PaN-data ODI

Deliverable D7.3

D7.3: Mechanisms and tools for integrity of Datasets.

Grant Agreement Number	RI-283556
Project Title	PaN-data Open Data Infrastructure
Title of Deliverable	Mechanisms and tools for integrity of datasets
Deliverable Number	D7.3
Lead Beneficiary	STFC
Deliverable Dissemination Level	Public
Deliverable Nature	Report
Contractual Delivery Date	01 January 2014 (Month 21)
Actual Delivery Date	16 Oct 2014

The PaN-data ODI project is partly funded by the European Commission under the 7th Framework Programme, Information Society Technologies, Research Infrastructures.

Abstract

How to validate integrity of Datasets from the data production to the end user analysis.

Keyword list

Data integrity, corrupted data, silent data corruption, checksum algorithms, sha1, md5, adler32, bit errors.

Document approval

Approved for submission to EC by all partners on 16.20.2014

Revision history

Issue	Author(s)	Date	Description
1.0	Jean-François Perrin	30th January 2014	Initial release
1.1	Fabien Pinet	28th February 2014	Algorithms evaluation
1.2	Jean-François Perrin	14th October 2014	Production deployment & Integration of feedback

Table of Contents

1 WHY CONSIDER DATA INTEGRITY?

Digital data is essential to modern Science, integrity of such data is vital. Corrupted data could lead to a vast waste of time and loss of credibility for researchers, it is our role as data producers and an archive centre, to ensure integrity of the data produced and stored in our research Infrastructure or at least provide the tools for identifying corrupted data as soon as possible in the scientific workflow.

When we started this project the first difficulty we encountered was the general feeling that data integrity is not an issue any more. Maybe you the reader have this impression that data corruption has disappeared, you may think about fraud or intentional corruption but what about technical or accidental data corruption?

20 years ago, checksums, algorithms for detecting or even correcting accidental data corruption, were present in most professional electronic solutions. As an example, at that time, they were present in the electronic hardware of detector or data acquisition at ILL, today, considering the low level of errors and the necessity to go fast, the search for millisecond or even nanosecond improvements, there is a tendency to remove or implement them very partially. This is the same situation for network devices, file systems … "data volume is too large", "disks are much more reliable", "modern raid systems take care of integrity".

The situation is not going to improve, with the current economic situation and the volume of data growing exponentially (i.e. Big Data), we, IT managers, are under very strong budgetary pressure and therefore are moving from high end, dedicated solution, to low cost solutions where data integrity protection is almost always not implemented natively.

The following chapters will analyse more deeply the data life cycle, technical problems and try to recommend solutions.

2 TYPES OF DATA CORRUPTION

2.1 TECHNICAL DATA CORRUPTION

The two causes of technical data corruption are the storage stack and the network transfer, this includes hardware and software components.

2.1.1 Storage stack

There are different types of data corruption, they have been studied in academic papers[1][2] and evaluated in a similar scientific environment by our colleagues from CERN IT [3], where during a test 33700 files were checked (~8.7 TB) and 22 errors found (an error rate of one bad file over 1500 files).

Our intention in this paper is not to thoroughly detail and explain such errors, which could necessitate a solid technical background in this field, but rather to let the reader know that they still exist and that some of them are even not detected by ECC and other partial checksum mechanism.

Those errors come from disk and memory physical errors, RAID systems and software including firmware errors. Most of them are single bit and silent errors, they are not always noticeable by end users, but they can lead to files being completely unreadable if for instance a file was previously compressed.

The most global and simplest approach is to ensure that the file system takes care of data integrity, currently only few of them really implement a complete data integrity monitoring solution, the most used in our community being ZFS and WAFL.

As a side note, an industry standard (T10-DIF) on data integrity exists since 2003[4] and was recently extended[5] (T10-P1) to cover the full storage stack (OS kernel, HBA, FC switchs, disk arrays and disk drives). They consist in extending the typical block size by some bits for storing the checksum alongside the data, in order to be verified by the different component of the stack. Such standards represent a very interesting approach but are currently only implemented by few proprietary solutions.

