Intelligent Tools for Policy Design

Deliverable 3.11

FUPOL CORE Platform
Final Core Platform

Project Reference No.	287119
Deliverable No.	D 3.11
Relevant workpackage:	WP 3
Nature:	Report
Dissemination Level:	Public
Document version:	FINAL
Editor(s):	Nikolaus Rumm, Robert Thaler , Bernhard Ortner, Andreas Noack, Michael Kaufmann, Peter Orgon
Contributors:	Peter Sonntagbauer, Herbert Löw, Susanne Sonntagbauer, Mario Neumann, Christopher Haigis, Hakan Kagitcioglu, Peter Mairhofer, Anton Jessner, Alexander Kamenicky, Ilja Hönigschnabel
Reviewers:	Herbert Löw, Oliver Siml, Abdukalikov Bakythzan
Document description:	The objective of this document is to provide a supplement to the final prototype release of the FUPOL Core Platform as of August 2015. The document covers the product on the requirements level, adds descriptions of the new features that were introduced since D3.9 and additional information on tests as long as selected architectural and design decisions.

History

Version	Date	Reason	Prepared / Revised by
0.1	2015-06-22	Initial release, based on D3.9	Rumm Nikolaus
0.2	2015-07-07	Added the HANA parts	Rumm Nikolaus, Robert Thaler
0.3	2015-07-24	Added information on the tests	Ortner Bernhard, Rumm Nikolaus
0.4	2015-08-11	Added information on the QlikView features	Rumm Nikolaus, Andreas Noack
0.5	2015-08-28	Final editing	Rumm Nikolaus, Bernhard Ortner

Table of Contents

Management Summary

The objective of the FUPOL project is the development of a new governance model to support the policy design and implementation lifecycle. The innovations are driven by the demand of citizens and political decision makers to support policy domains in urban regions with appropriate ICT technologies. Those policy domains are very important, since more than 80% of the whole population in Europe lives in urban regions and the share will further grow in the future.

Deliverable D3.11 is the final version of the FUPOL Core Platform as described in this document. Note that due to the nature of this deliverable almost all efforts that were necessary to produce it went into software development and that this document is only a description of what the software does and does not. So in order to benefit most from reading it we strongly suggest to use the software to get a better understanding of the features. There's a user manual available that will guide you.

Focus in year 4 (which ist the project's final year) was put on the review remarks from October 2014, especially on…

- cloud computing
- technical evaluation (benchmark testing)
- focusing on the selling-points

The FUPOL Core Platform is a central module of the FUPOL System, providing services to the FUPOL users and to the other FUPOL modules:

- Centralized access and account management (security, user management)
- Campaign management (support for research activity) including tools like opinion maps and questionnaires
- Client management (support for multi-client operations)
- Data and knowledge management including GIS data, semantic and statistical data

- Social media management including content crawling from Twitter, Facebook and other social media sites
- Operational support (services that support the reliable operations of the FUPOL System like logging)
- Integration services (messaging middleware, service coupling, …)

An important note is that this document covers the FUPOL Core Platform, but not the complete FUPOL System. Thus all requirements mentioned in this software requirements specification, the architecture and the design focus on the core platform. Interactions with the other FUPOL modules are explained on interface level, but lack any further detail, as these have to be specified for the respective modules separately.

In order to fully understand the FUPOL Core Platform we recommend reading…

- D3.6 Revised Requirements Specification and Use Cases (which is the successor of D3.1)
- D3.2 Preliminary Software Design Description to get an understanding of the technical design that realizes the requirements from D3.1
- To a lesser extent we recommend reading D3.7 Test Reports Prototype to get an understanding of some user scenarios and workflows
- …and finally there's our user manual that describes how to use the software product from the civic servant's point of view

This deliverable (D3.11) is based on D3.9, D3.6, D3.2 and D3.7, with parts (i.e. screenshots) taken over from the user manual, too.

There are significant dependencies between the content of D3.11 and other deliverables, especially in work packages WP5 (advanced visualization), WP6 (e-policy idea management and summarization system) and WP4 (policy simulation software) which have been respected to design the FUPOL Core Platform based on the requirements as documented in D3.1/D3.6.

