publications

MDPI

Article

DRAS-TIC Linked Data: Evenly Distributing the Past

Gregory Jansen [1,*], Aaron Coburn [2], Adam Soroka [3], Will Thomas [1] and Richard Marciano [1]

1 School of Information Studies, University of Maryland, College Park, MD 20742, USA
2 Information Technology Services, Amherst College, Amherst, MA 01002, USA
3 Office of the CIO, The Smithsonian Institution, Washington, DC 20002, USA
* Correspondence: jansen@umd.edu

Received: 1 March 2019; Accepted: 27 June 2019; Published: 4 July 2019

Abstract: Memory institutions must be able to grow a fully-functional repository incrementally as collections grow, without expensive enterprise storage, massive data migrations, and the performance limits that stem from the vertical storage strategies. The Digital Repository at Scale that Invites Computation (DRAS-TIC) Fedora research project, funded by a two-year National Digital Platform grant from the Institute for Museum and Library Services (IMLS), is producing open-source software, tested cluster configurations, documentation, and best-practice guides that enable institutions to manage linked data repositories with petabyte-scale collections reliably. DRAS-TIC is a research initiative at the University of Maryland (UMD). The first DRAS-TIC repository system, named Indigo, was developed in 2015 and 2016 through a collaboration between U.K.-based storage company, Archive Analytics Ltd., and the UMD iSchool Digital Curation Innovation Center (DCIC), through funding from an NSF DIBBs (Data Infrastructure Building Blocks) grant (NCSA "Brown Dog"). DRAS-TIC Indigo leverages industry standard distributed database technology, in the form of Apache Cassandra, to provide open-ended scaling of repository storage without performance degradation. With the DRAS-TIC Fedora initiative, we make use of the Trellis Linked Data Platform (LDP), developed by Aaron Coburn at Amherst College, to add the LDP API over similar Apache Cassandra storage. This paper will explain our partner use cases, explore the system components, and showcase our performance-oriented approach, with the most emphasis given to performance measures available through the analytical dashboard on our testbed website.

Keywords: distributed database; linked data platform; Fedora Commons repository; horizontal scaling

1. Introduction

This article will showcase the Digital Repository at Scale that Invites Computation (DRAS-TIC) Fedora research project [1], led by the University of Maryland's Digital Curation Innovation Center (DCIC) [2] and its immediate relevance to the Fedora community, as it proves and improves the performance of various implementations of the Fedora 5 API [3], a combination of the W3C Linked Data Platform (LDP), W3C Memento, and other web standards. In digital repositories, the Linked Data Platform standardizes a clean and flexible API for managing digital objects alongside fine-grained structural and descriptive metadata encoded in the Resource Description Framework (RDF). The Memento standard then mixes in a time-mapping and versioning API, such that resources and their descriptions can be retrieved as they were at a particular time and date. Beyond time-based applications, this feature also adds much needed transparency and provenance to otherwise untraceable digital collections. The DRAS-TIC scalability goals have been pursued with our partners in the Trellis Cassandra software project [4], which is a combination of the Trellis Linked Data Platform [5] and Apache Cassandra [6], a distributed database that can scale horizontally and incrementally to potentially

thousands of low-cost servers. Trellis Cassandra extends LDP capacities into the petabyte range with hundreds of millions of unique objects. It also extends repository systems to handle a large number of clients or client requests through incremental scaling of both frontend and storage servers. In order to make such capacity and client scaling sustainable for repository managers both storage migration and the addition of capacity must become routine, cluster-managed processes. Apache Cassandra makes this possible, adding capacity when it is needed at a predictable cost and avoiding big storage planning cycles that hinder collection development. Our work on Apache Cassandra was in part based upon a previous non-LDP repository project, called Indigo [7].

2. Materials and Methods

2.1. Requirements from Partner Institutions

As this project aims to address nascent technical demands for digital repositories, we are eager to qualify these demands through the lens of our four participating institutional partners. Their specific needs for repository technology were studied through extensive interviews with the repository managers and technologists at those institutions. We are grateful to the several staff members at partner institutions who allowed us to visit them, reviewed transcripts and summaries, and gave detailed answers to our follow-up questions. As a graduate student, Saba Aldughaither, helped create our interview script and conducted these interviews as part of her capstone course for a Master's degree in Information Management. She created written transcripts and summaries that helped to align our research questions and performance testing scenarios with real-world demand or demands anticipated in the next several years. We interviewed staff at Georgetown University Libraries, University of Maryland Libraries, and the Smithsonian Institution's (SI) Office of the CIO in spring of 2018. Amherst College has also been a significant institutional partner on this project. However, they became involved after the formal requirements interviews had been conducted.

The three institutions under study presented diverse use cases, but all the use cases had some key common characteristics. Concerns around scalability were expressed by all three institutions with varying degrees of urgency. UMD and SI reported scale as an immediate challenge, while Georgetown cited it as a potential challenge in the future. All three institutions were also interested in systems with open standards; all three currently use DSpace for at least a subset of their collections, and two of the three currently use versions of Fedora. UMD and SI both described significant challenges with their current systems, while Georgetown's current software successfully meets the library's needs. Although the requirements were complex, each institution had some defining characteristics regarding its requirements from digital repository software. For UMD Libraries, the desire to consolidate collections under one versatile system was a main feature. For SI, the ability to compute or perform analysis functions on large, complex, or manifold objects inside the repository was a high priority. Georgetown cited the stability and reliability of DSpace as an important feature. For further details, you may find these interviews or the summaries on our project website [8].

