBytes	(compressed) in GBytes	(transfer) in Watts	(standby) in Watts	(idle) in Watts		(native) in MBytes/s	(compressed) in MBytes/s
Fujitsu Siemens LTO1 (PRIMERGY Entry)	100	200	18	9	N/A	0.013	16	32
Quantum LTO4	800	1,600	28.8	9.5	6.4	0.018	120	240
Tandberg LTO5	1,500	3,000	24	N/A	6.9	0.014	140	280

Therefore, we introduce a new metric called *TEFL*—the Total Energy for the File Life cycle—and discuss practical methods of how this metric can be computed. After reviewing the latest concepts in the field of data life cycle management and energy efficient storage systems in Sect. 2, we introduce our model concept for computing the TEFL in Sect. 3. In Sect. 4 we discuss the practical realization of the model. The model is applied to an example life cycle in Sect. 5. Finally, Sect. 6 summarizes our work and concludes with an outlook on future development.

2 Related work

Energy consumption of disk drives under particular load has been modeled to estimate and to control power management effectively. In [6] a hardware and energy model is introduced and the effects of power management are discussed. In [9] activity of a single server is traced by a kernel module at the host disk scheduler level. The collected requests, LBA positions (offsets) and I/O size are replayed to estimate the energy consumption.

The grid middleware gLite is extended in [12] to allow accounting of I/O activity. Therefore, access to grid files is intercepted and traced. While the I/O activity and file sizes are taken into account, energy consumption of the file access and idle time are not considered.

In the Amazon Elastic Compute Cloud (EC2 [7]) the Elastic Block Store provides persistent storage capacity. Costs are calculated based on provisioned storage capacity and the number of I/O requests. I/O activity is tracked coarse-grained and not per file.

Dempsey [16] is an extension to DiskSim able to replay I/O traces and estimating energy consumption for the replay. Power management strategies can be implemented and compared by using Dempsey.

Energy efficiency and power consumption of MAID systems is evaluated in [3].

Several approaches exist to utilize the varying characteristics of hardware deployed in a storage landscape. Typically, data is arranged in a hierarchy with different price and performance characteristics. Either the hierarchy is used to cache data of lower levels or to store files on one storage and migrate data between the storage devices as in traditional HSM systems. Recently, flash storage extends the storage hierarchy. Hybrid hard disks and file systems are emerging, which shall utilize these storage efficiently with regards to power and energy. See [10] for an example.

The decision when data should be migrated between storage systems has a major impact on energy efficiency and performance. Current industry solutions like HPSS are policy-based and depend on variables like file size, file extension and duration of the last file access. The administrator must specify the conditions when data should be migrated based on these variables. Automatic decision making—that is, individual migration based on observed access patterns—is an important research subject [15]. However, due to the lack of practical metrics related to energy consumption only performance aspects are taken into account.

3 Modeling energy efficiency

To estimate the energy consumption of data during its life cycle a suitable energy model is introduced as a foundation to discuss implementation alternatives. The model shall allow to quantify the amount of energy consumed for each file individually. In addition, it must be robust against the influence of operations performed on different files in the storage landscape. For example, for a hard disk drive the energy consumption of an I/O operation depends on the position of the read-and-write heads and the block position on the disk. Therefore the order of I/O operations and third-party I/O matters. Also, storage systems might schedule multiple operations at once. While taking these operations into account increases the accuracy of the estimation, a user cannot influence the order of I/O operations when multiple operations are scheduled. A major difficulty is to divide the measured or estimated energy consumption among the concurrently scheduled operations. Therefore, a simpler model which still provides a good and especially fair estimation is required. This analytical model splits the energy consumption of files into *idle* and *active* phases and takes migration and replication into account.

3.1 Idle files

The energy consumption $E_{idle}(S, F)$ of a particular file (F) located on a storage system (S) can be computed with Eq. 1. In other words, the equation shares the idle energy cost among the files located on the storage system. By using this formula empty space on the storage is only accounted to the computing center and not to the user.