2.1.2 Network transfer

Like for the storage stack, data can easily be corrupted during the transfer, again, like ECC for memories, IP protocol implements a checksum on an individual frame, but errors could occur at a higher level of the stack where global corruption detection mechanism does not exist or exists only partially (not catching all possible types of errors).

---

[1] Bairavasundaram et al. "An Analysis of Data Corruption in the Storage Stack*." Proceedings of the 6th USENIX conference on File and Storage Technologies (FAST'08)*. February 2008

[2] Jiang et al. "Are Disks the Dominant Contributor for Storage Failures*?" Proceedings of the 6th USENIX conference on File and Storage Technologies (FAST'08)*. February 2008

[3] Panzer-Steindel. "Data Integrity." *Internal CERN/IT study*. 8 April 2007 (http://indico.cern.ch/getFile.py/access?contribId=3&sessionId=0&resId=1&materialId=paper&confId=13797)

[4] Keith Holt (July 1, 2003). "End-to-End Data Protection Justification". *T10 Technical Committee document*

[5] EMC Corporation (September 18, 2012). "An Integrated End-to-End Data Integrity Solution to Protect Against Silent Data Corruption". *White paper*. Oracle Corporation.

As a matter of example, data transferred by end users is often achieved using the HTTP protocol. A standard for data integrity using this protocol exists (RFC 1864 - The Content-MD5 Header Field) and is implemented in web servers (Apache[6]) but not in standard clients, i.e. web browsers (Firefox, Chrome, Internet Explorer, Safari …) and therefore not used. The noticeable exception is the S3 browser from Amazon, this is probably in response to the data corruption they experienced in 2008[7], where data corruption was generated by faulty load balancers during transfers.

HTTP is just an example, but the network stack is complex, even if TCP/IP provides a mechanism for checking and recovering data integrity at the lower level, data crosses billions of lines of code before reaching a final destination, every bug or misconfiguration (for instance transfer of binary data in text mode using FTP) in this stack could cause data corruption. This corruption will remain silent due to lack of mechanisms for ensuring data integrity.

2.2 INTENTIONAL CORRUPTION

Intentional data corruption exists, this is often the cause of retracted scientific publication[8] and even if the rate is low, very simple measures could probably be implemented in order to prevent such issues.

Analysing public cases of publication retraction where data corruption is concerned, like for instance in the well-known Schön case[9], raw data are simply not available when experts try to validate results. We were not able to identify a single case in scientific publications where data corruption is present and where a dishonest scientist has tried to forge digital data integrity in order to enforce his statements.

Even if the simplest checksum algorithms have been proven to be vulnerable to collisions (a collision is when two distinct files produce the same checksum result), it will be extremely difficult to forge meaningful checksums produced by simpler algorithms like CRC32 or Adler, when these checksums are generated alongside the experimental data and archived by the facility. The person will have not only to produce meaningful collisions but also bypass the IT security measures of the facility.

Producing collisions on small words of few characters using CRC or Adler checksum algorithm families or even the MD5 cryptographic hash algorithm is relatively feasible, exploiting such weaknesses on larger files, although it has been done in specialised laboratories[10], is extremely difficult for a non-specialist of cryptology.