While this document's predecessor (D3.9) was based on the software release from late September 2014 this document is based on release 0.59 from the end of August 2015.

1 Introduction and Goals

This is the description of the final version of the FUPOL Core Platform, having been developed by the team of work package 3 (WP3), based on the project state of late August 2015.

The FUPOL Core Platform is now finished with regard to the DOW, but it's still extended with features that will support future exploitation. The product's quality status can be described as "near production ready" – it's no longer a prototype but not yet fit for commercial customers, both in stability and in features. Most insufficiencies are related to enterprise integration and sales support (i.e. integration with a city's Active Directory for managing users).

This document shall describe what is available now that the project's deadline is near, what can be achieved when using the software and how it was done on a conceptual and technical level.

It's difficult to describe working software using text only, so we added some screenshots to this document, but we recommend using the product to get the whole user experience.

For an introduction to the project and the product under development we suggest to start with D3.10 Final Software Requirements Specification and Use Cases.

Finally there's the user manual which gives detailed instructions on how to use the software, following a very pragmatic and user-centric approach.

This deliverable's structure is based on its predecessor D3.9.

2 The Purpose of the Project

For an introduction to the project's purpose please refer to D3.1.

3 Features

This chapter describes the features that are implemented in release 0.59.

3.1 Product Scope of the Final Version

As already mentioned the core platform is now available as a final version (though minor updates might follow, i.e. to fix bugs should they be found until the end of the project). Some features that we envisaged in the early stages of the project could not be completed at all because they could not be realised for various reasons (i.e. access to Chinese social media data, using semantic web technology, …) or there was no need for them any more (i.e. because we access open data online instead of importing it).

The main changes to the original scope are:

- we preferred online data access over import/store functions (statistical data, geographical data, semantic data …)
- not all social media sources could be integrated (i.e. we were never able to access Chinese social media content for political/legal reasons), but we integrated other sources that were not on our original agenda (i.e. RSS)
- the semantic web technology that we wanted to use in the project could not meet our requirements (i.e. stability, performance) and so we had to replace it with a more traditional data store. However the ontologies are still in use, though only in the form of a relational data model.

For a list of all planned features please consult D3.5 (chapter 3.1). The following table lists the differences.

Id	Feature	Reason
WP3-338	Push campaign data back to the data base	Replaced by accessing Open Data online – no local store required
WP3-337	Pull campaign data from the data base	Replaced by accessing Open Data online – no local store required

WP3-77	Import statistical data	Replaced by accessing Open Data online – no local store required
WP3-76	Link geographical and statistical data	Replaced by accessing Open Data online – no local store required
WP3-75	Import geographical data	Replaced by accessing Open Data online (Google Maps or Open Streetmaps) – no local store required. Public data has advanced to a level that is sufficient for our use cases.
WP3-339	Import semantic data	Not required any more – knowledge base data is accessed online from Open Data sources
WP3-340	Browse data from the data base	Not required any more – statistical and semantic data can be browsed online using SEMAVIS
WP3-62	Publish to social media window	Not implement – regarded as spam by the social media companies and in violation of their T&Cs.
WP3-65	GIS model according to INSPIRE	Not required any more – geographical data is accessed online from Open Data sources
WP3-32	Data import from CORINE 2006 (urban atlas)	Not required – the simulators access their required data online from Open Data sources
WP3-31	LUCAS data import (land use data)	Not required – the simulators access their required data online from Open Data sources or they'll implement their own data import functions (ie.. commercial off-the-shelf simulators)
WP3-2	Support for importing and storing Eurostat data	Not required – accessed online
WP3-80	Anonymized citizen data	Not implemented because it's easy to search for the data using Google or other search engines – however we don't expose the user's data everywhere

WP3-20	Import of Linkedin data	Not implemented – not relevant for citizens
WP3-820	Import if Sina Weibo data	Not implemented for legal reasons – no access to Chinese social media data

A major technical change was applied in the final project year. As the RDS triple store that we used previously didn't perform well enough to meet even the most basic operational requirements we replaced it with a relational data store (Postgres).