During the life cycle the size of F might change. To account for the file properly the average file size over time (integral) can be used to compute the idle costs post-mortem. The modification times are discrete, therefore, by storing a timestamp of the last modification the idle costs can be updated after each modification of the file size, and accumulated (variable $f(F)$). Parallel storage—that is, RAID 0 concepts on parallel file system level—can be treated similar to low level RAID 0, that is, by estimation of the idle energy consumption and storage capacity of the whole storage system in the equation.

$$\begin{aligned} E_{idle}(S, F) &= \frac{\int_{t=0}^{T} s(F,t)}{c(S)} \cdot P_{idle}(S) \\ &= \frac{f(F)}{c(S)} \cdot P_{idle}(S) \end{aligned} \tag{1}$$

The variables and units are defined as follows:

- t is the time. T is the current time.
- $s(F,t)$ is the size of F at a given time, $[s] = Bytes$.
- $f(F)$ is the integrated file size over time, $[f] = Bytes \cdot seconds$.
- $c(S)$ is the capacity of the storage system S, $[c] = Bytes$.
- $P_{idle}(S)$ is the power consumption of S while S is idle, $[P] = W$.
- $E_{idle}(S, F)$ is the energy consumption of an idle file, $[E] = J$.

3.2 Active files

I/O to a file requires additional energy to access the data on the physical storage system and to transfer the information. Here we assume the independence of concurrent accesses to files, that is, the energy consumption is independent of the system's state and only depends on the state of the particular file.

Energy demands for I/O can simply be split into read and write, as shown in Eq. 2.

$$E_{access}(S, F) = E_{read}(S, F) + E_{write}(S, F) \tag{2}$$

For each type of access an individual cost can be defined. For one access type—in this case, read—the energy for one access can be computed with Eq. 3. This formula accounts the energy consumption for one random access (that is, one I/O operation) and the transfer cost for a sequential transfer. The energy for all reads of a file Eq. 4 can be used. Write energy can be computed analogously. For all accesses during the lifetime of the file we sum up the individual energy consumption of each I/O. Overhead for setting up the I/O is added to account for seeking on tapes or disks. Depending on the I/O system energy consumption of seeks could be rated differently.

$$E_{read}(S, a) = P_{read}(S) \cdot \left(\frac{1}{i(S)} + \frac{a}{b(S)} \right) \tag{3}$$

$$\begin{aligned} E_{read}(S, F) &= \sum_{a \in R(F)} E_{read}(S, a) \\ &= P_{read}(S) \cdot \left(\frac{|R(F)|}{i(S)} + \frac{1}{b(S)} \cdot \sum_{a \in R(F)} a \right) \end{aligned} \tag{4}$$

The variables are defined as follows:

- a is the amount of bytes of F accessed, $[a] = Bytes$.
- $i(S)$ is the IOPS of the system, i.e., the number of seek operations per second, $[i] = 1/s$.
- $P_{read}(S)$ is the typical power consumption of the system during the access, $[P_{read}] = W$.
- $b(S)$ is the transfer rate to sequentially access data, $[b] = Bytes/s$.
- $R(F)$ is the set of read operations performed on the file F.

This equation is still true if the I/O subsystem schedules or queues multiple I/O operations at a given time. If we could measure energy consumption of the system directly and multiply it with the access time, or accumulate the measured energy for the time the I/O happens, then concurrently scheduled operations would lead to a longer access time. In this case it would be hard to distribute the observed energy consumption to the performed I/O operations. Therefore, the simple model is used.

