Upscaling the MPT

Visualizing the Performance Impact from Application Configuration

|  |  |
| --- | --- |
| **Peter May** | **Kevin Davies** |
| *British Library**London, UK**Peter.May@bl.uk**https://orcid.org/0000-0001-8625-9176* | *British Library*Boston Spa, Wetherby*, UK**Kevin.Davies@bl.uk**https://orcid.org/0000-0001-6522-9568* |
|  |  |

**Abstract – The Minimum Preservation Tool (MPT), developed by the British Library, provides a local technical digital preservation environment to routinely fixity-check collections awaiting ingest into a long-term digital repository. Within the Library this was deployed on a standard-provision Virtual Machine using the same SHA-256 hash function as our long-term digital repository. This operates effectively for the collections currently under MPT control, but to be confident managing larger and more varied collections, we look to understand how performance can be improved, for example through use of different hash functions or through greater parallelization. This poster outlines experimental findings exploring the effect on performance of four different hash functions and four sizes of parallel processes, across three broad corpora (large, mixed, and small file-sizes).**

**Keywords – Minimum Preservation Tool, Performance, Checksum**

**Conference Topics – Innovation**

# Introduction

Trustworthy digital repository systems are a crucial component to maintaining long-term access to authentic digital content. Content obviously has to be deposited into such systems for them to perform, however the Library’s experience has been that there is often a delay between acquiring content and ingesting it into a long-term repository. In this interim period, backing up content helps replicate the data, somewhat securing content at the bit-level, but does not necessarily prevent, detect, or raise awareness of bit-level corruption.

In order to safeguard this ‘interim’ digital material, the Digital Preservation Team at the British Library have developed the Minimum Preservation Tool (MPT) [1, 2] to provide basic integrity checking across replicated interim data-stores. This open-source tool provides simple checksum generation, validation and cross-data-store comparison, combined with a reporting mechanism for each of these functions.

We have deployed MPT to several collections of various sizes, using a Library standard-provision virtual machine (VM) and the SHA-256 hash function to generate checksums. These choices where based on what could easily be provisioned and what hash functions are used within existing workflows.

During the time we have been using MPT under this setup it has performed effectively. We have gained experience surrounding the execution times and scheduling needs to service the collections in care. Staggering checksum validation tasks so that larger collections are validated on alternating weeks is one example of this. But as further collections are identified for MPT control, and as the amount of data to be protected increases, the question has arisen of how MPT’s performance can be improved. Code analysis can be undertaken to look for efficiencies, and the VM configuration could be enhanced, but what efficiencies can be gained due to application configuration? Could we increase parallel processing of files? Or could changing the hash function used to generate checksums improve performance?

With this in mind, we initiated an internal project to look at how to upscale the MPT service. Broadly this covers two main areas: 1) investigations in a test MPT environment to understand the impact of virtual machine and application configuration on performance; and 2) to understand the deployment and scheduling benefits afforded by containerizing the service (this latter work has yet to start).

# Investigations

This poster will focus on two application configuration investigations undertaken: 1) the effects of hash function choice on MPT checksum validation performance; and 2) the effects of the number of parallel processes on processing time.

## Environment Setup

Setup required creating a suitable test environment - a virtual machine, provisioned in alignment with standard VMs supplied by our IT department (4 x logical AMD Opteron 6276 cores, 8 GB physical memory, Windows Server 2019), with locally attached storage for test data.

In terms of test data, as the Library handles many different digital collections with varying makeup of files, we wanted to understand the overall performance across broad categories of collections. We generated three file corpuses of up to 1TB each – Large (262 files, >4GB/file avg.), Mixed (21k files, 8.5MB/file avg. (s.d. 116MB)), and Small (3.8m files, 138KB/file avg.) – each representing a broad variety of the collections held by the Library.

## Hash Function Choices

MPT uses the standard Python hashlib library to generate checksum digests, which provides support for most commonly-used cryptographic hash functions. Of these SHA-256 was chosen as the benchmark function due to its common use in the Library already, MD5 was chosen as another frequently used algorithm, and finally BLAKE2 was selected as it showed performance improvements over SHA-256 and MD5 [3].

Non-cryptographic hash functions are typically faster [4] and, given the nature of the MPT process is to detect changes to file bit streams, such algorithms were considered acceptable. The XXHASH algorithm [5] was selected as it is considered the fastest [6], but implemented in MPT through another library [7].

Algorithms selected, checksums were generated for all files in each corpus using the test environment. Timing information was provided by the MPT reporting mechanism.

## Number of Parallel Processes

MPT supports parallel processing using Python’s multiprocessing module. Each hash function was tested using 8, 16, 24 and 32 parallel processes in an attempt to find the optimal point where the balance of CPU usage versus disk response time provides the lowest overall processing time. Experiments were again performed using the selected algorithms on the test environment across all three corpora.

# Poster

The poster will present an overview of our on-going investigations and findings to date, with particular emphasis on hash function choice and optimal number of processes. It will outline the MPT to set the scene for those unfamiliar with the tool, give details about the experimentation setup and variables under scope, as well as present results obtained.

Our aim is to share knowledge of the MPT tool and experimental evidence demonstrating how to optimize its usage. The poster therefore aligns with the call for contributions by supporting colleagues across all organizations and sharing research that influences practice.

# Acknowledgements

We would like to thank the other MPT team members who helped contribute to this work.

# REFERENCES

1. K. Davies; P. May and D. Russo, Minimum Preservation Tool, 2020-07-22, Github: <https://github.com/britishlibrary/mpt>
2. M. Pennock, J. Beaman, P. May and K. Davies, “Back to Basics: The Minimum Preservation Tool,” In Proceedings of iPRES2021, 17th International Conference on Digital Preservation, 2021.
3. S. Chang, R. Perlner, W. E. Burr, M. S. Turan, J. M. Kelsey, S. Paul, L. E. Bassham, “Third-Round Report of the SHA-3 Cryptographic Hash Algorithm Competition”, November 2012, [https:/nvlpubs.nist.gov/nistpubs/ir/2012/NIST.IR.7896.pdf](https://nvlpubs.nist.gov/nistpubs/ir/2012/NIST.IR.7896.pdf)
4. T. Claeson, A. Sateesan, J. Vliegen, N. Mentens, “Novel Non-cryptographic Hash Functions for Networking and Security Applications on FPGA”, September 2021. <https://www.researchgate.net/publication/354610638_Novel_Non-cryptographic_Hash_Functions_for_Networking_and_Security_Applications_on_FPGA>
5. Yann Collett, xxHash, 2012 – 2021, Github: <https://github.com/Cyan4973/xxHash>
6. B Buchanan, “When Fast Just Isn’t Enough: Nobody inspects the spammish repetition”, 2018-08-08. <https://medium.com/asecuritysite-when-bob-met-alice/when-fast-just-isnt-enough-nobody-inspects-the-spammish-repetition-3458a96aa04e>
7. Yue Du, xxhash, 2014 – 2020, Github: <https://github.com/ifduyue/python-xxhash>