[6] http://httpd.apache.org/docs/2.4/mod/core.html#contentdigest
[7] https://forums.aws.amazon.com/thread.jspa?threadID=22709#
[8] http://retractionwatch.com/
[9] http://en.wikipedia.org/wiki/Sch%C3%B6n_scandal
[10] http://web.archive.org/web/20071226014140/http://www.cits.rub.de/MD5Collisions

3 EXPERIMENTAL DATA WORKFLOW

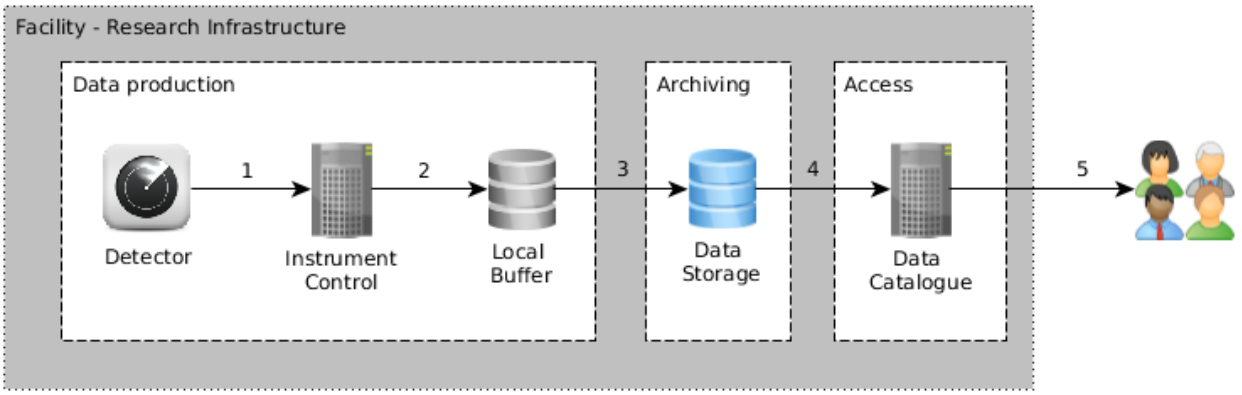


Figure 1 Typical data flow taking place in the analytical facilities.

During the production phase, experimental data are produced by the detectors installed on the instruments. Data are received by instrument control software (1), converted into the target output format and put temporarily into a local buffer (2) onto the instrument machine, in order to ensure their initial persistence and avoid introduction of latency due to network storage. Once locally recorded, the data are archived (3) by being copied in an asynchronous process into the network data storage, which guarantees its long-term preservation. Finally, the arrival of new files into the data storage triggers the process of their import into the data catalogues (4), which makes the data available for internal and external users.

We would like to be able to produce checksums of the raw data at the facility, ideally as soon as possible in this workflow (Figure 1) but without disturbing the data acquisition process and eventually integrate this information into the data portal so that users could easily validate that the files they have transferred have not been corrupted.

4 EXISTING SOLUTIONS

Many algorithms exist in this field, they fall essentially into two categories: simple hashes (Cyclic Redundancy Check, Adler …) and cryptographic ones (MD5, SHA, Tiger, WHIRLPOOL …). As it will be presented in the next chapter, the simple hashes present the benefits of performance, but at the cost of weak protection against collisions and the cryptographic ones are more robust to collision but slower to compute the hash.

Another aspect which is also important is the availability and popularity of software that implements these algorithms, the ultimate goal of the project is that users could verify themselves the integrity of the datasets when they perform analysis. Linux and Mac OS integrate natively a tool (command line) for computing MD5 or SHA1 hash of files, but not Microsoft Windows. Nevertheless software are available for the main three analysis platforms (Microsoft Windows,

Linux and Mac OS). Currently MD5 is by far the most popular and very well known, but due to its weakness the situation might change in the future.

5 ALGORITHM EVALUATION

The goal of this evaluation is to understand the real performance of these algorithms on a modern but standard computer with typical neutron scattering data.

5.1 TYPICAL NEUTRON SCATTERING DATA PROFILE

Typical experimental neutron scattering data are relatively small (except for nuclear physics experiments) and store over thousands of small files. Figure 2 presents the distribution by file size of the 1st 2013 cycle, half of the files are smaller than 94KB.

