In: 3rd International Workshop on Data Engineering Meets the Semantic Web (DESWEB), Washington, DC, USA, April 2012.

A Benchmark for RDF-based Metadata Management in Distributed Long-Term Digital Preservation

Ivan Subotic[#], Lukas Rosenthaler[#], Heiko Schuldt[*]

[#]*Imaging and Media Lab, University of Basel, Switzerland*

[*]*Databases and Information Systems Group, University of Basel, Switzerland*

{firstname.lastname}@unibas.ch

***Abstract*— In a large variety of applications, the long-term, guaranteed availability of data is becoming increasingly important. Thus, long-term digital preservation systems have to be inherently distributed to allow content to be replicated. This affects both the preservation of the actual digital objects and their associated metadata. For the latter, RDF has become the prevalent data model. Ensuring data integrity and consistency requires periodic checks to timely detect inconsistencies, for instance due to (partial) hardware failures, and trigger repair actions. Hence, the access characteristics to metadata in long-term digital preservation significantly differs from metadata management in other types of applications. In addition, the increasing size of digital archives challenges the consistency checks of the associated metadata. In this paper, we introduce a novel benchmark for triple store-based metadata management that jointly takes into account the specific access patterns of long-term preservation systems: i.) complex periodic consistency checks, ii.) concurrent read and write requests to the archive, and iii.) the actions to be taken on data to re-establish consistency if a violation has been detected. Furthermore, we present the results of this benchmark applied to our distributed long-term digital preservation system DISTARNET.**

I. Introduction

The long-term preservation of digital assets is becoming increasingly important. Various applications require digital objects to be available unaltered for potentially long periods. In business applications, for instance, data needs to be kept for several years due to legal constraints. Similarly, scientific data needs to be preserved to ensure the reproduction of results. In medical applications, health-related data on patients needs to be available at least for the lifetime of a human. Finally, data in cultural heritage digital libraries should be preserved for potentially unlimited time spans.

In order to ensure the ability to interpret data, and also interdependencies with other data, digital long-term preservation needs to take into account metadata together with actual digital objects. For the former, RDF has become the prevalent data model and triple stores are used for managing metadata, e.g., to describe how data has been produced (data provenance), how data representations can be interpreted, etc. The requirements on digital long-term preservation can best be met by systems that are inherently distributed, i.e., that apply replication to increase the availability of digital objects and their associated metadata. The vulnerability of computer hardware might lead to inconsistencies across different or same replicas of digital objects and metadata. Therefore, to guarantee integrity and consistency of digital objects, regular checks need to be applied to be able to timely detect inconsistencies and trigger repair actions (e.g., to re-install a corrupted replica from another, non-corrupted one). These consistency checks generate a non-negligible additional load to the system which forms the special characteristics of long-term digital preservation systems, compared to other RDF database applications. Consistency checks and corrective actions will be referred to as *system processes*. In addition, read and write requests (*user processes*) need to be considered in the archive.

In this paper, we introduce a novel benchmark for RDF-based metadata management that jointly takes into account specific access patterns of long-term preservation systems that stem from both system and user processes, as such currently not present in any existing benchmark. Furthermore, we present the results of this benchmark applied to our distributed long-term digital preservation system DISTARNET in two realistic archiving settings, an image archive, and for digital preservation of ancient documents.

The remainder of this paper is organized as follows. Section II discusses related work. In Section III, we introduce the DISTARNET system, in particular its data model and the processes implementing consistency checks and repair actions. Section IV presents our new benchmark for long-term digital preservation. Section V provides the benchmark results of DISTARNET and Section VI concludes.

II. Related Work

There are a number of different benchmarks for evaluating the performance of triple stores: the Berlin SPARQL benchmark [1] (BSBM), the Barton Library benchmark [2], the Lehigh University Benchmark [3] (LUBM), which focuses on inference and reasoning capabilities of RDF engines, and the SPARQL Performance Benchmark (SP^{2}Bench) [4]. All these benchmarks have in common that they do not, or only partly, address the specific access characteristics that can be found in a long-term digital preservation system.