2.2. Measuring Software Performance at Scale

Given the complexities of designing a distributed database schema and given that we know that a convincing platform must be supported by measured performance, we planned the project around a testbed for evaluating various LDP software systems under simulated workloads. Having studied the requirements from our partners, we let these needs inform our testing scripts. With each major change in our Cassandra schema or other candidate system components, we were able to run an extensive battery of simulated workloads and capture metrics for client-side performance and server metrics like CPU, memory, and disk activity. Each test run was linked to a well-defined configuration and code commit in the development history of the software. These data flowed into an analytics dashboard, where stakeholders can see the performance impact of changes in the software design. The design of

the performance testbed that met all of these goals was a major focus of this project and involved a complex stack of software choices.

For the hardware, we had a cluster of four 32-core Dell servers with attached 12-Gbps, high-speed NetApp storage. In order to use the full power of this cluster, we decided to use Linux containers to encapsulate both the test subjects (candidate DRAS-TIC systems) and the test workers or clients running user scenarios. We installed the Docker engine on each physical host and joined these together to make a Docker Swarm [9] environment, such that we could replicate any service, such as Cassandra, any number of times across the swarm. Since Cassandra generates its own significant network traffic between nodes, we had two switched local networks between the four Dell servers. Docker Swarm gave us the ability to scale up test workers or any of the candidate system's nodes independently.

Our test workers were based on the Gatling.io testing framework [10], which is a highly parallel Scala-based system for scenario-driven load testing. In Gatling.io's domain-specific language, you define a step-by-step test script called a simulation, which may consume feeds of test data and perform validation of server responses. Then, you define what is called a Gatling scenario, which dictates how many parallel users will enter and run that same simulation over the course of your test run. These scenarios or load profiles are tailored to performance demands, and a single Gatling worker node can run hundreds of simulated users at one time. While running simulations, these Gatling worker nodes do minimal work to record results data, appending events to a local log file. Through Docker Swarm replication, we can run dozens of these Gatling worker nodes at the same time, giving us a very high upper limit on the performance loads we can generate.

2.3. Performance Analysis and Candidate System Traceability

After all of the Gatling load scenarios were complete, we were left with a set of exhausted test worker nodes, each holding a simulation log file that recorded the key client-side performance metrics for each request made of the candidate system. At this point, the test workers move on to index these data for analysis. Each test worker node parses its local simulation log using the Logstash [11] tool, which unpacks the log format and sends test event data over the network to our Elasticsearch cluster. The Elasticsearch cluster builds a separate index for each test run, which contains three types of events: the simulation run itself; each simulated user as it begins and ends the script; and each individual request that is made of the candidate system.

Along with the client-side metrics, the test workers also recorded several other data points about the candidate system and the testbed system itself. They included the Docker image name, image tag, Git repository URL, and commit id of both the test worker and candidate system. These data points created traceability of our testbed by linking each test to a specific system configuration and software commit. Any stakeholder reviewing the performance data may follow these links to discover exactly what software was under test at the time. With this infrastructure supporting our iterative development process, we were free to try many designs in the pursuit of performance optimization, even for specific use cases, without losing our way.

After the data were indexed in Elasticsearch, we had access to the test run on our testbed website [8]. A simple table of recent tests provided links to a dashboarding program that we used, called Grafana [12]. This analytic dashboard software excels at presenting time series data. We created a customized dashboard to present our LDP test results; see Figure 1. For instance, it included graphs that show the number of requests processed per second and the average duration (for significant percentiles) of the server response. The graphs included red dashed lines that show any error responses or server timeouts encountered during the test.

Readers may wish to explore this and other published dashboards that are available from the testbed portal.

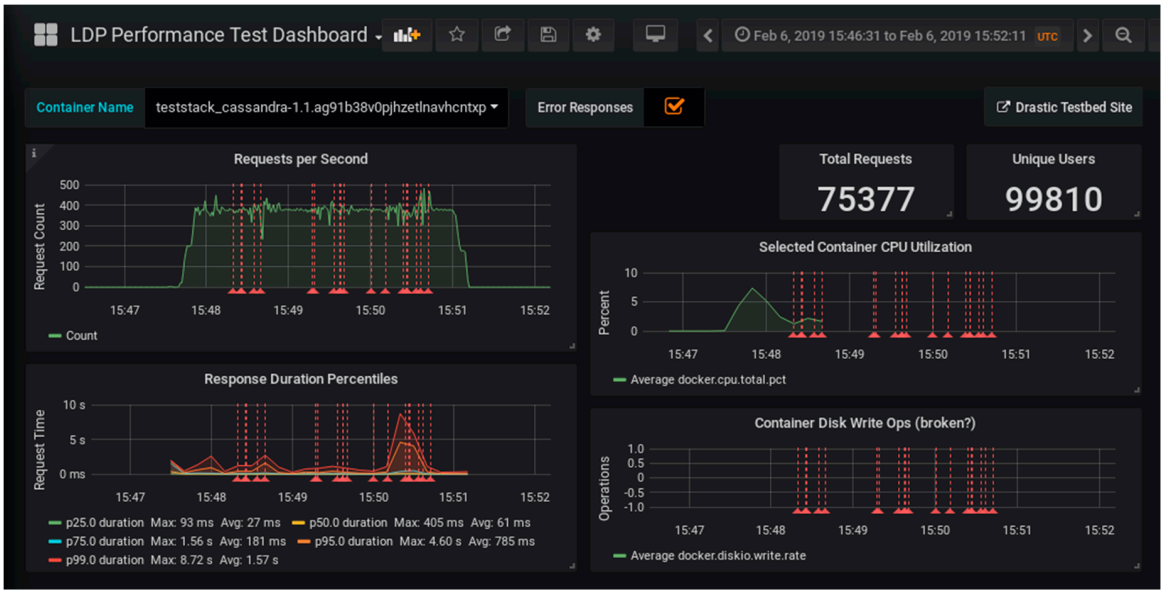

Figure 1. Dashboard display of a typical test run.