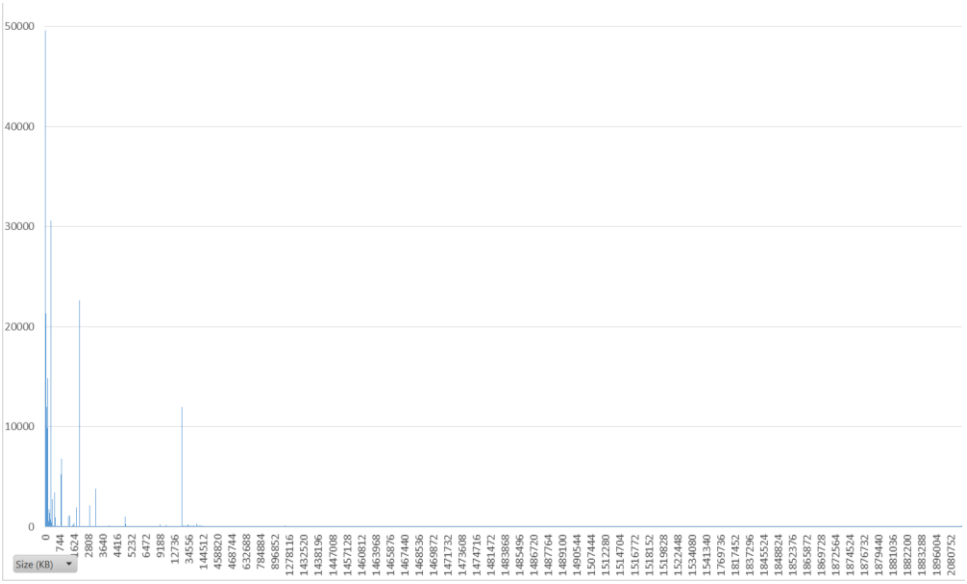


Figure 2 Distribution of file per size

5.2 RESULTS OF THE TESTS

The tests have been run on a modern (Intel Core I5 – 64bits architecture – 8GB of RAM) but standard (office type) computer. We tested the algorithms using three different programs:

- jacksum [11], a well-known, free and platform independent software that supports 58 algorithms.

[11] http://www.jonelo.de/java/jacksum/

- A python script written for the purpose of this test that uses the python native library *zlib* for CRC and Adler32 and the python native library *hashlib* for MD5 and the SHA family type algorithms.
- A C program which implements only the Adler algorithm. This was done mainly in order to see if we can improve the performance of it for production purposes.

Tests have been run five times each in order to ensure sufficient consistency of the results. The computer has been rebooted between tests in order to avoid cache mechanisms effects.

The following table presents the python script results in terms of the time spent (for this exercise we don't need information on the CPU usage or I/O waits) for the three most interesting data file sizes:

- 94 KB represents the median file size of ILL data files
- 80 MB represents the mean size value of ILL data files
- 3.8 GB represents the largest ILL data files.

	94 KB			80 MB			3.8 GB		
	Read time	Compute time	Total	Read time	Compute time	Total	Read time	Compute time	Total
CRC32	0.000036	0.000088	0.000124	0.428	0.061	0.489	19.06	2.86	21.92
ADLER	0.000036	0.000032	0.000068	0.428	0.025	0.453	19.06	1.17	20.22
MD5	0.000036	0.00015	0.000186	0.428	0.113	0.541	19.06	5.11	24.16
SHA1	0.000036	0.000141	0.000177	0.428	0.100	0.527	19.06	4.69	23.75
SHA256	0.000036	0.000344	0.000380	0.428	0.273	0.701	19.06	12.90	31.96
SHA512	0.000036	0.000227	0.000263	0.427778	0.175	0.603	19.06	8.22	27.28

Table 1Results of algorithms evaluation on typical ILL raw data files - results expressed in seconds

5.3 OBSERVATIONS

In the following presentation of the results we will leave out SHA256 because it is less secure and less performant than SHA512. The reason for this difference is that the SHA256 algorithm was created for 32bit operating systems whereas SHA512 came later on and took advantage of the 64bit architecture[12]. We ran the tests on a 64bit system which is currently the standard.