The e-commerce scenario used by BSBM takes into account a similar access pattern a user would also have in an archiving system (user processes). However, it lacks support for system

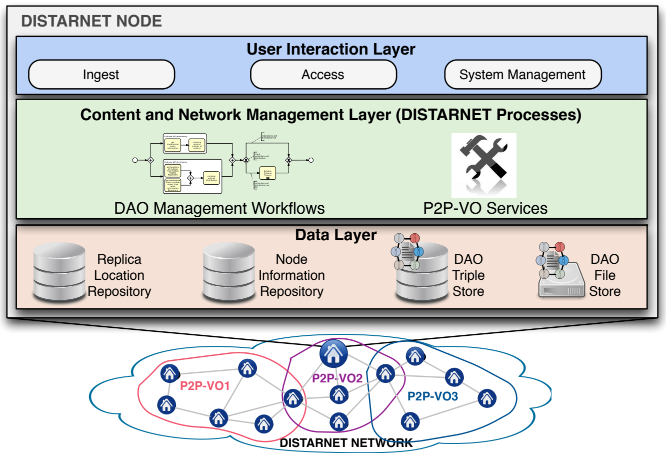

Fig. 1. DISTARNET System Architecture

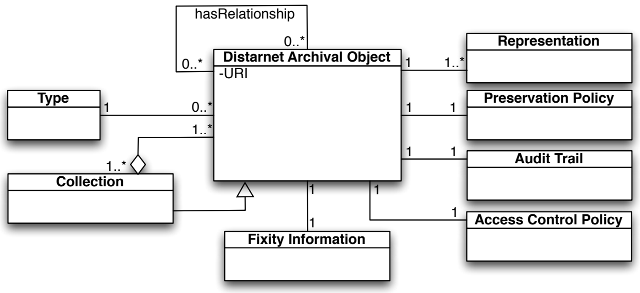

Fig. 2. Conceptual DISTARNET Data Model in UML Representation (attributes omitted for better readability)

processes which make the particular semantics of a preservation system, (e.g., consistency checking, data migration, etc.) and the recovery needed for corrupted data. Similarly, the Barton Library benchmark focuses on user access without special consideration of the access pattern imposed by system processes. The same also holds for LUBM and SP^{2}Bench.

There are a number of long term archiving initiatives underway. Some follow a centralized approach like Kopal[1] or SHERPA DP [5]. Others, like LOCKSS [6], Cheshire 3 [7], SHAMAN[2], and DuraCloud[3], take distributed approaches which in general have lower infrastructure and maintenance costs, the ability for virtually unlimited growth, and allow for a higher degree of availability and reliability. DISTARNET also follows a distributed approach.

III. DISTARNET System Model

DISTARNET is a fully distributed system consisting of a network of collaborating nodes (see Figure 1), organized into virtual organizations (VO) [8]. Each VO is formed of computer resources of organizations that collaborate based on mutual trust. The structure of the network inside the VOs is organized in a P2P fashion, i.e., a DHT overlay is used for data management. We refer to this combination of VO and P2P as a *P2P-VO*. A DISTARNET node can be part of one or more P2P-VOs. Resources provided by nodes within a P2P-VO can only be accessed by other member nodes. A detailed presentation can be found in [9].

A. DISTARNET Data Model

A *DISTARNET Archival Object* (DAO, see Figure 2) is a container holding an information object with possibly different *representations* (e.g., image, audio/video, text, etc.). Moreover, it contains *relationships* to other DAOs (e.g., links) or information about the object's membership in collections/subcollections. Thus, the DISTARNET data model allows the archiving of networks of complex digital objects. Further, a DAO can also contain semantically rich metadata that will provide supplementary descriptions on an information object (e.g., annotations). Finally, a DAO comprises an *access control policy*, an *audit trail* where any changes to an object are logged, a *preservation policy*, and *fixity information* where different checksums of the parts of a DAO are maintained.