2.4. The Trellis LDP and Trellis Cassandra Software Project

We are grateful to have engaged in this project with skilled repository developers at partner institutions. Having found the Smithsonian Institution as an institutional partner in our proposal writing process, we knew that Adam Soroka, a senior architect, would be involved in this effort. Later in 2017, we also reached out to Aaron Coburn at Amherst College, specifically to align with his project to implement the Linked Data Platform specification in his Trellis framework. Trellis LDP [5] is a Java software framework that makes it possible to implement many different backend stores behind a unified LDP/Memento API endpoint. The API endpoint code is exactingly bound to the LDP and Memento W3C specifications, while the underlying storage code may diverge in many directions to suit technical environments or use cases. We quickly realized that there was a strong alignment of interests and schedules between these projects. The first stable release of Trellis LDP was in early 2018, and the 1.0 milestone is coming soon. The Trellis project benefits as it gains a new storage implementation, with the addition of the Trellis Cassandra backend. Trellis LDP also received numerous code contributions as the Cassandra storage implementation pushes certain code optimizations to the forefront, such as streaming data and asynchronous operations. Finally, Trellis LDP is gaining insight from our extensive performance testing. The DRAS-TIC repository initiative (here embodied in the Trellis Cassandra backends for storing metadata and binaries) benefits from reuse of the Trellis LDP/Memento API endpoint, a complex application that must meet API specifications exactly. Overall, the effect of the software development partnership is that more attention is being paid to less code, promoting improvements in software quality across the board. This strong collaboration has improved code at all levels and brought more attention to implementation decisions and performance analysis.

3. Results

This collaboration produced a number of outcomes that we consider significant for the Linked Data community. In the Trellis Cassandra storage module, we helped create a software stack for LDP and Memento resources that scales horizontally to accommodate increased user demand or to add storage capacity. We proved the performance of this stack and several other candidate systems in our DRAS-TIC testbed. Along the way, we created and tested different Docker software stack configurations for the various candidate systems.

3.1. Software Stacks, Comparing APIs, and Configurations

The Trellis LDP Framework, created by Aaron Coburn at Amherst College, is big news for the linked data community. Its stateless frontend API server enables a Linked Data Platform service to scale out smoothly to serve more client connections. In our primary candidate system, Cassandra provides a similar scale out in the storage layer, together removing most of the performance bottlenecks that plague existing repository systems. We hasten to add that Trellis supports a variety of other storage implementations beyond Cassandra, including flat files, relational databases, and cloud-managed services such as Amazon S3. In addition to testing the Trellis Cassandra stack, we ran the same tests against the Trellis File System stack and the Trellis Database stack. Meanwhile, DuraSpace and their Fedora partners released the Fedora 5.0 reference software [13], based on the JBoss ModeShape [14] stack. ModeShape is also open to different storage implementations, and so, we likewise configured the Fedora ModeShape stack for testing. We tested the Fedora 5.0 stack with file system storage and with database storage. The Table 1. below shows the various software stack configurations that were most often subjected to performance tests.

Table 1. Software stack configurations most often subjected to performance tests.

Front-End	Frontend Scale	Storage System	Storage Scale
Trellis LDP	1	Local File System	1 [1]
Trellis LDP	1–N (4 tested)	PostgreSQL Database	1
Trellis LDP	1–N (4 tested)	Cassandra Cluster	1–N (4 tested)
Fedora 5.0	1	Local File System	1
Fedora 5.0	1	PostgreSQL Database	1

[1] Disregards sharing a network filesystem between frontend nodes; a possibility not yet tested.

It is important to note one thing about these contrasting software stacks before moving on, which is that they represent different design choices within the LDP ecosystem. Trellis and Fedora differ in their support for some parts of the Fedora 5.0 API specification. The Trellis LDP project chose early on to remove transaction support, in favor of stateless servers and minimal coordination between nodes. The Fedora 5.0 reference software continues to support a flexible transaction API with rollbacks and commits, but as a result, that system and the underlying ModeShape layer must perform more work to maintain and coordinate transaction states. Therefore, when we look at the performance results, we have to bear in mind that these software stacks do not always serve the same use cases. We explore transactions and their uses further in the Discussion Section. This project helps to reveal in clear metrics the weighty design trade-off between scalability and atomic transaction guarantees. The metrics will showcase the performance and scaling potential of various storage options for all candidate systems.

3.2. Repository Performance Measures

Our primary test scenario was one that ingested a consistent sample of files into the configured candidate system, along with a couple of descriptive triples. Each simulated user created an RDF Basic Container with a couple descriptive triples and a Non-RDF (binary) Resource. We ran this test at a variety of scaling factors, with each scaling increment representing another test worker node that brought another 2000 simulated users to bear. All simulated users initiated their scripted API calls within a 200-s window of time. It was a short test, but it created a consistent benchmark ingest load on the candidate systems.

Our tests of the Fedora ModeShape system began by using a file system as storage [15]. We were able to run a single ingest test worker against the Fedora server without any errors. It kept up with the performance load of approximately 20 requests per second. Then, we ran the test again, this time with two ingest test workers and a load of 40 requests per second. Towards the end of this test, we began to see the Fedora system return some internal server errors to our test clients. The maximum

sustained requests per second for this system were somewhere between 20 and 40 requests per second. In Figures 2 and 3, you can see that request duration started to rise $\frac{3}{4}$ of the way through the test load. The red lines in these graphs indicate the occurrence of errors.

Figure 2. Fedora file system begins to show error responses at 40 requests per second.