The problems that we faced when using Virtuoso were mostly related to its stability, especially when inserting triples/quads concurrently to the same resource. The datastore became unstable:

- availability decreased over time – the response time increased significantly under heavy load and the system was unable to restore normal operations once the load was reduced, so we had to restart it almost daily
- under certain circumstances that we were unable to understand fully the stored data became inconsistent, which was detected and reported by the datastore, leading to labour-intensive clean-up operations
- furthermore we faced issues with Virtuoso's JDBC driver that we had to fix manually

In fact, we had to restart Virtuoso daily and clean-up the database almost every week. So we decided to drop it and replace it by Postgres in D3.9. This move was somehow bold, but necessary, as the sytem's availability was too low for pilot operations.

The downside of using Postgres is of course that the semantic web approach has collapsed with it. We've reused the FUPOL ontology (based on SIOC, FOAF and DC), but its relational representation doesn't allow us to use inference for analyzing the data. Furthermore we had to redesign the API to SEMAVIS, as it was based on SPARQL and replace it with a JSON-based version (though, there were problems with authentication, authorization and data protection when we used SPARQL).

However, an opportunity that happened to the project in year 4 changed the game here and we were able to replace Postgres with an advanced cloud based datastore.

We presented FUPOL to SAP and were encouraged by them to propose it to their „SAP Hana Cloud Accelerator Programme", a marketing initiative that selects innovative products to be supported by SAP – and we won the grant. This move secured valuable resources for the project:

- a SAP HANA cloud platform (HCP) instance (a full grown cloud environment to run FUPOL)
- a SAP HANA in-memory database
- support and training for the project members
- the opportunity to present FUPOL at SAP events, an important source of information for public (IT) stakeholders
- the opportunity to advertise FUPOL in SAP's cloud store, providing an additional sales channel

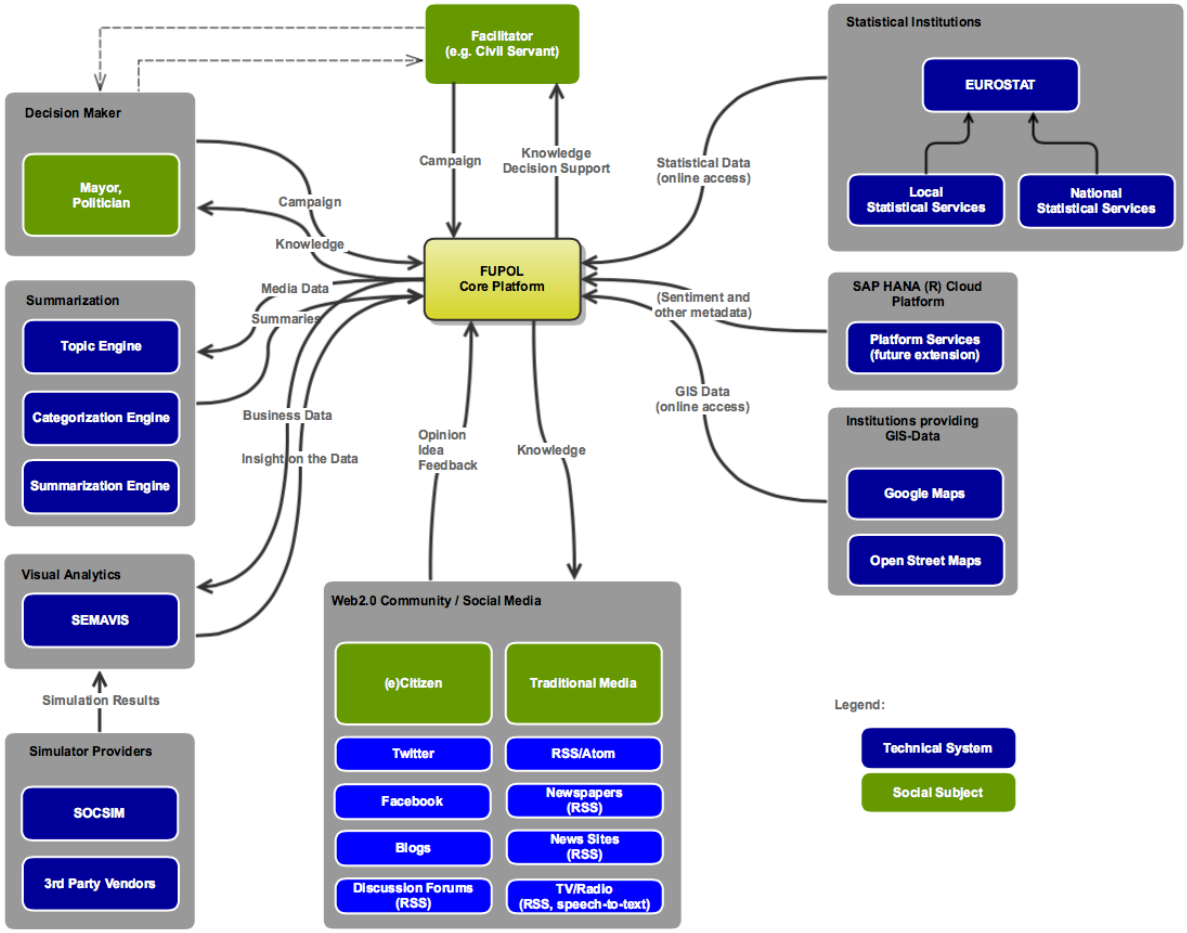
So we decided to migrate FUPOL from the less-advanced Postgres-database to SAP's HANA platform, which boosted the system's performance and realized our cloud computing approach in an elegant way. Migration was performed by Active and cellent and took several weeks. Most of the work was related to tweaks and adaptations of the data access layer (different datatypes, differences in SQL dialect, different Java-versions, encryption, ...).