To represent, for example, an annotation for an archived image, we will create a DISTARNET Archival Object of the corresponding type which contains the annotation and make a link to the DAO containing the image – note that an annotation can be anything from text to a full-fledged DAO. The DISTARNET data model provides support for different kinds of complex objects like documents, images, electronic books, and other compound information entities. Further, DISTARNET allows any combination of media types to be aggregated into complex objects. The container storing the information objects corresponds to the Archival Information Package (AIP) described in the OAIS Reference Model [10]. Although certain metadata like annotations are generated over time, and were not necessarily present at the time the original information object has been archived, they are treated equally since they provide additional descriptive information and thus need to be preserved as well.

B. DISTARNET Processes

The goal of DISTARNET is to guarantee the long-term availability of data by applying dynamic replication, automated consistency checks, and recovery of DAOs. This is done by means of automated *system processes*, governed by preservation policies, which are run without any centralized coordinator in a fully distributed network. In addition, there are a number of *user processes* which allow users to retrieve objects, to create arbitrary links between DISTARNET objects, to create and manage collections/subcollections, and create and edit annotations. In what follows, we briefly discuss the DISTARNET system processes (details can be found in [9]):

a) Self-Configuration: the ability of the DISTARNET system to automatically detect changes in the network. Events such as new nodes joining or nodes leaving are constantly monitored and taken care of by predefined processes.

Node Joining Process (NJP). These processes encompass the necessary steps that need to be executed after a node that

[1]http://kopal.langzeitarchivierung.de
[2]http://shaman-ip.eu/shaman/
[3]http://duraspace.org/duracloud.php

has joined the network to integrate it into the P2P-VO (e.g., configure node credentials, update list of neighbor nodes).

Periodic Neighbor-Node Checking Process (PNCP). Every node periodically pings its neighbors. If a node does not reply after a predefined period, it is marked as lost.

Automated Dynamic Replication Process (ADRP). This process is responsible for finding suitable storage nodes (e.g., based on geographic distribution), for estimating the optimal number of replicas and for initiating the creation of replicas by taking into account policy-based restrictions.

b) Self-Healing: Due to the continuous monitoring of nodes, DISTARNET is able to detect events reflecting abnormal conditions that may harm its proper functioning and it will be able to execute processes for automated recovery.

Node Lost Event. The system automatically initiates countermeasures by re-evaluating the affected DAOs. The ADRP will create new replicas, if necessary, by taking into account policy-defined redundancy and availability requirements.

Corrupted DAO Event. Periodic integrity checks are done by the *Periodic Integrity Checking Process (PICP)* which automatically trigger countermeasures like finding healthy replicas and copy them in place of the corrupted DAOs.

Obsolete Data-Format. Data formats of DAOs are constantly monitored and warnings are issued if a given data format is becoming obsolete. The *Data-Format Migration Process (DMPP)* can be used to automatically migrate data formats by following a predefined migration path.

c) Self-Learning: DISTARNET automatically reacts to changes in its environment by dynamically adapting the above processes, for example by lowering the invocation frequency of processes in the case no changes took place in the network for a longer period, or shortened if there were recent changes.

IV. Benchmark Design

In what follows, we present the benchmark we have developed for evaluating triple store systems in the context of the typical load of long-term digital preservation.

A. Scenarios

To cover a wide usage spectrum, the benchmark will be performed using data generated for two different scenarios.

Scenario 1 – Image Archive: The *Image Archive* consists of collections of images where each image has three representations: a TIFF and a JPEG representation, and a textual representation of the image in Dublin Core. Each representation is characterized by attributes such as mimetype, label, or creation-date-time. The TIFF and JPEG representations will only contain URIs to where the TIFF or JPEG files are stored. Figure 3 (a) depicts the Image DAO graph that describes the metadata graph that will be stored in the triple store. Note that this is not a complete illustration of the whole graph but rather an excerpt that visually represents the differences between the image and the manuscript graph (scenario 2).

