SPECIAL

Scalable Policy-awarE Linked Data arChitecture for prIvacy, trAnsparency and compLiance

Deliverable No 3.5

Scalability and Robustness testing report V2

Document version: 1.0

SPECIAL DELIVERABLE

Name, title and organisation of the scientific representative of the project's coordinator:

Ms Jessica Michel +33 4 92 38 50 89 jessica.michel@ercim.eu

GEIE ERCIM, 2004, route des Lucioles, Sophia Antipolis, 06410 Biot, France

Project website address: http://www.specialprivacy.eu/

Project	
Grant Agreement number	731601
Project acronym:	SPECIAL
Project title:	Scalable Policy-awarE Linked Data arChitecture for prIvacy, trAnsparency and compLiance
Funding Scheme:	Research & Innovation Action (RIA)
Date of latest version of DoW against which the assessment will be made:	17/10/2016
Document	
Period covered:	M18-M27
Deliverable number:	D3.5
Deliverable title	Scalability and Robustness testing report V2
Contractual Date of Delivery:	31-03-2019
Actual Date of Delivery:	31-03-2019
Author (s):	Javier D. Fernández (WU), P.A. Bonatti (CeRICT), U. Milosevic (TF), Jonathan Langens (TF)
Reviewer (s):	Rigo Wenning (ERCIM), P.A. Bonatti (CeRICT), Sabrina Kirrane (WU)
Contributor (s):	U. Milosevic (TF), Jonathan Langens (TF), P.A. Bonatti (CeRICT), Luca Ioffredo (CeRICT), Iliana M. Petrova (CeRICT), Luigi Sauro (CeRICT), Ida Rejeki Siahaan (CeRICT), Sabrina Kirrane (WU)
Participant(s):	WU, CeRICT, TF, ERCIM
Work package no.:	3
Work package title:	Big Data Policy Engine
Work package leader:	TF
Distribution:	PU
Version/Revision:	1.0
Total number of pages (including cover):	41

Disclaimer

This document contains description of the SPECIAL project work and findings.

The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated in the creation and publication of this document hold any responsibility for actions that might occur as a result of using its content.

This publication has been produced with the assistance of the European Union. The content of this publication is the sole responsibility of the SPECIAL consortium and can in no way be taken to reflect the views of the European Union.

The European Union is established in accordance with the Treaty on European Union (Maastricht). There are currently 28 Member States of the Union. It is based on the European Communities and the Member States cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice and the Court of Auditors (http://europa.eu/).

SPECIAL has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731601.

Contents

List of Figures

List of Tables

1 Summary

The aim of this deliverable is to test the scalability and robustness of the SPECIAL platform such that the results can be used to inform future releases of the platform. It also offers an update over the previous release by reflecting on (i) a new evaluation on a larger cluster consisting of 10 nodes, which (ii) considers the latest release of the platform. We also (iii) provide an in-depth evaluation of *PLReasoner*, the SPECIAL's compliance engine.

What is in this deliverable

In this version of the deliverable we pay particular attention to: (i) introducing the general benchmark scenario and the non-functional desiderata, in *Chapter* 1; (ii) setting up the methodology that guides our evaluation, including the preparation of the synthesised test data and the identification of key performance indicators, in *Chapter* 2; (iii) providing an evaluation of the third release of the SPECIAL platform both in terms of performance and scalability, in *Chapter* 3; (iv) specifically performing an in-depth evaluation of the SPECIAL's compliance engine, called PLReasoner, in Chapter 4. Finally, we conclude in *Chapter* 5.

This deliverable builds upon technical requirements from *D1.7 Policy, transparency and compliance guidelines V2*, *D1.8 Technical Requirements V2*, the SPECIAL policy language which is described in *D2.5 Policy Language V2*, the SPECIAL transparency and compliance framework presented in *Deliverable D2.7 Transparency Framework V2* and *D2.8 Transparency and Compliance Algorithms V2*. The System Under Test (SUT) refers to the current third release of the SPECIAL platform, presented in *D3.4 Transparency & Compliance Release*.

What is *not* in this deliverable

Considering the iterative and agile nature of the project, this deliverable is not meant to serve as a complete evaluation of the SPECIAL platform, but rather as a summary of our current tests and results that will be updated regularly as the project advances. Thus, we do not deal here the security aspects, which are subject of the public penetration/hacking challenges in WP5 (*D5.3 Public penetration/hacking challenges*). Note also that the usability testing is provided in WP4 (*D4.4 Usability testing report V2*). Instead, this document aims to describe the performance and scalability tests to be performed in current and future version of the platform.

Similarly, we do not deal with any issue related to compliance checking (based on business rules) of existing Line of Business and Business Intelligence / Data Science applications (described in *D2.7 Transparency Framework V2*). It is worth noting that the implementation and testing plans of the pilots are devoted to WP5 (*D5.1 Processing and aggregation pilot and testing plans V1*, *D5.3 Sharing Pilot and testing plans V2* and *D5.5 Final Pilot implementations and testing plans V3*). The information of this deliverable, and its future versions, will be used to guide these evaluations.

Chapter 1

Introduction

In this chapter we introduce our benchmark scenario by summarising the current functionality and components of the SPECIAL platform, our System Under Test (SUT). Then, we collect requirements and considerations that will guide our benchmark approach, which is presented in the next chapter. Finally, we review relevant state of the art.

1 The SPECIAL platform

One of the core technical objectives of SPECIAL is to implement consent, transparency and compliance mechanisms for big data processing. The SPECIAL platform uses Semantic Web technology in order to model the information that is necessary to automatically verify that data is processed according to obligations set forth in the GDPR (i.e. usage policies, data processing and sharing events, and the regulatory obligations).

As presented in *D1.8 Technical Requirements V2*, the SPECIAL platform consists of three primary components:

(i) *The SPECIAL Consent Management Component* is responsible for obtaining consent from the data subject and representing it using the *SPECIAL usage policy vocabulary* (*D2.5 Policy Language V2*);

(ii) *The SPECIAL Transparency Component* is responsible for presenting data processing and sharing events to the user in an easily digestible manner following the SPECIAL policy log vocabulary (D2.7 Transparency Framework V2); and

(iii) *The SPECIAL Compliance Component* focuses on demonstrating that data processing and sharing complies with usage control policies (*D2.8 Transparency and Compliance Algorithms V2*).

This deliverable specifically focuses on evaluating the scalability and robustness of the SPECIAL transparency and compliance components. Note that the SPECIAL consent management component is mostly related to our efforts on user interaction in WP4 (cf. see *D4.4 Usability testing report V2*).

In *D3.4 Transparency & Compliance Release*, the SPECIAL transparency and compliance components are materialised in a practical implementation of the SPECIAL platform. Therefore, this deliverable will report on the evaluations of the third release of the platform.

The system architecture of our current system is depicted in Figure 1.1.

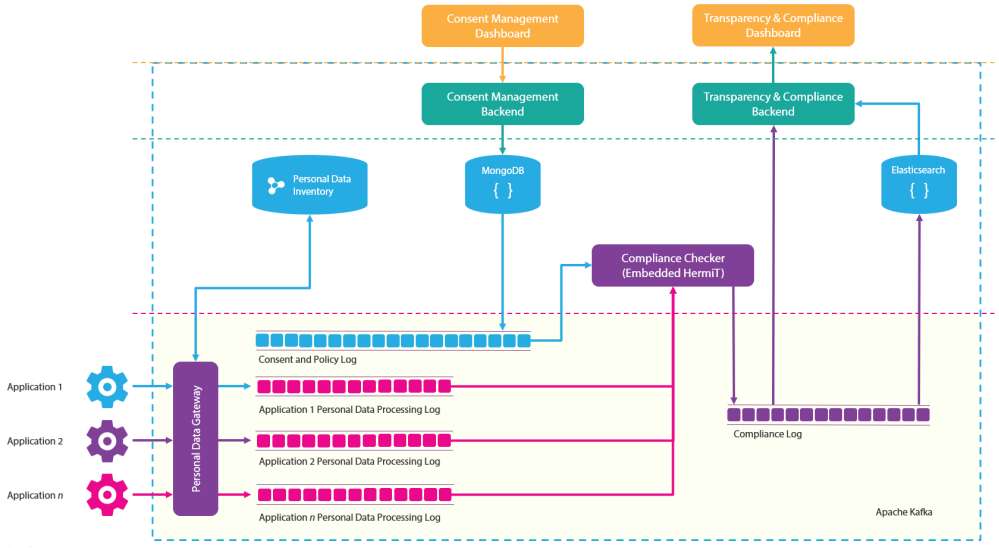

Figure 1.1: SPECIAL-K architecture setup for ex post compliance checking

SPECIAL Transparency Component. Data processing and sharing event logs are stored in the Kafka[1] distributed streaming platform. A Kafka topic is used to store application logs, while a separate compliance topic (called *Compliance Log*) is used to store the enriched log after compliance checks have been completed.

As logs can be serialised using JSON-LD, it is possible to benefit from the faceting browsing capabilities of Elasticsearch[2], and the out of the box visualisation capabilities provided by Kibana.

Compliance Checker. The compliance checker, which currently includes an embedded HermiT[3] reasoner uses the consent saved in MongoDB, together with the application logs provided by Kafka to check that data processing and sharing complies with the relevant usage control policies. The results of this check are saved onto a new Kafka topic. In addition to HermiT, we have developed and integrated a specific SPECIAL's compliance engine, called PLReasoner (see [4] for a description of the algorithms and their complexity analysis). A specific evaluation is provided in Chapter 4.

To the best of our knowledge, no benchmark exists for the GDPR-based compliance and transparency services such as the ones provided by the SPECIAL platform. However, the existence of such systems and benchmarks is of utmost importance to identify shortcomings, optimise the performance and guide future directions. In the following, we provide additional considerations and technical requirements that are relevant in order to benchmark compliance and transparency components emerging from our efforts in SPECIAL, and we review relevant state of the art. Our benchmarking approach and evaluation is presented in the next chapters.

[1] https://kafka.apache.org/
[2] https://www.elastic.co/products/elasticsearch
[3] http://www.hermit-reasoner.com/

Table 1.1: Transparency and compliance services.

Component	Functionalities	Current support in SPECIAL platform (third release)
Transparency component	List the data processing and sharing events that happened	Total
	Find data processing and sharing events by data subject, by consent, by temporal window	Partial (temporal filter is not supported)
	Add data processing and data sharing events to the transparency ledger	Total
	Export the transparency data in an interoperable format	Total
Compliance component	Coherency validation of transparency data and consent data	Total
	Can be called by an access control system for ex-post and ex-ante compliance checking	Total
	Can process the transparency ledger for ex-post compliance checking	Total
	Get statistics for key parameters (#consents, #revocations, #data sharing events, #data processing events ...)	Partial (supported for most parameters)

1.1 Considerations and Technical Requirements

Table 1.1 recalls the services we foresee for the transparency and compliance components (see *D1.8 Technical Requirements V2*), and the current support in the SPECIAL platform (third release - *D3.4 Transparency & Compliance Release*). As can be seen, most of the transparency services are already in place. However, our current prototype only supports basic filtering of processing and sharing events. Our current benchmark, presented in the next chapter, will consider this basic functionality, while more expressive queries, if needed, are deferred to pilot evaluations in WP5. In turn, the compliance component implements the core functionality. Given that our engine performs ex-ante compliance checking using the same core components of the ex-post compliance checking (as described in the third release), we focus on evaluating the core compliance checking mechanisms.