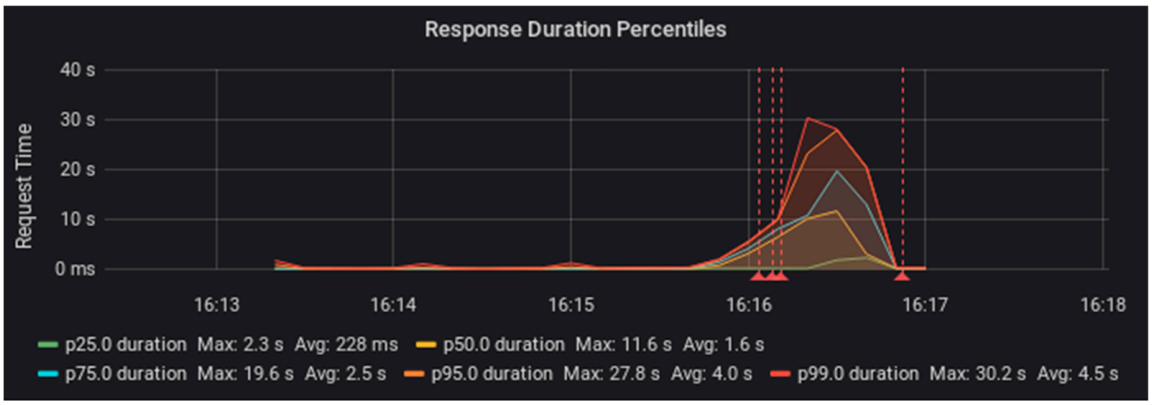

Figure 3. Fedora file system performance begins to degrade at 40 requests per second.

We ran the same set of tests with Fedora configured in a stack with PostgreSQL database storage [16]. This configuration was also able to handle the 20 requests per second load of one test worker. However, when we tried using two test workers to supply 40 requests per second, the performance degraded sooner and more dramatically than the file system configuration (see Figures 4 and 5).

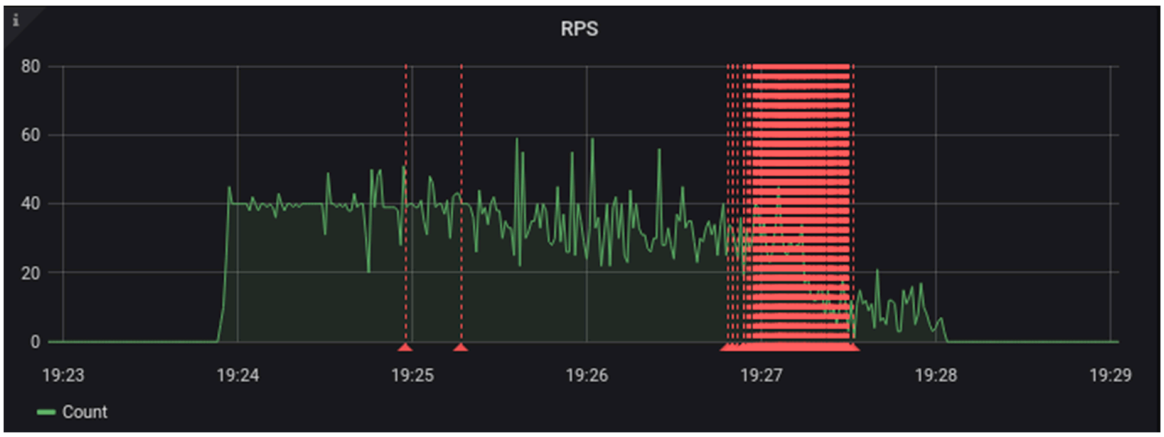

Figure 4. Fedora PostgreSQL system shows error responses at 40 requests per second.

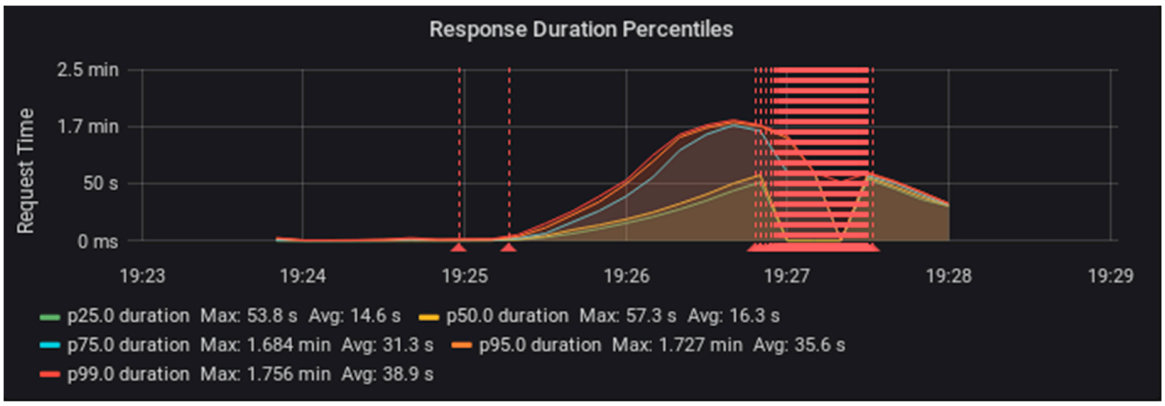

Figure 5. Fedora PostgreSQL system performance degrades at 40 requests per second.

By looking into the system logs, we were able to determine that the cause of these errors was always a timeout while the system waited for a database connection. With a limited number of database connections, it seems like Fedora operations were not releasing these connections fast enough to keep up with requests.

Moving on to the Trellis LDP system, let us look at the performance with PostgreSQL database storage. This system was able to handle the load from a total of 15 test workers (300 requests per second) while maintaining an average response time of just 51 milliseconds and a maximum response time of two and a half seconds (See Figure 6).