This scenario is motivated by its generality, since the images can be exchanged by any data type (e.g., documents, audio/video, etc.) that can have multiple representations that need to be jointly archived.

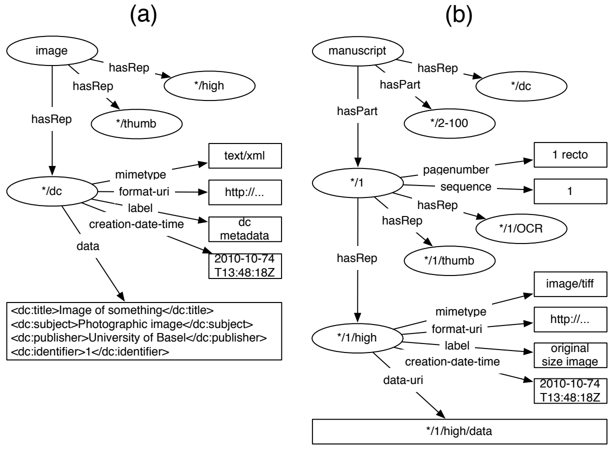

Fig. 3. Image (a) and Manuscript (b) DAO Graph

Scenario 2 – Manuscript Archive: This scenario addresses collections of digitized manuscripts. Each manuscript consists of several pages, each one having multiple representations. Additional metadata describes the manuscript as a whole. For our scenario we set the number of pages per manuscript to 100. Every page has three representations, namely TIFF, JPEG, and OCR Text. The metadata describing the whole manuscript consists of Dublin Core formatted text. Figure 3 (b) depicts the Manuscript DAO graph.

B. Scaling Factor

The *RDF generator* creates the metadata representing the archives of different sizes for both scenarios. For this we have defined a scaling factor F that specifies a multiple of a collection containing 100 archived objects. In both scenarios, we will perform the benchmark runs for scaling factor sizes of 10, 100, 1,000 and 10,000. The generated metadata corresponds to archive sizes of 1,000, 10,000, 100,000 and 1,000,000 objects. If we take the conservative assumption of a total of 100 MB of digital data per image object (including all representations), this implies for Scenario 1 required disk spaces of 100 GB, 1 TB, 10 TB and 100 TB, respectively.

For Scenario 2, if we again assume 100 MB of digital data per image object (which will be needed for all representations of each of the 100 pages), then the total disk space amounts to 10 TB, 100 TB, 1 PB and 10PB, respectively. This total disk space represents the size of the whole archive managed by the metadata store. Table I shows the different scenarios (S.), the scaling factor values (F), the number of image or manuscript objects (# Obj.), the needed storage space for the whole archive (Coll.), the size of the triple store on disk (Meta Data), and the number of triples the graph consists of.

C. Benchmark Queries

The benchmark queries are derived from the (system and user) processes' access to metadata. The goal is to evaluate the scalability characteristics – and possible limitations – of the metadata store with increasing number of objects. We assume that the benchmark load reflects the system and

TABLE I

CHARACTERISTICS OF S 1 AND S 2 WITH DIFFERENT SCALING FACTORS

S.	Factor	# Obj.	Coll.	Meta Data	# Triples
1	10	1 k	100 GB	201 MB	29 k
1	100	10 k	1 TB	204 MB	290 k
1	1,000	100 k	10 TB	450 MB	2,902 k
1	10,000	1,000 k	100 TB	2.9 GB	29,020 k
2	10	1 k	10 TB	425 MB	2,515 k
2	100	10 k	100 TB	2.6 GB	25,150 k
2	1,000	100 k	1 PB	23.9 GB	251,502 k
2	10,000	1,000 k	10 PB	250 GB	2,515,020 k

user processes that need to be considered in the course of one day (i.e., a number of system processes is supposed to run daily). The focus of this benchmark is to evaluate the triple store performance for metadata management. Therefore, the processes that have been implemented in the benchmark contain only the parts that pertain to the interaction with the triple store.