See D3.10 for an overview of the technical concepts that we use now and – even more so – in the future. We're confident that the move to HCP has opened a treasure chest of features that we'll be able to integrate during exploitation (i.e. sentiment analysis for many languages, better enterprise integration, ...).

3.2 Business Context

For a complete description of the FUPOL core platform's business context, including a representative business event list, read D3.6.

The following diagram illustrates the business context of the FUPOL core platform. Note that this is not the context diagram of the FUPOL system, but just the part of it covering the important aspects of the FUPOL Core Platform.



The changes to the previous prototype are related to...

- the simulators communicate directly with SEMAVIS – the core platform is no longer required for that

- there's no direct connection between the core platform and the simulators any more (besides user authentication)
- text summarization was reintroduced by Xerox and integrated in the core platform
- as the core platform is now hosted on SAP HANA we will add some of the advanced HCP features to the core platform later (i.e. it supports a wide range of languages when analyzing sentiment)

The business context shows other systems and units which are connected to the FUPOL core platform. In the following the interaction between the FUPOL-system and the surrounding systems is described.
For a description of the context diagram's building blocks please refer to D3.9.

The core platform acts as a middleware, connecting the modules from WP2/WP4, WP5 and WP6 to form a common service to the user.

3.3 The Hands-On Users of the Product

For a description of the hands-on users oft he product please read D3.4.

3.4 Development Team

The WP3 core development team is formed of the following people. All members are working on-site in the FUPOL Project Office in Vienna at cellent or Active. Note that as planned the team has been reduced at the start of year three.

Team Member	Role	Relevant Skills	FUPOL Participant
Andreas Noack	Team Member	• Developer (Business Intelligence, QlikView)	• Active
Bernhard Ortner	Team Member	• Developer, Tester	• Active
Nikolaus Rumm	WP Manager	• Architect • Requirements engineer	• Cellent
Robert Thaler	Team Member	• Developer	• Cellent

Besides the development team other individuals performed support tasks (reviewing deliverables, testing, translating, support for exploitation, …).