[12] https://community.emc.com/community/edn/rsashare/blog/2010/11/01/sha-2-algorithms-when-sha-512-is-more-secure-and-faster

5.3.1 Small data files – Typical neutron data files

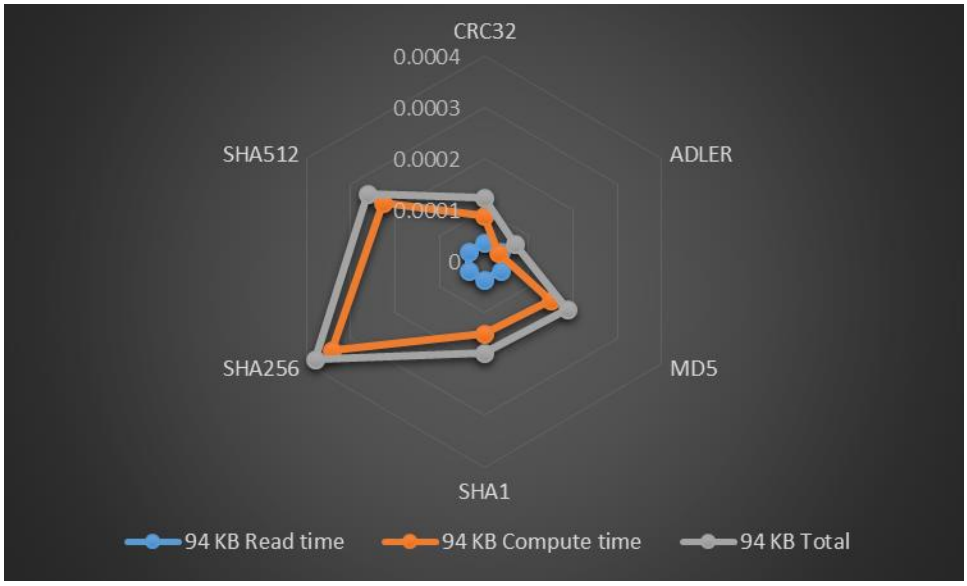
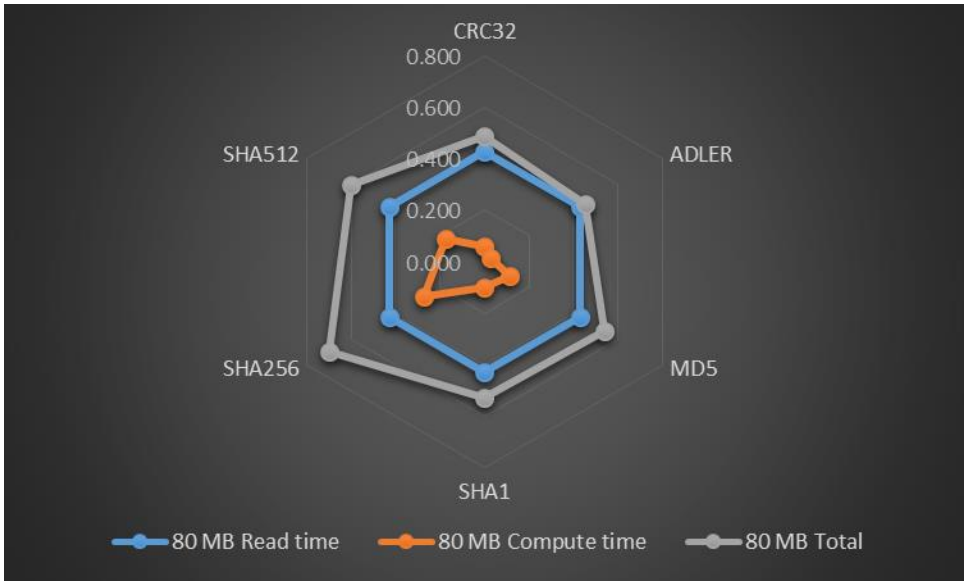