3.3 Migration and replication

For replication among multiple systems the energy required is the amount of energy to read the file with the given file size on the source system (S_1) and to write it on the target system (S_2), as shown in Eq. 5. We assume that the I/O system integrates the whole storage landscape and stores metadata only once in a global catalog.

$$E_{Rep}(S_1, S_2, F) = E_{read}(S_1, s(F)) + E_{write}(S_2, s(F)) \tag{5}$$

Migration can be thought of as a replication with an additional cost for deleting the file, as shown in Eq. 6.

$$E_{Mig}(S_1, S_2, F) = E_{Rep}(S_1, S_2, F) + E_{delete}(S_1, F) \tag{6}$$

In modern I/O systems the deletion of files is handled efficiently and thus can be neglected. Tape archives are a special case as deleting a file does not remove the file from the tape immediately. Typically, tapes are compacted by copying the old contents of several tapes—which contain data of deleted files—onto a new tape. Afterwards the old tapes can be reused.

4 Collecting energy consumption in file systems

Existing file systems can be modified to compute the energy consumption of individual files. Several aspects must be considered: accuracy of computed results, performance of the file system, logic and implementation effort, compatibility to existing systems and suitability in storage landscapes.

These aspects are addressed when we discuss where the metrics to compute energy consumption are stored, which values will be stored and how these values are updated.

4.1 Backend store for energy metrics

First, it is discussed how energy values can be stored. As our goal is to quantify the energy consumption over the life time of each file, this cannot be done on the application level. Modifications to the file systems are necessary to accumulate the energy consumption from creation up to the current time.

Information could be kept in external databases, but due to the additionally required infrastructure and induced overhead this seems inappropriate. A close coupling to the file system is needed. One possibility is to keep this information in the file's inode—like `atime`, etc.—, another is to store it in the file's Extended Attributes (EAs). Common local file systems—for example, ext3 and ext4—and parallel file systems like GPFS and PVFS2 support EAs.

EAs are widely used to realize Access Control Lists (ACLs). However, there is a lack of common file system tools supporting extended attributes. Even many basic tools—like `cp`—are still not capable of handling these attributes properly, but it is expected that tools increasingly support EAs. EAs are implemented efficiently, especially due to the increasing use of ACLs. Therefore, we decide to store the energy consumption in EAs as system-specific attributes.

4.2 Update of energy metrics

Each I/O access requires update of the energy-related metrics. Similar to file access time, each read will cause a modification of the file metadata. For example, to avoid the negative impact of metadata modifications due to file access time updates, the access time semantics got relaxed to the relative `atime`. Caching of the metadata and deferring the write-back of the modifications to a file's energy metrics are methods to minimize the impact of maintaining this additional metadata or EAs. In parallel or distributed file systems the values could be either updated on each storage server, or in the global metadata similar to the `atime`.

Modifications to the file size are more difficult to handle as the idle time and energy costs of the storage must be considered. Basically, when the physical file size changes, then the idle energy costs must be recalculated and updated. Consequently, the characteristics—especially the energy consumption—of the storage must be known on the file system level. Figure 2 shows an example of file size modifications during the life-time of an object.

4.3 Metrics to store

There are two possibilities to gather the energy consumption for a file. First, we can store the accumulated energy consumption. Second, basic information can be stored which then can be used to derive energy consumption.

By using the first metric, the stored energy consumption can be updated with every I/O access. A timestamp storing the end time of the last operation corresponds to the idle time. After the I/O is performed the energy consumption of the I/O and the idle energy costs is computed and added to the counter and—finally—the timestamp can be set to the current time. An advantage of using this metric is that it requires only two additional pieces of data to store per file. However, the file system requires information on how to compute these energy values.

A benefit of storing basic information is that we get additional insight about file usage. Therefore, this solution is more versatile for analysis of file operation. Energy consumption can be computed by a user space program at any given time with the equations provided in Sect. 3. Energy characteristics of all storage systems must be known only to the user space application and can be modified. One drawback is that energy consumption of the storage system must not change during the lifetime of the files in this scheme. Upgrades of existing storage systems lead to different energy profiles and thereby cause inaccurate results. Embedded in the file system, the system could update all energy metrics after a system modification occurred, consequently that system would be more accurate.

Production systems are rarely upgraded and the benefit of gathering the access statistics and the independence of the storage system from energy aspects outweigh the inaccuracy, therefore, the basic statistics are a candidate for an implementation.