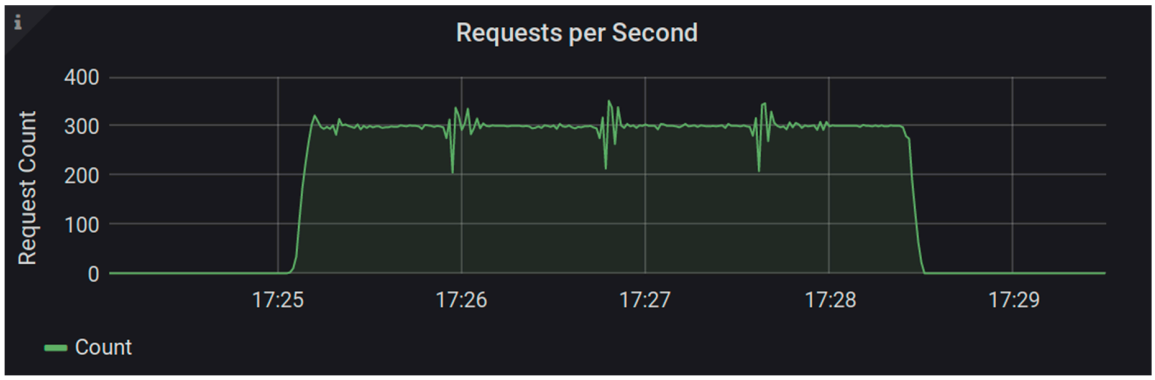

Figure 6. Trellis PostgreSQL system serves 300 requests per second.

When we pushed our test load up to 16 workers (320 requests per second), we saw performance degrade and were able to produce error responses from the Trellis PostgreSQL stack [17]. We have not yet investigated the root cause for this pattern (See Figure 7).

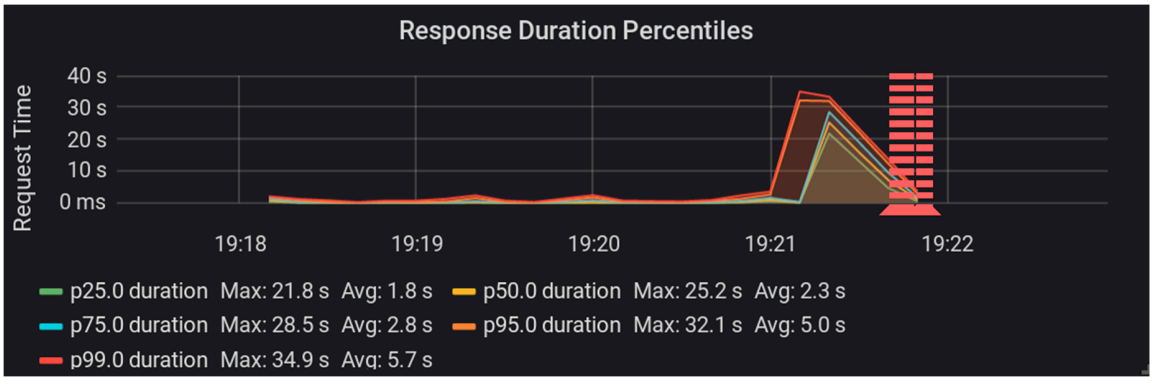

Figure 7. Trellis PostgreSQL performance degrades at 320 requests per second.

Finally, let us look at the performance we were able to measure with the Trellis Cassandra system. Figures 8 and 9 below show a scenario that stressed the Trellis Cassandra candidate system with 24 test workers producing 480 requests per second [18]. This candidate system included four frontend Trellis nodes and four Cassandra storage nodes, and it handled this rate without errors.

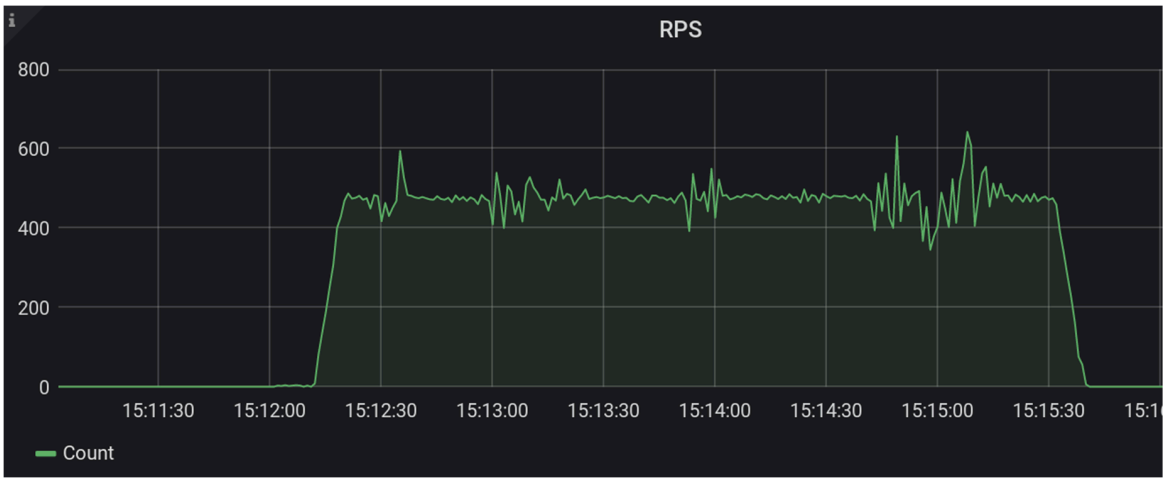

Figure 8. Trellis Cassandra (four and four) system performs at 480 requests per second.