Figure 3 Typical ILL raw data file results

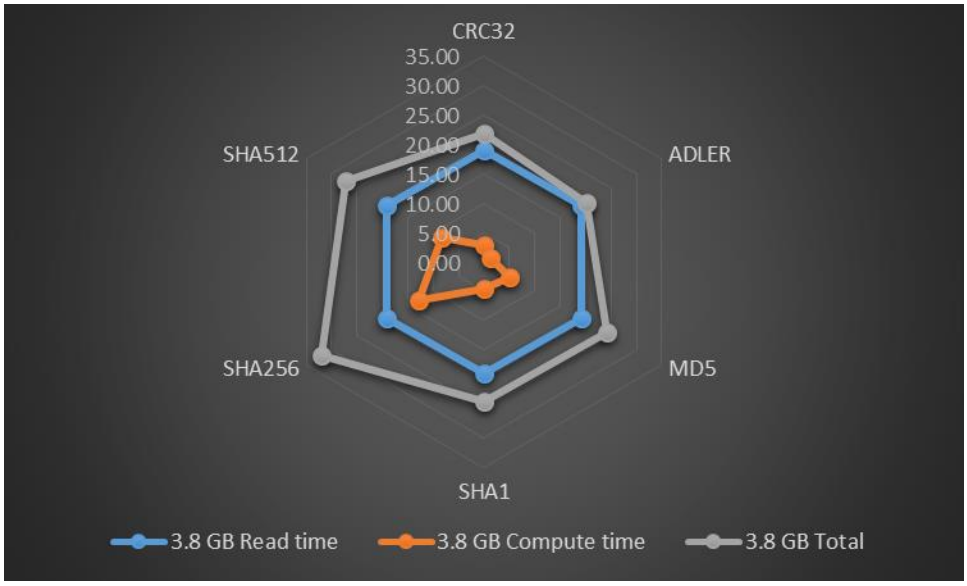
On small files ADLER is the quickest algorithm (68 µs). Most of the time is spent on computing the hashes. The total processing time is in the order of 100s of µs.

5.3.2 Medium files



Here the time necessary to read the file takes precedence over the time necessary to compute the hashes.

5.3.3 Largest files



Large files scenario is similar to the medium one, this tendency is highlighted on the Figure 4. Independently of the algorithm chosen the percentage of computational time over the total time remains the same for files larger than 80MB.