In our model the number of I/O operations and the amount of accessed data per storage system are required to

```
system.iocount_read          = 10
system.iocount_write         = 50
system.blocks_read           = 1000
system.blocks_written        = 1000
system.file_size_over_time   = 200000                  (Byte * Seconds)
system.last_file_size_update = 2010-05-12 17:00:00.01  (timestamp)
```

Fig. 3 Example extended attributes for one storage system

compute an estimated energy consumption. Due to practical reasons the number of blocks involved in the I/O operation are used. Modifications of file size must be handled by updating the integral of the file size over the life time, in our case *file_size_over_time* is an approximation. At file creation the size is 0. Upon each update of the file size the *file_size_over_time* is incremented by the delta:

$$(current\_time - last\_file\_size\_update) \cdot current\_size$$

Then the *last_file_size_update* is set to the new time and file size can be updated accordingly. One 128 bit counter seems sufficient. Rounding errors will occur, but the precision in bookkeeping the time can be improved by using milliseconds instead of seconds when necessary.

For a single storage system the entries are as simple as shown in Fig. 3. Keys and example values for the extended attributes are provided.

In a storage landscape with migration and replication of files the situation is more complicated. Consequently, each storage system needs a unique identifier in the storage landscape to maintain correct information after migrations. Additionally, to calculate idle energy consumption correctly, counters are needed to accumulate the time the object resided on each storage system over the whole life time of the object. Consequently, the extended attributes shown in Fig. 3 are stored for each storage system and prefixed with their unique identifier. Once an object gets replicated the time must be updated accordingly. It must be updated once the file is removed/migrated again from the file system. The described mechanism can be directly embedded into the file size update mechanism by changing the size of the file on the migration source to 0 and to the current file size on the target.

With this modifications multiple migrations and replications between file systems are accounted for properly. Furthermore, a fixed number of replicates per file system can be handled by knowing the replication policy of each the storage system.

5 Example scenarios

An theoretical assessment of the solution is performed by calculating the derived metrics for one scientific data file on two distinct hardware scenarios.

5.1 Access pattern

Consider an iterative application which appends 10 GBytes of data to a file every 10 minutes. For instance the application could be a climate simulation which runs for 1,000 minutes, that is, performs 100 iterations. In total a single file of 1 TByte is created. After completion of the program, in a post-processing phase, data is extracted, post-processed and 100 minutes later the whole data set is visualized—that is, data is read completely two times. Then—at the end of the post-processing phase—the data is migrated to tape.

In a phase of further analysis the data is requested one year later to perform new analysis and compare the simulation run with new data. Between each run 100 minutes elapse. Therefore, data is staged from tape to disk and read sequentially two times by various applications. Then the data on disk is deleted immediately.

5.2 Storage landscape: online storage and tape archive

In this scenario a hierarchical storage system can migrate data between a disk storage and a tape archive. The file system consists of 2,080 Western Digital Caviar Green hard disks embedded in 130 shelves of 16 disks. For redundancy, each shelf uses RAID-5 and is mirrored. Capacity of the online storage is about 1 Petabyte for disks. The tape archive is a StorageTek SL8500 equipped with 10 tape drive trays and a maximum capacity of 10,000 tapes.

Qualitative behavior of the extended attributes for the access pattern are shown in Fig. 4. The time for a read and write operation is not taken into account in the diagram. In this figure the metric *file size over time* is provided for each timestamp. With our scheme it is computed only during file size updates—marked with *X* in the figure and during the append phase. I/O on disk is performed in blocks of 10 MBytes. On tape the whole file is copied in one access.

Results are compared between the disks and the same number of shelves equipped with Intel X25-E Extreme flash drives. In this case the overall capacity is lower (64 TByte), though.