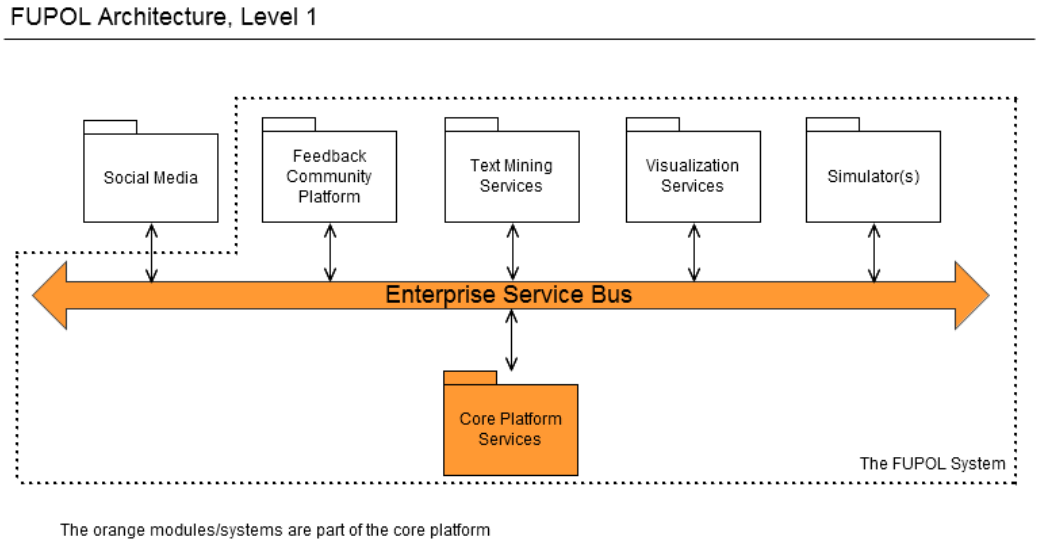
4 Architecture and Design

For an introduction to the architecture we recommend reading D3.5 (chapter 4).

A more detailed description can be found in D3.2.

4.1 Architectural Overview

The actual architecture is based on an enterprise service bus (ESB). Note that most connections between the modules will be (logically) point-to-point, but technically the communication is done through the ESB.