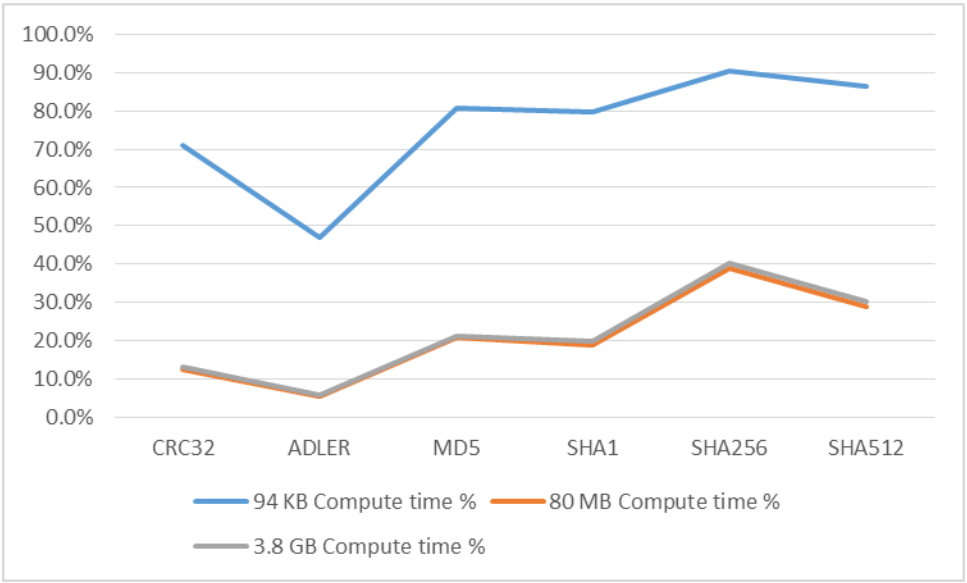


Figure 4 Percentage of compute time over total time.

5.4 CHOICE OF THE ALGORITHM.

If we were only looking for speed, Adler would have been the best choice, if we were only looking for security SHA512 would have been the best option. We chose MD5 as a compromise where security, especially in the context of the archive being managed by the facilities, and performance

are acceptable. The choice was largely driven by the fact that today MD5 is the most well-known system and shouldn't represent a barrier for the usage by scientists.

6 PRODUCTION SET UP AT ILL

We wanted to generalise for all of the ILL instruments this integrity control and to introduce the initial checksum of the generated data file as soon as possible in the workflow described by Figure 1. Unfortunately due to the diversity of detectors on one side and the quest for very high performance on the acquisition control on the other side, we finally introduced it at step 3 of the process (i.e. when the files reach the archive server).

Files are transferred from the acquisition control computer to the archive using the rsync protocol. In this initial deployment, we created a python script which simply monitors the logs of the rsync server and starts the MD5 checksum as soon as a new file has been transferred. This checksum is then recorded in a database and added to the manifest file stored in the folder of the data files in the archive.

The manifest, which is simply a catalogue of key value pairs (name of the file – checksum), is created mainly for the users who want to validate the integrity of the whole datasets. Tools like md5sum available on most Linux computers are able to process them automatically. Running 'md5sum -c' compares the MD5 hash value of each file listed in the manifest file with the computed value of the file hash. If data integrity of the archive has been ensured, all files should be listed in the output with 'OK'.

The database record is created for two purposes, firstly in order to present hashes on the web interface of the data portal and secondly for automated regular checks of the archive files. Here is a small example of the content of the database.

ID	FILENAME	CHECKSUM_MD5	DATE	CYCLE_ID	INSTRUMENT_ID	PROPOS_ID
285	233895	d41d8cd98f00b204e9800998ecf8427e	01/08/14 10:33:54	133	92	590
286	233896	d41d8cd98f00b204e9800998ecf8427e	01/08/14 10:33:54	133	92	590
287	233897	d41d8cd98f00b204e9800998ecf8427e	01/08/14 10:33:54	133	92	590

This setup will be modified by the end of 2014 in order to introduce an additional check on the instrument control machine therefore we will move the initial verification from step 3 to step 2 of the diagram Figure 1.

7 CONCLUSION AND RECOMMENDATION

This work was really useful in order to raise the general awareness of the community regarding the reliability of the storage and network processes involved in the data production and archiving.

We could only recommend to put in place mechanisms that allow verification of the data integrity all along the workflow.

A solution does not exist out of the box, but Adler32 is a good candidate for technical checks as long as security is not a major concern (i.e. ensured by other means). With the increasing volume of data, this is probably the only real solution for large data producers. On the user side only MD5 or SHA1 are really easily accessible, consequently we need to provide at least either MD5 or SHA1 hashes. For small volumes of data MD5 (or SHA1) is probably the easiest choice, in case of large volumes a hybrid solution (Adler for the technical checks at the facility premises and MD5 for user downloads) could suit our use case.

On large files, computing hashes occupies only 10 to 40% of the totally necessary time for producing the check, the rest is only reading bytes from disk. For research infrastructures producing such files at a high pace, the solution is probably to try to integrate this hashing process when files are already in the memory of the systems for other purposes and avoid the read performance cost.