Characteristics of the two storage systems are provided in Table 4. Hardware characteristics were inspired by vendor information from Storage Tek—for example, see [11]. Overhead for building the infrastructure, in particular for shelves, interconnects, RAID controllers are estimated and added to the energy costs to power the parallel file system. Additionally, the IOPS and transfer rate of the overall system are reduced due to overhead. Due to the theoretical nature, an estimation of the real power consumption and performance is sufficient. For instance, an energy consumption of 100 Watts per shelf is assumed. A compression rate of 50% for tape archival is used. Performance of the overall

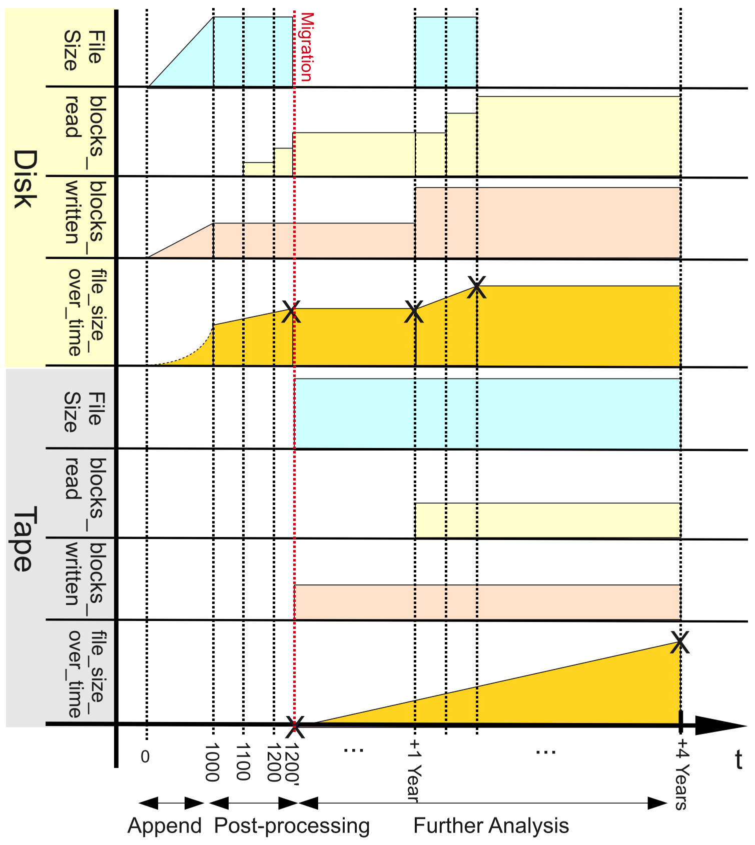

Fig. 4 File attributes for the scenario with migration between tape and disk (qualitative view)

Table 4 Hardware characteristics of the storage components with the estimated energy consumption during operation (active estimate) and qualitatively considered overhead of enclosures, controllers and metadata handling

Storage system	Power consumption			IOPS		Transfer rate	
	(active max) in Watts	(active estimate) in Watts	(idle) in Watts	(read)	(write)	(read) in MBytes/s	(write) in MBytes/s
Hard disk	9.4	10	5	80	80	125	125
Parallel file system (with disks)	90,000	40,000	24,000	60,000	60,000	15,000	15,000
Flash disk	2.6	3	0.06	35,000	8,600	250	170
Parallel file system (with flash)	75,000	24,600	13,132	10,000,000	2,500,000	30,000	20,000
LTO tape drive	53	53	35	0.01	0.01	210	210
Tape archive	4,000	1,200	800	0.10	0.10	2,100	2,100

system does not match the aggregated performance of all disks. For example, disk enclosures and controllers cache data, but RAID-5 (and RAID-1) reduce concurrency. Additional handling of the file system degrade performance as well.

Quantitative attribute values of the access pattern are provided in Table 5. The amount of accessed data, the number of performed I/O operations and the *file size over time* depend on the access pattern. Computed energy consumption depends on the hardware infrastructure and are computed with the model presented in Sect. 3 and the hardware characteristics from Table 4. The two scenarios, online storage provided by disk or flash are shown.