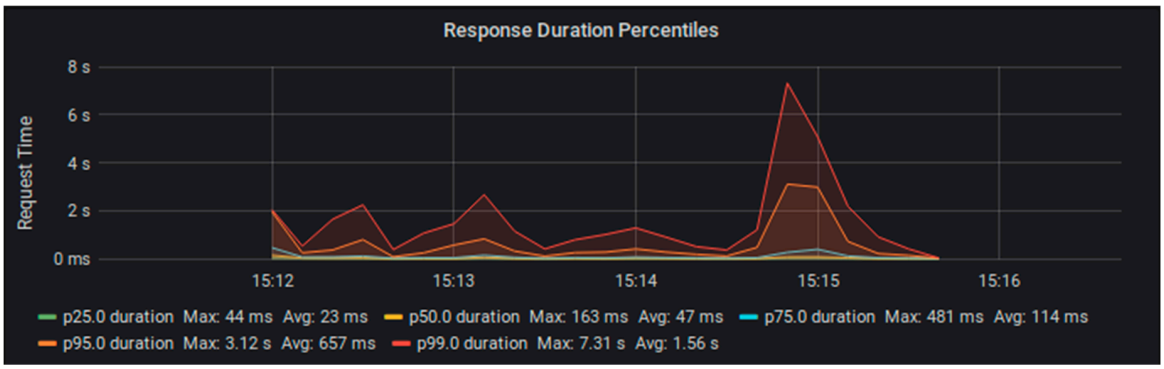

Figure 9. Trellis Cassandra (four and four) response duration percentiles at 480 RPS.

We found the limit of this four Trellis, four Cassandra system when we ran 28 test workers demanding 560 requests per second [19]. At that rate, we started to see error responses as the system failed to keep up with the request load (See Figures 10 and 11). The candidate system stopped waiting for Cassandra connections that were busy processing other requests, resulting in error responses to the LDP client. This is probably a typical failure mode for a Trellis Cassandra system when subjected to too much load. It can be avoided by tuning the Cassandra timeout settings or by adding more Cassandra nodes to the cluster.

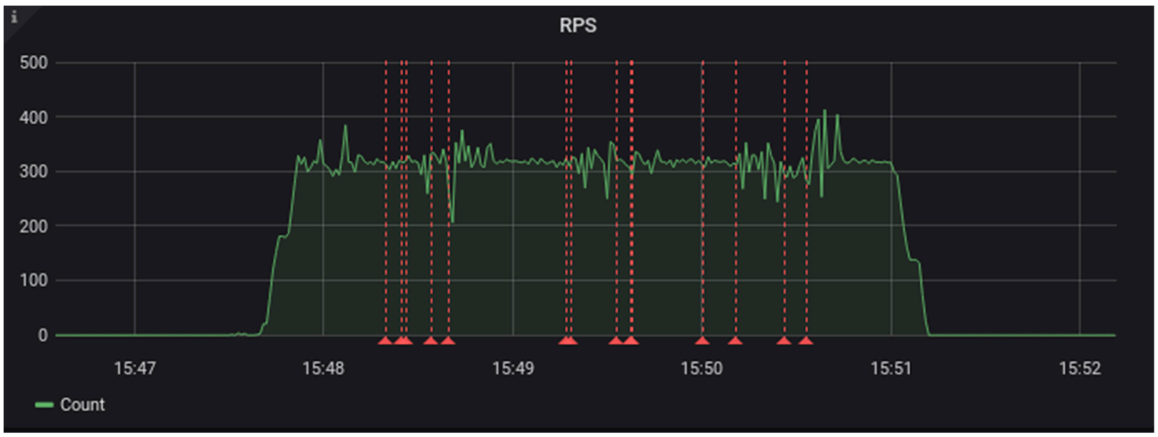

Figure 10. Trellis Cassandra shows error responses with 28 test workers.

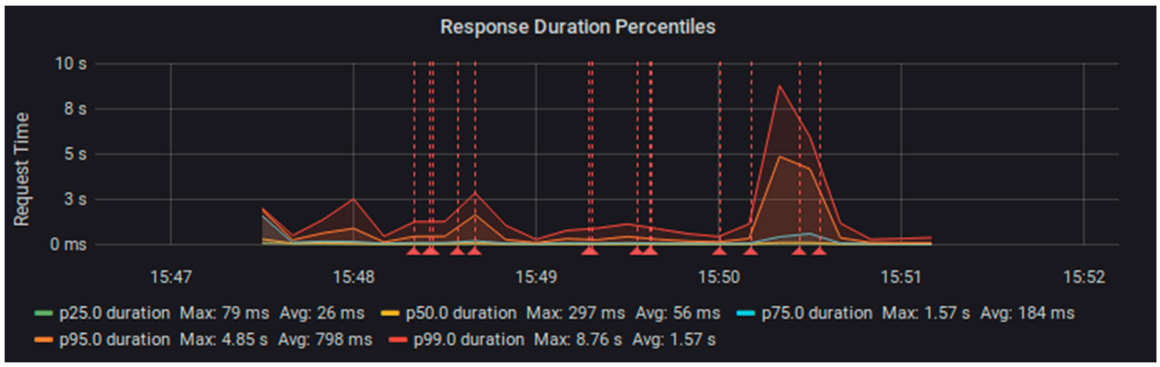

Figure 11. Trellis Cassandra performance degrades at 28 test workers.

4. Discussion

Let us now explore the architectural style of the Trellis LDP and Trellis Cassandra software and the implications of running it as the LDP/Memento component in a repository stack. The reasoning and ramifications of this distributed and componentized architectural style are wide ranging and impact system performance, software sustainability, collection development, capacity planning, hardware migration, and disaster recovery.

4.1. Architectural Style: Functional Decomposition

One of the fundamental concerns addressed in the Trellis LDP Framework and posed to the repository community by the Fedora 5.0 API specification process is one of component scope. If the Fedora 5.0 API specification is inviting implementations and modular, then implementations are free to perform their own analysis of fit and function between software components, finding the most appropriate part of a stack to situate any feature or leaving some out entirely.

The Fedora 5.0 reference implementation locates all Fedora API functions in a single Java virtual machine. While Fedora itself is not a search engine, triple store, or end user application, it does hold to a monolithic implementation of the repository API functions. As the primary reference implementation, it also implements each and every feature described in the specification, leaving none aside. One result is that the Fedora software application is highly customized to its own community, providing a "one size fits all" implementation. For software sustainability reasons, these broad functions were not all implemented from scratch. Instead, an underlying storage software was selected, which then had to be even broader in its functions in order to encapsulate the Fedora functions fully. That underlying store is JBoss ModeShape, which implements the Java Content Repository (JCR) specification [20], a set of Java functions targeting content management that dwarf the LDP specification in complexity.