System Processes

P1: For each DAO, find out if it was checked in the last 24h. If not then check if the checksums of the objects are OK, and mark as checked. If a checksum is not OK, then lock and mark DAO as corrupt. The whole archive must be checked once a day. P1 has read-write access to the triple store.

P2: For all DAOs marked as corrupt, replace subgraph with healthy version from another system retrieved with P3. We assume a failure rate of 10% of all DAOs. These 10% will be detected together by P1 and P3. The assumption of a 10% failure rate is a very conservative one and corresponds to the worst case that we anticipate based on the discussion in [11] and [12]. P2 has read-write access to the triple store.

P3: Queries a local node will receive from other remote nodes on which P2 runs. It returns the subgraphs and their checksums of a DAO identified by URI. Before sending the subgraphs, checksums are again calculated to check whether the DAO is OK. If not OK, then DAO is locked and marked as corrupt. We assume that at least three copies per object are stored in the network. Therefore the other remote nodes can run this query against the local node in consideration. Hence, we conservatively assume two concurrent requests. P3 has read-write access to the triple store.

P4: Addresses data format migration, i.e., the actions to bring preserved digital content up-to-date when a new data format is available, or an existing format is deprecated. P4 converts each DAOs TIFF representation to JPEG2000. This process must finish within 90 days (for the complete collection) – so the whole collection could in principle be migrated four times a year, which is a very conservative assumption. P4 implements read-write access to the triple store.

P5: Encapsulates the corrective actions needed to repair and maintain digital objects. For each affected subgraph of a DAO, the changed subgraph is retrieved, and the checksum of the subgraph is recalculated and updated after P4, P8, P9, and P10. This process accesses the triple store in read-write mode.

User Processes:

P6 (Simple Read): Query and return a certain number of objects belonging to a collection. The number of objects to be returned is determined by a normally distributed random variable. The number of process instances are calculated by multiplying $1/10$ with the number of users and the scaling factor. P6 has read-only access.

P7 (Complex Read): Find objects with creation date between two dates (randomly chosen from a list), that have a particular author, and some keywords that occur in the annotations. The number of process instances is calculated by multiplying $1/10$ with the scaling factor and the number of users. P7 has read-only access to the triple store.

P8: User-created collection with some (randomly) selected objects. We conservatively assume that each user creates one collection per week (i.e., $1/7$ per day). P8 needs read-write access to the triple store.

P9: Creation of an annotation. We assume that every user creates one annotation per day. Thus, P9 requires read-write access to the triple store.

P10: Represents the user creating a link between two DAOs. We assume that each user creates two links per day. P10 has read-write access.

D. Benchmark Mix

Table II shows the frequency and concurrency per process type that are used for creating the benchmark mix. Frequency denotes how often a specific process type will run, and concurrency denotes how many possibly concurrent instances will be created each time the process runs. Multiplying the frequency and the concurrency yields the total number of instances of each process type created. Each process type in turn is implemented by means of a number of SPARQL queries. The factor F represents the scaling factor that is used to generate the test data. Increasing the scaling factor linearly increases the size of the metadata representing the archive, which in turn also increases the number of process instances. The variable b is the base number of objects (e.g., image or manuscript) that is created for each F, and variable u the number of users that access the system per F. The degree of concurrency of the user process types depends solely on u and F. r denotes the number of replicas (minimum $r = 3$)

For the evaluation, we have set b to 100 (i.e., 100 image or manuscript objects are contained in a collection). The size of b is by itself not important, but rather the ratio of b and u as it defines the relation between system and user processes. For the benchmark, we set the ratio between system and user processes to approx. $1/4$ user and $3/4$ system processes which reflects the special characteristics of a digital long-term preservation system. Table III contains relative and absolute numbers of process instances for the base case with $F = 1$, $b = 100$, $u = 20$ and $r = 3$.