1.1.1 Non-functional requirements

Before discussing the practical benchmark and its results, let us recall and discuss some of the non-functional desiderata presented in *D1.7 Policy, transparency and compliance guidelines V2* (also reviewed by Bonatti et al. [3]) and *D1.8 Technical Requirements V2*:

Storage: Given the volume of events and policies that will need to be handled, the scalability of event data processing is a major consideration. Parameters such as the number of data subjects, the number of consent requests and the number of data processing steps, have a multiplicative effect.

In this respect, as described in *D3.4 Transparency & Compliance Release*, the SPECIAL platform makes use of a specific Kakfa feature, referred to as *log compaction*, which reduces storage needs. In particular, the compliance checker feeds on a compacted Kafka topic which holds the complete policies for all data subjects, where duplicates are removed. We can expect other platforms to use similar features in order to reduce the storage footprint.

It is also worth mentioning that the replication factor of the underlying distributed filesystem can increase the storage needs significantly (but improves the overall fault-tolerance of the system), hence this information is crucial for benchmarking. In our current scenario, we consider a replication factor of two, i.e., data is written to two nodes.

Finally, note that we consider instantaneous data sharing and processing events. In *D2.7 Transparency Framework V2*, we discuss a grouping feature for the events, which is not currently supported by the SPECIAL platform and will be considered as part of future work.

Scalability: Because of the multiplicative effect is it important that the SPECIAL architecture can adapt to larger volumes i.e. via both horizontal and vertical scaling.

As shown in Figure 1.1, the SPECIAL platform runs on proven open source software that is used at large scale by some of the largest companies in the world[4] [5, 6, 14]. In *D3.4 Transparency & Compliance Release*, we provide details on how the system can scale to support a load beyond what a single instance can handle.

Thus, the benchmark tasks should build upon a real-word large-scale scenario, where the ability of the system to scale horizontally and vertically can be validated.

Performance & responsiveness: The total volume of data should only marginally impact the performance and responsiveness of the services. Creating a single data store will destroy the data locality for some services, impacting responsiveness.

As discussed in *D3.4 Transparency & Compliance Release*, Kafka is specifically dedicated for high-performance, low-latency commit log storage. Given its streaming focus (yet it efficiently supports batch oriented workload), the system can perform near real time data processing. Similarly, the SPECIAL transparency component is based on Elasticsearch, which provides efficient query times, heavily relying on the filesystem cache.

Our benchmarking scenario is designed to assure that the SPECIAL platform can cope with such requirements, assuring an overall efficient performance and low latency.

Availability, robustness & long-term applicability: Since transparency and compliance management is bound to a legal obligation, solutions should be guaranteed to work for many years. For personal data, the GDPR calls for a long-term durable solution. If changed, the new system should be capable of importing the existing transparency and compliance data.

The SPECIAL platform makes use of the ability of Kafka to store records in a fault-tolerant durable way. For example, as described in *D3.4 Transparency & Compliance Release*, in case of catastrophic failure where all consumers die, the system can recover the last processed event from a special *state* topic. This prevents redoing work which was already done previously and avoids data loss.

The evaluation of fault-tolerance aspects is deferred to future work.

Security: In addition to the above requirements, all components in the ecosystem must adhere to a general requirement of data security, as it is imperative that a breach of security does not hinder the operations of the systems.

D3.4 Transparency & Compliance Release discusses current authentication and authorisation methods for the SPECIAL platform. While, *D1.8 Technical Requirements V2* identifies data privacy threats mitigation. In this deliverable we do not directly address this aspect, as security aspects will be subject of the public penetration/hacking challenges in WP5 (*D5.3 Public penetration/hacking challenges*).

[4] Elasticsearch use cases:[5]

1.1.2 Considerations for Compliance Checking

SPECIAL policies are encoded using a fragment of OWL2-DL. As discussed in *D2.7 Transparency Framework V2* and *D2.8 Transparency and Compliance Algorithms V2*, the main policy-related reasoning tasks are reduced to subsumption and concept consistency checking. That is, checking whether a data controller's policy P_0 complies with a data subject policy (i.e. consent) P_1 amounts to checking whether the inclusion $P_0 \sqsubseteq P_1$ is entailed by the ontologies for the policy language and the vocabulary.

As mentioned above, and depicted in *Figure* 1.1, our prototype performs the compliance checking on the HermiT reasoner, by default, using OWL API 3.4.3 1. In this deliverable, we also evaluate the PLReasoner, which uses OWL API 5.1.7.

1.2 State of the Art

To the best of our knowledge, no established benchmark covers the identified transparency and compliance operations, summarised in Table 1.1 nor the requirements listed in Section 1.1.1, which are the main objective of the SPECIAL platform. This motivates our proposed benchmark (presented in the next chapter), which covers most of the core operations and requirements, and it is designed to be flexible and extensible in the future.

Nevertheless, much work has been done in benchmarking OWL2 reasoners, which is a central aspects for the compliance component, as discussed above. Traditionally, the elements in OWL benchmarking are classified in *data schema*, *workload* and *performance metrics* [2, 9, 10, 11]. The former mostly refers to the structural complexity of the data schema and the usage of particular ontology constructs. The workload comprises (i) the data generation process, which often produces datasets of different sizes, and (ii) the queries or reasoning tasks to be performed by the reasoner, which should be able to evaluate the inference capability and scalability of reasoner. Finally, the performance metrics describe the quantitative measures that should be evaluated, such as: loading time, which can include different subtasks such as loading ontologies and checking ABox consistency [2], query response time, i.e. the time needed to solve the given reasoning task, completeness and soundness [8].

When it comes to well-established OWL benchmarks, the Lehigh University Benchmark (LUBM) [9] is one of the first and most popular proposals. LUBM considers an OWL Lite ontology with different ABox sizes, where different reasoning tasks of answering conjunctive queries are proposed. The University Ontology Benchmark (UOBM) [11] extends LUBM to include both OWL Lite and OWL DL ontologies and constructs. In turn, Weithöner et al [15] discuss deficiencies and challenges of OWL benchmarks, listing a set of potential requirements such as separating measurements in each step of the process, allowing for different ontology serialisations, or disclosing the reasoners capabilities with respect to query caching.

Recently, the OWL reasoner evaluation (ORE) competition [13] provides different reasoning challenges. ORE is generally based on the tasks of consistency, classification, and realisation, on two OWL profiles (OWL DL and EL). Regarding the data corpus, ORE considers (i) different ontologies submitted by users and (ii) sampled ontologies from different domains.

Chapter 2

Evaluation strategy for the SPECIAL platform

In this chapter we present the benchmark for the GDPR-based transparency and consent we developed in the context of the SPECIAL project, referred to as the *SPECIAL Transparency and Consent Benchmark* (`STC-bench`) hereinafter.

The application scenario considers the SPECIAL BeFit scenario of fitness tracking presented in *D1.7 Policy, transparency and compliance guidelines V2*, which deals with a potential large volume of streaming content, namely location and heart data from BeFit devices.

As we motivated in the previous chapter, there is a lack of benchmarks to evaluate the GDPR-based compliance and transparency services such as the ones provided by the SPECIAL platform. Thus, in addition to serving our evaluation purposes, we expect `STC-bench` to become a valuable asset for similar systems implementing GDPR-based transparency and compliance.

We design `STC-bench` following the same methodology as most of the benchmarks under the H2020 HOBBIT[1] (*Holistic Benchmarking of Big Linked Data*) project [12]. Thus, the design of the benchmark considers three main aspects:

(i) First, we *identify the choke points*, that is, the identified technical difficulties that the benchmark should consider to challenge the system under test (our SPECIAL platform). We present our choke points in Section 1.

(ii) Then, the benchmark *data* is selected. In our case, and given our scenario, we propose a generator of synthetic data, described in Section 2.

(iii) Finally, we design *benchmarking tasks* to cover the identified choke points. Section 3 presents and discusses the current tasks in `STC-bench`.

The `STC-bench` data generator and the results of the evaluation (presented in the next chapter) are publicly available in our website[2], which will be continuously updated with the last results of our tests.

[1] `https://project-hobbit.eu/`
[2] `https://www.specialprivacy.eu/benchmark`

1 Choke Point-based Benchmark Design

We design `STC-bench` following the same methodology as most of the benchmarks under the H2020 HOBBIT project [12]. Thus, the development of the benchmark is driven by so-called "choke-points", a notion introduced by the Linked Data Benchmark Council (LDBC) [1, 7]. A choke-point analysis is aimed at identifying important technical challenges to be evaluated in a query workload, forcing systems onto a path of technological innovation. This methodology depends on the identification of such workload by technical experts in the architecture of the system under test.

Thus, we analysed the SPECIAL platform with the technical experts involved in the SPECIAL policy vocabulary, the transparency and the compliance components. Following this study, we identified the transparency and compliance choke points described below.

Transparency choke points:

CP1 - Concurrent access. The benchmark should test the ability of the system to efficiently handle concurrent transparency requests as the number of users grows.

This choke point mostly affects the *scalability* and the *performance and responsiveness* requirements identified in the previous chapter (see Section 1.1). On the one hand, the system must scale to cope with the increasing flow of concurrent transparency requests. Ideally, the system can dynamically scale based on the work load without interruptions, being transparent to users. On the other hand, the performance and responsiveness (in particular, the latency of the responses) should be unaffected irrespective of the number of users or, at worst, being affected marginally.

In the current third release of the SPECIAL platform, the transparency component relies on Elasticsearch, where different thread pools can be specified[3].

CP2 - Increasing data volume. The system should provide mechanisms to efficiently serve the transparency needs of the users, even when the number of events in the system (i.e. consents, data processing and sharing events) grows.

In this case, in addition to the previous consideration on *scalability* and the *performance and responsiveness*, special attention must be paid to the *storage* requirements and the indexing mechanisms of the system, such that the accessing times do not significantly depend on the existing data in the system (e.g. the number of events).

As mentioned in the previous chapter, the SPECIAL platform makes use of *log compaction* to reduce the space needs (see (*D3.4 Transparency & Compliance Release* for further details). As for Elasticsearch, we use the default configuration, where further inspection on different compression options (e.g. using the DEFLATE algorithm[4]) is deferred to future work.

CP3 - Ingestion time in a streaming scenario. The benchmark should test that the transparency needs are efficiently served in a streaming scenario, i.e. the user should be able to access

[3] See Elasticsearch documentation: `https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-threadpool.html`

[4] See compression in Elasticsearch: `https://www.elastic.co/blog/store-compression-in-lucene-and-elasticsearch`

the information of an event (and the result of the compliance check) shortly after the event arrives to the system.

This choke point implies that no significant delays are introduced (i) by the compliance checker, and, specifically (ii) by the ingestion of the event in the transparency system.