In contrast, the Trellis framework implements some of the Fedora API features, leaving aside those that are not part of existing web standards and focusing instead on LDP and Memento. Indeed, the Trellis project makes no claims about the Fedora 5.0 specification, but it may nevertheless be deployed within a larger stack that brings along other software to provide additional Fedora features. For example, the LDP specification does not include checksum verification, but this feature is straightforward enough to implement, given an LDP-compliant server. A component architecture blurs the notion of a "one size fits all" software server, but Fedora repositories already typically sit within a larger stack of search engines, triple stores, and user-facing software. We see several advantages in the Trellis software encapsulation of LDP and Memento functions.

Modern data centers and cloud providers continue to diversify their hosting services and find new ways to move compute and storage resources to the services where they are needed. These include both vertical (grow a server) and horizontal (add a server) scale up, which may even be dynamic, i.e., scaling on demand. In addition, the configuration and management of fleets of servers are increasingly effective through deployment and orchestration tools. Whereas it was once an ordeal for operations to run several additional servers, composing your service from diverse software, it now requires minimal effort and is portable. For instance, we can issue a single Docker Swarm command to add either Trellis frontend nodes or Cassandra storage nodes at runtime. Using Kubernetes [21] or any other orchestration framework, one can configure dynamic scaling of the compute nodes based on runtime performance metrics. Furthermore, Trellis provides some experimental code for running the frontend nodes entirely within an Amazon Lambda function, which allows the scale of the web service to be based entirely on traffic flow [22]. With monolithic software in a "one size fits all" server, all optimization of components must necessarily take place within that server software and deal with the internal complexity of the application. This requires specialized knowledge of the application domain. It is not possible to use industry standard server orchestration tools to scale up the inner parts of a software application dynamically.

4.2. Architectural Style: Always Already Distributed

In the last twenty years, much has been written and many advances made with regard to distributed systems, both for storage and computation [23]. We characterize these system architectures as clusters of servers that work together to deliver a service. They generally share these qualities:

- Fault tolerant: failure of individual server nodes is the norm and expected.
- Highly available: more servers can be added to the cluster without interrupting service.
- Peer-to-peer: there is no central coordinating server or central point of failure.

The Apache Cassandra database, technically described as a wide column store, is fully distributed and has these qualities. Cassandra clusters have no central coordinating node and can be configured with varying levels of data replication per keyspace. Cassandra's particular strength is the ability to tune read and write operations to a certain level of consistency. This allows operators to substitute consistency with eventual consistency, not waiting for all copies of a row to be written, in favor of higher availability and speed. Cassandra is open source (Apache License 2.0) and has been successfully used in many large industry deployments, such as NetFlix and Apple Computer, sometimes with thousands of server nodes. The repository community of libraries and archives is strong on guarantees for their data, entrusted as they are with data preservation and access over the long term. Having collectively built several repository stacks and experimented with Cassandra for a number of years, this team feels that eventual consistency is able to satisfy the needs of the community and that it may be preferable to other storage. Distributed storage, being fault tolerant, is able to provide a level of reliable write operations and durability that far outstrips local storage and vertical database systems.

Uniquely, Cassandra systems can be tuned to provide a chosen level of fault tolerance through data replication, creating clusters that remain fast, but can withstand a high number of server node failures. Eventual consistency makes this possible, as we would otherwise have to wait for all data

replicas to be written to so many Cassandra servers. Consistency in Trellis Cassandra is adjustable independently for read and write requests, and for RDF and non-RDF data. If a consistency less than full is selected, performance can be radically increased, but at the possible price of requiring clients to coordinate work to avoid clashes. In our experience, it is trivial to design repository workflows to meet this requirement or simply count on the premise that a temporary data inconsistency in Cassandra, on the order of milliseconds for the chunk sizes we use, is not problematic for most applications.

Another implication of using Cassandra storage is that any database transactions require more overhead since they involve coordination across the nodes in the cluster. As mentioned, transaction support may be a critical feature to some repository systems and of minimal interest to others. Trellis LDP does not include the Fedora transactions API, which supports a user holding an arbitrary number of objects in a suspended edit state until a commit or rollback is issued. Transactions are a functional requirement when two end users have conflicting instructions to a system and only one set of instructions may be allowed to proceed. For instance, banks are concerned that two transactions not spend the same limited account balance. We assert that in most cases, we do not see the same high risks and consequences in repository operations. If, for a brief unlikely moment, a user sees metadata that have been superseded, that is not of great concern. If an institution has a compelling, functional, and non-technical use case for repository transactions, then they will need a system that provides transaction support. However, if the reasons are technical, such as guarding a large batch of ingests or edits against technical failures, then we think that the simple answer is to engineer the system for more reliable write operations. Once again, Cassandra has reliable and high throughput write operations because it is fault tolerant in the face of individual server node failures and because each destination node for data replication is equally and redundantly available to fulfill a given write request [24].

5. Conclusions

Through the performance test results and discussions of systems architecture above, we hope to have illustrated the important decisions and trade-offs involved in next-generation repository systems. We find that for most purposes, a horizontal scaling, stateless LDP server will outperform other systems and provide greater availability as needs change. Similarly, a mix of backend Trellis storage extensions allows implementers to choose one according to their own institutional needs and resources. We have measured the performance of many storage systems behind Trellis LDP, and those metrics are summarized above and provided in detail on our testbed website. In particular, with the Trellis Cassandra storage option, we see that quantity has a quality all its own. We see the potential for near limitless storage growth in a Cassandra storage cluster, with smooth incremental costs for additional capacity, in-band migration, and uninterrupted availability. Cassandra also offers highly resilient storage for data where durability is the key requirement, without sacrificing the performance of reads and writes.