E. Benchmark Implementation

The benchmark (RDF generator and driver) implemented in Scala [13] runs on any current JVM. The benchmark test driver

TABLE II

QUERY FREQUENCY AND CONCURRENCY. F - SCALING FACTOR, b - BASE NUMBER OF OBJECTS, u - BASE NUMBER OF USERS, R - NUM. OF REPLICAS

P-Type	Frequency	Concurrency	Access
P1	1	$b * F$	R/W
P2	$0.10 * b * F$	1	R/W
P3	$0.10 * b * F$	$r - 1$	R
P4	1	$1/90 * b * F$	R/W
P5	$P4 + P8 + P9 + P10$	1	R/W
P6	$1/10$	$u * F$	R
P7	$1/10$	$u * F$	R
P8	$1/7$	$u * F$	R/W
P9	1	$u * F$	R/W
P10	2	$u * F$	R/W

TABLE III

NUMBER OF PROCESS INSTANCES ($F = 1$, $b = 100$ AND $u = 20$)

P-Type	Frequency	Concurrency	Absolute	Relative
P1	1	100	100	38.43 %
P2	10	1	10	3.84 %
P3	10	2	20	7.69 %
P4	1	1	1	0.43 %
P5	63	1	63	24.25 %
				74.64 %
P6	2	1	2	0.77 %
P7	2	1	2	0.77 %
P8	2	1	2	0.77 %
P9	20	1	20	7.69 %
P10	40	1	40	15.37 %
				25.36 %
TOTAL	151	110	260	**100.00 %**

takes as parameter the location of the SPARQL endpoint. For the communication to the triple store, the SPARQL 1.1 HTTP protocol[4] is used. The system and user processes implement their operations on the RDF graphs by using SPARQL 1.1 Query[5] and Update[6]. The benchmark mix is implemented by serializing the system and user processes described in Table III, depending on the size of F which is given by the number of DAOs found in the triple store. This guarantees at runtime an evenly distributed access pattern to the triple store, where the system and user processes are evenly intermixed and spread over the whole duration of the benchmark.

The benchmark can be run on any triple store that exposes a SPARQL update enabled endpoint, and into which the RDF data from the data generator is loaded.

V. BENCHMARK EVALUATION RESULTS

Evaluation Setup

Since one of the targeted types of organizations for DISTARNET are small archives that do not possess a large IT environment, we decided to perform the evaluation on commodity hardware. The evaluation was performed on an Apple Mac Pro with 2 x 3GHz Dual-Core Intel Xeon CPUs, 24GB RAM and a dedicated 2TB SATA disk drive. The triple store used in the evaluation was Jena TDB[7], a natively implemented triple store, with a SPARQL endpoint provided by the Fuseki server[8].

[4]http://www.w3.org/TR/2011/WD-sparql11-http-rdf-update-20110512/

[5]http://www.w3.org/TR/2011/WD-sparql11-query-20110512/

[6]http://www.w3.org/TR/2011/WD-sparql11-update-20110512/

TABLE IV

BULK LOAD TIMES FOR SCENARIOS 1 AND 2

S	# of DAOs	# of Triples	TS Size	Duration	TPS
1	1,000	29,020	0.2 GB	00m 09s	3,224
1	10,000	290,200	0.2 GB	00m 42s	6,910
1	100,000	2,902,000	0.5 GB	06m 59s	6,926
1	1,000,000	29,020,000	2.9 GB	01h 07m	7,206
2	1,000	2,515,020	0.4 GB	06m 32s	6,416
2	10,000	25,150,200	2.6 GB	01h 14m	5,608
2	100,000	251,502,000	23.9 GB	13h 38m	5,120
2	1,000,000	2,515,020,000	250.0 GB	97h 20m	7,177