Surprisingly flash idle costs are much more expensive due to the lower capacity. Busy costs are lower by a factor of three, because flash is faster. In this scenario (disk and tape) the TEFL is 23.3×10^6 Joules. With an energy cost of 0.20 € per kWh the total cost is 1.30 €.

Table 5 Energy and I/O calculation for the scenario with migration between online file system and tape archive (the values of the file attributes are specified for each phase)

	Processing phase	Accessed data		I/O operations		File size over time in Bytes × s	Energy consumption in Joules			
		(read) in Bytes	(write) in Bytes	(read)	(write)		(disk & tape)		(flash & tape)	
							Idle	Busy	Idle	Busy
Online	Append	0	1×10^{12}	0	1×10^5	30.3×10^{15}	0.7×10^6	2.6×10^6	6.2×10^6	1.2×10^6
	Post-processing	3×10^{12}	1×10^{12}	3×10^5	1×10^5	42.3×10^{15}	1.0×10^6	10.4×10^6	8.7×10^6	3.5×10^6
	Further analysis	5×10^{12}	2×10^{12}	5×10^5	2×10^5	54.3×10^{15}	1.3×10^6	18.3×10^6	11.1×10^6	6.3×10^6
Tape	Append	0	0	0	0	0	0	0	0	0
	Post-processing	0	1×10^{12}	0	1	0	0	0.6×10^6	0	0.6×10^6
	Further analysis	1×10^{12}	1×10^{12}	1	1	15.8×10^{19}	2.5×10^6	1.1×10^6	2.5×10^6	1.1×10^6

Table 6 Energy and I/O calculation for the disk in the storage landscape consisting of only one online storage system

Processing phase	Accessed data		I/O operations		File size over time in Bytes × s	Energy consumption in Joules			
	(read) in Bytes	(write) in Bytes	(read)	(write)		(disk)		(flash)	
						Idle	Busy	Idle	Busy
Append	0	1×10^{12}	0	1×10^5	30.3×10^{15}	0.7×10^6	2.6×10^6	6.2×10^6	1.2×10^6
Post-processing	2×10^{12}	1×10^{12}	2×10^5	1×10^5	42.3×10^{15}	1.0×10^6	7.8×10^6	8.7×10^6	2.7×10^6
Further analysis	4×10^{12}	1×10^{12}	4×10^5	1×10^5	15.8×10^{19}	3.8×10^9	13×10^6	32.4×10^9	4.3×10^6

5.3 Storage landscape: online storage

In this scenario data is kept online—on disk or flash, the whole time. Table 6 shows the statistics in this case. Compared to the scenario with tape the energy costs are much higher. In detail the TEFL is 3.8×10^9 Joules, leading to a cost of 211 €.

6 Summary and conclusions

In this paper a model to compute and estimate the energy consumption of a file during its life cycle is introduced. This model shares the costs of the storage fairly among all files, depending on the size of the files themselves. Several design issues of how the file system could be modified to provide required information are discussed. For example, additional statistics about file accesses can be kept in extended attributes. In two scenarios the statistics and energy consumption are computed to provide additional insight on how they behave.

Concluding, extending a file system to account I/O accesses is technically possible and allows to estimate the TEFL. This increases awareness of energy consumption of scientific data and enables optimizations of storage landscapes and hierarchical storage systems to optimize total energy for the file life cycle.

7 Future work

In the near future we will modify a local file system to account for file access statistics and supply tools to derive energy consumption by using the access statistics. Obtained results will be compared to directly measured energy consumption by precision power meters from ZES Zimmer Electronics, which are available in our research group. Embedding these methods into the parallel file system GPFS and HPSS is considered to provide insights into user activity of the DKRZ and to estimate energy consumption of the files in our data center in order to optimize migration policies between GPFS and HPSS.

Acknowledgements Thanks to Manuela Ernst and Timo Minartz for their contributions.