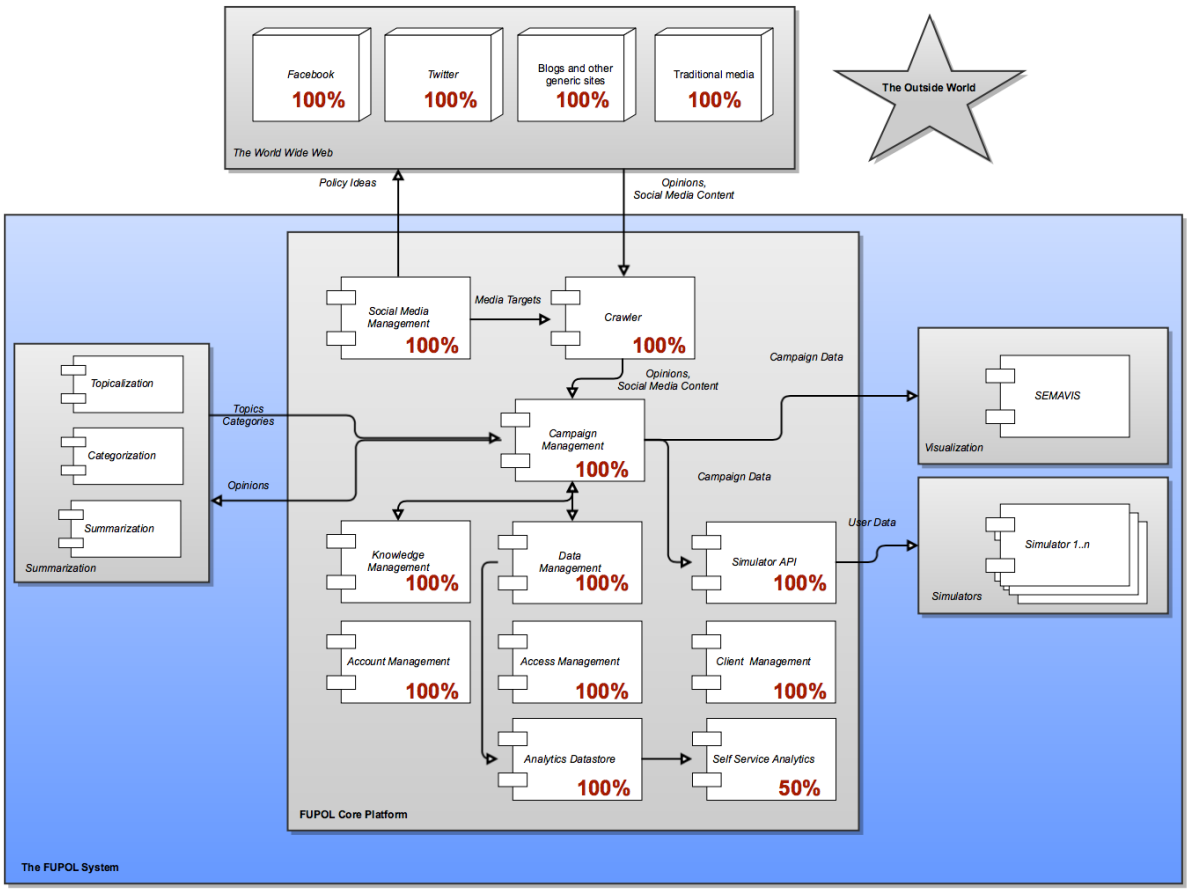


A description of the modules can be found in D3.2.

4.2 Components and Level of Completion

The following diagram provides a bird's eye view on the logical components and the main data flows in the FUPOL system. Please note that this view doesn't represent the choosen architecture but it's here for understanding the relation/interaction between the various FUPOL modules and the data flows between them. The actual design doesn't use point-to-point-connections but instead of that it's based on a SOA architecture using an enterprise service bus.

The numbers indicate an estimation of the feature completeness of these modules (numbers on the interfaces represent the level of the current technical integration) based on release 0.51 from September 2014.



As illustrated in the diagram the core platform is now finished and we'll focus on adding features relevant to selling the solution until the end of the project (i.e. self service BI.

Social media connectivity is unchanged. Content from the following sites can be crawled:

- Facebook (public posts)
- Facebook groups (wall board including comments)
- Twitter advanced search
- Blogspot.com
- RSS/Atom (including http basic authentication)

Unfortunately Facebook has changed its data sharing and privacy policy in May 2015 and subsequently shut down full text search and access to personal Facebook profiles – a fact that was discussed controversially in the social media monitoring community. Thus Facebook can now only be used to monitor group profiles (i.e. profiles curated by cities, politicians, companies, …) and the comments that citizens place there, but not the citizens' personal pages.

The following chapters show some details and screenshots of D3.11. We focused on those parts that are either new (as compared to D3.9) or that had some significant changes.

4.2.1 Text Mining, Topics and Categorization

The social media tools were enhanced by integrating WP6's summarization feature and making it available to the process of category creation.

This was requested by the users, as they found it difficult to fully understand a topic just from looking at a list of related keywords.

Now the user can access a WP6-generated summary, which selects representative sentences from the content that defines the category.