Bulk Load Times

The Jena TDB triple store has two command line utilities for fast initial loading (i.e., build TDB indexes) of RDF data into an empty TDB store, namely `tdbloader` and `tdbloader2`. They differ in the way indexes are generated. `tdbloader` builds the node table and the primary indexes first. After that, it builds the secondary indexes. `tdbloader` can also be used for incremental loads. `tdbloader2`, in contrast, builds just the node table and text files for the input data using Node IDs. It then uses UNIX `sort` to sort the text files and produce text files ready to be streamed into `BPlusTreeRewriter` to generate B$^+$Tree indexes. `tdbloader2` cannot be used to incrementally load new data into an existing TDB database but it is faster when working with larger data sets[9].

For the two scenarios we have used the `tdbloader2` command. Table IV contains the load time durations and the rate in triples per second (TPS) for the two scenarios and different scaling factors. This table shows that there is a substantial amount of data that needs to be managed, even though we are only dealing with the metadata of the archival system.

S1 and S2 Evaluation Results

In the following we will discuss the results of our evaluation runs made with scaling factor F set to 10, 100, 1,000 and 10,000 for both scenarios (summarized in Table V). Figure 4 depicts the durations of the evaluation runs per scale factor. The change in duration from $F = 10$ to $F = 100$ is sub-linear but can be explained through the large impact of the program overhead of the short running $F = 10$. The change in duration is rising slightly between $F = 100$ and $F = 1,000$ for both scenarios, and then between $F = 1,000$ and $F = 10,000$, S1 stays almost linear while S2 doubles in duration. This doubling can be explained by the high load, i.e. 2.5bn triples and adding

[7]http://openjena.org/TDB/

[8]http://openjena.org/wiki/Fuseki

[9]http://seaborne.blogspot.com/2010/12/performance-benchmarks-for-tdb-loader.html

TABLE V

EVALUATION RESULTS FOR S1 AND S2

Factor	10	100	1,000	10,000
S1				
New Triples	3,438	34,476	344,914	3,449,197
P-Type Inst.	2,618	26,192	261,936	2,616,364
Duration	19s	2m 40s	46m 4s	7h 46m
S2				
New Triples	11,060	111,405	1,106,400	11,145,304
P-Type Inst.	2,618	26,192	261,936	2,619,364
Duration	1m 37s	13m 28s	2h 35m	2d 10h 16m

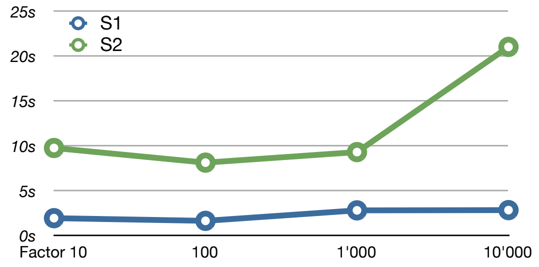

Fig. 4. Evaluation Run Durations in Seconds per Scale Factor in S1 and S2

11m new ones, combined with the limitations for executing parallel write operations.

Discussion

The benchmark evaluation times for both scenarios and all F values are well below the 24h mark except for $S2$ with $F = 10,000$ where the duration is over 48h. This means that the evaluated triple store can be used for the storage and management of metadata for DISTARNET archives of a size of approx. 5 PB (as this corresponds to the case where all benchmark queries terminate within the 24h margin).

Meta-Data-Only vs. All-Data Benchmark: The benchmark and results presented here take only the metadata side of the archival processes into consideration. The question now arises how the results would differ if not only the operations on the metadata but also on the archived data were taken into account. Furthermore what sizes of F would still be feasible while still fulfilling the 24h maximum running time constraint. For this, mainly system processes $P1$ through $P3$ would be affected. The additional run-time strongly depends on the size of the archived objects (for the calculation of the checksums), and the speed of the network connection. The calculation of the checksum can be easily parallelized, which in light of the widespread use of multicore CPUs will mitigate the effect. The process instances can be run in parallel which will be discussed in the next section. When taking the archived data into account as well (which is planned in future work), the archive sizes that can be handled in the predefined benchmark period of 24h will shrink – but this is not because of the performance of the triple store-based metadata management, but because of the additional computational load imposed by some of the system processes.