We continue to modify our candidate systems and to test these systems under new testing scenarios and for longer durations. As we add more and more test results to the testbed website, we also plan to further develop that website to allow easier filtering of results by candidate system, test scenario, scale, etc.

We continue to explore research topics around the theme of distributed systems as detailed in our Institute for Museum and Library Services (IMLS) proposal, with special emphasis on these remaining key research questions:

- Can Trellis and Trellis Cassandra mitigate the "super-node problem" encountered in other repository systems (e.g., containers that hold thousands or millions of child objects).
- Does distributed storage offer newly-distributed or decentralized modes of building repositories?

We hope that readers have gained insight into the benefits that horizontal scaling and distributed databases can deliver for repository systems in their institutions. They may have seen that Trellis LDP and Trellis Cassandra are fully proven options for their own repository projects. Please share our

testbed website with colleagues and investigate our partner use cases and the performance measures that are documented at various scales. We invite others in the community to work with us to develop additional candidate systems and configurations for performance testing. This work will encourage readers to collaborate in moving forward with digital repositories at scale and try Trellis Cassandra for themselves, perhaps using a ready-made Docker compose stack configuration. The ideal outcome for this article is to expand the community of interest around the DRAS-TIC research initiative and Trellis LDP, thereby fostering more adoption, collaboration, and software sustainability.

Author Contributions: Writing, G.J., R.M.; Testing and Metrics, G.J. and W.T.; Software, A.C., A.S. G.J.

Funding: This research was funded by the Institution for Museum and Library Services (IMLS).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Institute of Museum and Library Services. National Digital Platform Research Grant LG-71-17-0159-17. Available online: https://www.imls.gov/grants/awarded/lg-71-17-0159-17 (accessed on 28 February 2019).
2. The Digital Curation Innovation Center Website. Available online: https://dcic.umd.edu (accessed on 28 February 2019).
3. The Fedora 5.0 API Specification. Available online: https://fedora.info/2018/11/22/spec/ (accessed on 28 February 2019).
4. The Trellis Cassandra Software Project. Available online: https://github.com/trellis-ldp/trellis-cassandra (accessed on 28 February 2019).
5. The Trellis Linked Data Platform Project Website. Available online: https://www.trellisldp.org/ (accessed on 28 February 2019).
6. The Apache Cassandra Software Project Website. Available online: https://cassandra.apache.org/ (accessed on 28 February 2019).
7. The DRAS-TIC Indigo Repository Software Project. Available online: https://github.com/UMD-DRASTIC/drastic (accessed on 28 February 2019).
8. The DRAS-TIC Project Website. Available online: http://drastic-testbed.umd.edu (accessed on 28 February 2019).
9. "Swarm Mode Overview", Docker Documentation Guide. Available online: https://docs.docker.com/engine/swarm/ (accessed on 28 February 2019).
10. Gatling.io Open Source Documentation. Available online: https://gatling.io/docs/current/ (accessed on 28 February 2019).
11. Logstash 6.6 Product Documentation. Available online: https://www.elastic.co/guide/en/logstash/current/index.html (accessed on 28 February 2019).
12. Grafana Company Website. Available online: https://grafana.com/ (accessed on 28 February 2019).
13. The Fedora Commons Repository 5.0.2 Software Implementation. Available online: https://github.com/fcrepo4/fcrepo4/tree/fcrepo-5.0.2 (accessed on 28 February 2019).
14. The JBoss ModeShape Website. Available online: http://modeshape.jboss.org/ (accessed on 28 February 2019).
15. Fedora Modeshape with Filesystem and 2 Test Workers. Available online: http://drastic-testbed.umd.edu:3000/dashboard/snapshot/bYsCpBWP20xmyXkb3CW4x7qMYx7rW4L5 (accessed on 28 February 2019).
16. Fedora Modeshape with PostgreSQL and 2 Test Workers. Available online: http://drastic-testbed.umd.edu:3000/dashboard/snapshot/cLG3Arsqs1PtVbGUPshhpcCbOUFmftWE (accessed on 28 February 2019).
17. Trellis LDP with PostgreSQL and 16 Test Workers. Available online: http://drastic-testbed.umd.edu:3000/dashboard/snapshot/Re0RiG2dgPbuUBYG7DICAygBQiis4o61 (accessed on 28 February 2019).
18. Trellis LDP with Cassandra and 24 Test Workers. Available online: http://drastic-testbed.umd.edu:3000/dashboard/snapshot/fdV15D6HjP7kjN7wNbonSwUr3TiC6ohI (accessed on 28 February 2019).
19. Trellis LDP with Cassandra and 28 Test Workers. Available online: http://drastic-testbed.umd.edu:3000/dashboard/snapshot/bYsCpBWP20xmyXkb3CW4x7qMYx7rW4L5 (accessed on 28 February 2019).
20. The Java Content Repository (JCR) Java Specification Request (JSR-283). Available online: https://www.jcp.org/en/jsr/detail?id=283 (accessed on 28 February 2019).
21. The Kubernetes Website. Available online: https://kubernetes.io/ (accessed on 28 February 2019).

22. The Amazon AWS Lambda Website. Available online: https://aws.amazon.com/lambda/ (accessed on 28 February 2019).
23. Tanenbaum, A.S.; van Steen, M. Distributed Systems: Principles and Paradigms. 2006. Available online: http://barbie.uta.edu/~{}jli/Resources/MapReduce&Hadoop/Distributed%20Systems%20Principles%20and%20Paradigms.pdf (accessed on 28 February 2019).
24. How Are Write Requests Accomplished? DataStax Documentation for Apache Cassandra 3.0. Available online: https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlClientRequestsWrite.html (accessed on 28 February 2019).