Interestingly, engines such as Elasticsearch are mostly focused on read-intensive operations. Thus, the benchmark should consider this choke point to evaluate whether write-intensive streaming scenarios can become a bottleneck in the system.

Compliance choke points:

CP4 - Different "complexities" of policies. In general, policies can be arbitrarily complex, affecting the overall performance of any compliance checking process. Thus, the benchmark must consider different complexities of policies, reflecting a realistic scenario.

In our case, as discussed in the previous chapter, SPECIAL policies are encoded using a fragment of OWL2-DL, where the main task of the reasoner is to perform subsumption and concept consistency checking. Although this process could be very efficient, the complexity of the policy can be determined by: (i) the number on intersecting concepts in each category (*data*, *processing*, *purpose*, *storage* and *recipients*) of the SPECIAL *Minimum Core Model* (MCM), given that each of them has to be considered to perform the compliance checking, and (iii) the number of *UNION* policies that conform to the user consent, given that the compliance checker must analyse all of them before assuring that one event is not compliant with a given consent.

CP5 - Increasing number of users. The benchmark should test the ability of the system to efficiently scale and perform as increasing number of users, i.e. data processing and sharing events, are managed.

As previously discussed, the current version of the SPECIAL platform relies on Kafka to implement the compliance component. Kafka, can scale both horizontally and vertically, balancing topic partitions between cluster nodes. In this scenario, the benchmark must be able to provide a stress test to evaluate the the performance of the system when the number of users grows and starts to exceed the resource capabilities of the system.

CP6 - Expected passed/fail tests. In general, the benchmark must consider a realistic scenario where policies are updated, some consents are revoked, and others are updated. The benchmark should provide the means to validate whether the performance of the system depends on the ratio of passed/fail tests in the work load.

Note that our current version of the SPECIAL platform preserves the full history of policies and consents. However, the transparency component only considers the last consent of the system users in order to evaluate the compliance of the processing and sharing events.

CP7 - Data generation rates. The system should cope with consents and data processing and sharing events generated with increasing rates, addressing the "velocity" requirements of most big data scenarios.

In our case, Kakfa provides the necessary toolset to deal with real-time streaming needs. However, the capacity of the system is delimited by the infrastructure (the underlying

cluster). The benchmark should be flexible enough to test the capabilities of the deployed system and its *scalability*.

Note also that this choke point is of particular interest for "online users" in ex-ante compliance checking scenarios (as shown in Table 1.1).

CP8 - Performant streaming processing. The benchmark should be able to test the system in a streaming scenario, where the compliance checking should fulfil the aforementioned requirements of *performance and responsiveness* (latency).

Note that the SPECIAL platform is specifically designed to cover such streaming needs. Nonetheless, the benchmark should help in determining the expected latency distribution for a given work load on a supporting infrastructure.

CP9 - Performant batch processing. In addition to streaming, the system must deal with performant compliance checking in batch mode.

In our case, this choke point is particularly interesting as SPECIAL is based on the streaming-based Kafka framework, but it can also manage batch processing. Note that, our initial SPIRIT proposal (see *D2.8 Transparency and Compliance Algorithms V2*), which leverages the SANSA[5] stack for transparency and compliance, is more suitable for batch processing. Nonetheless, all of the SPECIAL uses cases require stream processing.

2 Data Generation

In the following we present the `STC-bench` data generator to test the compliance and transparency performance of the SPECIAL platform.

First, and foremost, note that the data generation should consider two related concepts: the controllers' policies and the data sharing and processing events that are potentially compliant with user consent.

When it comes to the policies, we distinguish three alternative strategies to generate pseudo random policies:

(a) Generating policies in the PL fragment of OWL 2, disregarding the SPECIAL minimum core model (MCM);

(b) Generating random policies that comply to the SPECIAL minimum core model (MCM);

(c) Generating not fully random (i.e. pilot oriented policies) subsets of the business policies.

In this benchmark, we focus on the second alternative, providing a synthetic data generator following the BeFit scenario. Chapter 4 uses the third strategy to specifically perform an in-depth evaluation of the PLReasoner SPECIAL's compliance engine, called PLReasoner.

In addition, the classes in the policies and the log events can come from the standard SPECIAL policy vocabulary, or can be extended with new terms from an ontology. At this stage, we consider the SPECIAL policy vocabulary as the core input.

Thus, the `STC-bench` data generator can produce both policies and data sharing and processing events. The following parameters can be set:

[5] `http://sansa-stack.net`

Table 2.1: Transparency queries for the data subject and the data controller

ID	User	Query
Q1		All events of the user
Q2		Percentage of events of the user passed
Q3		Percentage of events of the user failed
Q4	Data subject	All events of the user passed
Q5		All events of the user failed
Q6		Last 100 events of the user
Q7		All events of the user from a particular application
Q8		All events
Q9		Percentage of events passed
Q10		Percentage of events failed
Q11	Data controller	All events passed
Q12		All events failed
Q13		Last 100 events
Q14		All events from a particular application

- *Generation rate:* The rate at which the generator outputs events. This parameter understands golang duration syntax eg: 1s or 10ms.
- *Number of events:* The total number of events that will be generated. When this parameters is <=0 it will create an infinite stream .
- *Format:* The serialisation format used to write the events (json or ttl).
- *Type:* The type of event to be generated: *log*, which stands for generating data sharing and processing events, or *consent*, which generate new user consents.
- *Number of policies:* The maximum number of policies to be used in a single consent.
- *Number of users:* The number of UserID attribute values to generate.

3 Benchmark Tasks

In the following we present the set of concrete benchmark tasks for the SPECIAL compliance and transparency components. As for transparency tasks, note that the envisioned user stories in *D1.8 Technical Requirements V2* list potential interaction with users, but they are too general to describe functionality to be considered in our current quantitative approach for benchmarking. A qualitative analysis can be deferred to pilot evaluations in WP5.

Thus, we establish here a set of simple tasks to be performed by the SPECIAL transparency component. The transparency tasks are illustrated in Table 2.2. In this case, the system is aimed at resolving *user and controller transparency queries*. Further work is needed to identify the expressivity of these queries. We consider a minimum subset of queries, described in Table 2.1.

In turn, Table 2.3 shows the tasks to be performed by the SPECIAL compliance component in order to cover all choke points identified above. Each task delimits the different parameters involved, such as the scenario (streaming or batch processing), the number of users, etc. These parameters follow the choke points, and their values are estimated based on consultation with the SPECIAL pilot partners. Note that all tests set a test time of 20 minutes, which delimits the number of events generated given the number of users and event generation rate in each case.

Table 2.2: Transparency tasks, all referring to user and controller transparency queries

Task	#Users	Event Rate	Policies	#events	Pass Ratio	Choke Point
T-T2	100 1K 10K 100K 1M	none	UNION of 5 p.	500M events	Random	CP1
T-T3	1000	none	UNION of 5 p.	1M 50M 100M 1B 10B	Random	CP2
T-T4	1000	1 ev./60s 1 ev./30s 1 ev./10s 1 ev./s 10 ev./s	UNION of 5 p.	500M events	Random	CP3

Table 2.3: Compliance tasks.

Task	Subtask	Scenario	#Users	Event Rate	Policies	Test Time	Pass Ratio	Choke Point
C-T1	C-T1-1 C-T1-2 C-T1-3 C-T1-4 C-T1-5	Streaming	1000	1 ev./10s	1 policy UNION of 5 p. UNION of 10 p. UNION of 20 p. UNION of 30 p.	20 minutes	Random	CP4,CP8
C-T2	C-T2-1 C-T2-2 C-T2-3 C-T2-4 C-T2-5	Streaming	100 1K 10K 100K 1M	1 ev./10s	UNION of 5 p.	20 minutes	Random	CP5,CP8
C-T3	C-T3-1 C-T3-2 C-T3-3 C-T3-4 C-T3-5	Streaming	1000	1 ev./10s	UNION of 5 p.	20 minutes	0% 25% 50% 75% 100%	CP6,CP8
C-T4	C-T4-1 C-T4-2 C-T4-3 C-T4-4 C-T4-5	Streaming	1000	1 ev./60s 1 ev./30s 1 ev./10s 1 ev./s 10 ev./s	UNION of 5 p.	20 minutes	Random	CP7,CP8
C-T5	C-T5-1 C-T5-2 C-T5-3 C-T5-4 C-T5-5	Batch	100 1K 10K 100K 1M	-	UNION of 5 p.	100K events 1M events 10M events 100M events 1B events	Random	CP9

4 Key Performance Indicators (KPIs)

In order to evaluate the ability of the SPECIAL platform to cope with the previously described tasks we defined the following key performance indicators (KPIs):

- *Compliance Latency:* the amount of time between the point in which the compliance check of an event was performed and the time when the event was received. In our case, we consider that the compliance check is performed when the result is written to the appropriate Kafka topic storing the results of the process.
- *Compliance Throughput:* The average number of events checked per second.
- *Average transparency query execution:* The average execution time for the query.
- *CPU Usage by Node:* The average CPU usage by nodes in the system.
- *Memory Usage by Node:* The average memory usage by nodes in the system.
- *Disk Space:* The total disk space used in the system.

In addition to these indicators, when the system is deployed in a real-world scenario, the overhead with respect to the Line of Business application can be provided. This indicator can be considered in the future testing plans of the pilots, to WP5 (*D5.3 Sharing Pilot and testing plans V2* and *D5.5 Final Pilot implementations and testing plans V3*).

Chapter 3

Evaluation

This chapter shows the results on the evaluation of `STC-bench` on the current version of the SPECIAL platform (third release - *D3.4 Transparency & Compliance Release*).

In this second version of the deliverable, we focus on compliance, as it is the most data and processing intensive task of the project, showing how `STC-bench` can be applied to measure the capabilities of a particular installation of the SPECIAL platform. The transparency tasks can serve as guidelines for future developments and evaluations of the transparency framework provided within the SPECIAL platform as well as the SPECIAL pilots.

The remaining of the chapter is organised as follows. Section 1 provides details on the specification of the system running the SPECIAL system under test. In Section 2 we perform a first analysis of scaling the number of compliance checking processes. Then, we present the results on the aforementioned `STC-bench` compliance tasks, presented in the previous chapter. An in-depth evaluation and comparison of PLReasoner, the tailored SPECIAL's compliance engine, with respect to HermiT is presented in the next chapter.

1 Experimental Framework

Our experiments were executed against an installation of the third release of the SPECIAL platform (*D3.4 Transparency & Compliance Release*) on a cluster consisting of 10 nodes. Although, it is expected that large-scale companies could provide more computational resources, this installation (i) can serve many data-intensive scenarios as we will show in the results, (ii) is meant to provide clear guidelines on the scalability of the platform, which can help to plan future installations and evaluations.

The characteristic of the cluster are the following:

- *Number of Nodes:* 10.
- *CPUs:* Each node consists of 4 CPUs per machine (2 cores per CPU).
- *Memory:* 16 GB per node.
- *Disk Space:* 100 GB per node.
- *Operating System:* CoreOS 2023.5.0 (Rhyolite).