References

1. Astrophysical Research Consortium (2010) The sloan digital sky survey. http://www.sdss.org/
2. CERN (2008) The large hadron collider. http://public.web.cern.ch/public/en/LHC/LHC-en.html
3. Colarelli D, Grunwald D (2002) Massive arrays of idle disks for storage archives. In: Supercomputing '02: proceedings of the 2002 ACM/IEEE conference on supercomputing. IEEE Computer Society, Los Alamitos, pp 1–11
4. Deelman E, Chervenak A (2008) Data management challenges of data-intensive scientific workflows. In: CCGRID '08: proceedings of the eighth IEEE international symposium on cluster computing and the grid. IEEE Computer Society, Los Alamitos, pp 687–692

5. Gil Y, Deelman E, Ellisman M, Fahringer T, Fox G, Gannon D, Goble C, Livny M, Moreau L, Myers J (2007) Examining the challenges of scientific workflows. Computer 40(12):24–32
6. Greenawalt P (1994) Modeling power management for hard disks. In: The conference on modeling, analysis, and simulation of computer and telecommunication systems, pp. 62–66
7. Hazelhurst S (2008) Scientific computing using virtual high-performance computing: a case study using the Amazon elastic computing cloud. In: SAICSIT '08: proceedings of the 2008 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries. ACM, New York, pp 94–103
8. Kuntz SK, Murphy RC, Niemier MT, Izaguirre JA, Kogge PM (2001) Petaflop computing for protein folding. In: Proceedings of the tenth SIAM conference on parallel processing for scientific computing
9. Molaro D, Payer H, Le Moal D (2009) Tempo: disk drive power consumption characterization and modeling. In: Consumer electronics. IEEE Computer Society, Los Alamitos
10. Nijim M, Manzanares A, Ruan X, Qin X (2009) HYBUD: an energy-efficient architecture for hybrid parallel disk systems. In: ICCCN '09: proceedings of the 18th international conference on computer communications and networks. IEEE Computer Society, Los Alamitos, pp 1–6
11. Oracle (2010) StorageTek SL8500—power calculator. http://www.sun.com/calc/storage/tape_storage/tape_libraries/sl8500/index.html
12. Scibilia F (2007) Accounting of storage resources in glite based infrastructures. In: WETICE '07: proceedings of the 16th IEEE international workshops on enabling technologies: infrastructure for collaborative enterprises. IEEE Computer Society, Los Alamitos, pp 273–278
13. Steinke T (2000) Tools for parallel quantum chemistry software. In: Modern methods and algorithms of quantum chemistry. NIC series, vol 1. John von Neumann Institute for Computing, Jülich, pp 49–67
14. Valle M (2004) Scientific data management. http://www.cscs.ch/mvalle/sdm/scientific-data-management.html
15. Vengerov D (2008) A reinforcement learning framework for online data migration in hierarchical storage systems. J Supercomput 43(1):1–19
16. Zedlewski J, Sobti S, Garg N, Zheng F, Krishnamurthy A, Wang R, Wang O (2003) Modeling hard-disk power consumption

Julian M. Kunkel received his M.Sc. degree in computer science at the University of Heidelberg in 2007. Currently he is employed at the German High Performance Computing Centre for Climate- and Earth System Research. In his leasure time he works on his Ph.D. Interests cover high performance file systems and modeling of cluster systems' performance.

Olga Mordvinova is employed at SAP AG in Walldorf and works on her Ph.D. in the area of distributed storage. Her professional interests cover storage and data management solutions for data-intensive applications, especially on the field of business intelligence.

Michael Kuhn received his M.Sc. degree in computer science at the University of Heidelberg in 2009. Currently, he is employed at the University of Hamburg and works towards his Ph.D. His research interests are in high performance input/output, file systems and distributed systems in general.

Thomas Ludwig became Professor at the Ruprecht-Karls-Universität Heidelberg and lead the research group Parallel and Distributed Systems in 2001. Since 2009 he is Professor at the university of Hamburg and CEO of the German High Performance Computing Centre for Climate- and Earth System Research. His major research interests are high performance storage and energy efficiency in HPC.