Parallelism: The implementation of the benchmark utilizes parallel execution of the process instances whenever possible. As described in Section IV-E, the execution is done in cycles where one cycle is equivalent to the benchmark mix for $F = 1$ and where all processes are run in a serial fashion. The different process instances of the same type are always run in parallel where possible. The process instances for P1, P4, P6 and P7 are all run in parallel. Presently, the degree of parallelization of the write processes is limited, as only a multiple-reader / single writer locking model is supported.

VI. CONCLUSION

Long-term digital preservation is becoming increasingly important. It requires that the actual digital content is archived together with its metadata. In order to automatically cope with possible failures, the DISTARNET system provides processes that ensure redundant and consistent metadata storage. In this paper, we have defined a benchmark for process-based metadata management tailored to long-term digital preservation settings, and we have evaluated the scalability characteristics of metadata management using the Jena triple store to find out potential scalability restrictions. The evaluation results provided in this paper show that even with commodity hardware, archive sizes up to approx. 5 PB can be supported with triple-store based metadata management.

We also plan to apply this benchmark to other triple stores and to create a benchmark for the evaluation of complete long-term digital preservation systems. This will allow us to evaluate the scalability characteristics of the distributed DISTARNET system as a whole.

REFERENCES

[1] C. Bizer and A. Schultz, "The Berlin SPARQL Benchmark," *Intl. Journal on Semantic Web Information Systems*, vol. 5, no. 2, 2009.
[2] D. Abadi, A. Marcus, S. Madden, and K. Hollenbach, "Using The Barton Libraries Dataset as an RDF Benchmark," MIT, Tech. Rep., 2007.
[3] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A benchmark for OWL Knowledge Base Systems," *J. of Web Semantics*, vol. 3, Oct. 2005.
[4] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel, "SP^{2}Bench: A SPARQL Performance Benchmark," in *Proc. of ICDE*, Apr. 2009.
[5] G. Knight, "SHERPA DP: Establishing an OAIS-Compliant Preservation Environment for Institutional Repositories," in *Dig. Repositories*, 2005.
[6] V. Reich and D. Rosenthal, "LOCKSS (Lots of Copies Keep Stuff Safe)," *The New Review of Academic Librarianship*, vol. 6, pp. 155–161, 2000.
[7] P. Watry and R. Larson, "Cheshire 3 Framework White Paper," *Intl. Symposium on Mass Storage Systems and Technology*, pp. 60–64, 2005.
[8] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Jounral of Supercomputer Applications*, vol. 15, no. 3, 2001.
[9] I. Subotic, H. Schuldt, and L. Rosenthaler, "The DISTARNET Approach to Reliable Autonomic Long-Term Digital Preservation," in *Proc. DASFAA*, 2011, vol. 6588, pp. 93–103.
[10] Ccsds, "Reference Model for an Open Archival Information System (OAIS). Blue book, Tech. Rep. 1, January 2002.
[11] B. Schroeder and G. Gibson, "Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?" in *5th USENIX Conference on File and Storage Technologies*, 2007, pp. 1–16.
[12] E. Pinheiro, W.-D. Weber, and L. Barroso, "Failure Trends in a Large Disk Drive Population," in *5th USENIX Conference on File and Storage Technologies*, 2007, pp. 17–28.
[13] M. Odersky, P. Altherr, V. Cremet, I. Dragos, and G. Dubochet, "An Overview of the Scala Programming Language," EPFL Lausanne, Switzerland, Tech. Rep., Aug. 2004.