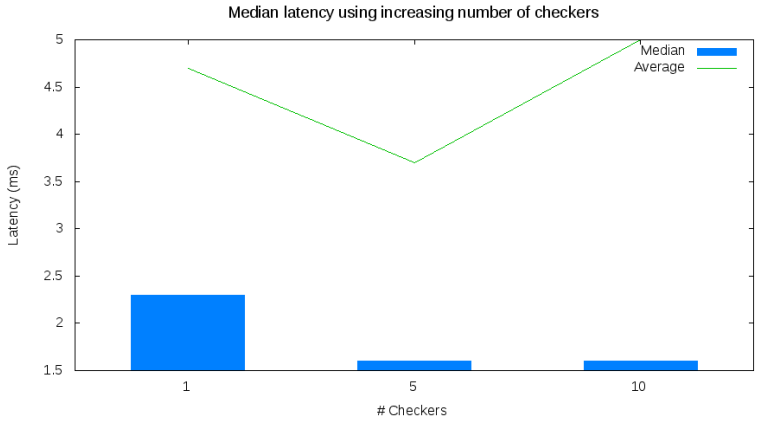

Figure 3.1: Median and average latencies with increasing number of compliance checkers

- *Replication Factor:* 2. As mentioned this implies that data is written to 2 nodes, enhancing fault-tolerance at the cost of additional space requirements and a minimum time overhead.

We report the averaged results of 3 independent executions.

2 Scaling the Compliance Checking Process

Before delving into the concrete results on the `STC-bench` tasks (shown in the previous section) we present here a first study on the scalability of the system with respect to the number of processes executing compliance checking.

As stated in *D3.4 Transparency & Compliance Release*, topics in Kafka are divided into partitions, which are the actual log structures persisted on disk. The number of partitions establishes an upper limit to how far the processing of records can be scaled out, given that a partition can only be assigned to a single consumer (in a consumer group). Thus, the total number of partitions of the application log topic will decide how many instances of the compliance checker can process the data in parallel.

Given the available resources of the cluster, we decided to set up 10 partitions, which puts an upper limit of 10 compliance checkers running in parallel.

As a first evaluation, we show how the system behaves with increasing compliance checkers running in parallel. We perform the test in a streaming (Section 2.1) and batch processing (Section 2.2) scenario.

2.1 Streaming

For this scenario, we evaluate the streaming task *C-T4-4* from `STC-bench`, shown in Table 2.3. Note that the task considers a stream of 1,000 users, where each user generates 1 event every second. That is, we evaluate an event stream that, on average, generates *1 event every 1ms*, producing a total of 1,200,000 events. Given that we expect a performance on the level of ms per check, the streaming flow is close to the limit of one compliance checker.

Figure 3.1 shows the median and average latencies (in milliseconds, with logarithm scale) with different number of compliance checkers in parallel, ranging from 1 to 10 (with 10 being

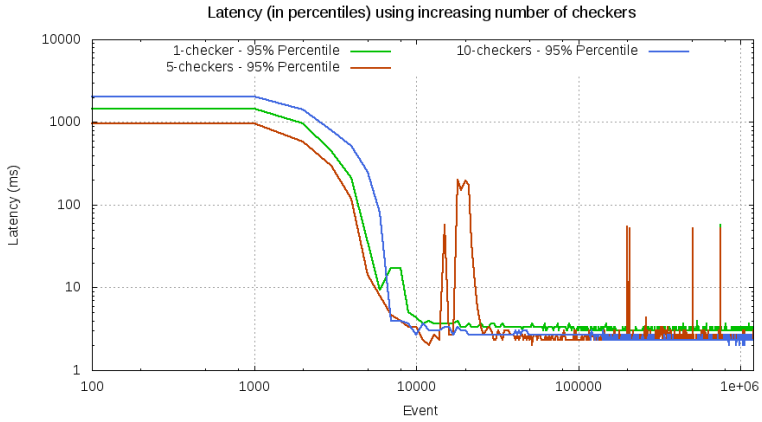

Figure 3.2: Latencies (in 95% percentile) with increasing number of compliance checkers (1, 5, 10 checkers)

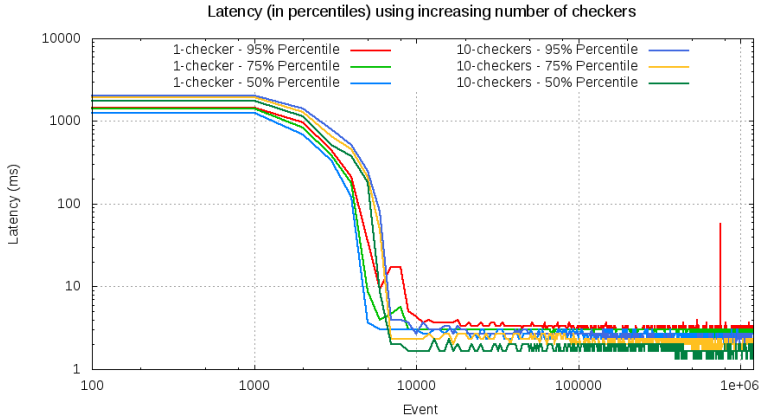

Figure 3.3: Latencies (in 95%, 75% and 50% percentile) with increasing number of compliance checkers (1, 10 checkers)

the upper limit defined by the number of partitions as explained above). Note that the median is usually preferred to the average given that the latency distribution can be skewed. Results show that the (median) latency is always at the level of milliseconds (in particular, less than 2.5 ms), with a noticeable improvement when more compliance checkers are running in parallel, providing a *stable latency of 1.5 ms*. As expected, the slightly higher average figures denote the expected skewed distribution.

Given this behaviour, we inspect the percentile latency, i.e, the value at which a certain percentage of the data is included. Figure 3.2 represents (in milliseconds and logarithm scale) the latency at 95% percentile, using 1, 5 or 10 parallel checkers. For instance, a value of '100' ms means that 5% of the events have a latency greater than or equal to '100' ms. The distribution of 95% percentiles first shows an initial *warm-up* effect, with higher latencies until the first 10,000 events. Then, the latencies are stable with 1-2 ms in all cases, even at the high streaming rate of 1 event every ms. That is, in general, only 5% of the events can experience latencies over 1-2 ms. As expected, latencies are slightly greater if only 1 checker is used. It is worth noting

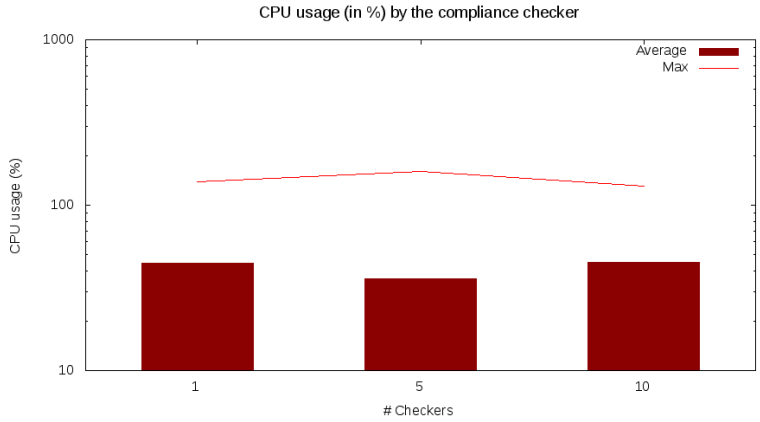

Figure 3.4: CPU usage (in %) with increasing number of compliance checkers

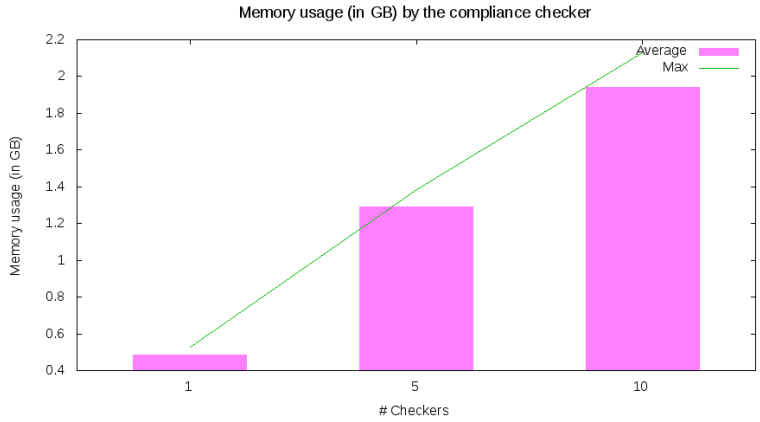

Figure 3.5: Memory usage (in GB) with increasing number of compliance checkers

that the evaluation uncovered a recurrent peak with 5 checkers around 20,000 events, which is subject of future inspection.

Figure 3.3 completes this analysis, depicting 50, 75 and 95% percentiles for the extreme cases of having 1 or 10 checkers in place. In this case, the 50 and 75 % percentiles are close to the 95%, which reflects that most of the data is in the range of the 95% percentile.

In the following, we evaluate the CPU usage (in percentage) and memory usage (in GBs) with increasing number of parallel compliance checkers (1, 5 and 10), shown in Figures 3.4 and 3.5 respectively. We report the average and the maximum number.

Results show that, thanks to latest improvements in the third release of the platform, (i) memory usage increases sublinearly (and remains under 2 GBs) as more parallel compliance checkers are running in parallel, and (ii) CPU consumption remains stable around 50%, with no major influence of the number of checkers. Both results show that Kafka is able to optimise the use of resources and to adapt to the number of parallel checkers. In addition, it is worth mentioning that Kafka is able to add compliance checkers dynamically.

Overall, although different application scenarios can have highly demanding real-time requirements, we expect that these figures, e.g. serving a 95% percentile latency of 1-2ms with an

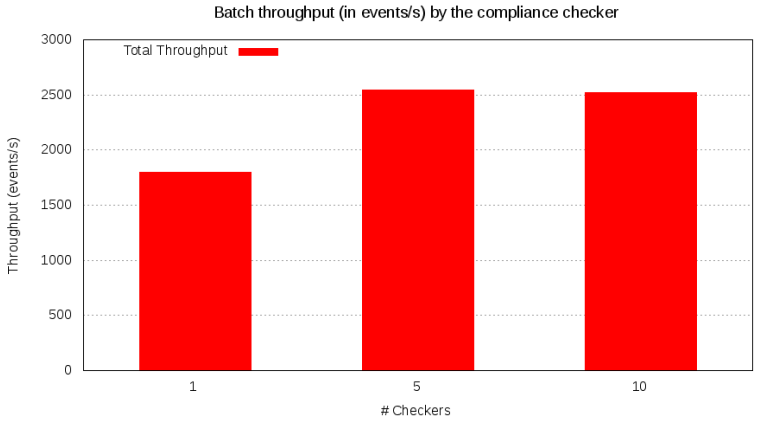

Figure 3.6: Total batch throughput (in events/s) by the compliance checker with increasing number of compliance checkers

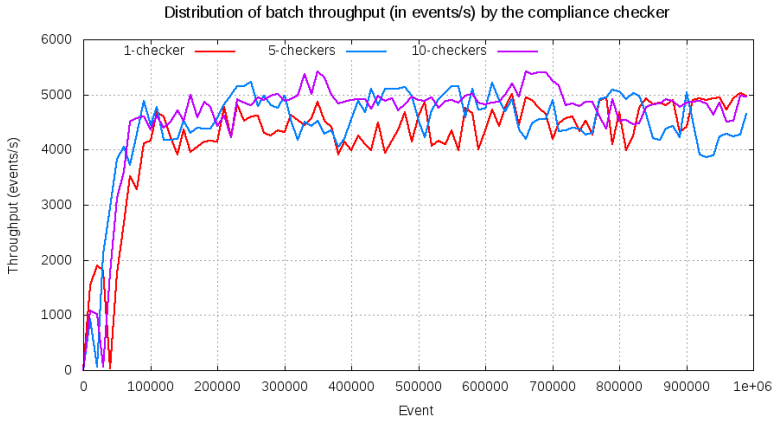

Figure 3.7: Distribution of batch throughput (in events/s) by the compliance checker with increasing number of compliance checkers

event stream of 1 event every 1ms, can cover a wide range of real-world scenarios. Recall that the limit of 10 parallel compliance checkers is solely bounded to the number of partitions in the installation, which depends on the resources of the cluster.

2.2 Batch processing

As stated in choke point CP9, the system must also deal with performant compliance checking in batch. Thus, we repeat the previous analysis looking at different number of compliance checkers for the case of batch processing. To this aim, we evaluate the batch task *C-T5-2* from `STC-bench`, shown in Table 2.3. This task considers 1,000,0000 events that are already loaded in the system. Given that we process events in batch, we inspect the provided throughput (processed events per seconds) using an increasing number of compliance checkers.

Figure 3.6 shows the total batch throughput (in events/s) for 1, 5 and 10 compliance checkers running in parallel. Similarly to the streaming scenario, the performance is improved significantly as more instances are running concurrently. In this case, we can observe a sublinear

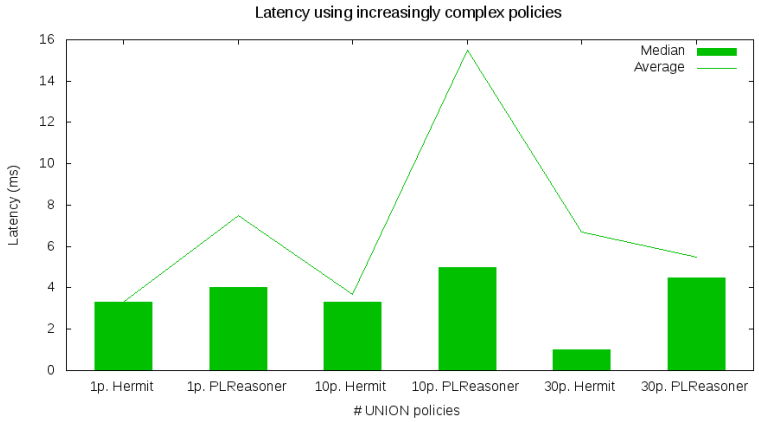

Figure 3.8: Median and average latencies with increasing complex policies

behaviour, where the throughput ranges from 1796 events/s with 1 checker to 2523 events/s with 10. The difference between 5 and 10 checkers is negligible.

Figure 3.7 shows the distribution of batch throughput (in events/s) across time, for 1, 5 and 10 compliance checkers. Results are consistent with the throughput reported above, showing a general constant behaviour and a better performance with 5 and 10 checkers running in parallel.

3 Results on STC-bench Compliance Tasks

This section provides results on the `STC-bench` tasks, shown in the previous section. As mentioned above, rather than showing a complete evaluation on an optimised and performant infrastructure, we focus on testing an installation of the SPECIAL platform and pinpointing good spots for optimisation.

We limit our scope to the functionalities provided by the current third release of the SPECIAL platform and the scaling capabilities of the infrastructure (see the specifications in Section 1). In the following we present the results for all the compliance tasks (*C-T1* to *C-T5* from Table 2.3). We disregard *C-T3* as no significant differences were found in our tests and we opt for a more realistic random generation of policies. The description of each task and subtask is provided in the previous chapter (see Table 2.3).

3.1 C-T1: Different Complexities of Policies

Recall that this task regards the behaviour of the system in a streaming scenario (at 1 event/10s per user and 1K users) when different complexities of policies, measured as the number of union policies, are considered. In this scenario, we make use of 1 compliance checker in order to isolate the performance of one instance. We also compare the implementation of the Hermit reasoner with our engine PLReasoner.

Figure 3.8 shows the median and average latencies (in milliseconds) with 1, 10 and 30 union policies. Results show that the median *latency ranges between 1.5-5 ms*, with relatively small differences as the number of union policies grows, except for the union of 30 policies. In this case, the higher number of union policies allows Hermit to quickly find a match (1.5 ms).

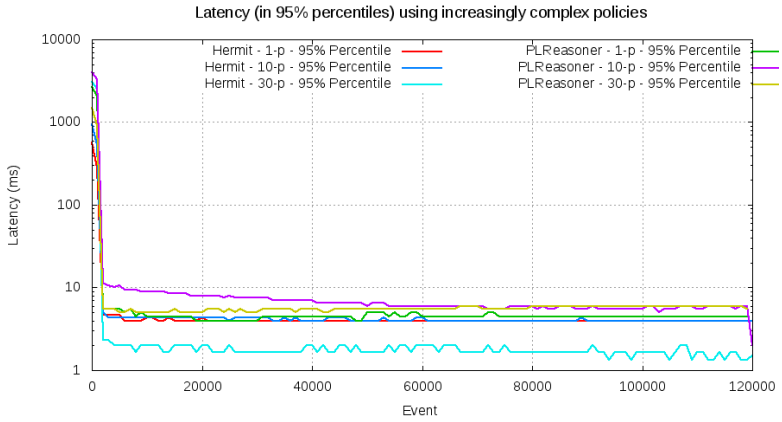

Figure 3.9: Latencies (in 95% percentile) with increasing complex policies

As for the comparison of reasoners, Hermit seems to slightly outperform PLReasoner in the scenario under test. Nonetheless, Chapter 4 provides a much more detailed comparison of both in isolation, showing different figures. The study of this discrepancy, revealed in these results, is subject of future work. A first analysis shows that different parsing and deserialisation of policies can affect the times of PLReasoner in the SPECIAL platform. In addition, our isolated study generally considers richer and more complex policies than STC-bench, also including different time intervals (for the duration of the storage). As shown in the next chapter, in that cases, the tailored PLReasoner engine is several times faster than Hermit.

Finally, the higher figures for the average latency again denote a skewed distribution. Thus, we inspect the latency at 95% percentile (the value at which 95% of the data is included), depicted in Figure 3.9 for 1, 10 and 30 policies. The distribution shows that, in all scenarios, the latency at 95% percentile is stable after the warm-up, with small differences with more union policies. Results also show that only 5% of the events can experience latencies over 5 ms.

3.2 C-T2: Increasing Number of Users

The second task in `STC-bench` focuses on evaluating the scalability of the system with increasing number of users, from 100 to 1 million. These users are considered to be generating events in parallel, each of them at a rate of 1 event every 10 seconds. In the following evaluation, we study the first four subtasks, covering up to 100,000 users given the characteristics of the experimental infrastructure (see Section 1). Note that serving 100,000 users at the aforementioned rate already implies to manage a stream of 10,000 events every second. In this scenario, we consider 10 compliance checkers (see Section 2.1) running in parallel in order to cope with such demand. As mentioned above, we expect that this evaluation can serve as a baseline to shed light on the potential of the SPECIAL platform, guiding our current efforts.

Figure 3.10 shows the median and average latencies for 100-100,000 users. Results show that the system is able to provide a median latency of less than 1ms with 1,000 users (each user with 1 event every 10 seconds, hence overall the system receives 1 event every 10 ms simultaneously), and 1.6ms with 10,000 users (overall, 1 event very ms). However, with 100,000 users, the current infrastructure needs to manage 1 event every 0.1ms (less than the checking

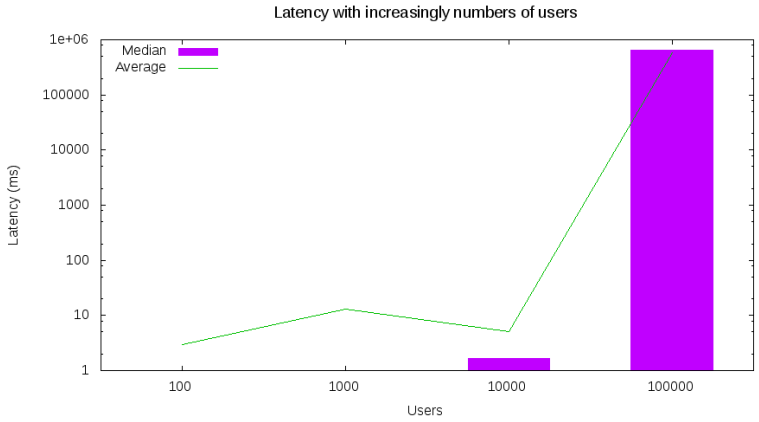

Figure 3.10: Median and average latencies with increasing number of users

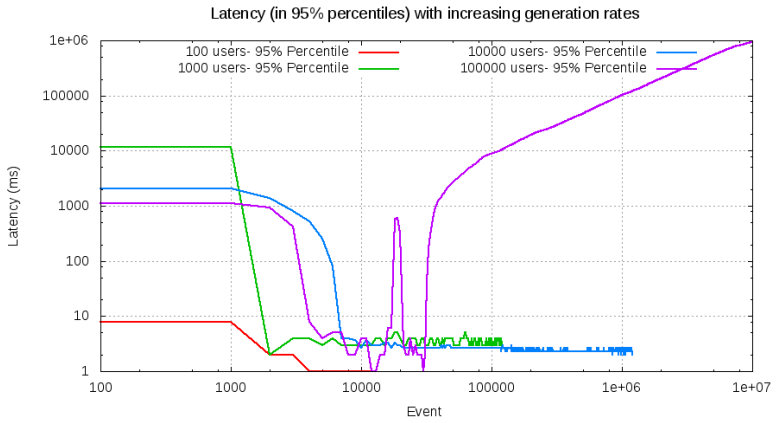

Figure 3.11: Latencies (in 95% percentile) with increasing number of users

time of 1ms), which causes delays of several seconds.

In order to highlight potential worst-case scenarios, we represent the latency at 95% percentile in Figure 3.11. Note that an increasing number of users results in more events, hence the different number of events in each scenario. As expected, results show two different scenarios. On the one hand, a number of users between 100-10,000 results in a *95% percentile around 1 ms*, with an initial warm-up step that produces higher latencies. On the other hand, a higher number of users (100,000) leads to increasing latencies as the number of events grows, i.e. events are queued for several seconds. The main reason is that the number of compliance checkers (10, given the amount of computational resources in the cluster) cannot cope with the overall actual ratio of 10,000 events every second. As discussed in Section 2, Kafka is able to optimise and adapt to the number of parallel checkers, which is solely limited by the number of partitions in the cluster, hence a more powerful infrastructure could cope with a greater number of users.

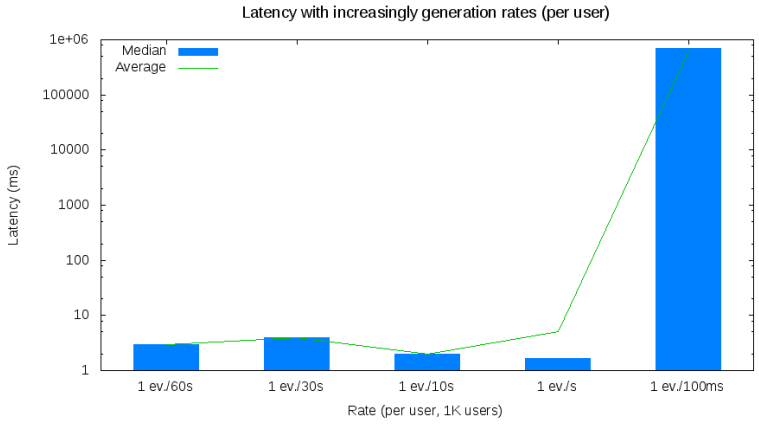

Figure 3.12: Median and average latencies with increasing generation rates. The rate refers to events per user, with 1K users

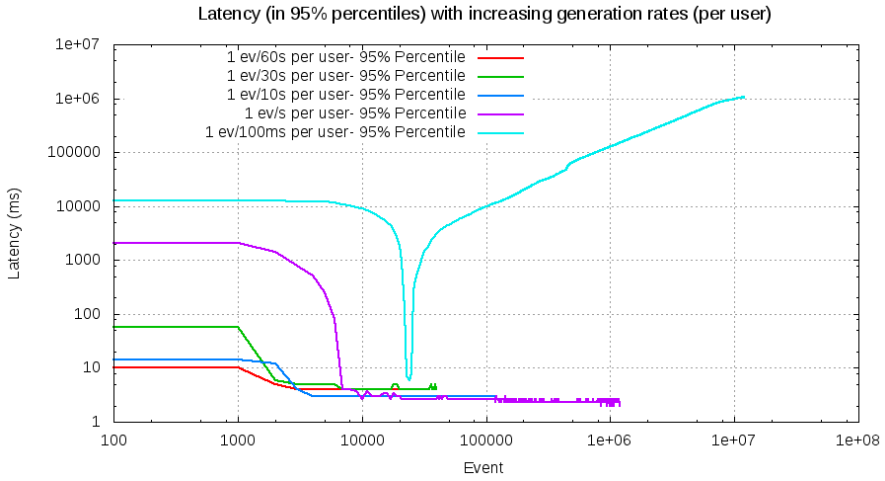

Figure 3.13: Latencies (in 95% percentile) with increasing generation rates. The rate refers to events per user, with 1K users

3.3 C-T4: Increasing Data Generation Rates

This task evaluates the performance of the system with increasing streaming rates. We consider 10 compliance checkers running in parallel in order to try to cope with the biggest rates in the defined tasks.

Figure 3.12 represents the median and average latencies (in milliseconds and logarithm scale), while the latency at 95% percentile is shown in Figure 3.13 (in logarithm scale). Several comments are in order. First, note that the median values in Figure 3.12 are consistent with our previous latency measures (Sections 2.1 and 3.1), obtaining values between 1-2ms for rates up to 1 ev/s (per user). Then, as expected, the median latency increases up to several seconds at the highest rate of 1 ev/100ms per user, that is, the system receives a total of 1 ev/0.1ms.

The huge skewed distribution for the highest rate is revealed by the 95% percentile shown in Figure 3.13. Note that we fix the benchmark time at 20 minutes, so more events are generated

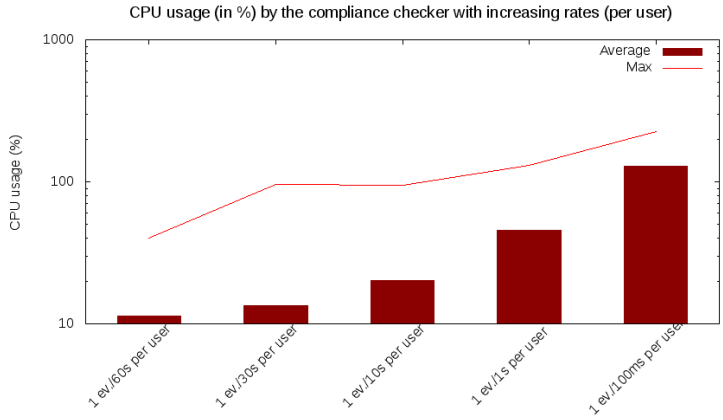

Figure 3.14: CPU usage (in %) for compliance checking with increasing generation rate (1K users)

Table 3.1: Space requirements (MB) with increasing generation rate.

# Users	Event Rate (per user)	# Events	Disk Space (MB)
1,000	1 ev./60s	20,000	819
1,000	1 ev./30s	40,000	1,563
1,000	1 ev./10s	120,000	1,954
1,000	1 ev./1s	1,200,000	5,355
1,000	1 ev./100ms	12,000,000	59,664

with increasing generation rates. Results shows that, the latency reaches a stable stage for rates up to 1 ev/1s per user, i.e. a total of 1 ev/1ms. In contrast, the latency at 95% percentile grows steadily for streams at 1 ev/100ms per user. This fact shows that the current installation cannot cope with such high rates and new events have to queue until they can be processed. The maximum latency reaches 17 minutes for 12 million events.

Finally, in this case, we also inspect the CPU usage and the overall disk space of the solution. The CPU usage (in percentage) is represented in Figure 3.14. As expected, the results show that the CPU usage increases (but sublinearly) with the generation ratio. The disk space requirements are given in Table 3.1. It is worth mentioning that the disk space depends on multiple factors, such as the individual size of the randomly generated events, the aforementioned level of replication, the number of nodes and the level of logging/monitoring in the system. The reported results already show the log compaction feature of Kafka as, on average, less bytes are required to represent each of the events with increasing event rates.

3.4 C-T5: Batch Performance

Recall that this task considers a batch processing scenario, i.e. events are already loaded in the system, with increasing number of events and users. In this evaluation, we consider the first three subtasks, testing up to 10 million events[1] (considering 100K events per user). We

[1]Note that the system is able to generate and process an arbitrary number of events in batch. Further results can be found in our companion website.

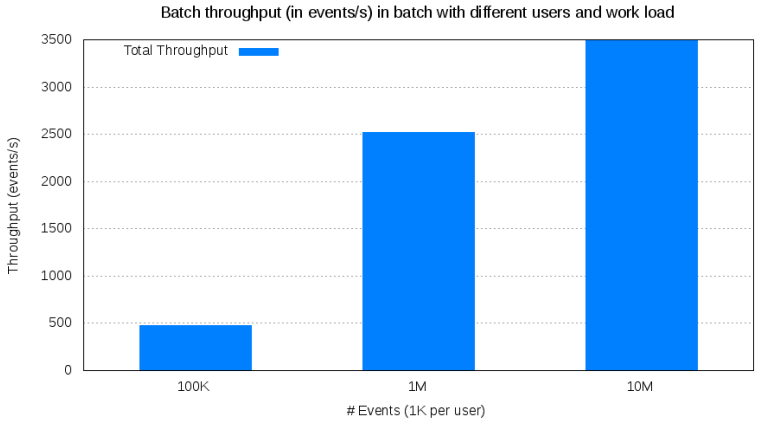

Figure 3.15: Total batch compliance checking throughput (in events/s) with increasing number of compliance checkers

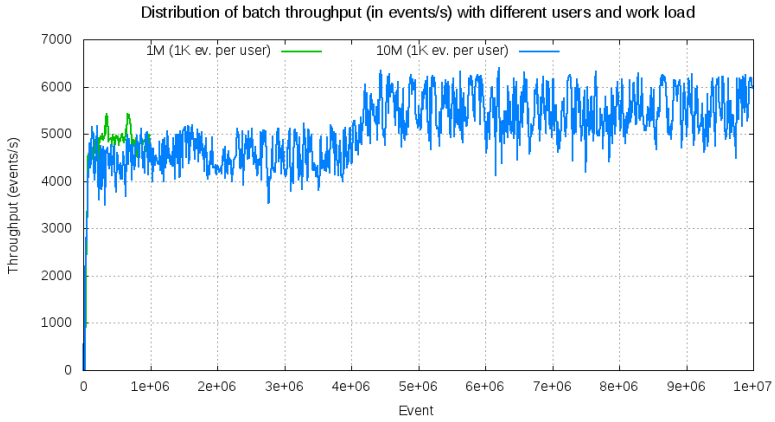

Figure 3.16: Distribution of batch compliance checking throughput (in events/s) with different users and work load. We consider 1000 events per user

inspect the provided throughput (processed events per seconds) using an increasing number of compliance checkers. As in previous cases, we here consider 10 compliance checkers running in parallel.

Figure 3.15 shows the total batch throughput (in events/s) for 100K, 1M and 10M events. The total throughput increases with the number of events, being over 474 processed events/s in all cases, with a maximum of 3,489 events/s in the case of 10M events.

Finally, Figure 3.16 looks at the distribution of the throughput for the case of 1M and 10M events. Both cases shows similar initial figures, with increased performance around 4M events.

Chapter 4

Compliance Checking

This section is devoted to the evaluation of SPECIAL's compliance checker, called *PLReasoner*, in terms of performance and scalability. This engine is compared with the general engine Hermit for OWL2-DL.

SPECIAL's engine is tested on two randomly generated sets of inputs. The first set is based on the actual knowledge base and policies developed for Proximus and Thomson Reuters. Consent policies are generated by modifying the business policies, mimicking a selection of privacy options from a list provided by the controller. This first set of test cases is meant to assess the performance of the engines in the application scenarios that we expect to arise more frequently in practice. The second set of experiments, that makes use of larger knowledge bases and policies, is meant to predict the behaviour of the engines in more complex scenarios, should they arise in the future.

The next section provides more details on test case generation. Then we briefly describe the implementation of SPECIAL's reasoner and its optimisations. The third section reports the experimental results.

1 Test case generation

The first set of test cases is derived from the business policies for Proximus and Thomson Reuters. In each compliance check $\texttt{SubClassOf}(P_B, P_C)$, P_B is a union of simple business policies randomly selected from those occurring in the pilots' policies (as if such simple policies were randomly associated to the business processes that trigger the compliance checks). The consent policies P_C are the union of simple policies randomly selected from the pilots, that are randomly perturbed by replacing some vocabulary terms with a different term. The modified terms simulate the opt-in/opt-out choices of the data subject in the following sense: if the term in P_C is a superclass (resp. a subclass) of the corresponding term in the original business policy, then the data subject opted for a broader (resp. more restrictive) usage option. In this set of experiments, the knowledge base is essentially the union of the ontologies reported in deliverable D2.5.

In the second set of experiments, both the ontologies and the policies are generated randomly, in order to set up a stress test where the increasing size of policies and ontologies allows to verify the scalability of SPECIAL's reasoner and Hermit. Five ontologies of increasing size are generated, using the following parameters:

Ontology	classes	object prop.	data prop.	max children	classification height
OP1	5	5	10	2	5
OP2	50	50	20	6	10
OP3	500	500	30	10	15
OP4	5000	5000	40	14	20
OP5	50000	50000	50	18	25

The column named "max children" sets the maximum number of direct subclasses that a class may have, while "classification height" is the maximum length of the chains of subclasses $A_1, A_2, \ldots, A_n$ such that $A_i \sqsubseteq A_{i+1}$ is in the ontology ($i = 1, 2, \ldots, n-1$). Note that the ontologies for SPECIAL's policy language and vocabularies, illustrated in D2.2, lie between OP1 and OP2. Policies have been generated with the following parameters:

Policy	classes	obj. prop.	data prop.	interval range	nesting	conj.	disj.
CP1	5	5	10	[1,365]	4	10	10
CP2	50	50	20	[1,365]	9	20	100

Each row corresponds to a class of generated policies (100 in CP1 and 1000 in CP2). The second, third, and fourth columns specify how many simbols of each sort are used in the policies. "Interval range" specifies the values from which interval endpoints are selected. The next column specifies the maximum nesting level of object properties. The last two columns specify the maximum number of concepts that $\mathscr{OIO}$ and $\mathscr{OUO}$ may contain, respectively. In particular, note that each policy in CP2 may contain up to 100 simple policies. Moreover, an ex-post inspection of the generated policies shows that the number of interval constraints per simple policies (that cause the hardness of reasoning in $\mathcal{PL}$, as proved in [4]) grows up to 14. So the complexity of the subsumption tests over CP2's policies is significantly harder than those arising in the pilots, as required.

2 The implementation of SPECIAL's engine

SPECIAL's compliance engine, called *PLReasoner*, is implemented in Java and it is distributed as a .jar file. PLReasoner supports the standard OWL APIs. The package includes both the implementation of the structural subsumption algorithm STS, and the preliminary normalisation phases, based on the 7 rewrite rules and on the interval splitting method for interval safety illustrated in D2.4.

Several optimisations have been tried and assessed:

1. Two caches for keeping normalised concepts and interval safe queries, respectively. Their purpose is reducing the overhead of the two normalisation phases, by re-using already normalised concepts and queries when possible.

2. A third cache that remembers the pairs of classes that have been found to be disjoint. Its purpose is reducing the cost of knowledge base inspection.

3. Pre-computation of normalisation for all policies. When this is possible (see the discussion at the end of this chapter), then run-time reasoning collapses to the calls to STS, which is typically much faster than normalisation.

3 Performance analysis

The experiments have been run on a server with an 8-cores processor Intel Xeon Silver 4110, 11M cache, 32GB, running Ubuntu 18.04 and JVM 1.8.0_181. PLReasoner has been compared with Hermit 1.3.8.510. We tried a comparison with Pellet, too, but when the randomly generated policies contained empty intervals, then Pellet stopped with a parsing error, instead of treating the expression as an empty class.

When the engine is started, it loads the specified ontology and *classifies* it (i.e., it finds all the implicit subclass relationships implied by the axioms). We start with a comparison of PLReasoner and Hermit in this initial phase (before compliance checking is started). The following results apply to all versions of PLReasoner, since optimisations apply only after classification:

Loading & Classification times for PLReasoner		
Ontology	time (ms)	time ratio PLReasoner/Hermit
Pilots	47	21%
OP1	28	15%
OP2	41	22%
OP3	79	30%
OP4	328	52%
OP5	2087	71%

Now we illustrate the performance results for the experiments based on the pilots. On this test set, the performance of PLReasoner without optimisations is worse than Hermit's. Things change if the caches for normalised expressions are used, and further improve if also the cache for disjoint concepts is applied, too. Here we report the results for the latter configuration:

Test set	TR		PROXIMUS	
Reasoner	PLReasoner with 3 caches	Hermit	PLReasoner with 3 caches	Hermit
# checks	9999		12000	
tot. time (ms)	8605	46039 (+435%)	8470	43984 (+419%)
avg. memory (MB)	613	766 (+25%)	585	767 (+31%)
avg. time for 1 check (ms)	0.86	3.96	0.71	3.67

We will show later that precomputation of normalisation further improves performance.

The stress test – i.e. the second set of experiments, with increasingly large ontologies and policies – shows the advantages of PLReasoner in all of its versions. In the next table, PLR stands for PLReasoner *with no optimisation*. The table shows the behaviour of the two engines on small policies as the knowledge base grows. All times are in milliseconds.

Policy class	CP1									
Ontology	OP1		OP2		OP3		OP4		OP5	
Reasoner	PLR	Hermit	PLR	Hermit	PLR	Hermit	PLR	Hermit	PLR	Hermit
# checks	8400									
tot. time (ms)	3646	21584 +492%	5873	24929 +324%	21585	52892 +145%	8347	377836 +4426%	7747	4952136 +52206%
avg. mem (MB)	154	768 +399%	614	768 +25%	614	768 +25%	614	922 +50%	461	768 +67%
avg. time per check	0.43	2.57	0.70	2.97	2.57	6.30	0.99	44.98	0.92	589.54

Clearly, Hermit's general algorithm quickly slows down as the size of the ontology increases. The non-linear behaviour of PLReasoner is due to the non-uniform structure of the random ontologies, and in particular the number and placement of their disjointness axioms. We are planning more experiments with a larger variety of ontologies, to address this issue, but it is not possible to include them in this deliverable, since large scale experiments (including Hermit) take months to be completed.

The effectiveness of optimisations is illustrated in the next table. We adopt the following notation:

- **PLR**: PLReasoner with no optimisations;
- **PLR 2c**: PLReasoner optimised with 2 caches for normalised concepts and interval-safe queries, respectively;
- **PLR 3c**: PLReasoner optimised with 3 caches: two as in PLR 2c and one for disjoint terms;
- **PLR pre**: all normalisations are pre-computed (given all the policies) and the two caches of PLR 2c are filled in with the results of normalisation before starting the compliance checks. The response times reported are those for the subsequent run-time checks (that involve only STS).

The cells of the next table contain for each test set but those involving CP2 (discussed later) the total time and the average time per compliance check (between parentheses).

Reasoner	PLR	PLR 2c	PLR 3c	PLR pre
TR	137997 (13.80)	44644 (4.46)	8605 (0.86)	1874 (0.19)
Proximus	177283 (14.77)	53776 (4.48)	8470 (0.71)	1849 (0.15)
CP1 - OP1	3646 (0.43)	180 (0.02)	148 (0.02)	22 (0.00)
CP1 - OP2	5873 (0.70)	732 (0.09)	740 (0.09)	333 (0.04)
CP1 - OP3	21586 (2.57)	573 (0.07)	222 (0.03)	93 (0.01)
CP1 - OP4	8347 (0.99)	856 (0.10)	1053 (0.13)	422 (0.05)
CP1 - OP5	7747 (0.92)	703 (0.08)	717 (0.09)	416 (0.05)

The first observation is that the cache for disjoint concepts is effective on the pilot-based test sets but not on the random test sets. This phenomenon is probably a consequence of the low number of disjointness axioms contained in the random ontologies O1–O5, that make the operations on the third cache a useless overhead. The number of disjointness axioms has been kept small in

order to get a better balance between compliance and non-compliance results over the random policies, and in order to limit the number of inconsistent policies.

The second observation is that – as it could be expected – the precomputation of normalised concepts significantly decreases response time during compliance checking. This optimisation, when applicable, is by far the most effective in terms of scalability. Its memory requirements are reasonable: in our test sets the average storage needs are always less than 600 MB.

Concerning the prerequisites for precomputation, all business policies can be easily normalised in advance with the seven rewrite rules illustrated in D2.4, since the inputs of this transformation procedure (i.e. the ontology and the business policies themselves) are relatively stable and known a priori. The crucial part is predicting the intervals occurring in the consent policies, that determine how the intervals in the business policies are to be split. If they are completely undetermined, then interval splitting cannot be precomputed. The resulting performance can be reasonably expected to lie between the response times of PLR 2c and PLR pre (since at least the 7 rewrite rules are applied in advance, while interval splitting is executed during compliance checks).

If, however, the options for the data subjects provide a limited set of durations, established by the controller, then the set of intervals that may occur in the consent policies is known in advance and full precompilation is possible.

In any case, PLReasoner addresses the need of some pilot leaders for real-time compliance checking. On TR's and Proximus' test sets, both "PLR 3c" and "PLR pre" can process a single compliance check in less than a millisecond. By processing multiple compliance checks in parallel, it should be easy to process a compliance check in a few tens of microseconds. Parallelization, in this scenario, involves minimal synchronisation overhead because each compliance check is independent from the others; just distributing the compliance checks over $\sim$10 processors should speed up response times by approximately one order of magnitude. Another option is the direct compilation of the engine on object code interpretable by hardware (as opposed to Java bytecode). Java provides such compilation facilities; alternatively, one may consider re-engineering the system with a different language, such as C++. The drawback of the latter approach is that currently there are not any analogues of the OWL APIs for C++, in terms of functionality and adoption.

We are left to illustrate the behaviour of the engines as policies become more complex. We do this by running PLR (PLReasoner without optimisations) and Hermit on ontology OP1 and on the class of policies CP2. Recall that the policies in CP2 are one order of magnitude larger than those in CP1. As a consequence, also the number c of interval constraints per simple policy may reach higher values. The complexity analysis of PLReasoner tells us that interval splitting may inflate business policies exponentially as c grows. The theory tells us also that this is inevitable (unless P=NP) since general subsumption in $\mathcal{PL}$ is coNP-hard. The graph in Figure 4.1 shows the performance of PLR and Hermit as $\max c$ grows, where $\max c$ is the maximum number of intervals occurring in a simple policy. The same data are plotted in Figure 4.2 to highlight the relative speed of the two engines.

PLR significantly outperforms Hermit until $\max c$ reaches 8. Then the probability of triggering the combinatorial explosion becomes high enough to generate concepts with 1,814,400 disjuncts ($\max c = 8$) and 2,156,544 disjuncts ($\max c = 8$). When $\max c = 10$, some policies have over 10M disjuncts; PLR's normalisation goes out of memory in 22% of these test cases. Recall, however, that SPECIAL's policies have by definition $\max c \leq 1$.

Figure 4.3 shows the performance of the optimised versions "PLR 2c" and "PLR pre" of

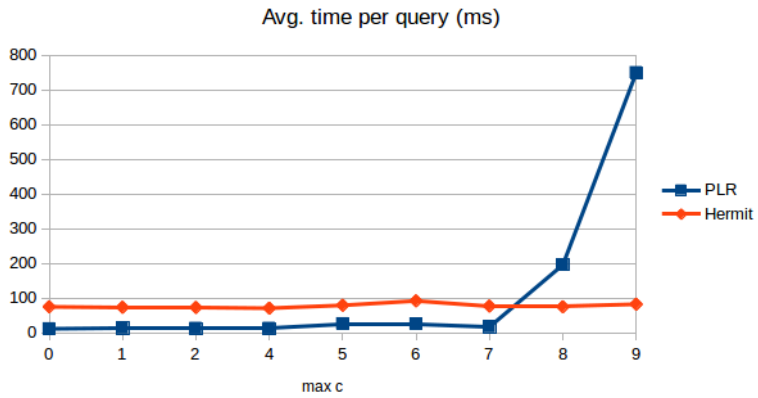

Figure 4.1: Relative performance PLR/Hermit on larger policies

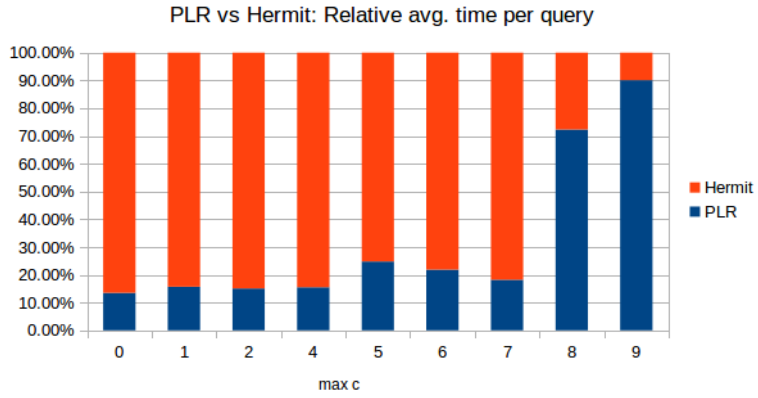

Figure 4.2: Relative performance PLR/Hermit on larger policies

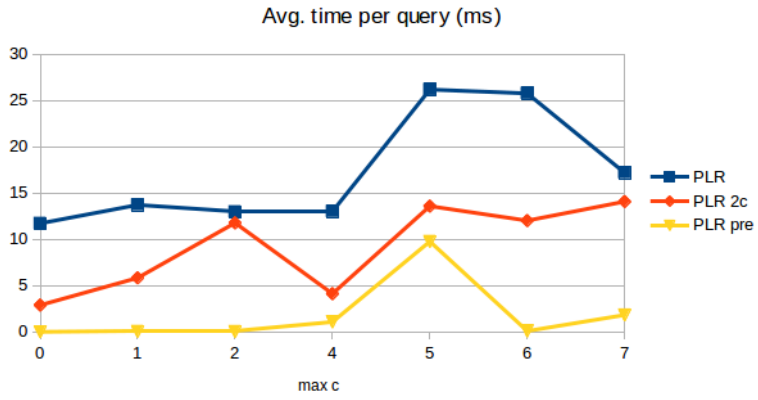

Figure 4.3: Effectiveness of optimisations on larger policies

PLReasoner, for $\max c = 1, \ldots, 7$.[1] The experiments confirm the effectiveness of these two optimisations. The numeric details are reported in the following tables:

			Avg. time per query (ms)			
max *c*	business policies	consent policies	PLR	PLR 2c	PLR pre	Hermit
0	4	700	11.721	2.890	0.004	75.055
1	2	700	13.711	5.831	0.049	73.714
2	1	700	12.961	11.770	0.033	73.054
4	4	700	13.006	4.122	1.074	71.013
5	6	700	26.180	13.578	9.759	79.389
6	2	700	25.765	12.019	0.089	92.172
7	1	700	17.183	14.074	1.826	77.129

			Max memory usage (MB)			
max *c*	business policies	consent policies	PLR	PLR 2c	PLR pre	Hermit
0	4	700	614	461	614	614
1	2	700	614	614	461	614
2	1	700	614	614	768	614
4	4	700	461	461	614	614
5	6	700	768	614	614	768
6	2	700	461	614	614	614
7	1	700	461	461	768	614

Note the remarkable efficiency of PLR pre.

[1] PLR 3c is not interesting in this set of experiments, given the negligible size of the ontology OP1.

Chapter 5

Conclusions

This deliverable presents the methodology that guides the scalability and robustness tests of the SPECIAL platform. First, we set up the scenario and discuss some of the non-functional desiderata. Then, we describe our benchmark for transparency and compliance, referred to as `STC-bench`, which (i) is designed on the basis of well-identified choke points (challenges) that would affect the performance of the SPECIAL platform and similar systems, (ii) provides a synthetic data generator that generates SPECIAL policies and data processing and sharing events, and (iii) describes key performance indicators and well-defined transparency and compliance tasks. We expect that `STC-bench` can become a valuable asset beyond SPECIAL for those tools aimed at GDPR-based transparency and compliance.

Then, we provide an evaluation of the third release of the SPECIAL platform on compliance tasks and an infrastructure consisting of a cluster of 10 nodes (each of them with 8 cores, 16GB memory and 100GB disk space per node).

Our evaluation focuses on illustrating the use of `STC-bench` and identifying spots for optimisation of our platform. In particular, our results show that:

- The SPECIAL platform scales (sublinearly) with the number of compliance checkers running in parallel (see Section 2.1), both in a streaming and a batch scenario. Although these results are promising, further work is needed to inspect and optimise the usage of multiple checkers.
- The system in place is able to serve a 95% percentile latency of 1-2ms with an event stream of 1 event every 1ms, which can cover a wide range of real-world scenarios.
- The system presents non-negligible delays (several seconds) when the event generation rate is faster than 1 event every 1ms. We expect to cover this scenario following two complementary strategies: (i) adding computational resources to the cluster, which will increase the number of partitions and thus compliance checkers, (ii) optimising the integration of our tailored reasoner, PLReasoner (discussed below).
- The performance is marginally affected by the increasing complexity of the policies, i.e. where user consent can consist of several union policies.
- The system scales with increasing number of users, but the increased generation ratio can affect negatively the latency as mentioned above.

- The system is able to perform compliance checking in batch mode, obtaining (median) throughputs of up to 3,489 events per second.

These results show several improvements with respect to the previous version of this deliverable. Thus, the third release of the platform significantly improves on compliance checking latency (from a previous latency of 100ms in a stream of 1ev/10ms to 1-2ms in a stream of 1ev/1ms in the latest version) as well as memory management (from 10 to 2 GB) and more stable CPU usage.

Finally, we provide an in-depth evaluation of PLReasoner – i.e. SPECIAL's engine tailored to the fragment of OWL2 used for business policies consent policies and the GDPR, using test cases derived from the business policies for Proximus and Thomson Reuters. The PLReasoner compares favorably with Hermit. In particular:

- Ontology classification time, compliance checking time, and memory consumption are all significantly reduced.
- On pilot-derived tests, the PLReasoner is more than 4 times faster than Hermit, even without pre-computing the normalisation phase. The average time per compliance check is well below a millisecond.
- If normalisation is precomputed, then each compliance check takes less than 200 μ-seconds and gets close to the real-time needs of some pilot leaders.
- The differences between Hermit and the PLReasoner are even wider as the size of the knowledge base grows. The stress tests carried out in the second set of experiments show that, as the ontology gets larger, the PLReasoner can be up to 5 orders of magnitude faster.
- Similarly, as policies get larger, the PLReasoner is significantly faster than Hermit. When the maximum number of intervals per simple policy is below 9, the PLReasoner without precomputation is up to 26 times faster than Hermit. With precomputation, the PLreasoner becomes 5 orders of magnitude faster than Hermit. Beyond 8 intervals per simple policy the intrinsic intractability of $\mathcal{PL}$ subsumption shows its effects and Hermit becomes more efficient. This is due to the cost of interval splitting, needed to guarantee interval safety. Recall, however, that SPECIAL's policies have at most one interval per simple policy. This guarantees that after interval splitting business policies grow at most linearly in the size of the given consent policy.

Overall, we expect that these insights can guide our future research and development steps of the SPECIAL platform. In particular, current results lead us to further investigate on the optimised use of PLReasoner within the SPECIAL platform and to solve or mitigate potential latency peaks and bottlenecks to improve the scalability of the system.

Bibliography

[1] R. Angles, P. Boncz, J. Larriba-Pey, I. Fundulaki, T. Neumann, O. Erling, P. Neubauer, N. Martinez-Bazan, V. Kotsev, and I. Toma. The linked data benchmark council: a graph and rdf industry benchmarking effort. *ACM SIGMOD Record*, 43(1):27–31, 2014.

[2] J. Bock, P. Haase, Q. Ji, and R. Volz. Benchmarking owl reasoners. In *ARea2008-Workshop on Advancing Reasoning on the Web: Scalability and Commonsense*. Tenerife, 2008.

[3] P. Bonatti, S. Kirrane, A. Polleres, and R. Wenning. Transparent personal data processing: The road ahead. In *International Conference on Computer Safety, Reliability, and Security*, pages 337–349. Springer, 2017.

[4] P. A. Bonatti. Fast compliance checking in an OWL2 fragment. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018*, pages 1746–1752, 2018.

[5] L. Engineering. Running kafka at scale, 2015. URL `https://engineering.linkedin.com/kafka/running-kafka-scale`.

[6] N. Engineering. Kafka inside keynote pipeline, 2016. URL `https://medium.com/netflix-techblog/kafka-inside-keystone-pipeline-dd5aeabaf6bb`.

[7] O. Erling, A. Averbuch, J. Larriba-Pey, H. Chafi, A. Gubichev, A. Prat, M.-D. Pham, and P. Boncz. The ldbc social network benchmark: Interactive workload. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 619–630. ACM, 2015.

[8] Y. Guo, Z. Pan, and J. Heflin. An evaluation of knowledge base systems for large owl datasets. In *International Semantic Web Conference*, pages 274–288. Springer, 2004.

[9] Y. Guo, Z. Pan, and J. Heflin. Lubm: A benchmark for owl knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2-3):158–182, 2005.

[10] S. A. Khan, M. A. Qadir, M. A. Abbas, and M. T. Afzal. Owl2 benchmarking for the evaluation of knowledge based systems. *PloS one*, 12(6):e0179578, 2017.

[11] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. Towards a complete owl ontology benchmark. In *European Semantic Web Conference*, pages 125–139. Springer, 2006.

[12] A.-C. N. Ngomo and M. Röder. Hobbit: Holistic benchmarking for big linked data. *ERCIM News*, 2016(105), 2016.

[13] B. Parsia, N. Matentzoglu, R. S. Gonçalves, B. Glimm, and A. Steigmiller. The owl reasoner evaluation (ore) 2015 competition report. *Journal of Automated Reasoning*, 59 (4):455–482, 2017.

[14] B. Svingen. Publishing with apache kafka at the new york times, 2017. URL `https://www.confluent.io/blog/publishing-apache-kafka-new-york-times/`.

[15] T. Weithöner, T. Liebig, M. Luther, S. Böhm, F. Von Henke, and O. Noppens. Real-world reasoning with owl. In *European Semantic Web Conference*, pages 296–310. Springer, 2007.