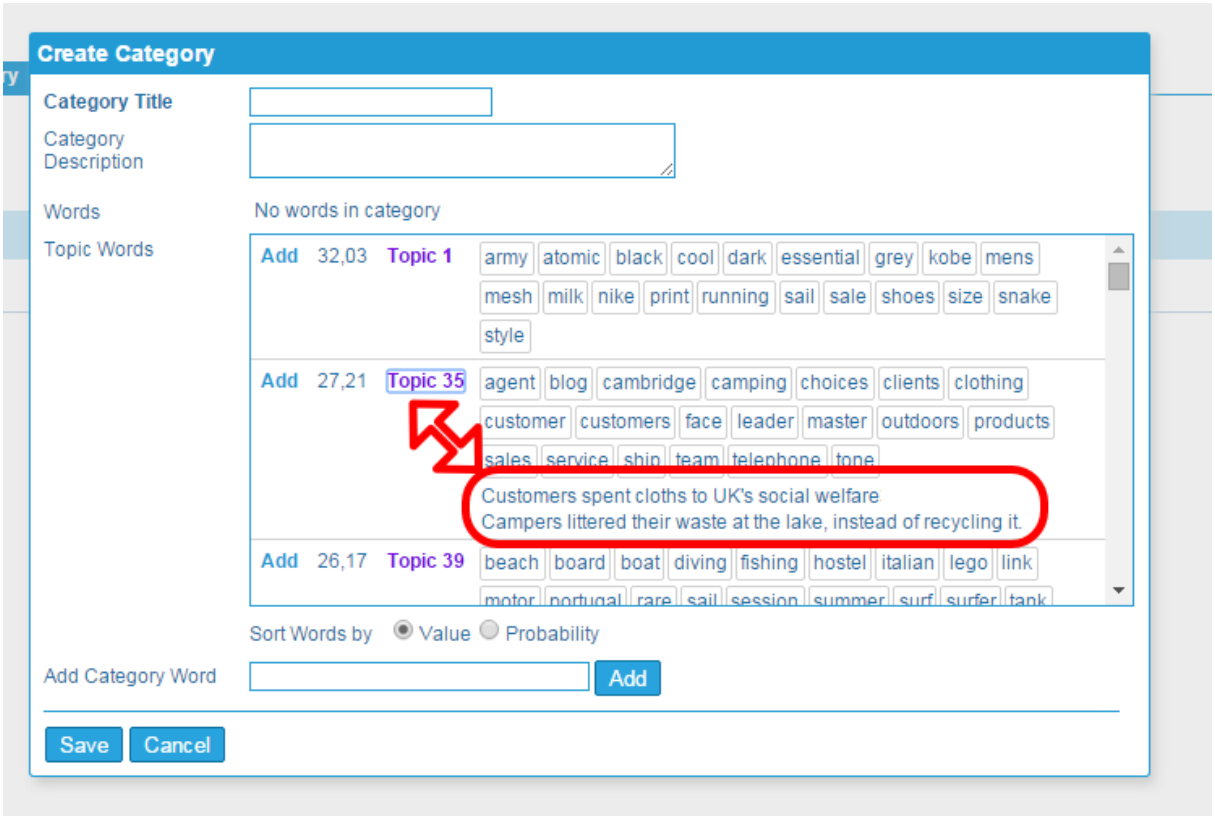


Image 1: summaries help to understand a topic by listing representative sentences from the content

4.2.2 Campaign Dashboard

The campaign dashboard provides an overview about the campaign's data and some selected details. Its purpose is to give the user information about the campaign's history and some insights.
Unlike SEMAVIS, which is a visual analytics tool, the previous campaign dashboard did not enable the user to…

- interact with the visualization (besides setting the timeframe)
- allow to change the detail level of the visualization (drill down)
- identify single posts
- combine more data than the one that's used to render that specific diagram

However, as part of switching the data store from Postgres to SAP HANA Cloud Platform (which is optimized for column-based analytical data access) we staretd to test a self service BI tool (QlikView), which has been finalized as a prototype.
The reason for this move is that FUPOL lacks a sophisticated user-configurable dashboard. While most existing media monitoring tools (i.e. Meltwater) provide dashboards, most of them are static (users can add/remove widgets or change the widgets' configuration, but most social media tools don't provide the feature to create individual, campaign specific dashboards). We felt that by adding self service BI, a technique that allows the end-user (i.e. the facilitator) to create individual views on the collected data and on the analysis results will give FUPOL a competitive advantage for later exploitation.
Our vision is that a facilitator will be able to create a problem-specific dashboard for every campaign and use widgets from a wide selection of pre-defined examples. Data access is already defined and the users select the relevant information based on their experience with the political issue instead of playing around with technical details (queries etc.).

What's already finished in D3.11 is:

- the data has been mapped from HANA HCP to QlikView's cube and as thus it's accessible to the end user
- we created several sample dashboards and experimented with the visualization

The move to QlikView will allow us to modify the visualizations rapidly and with minimum technical overhead in the future (i.e. by domain-experts instead of developers). New visualizations will be brought into the product by QlikTech.

As QlikView has direct access to all data that's stored in SAP HCP's column oriented i-memory database and uses sophisticated under-the-hood optimizations it's considerably faster than the previous solution.

The following screenshots shows the new prototype dashboard

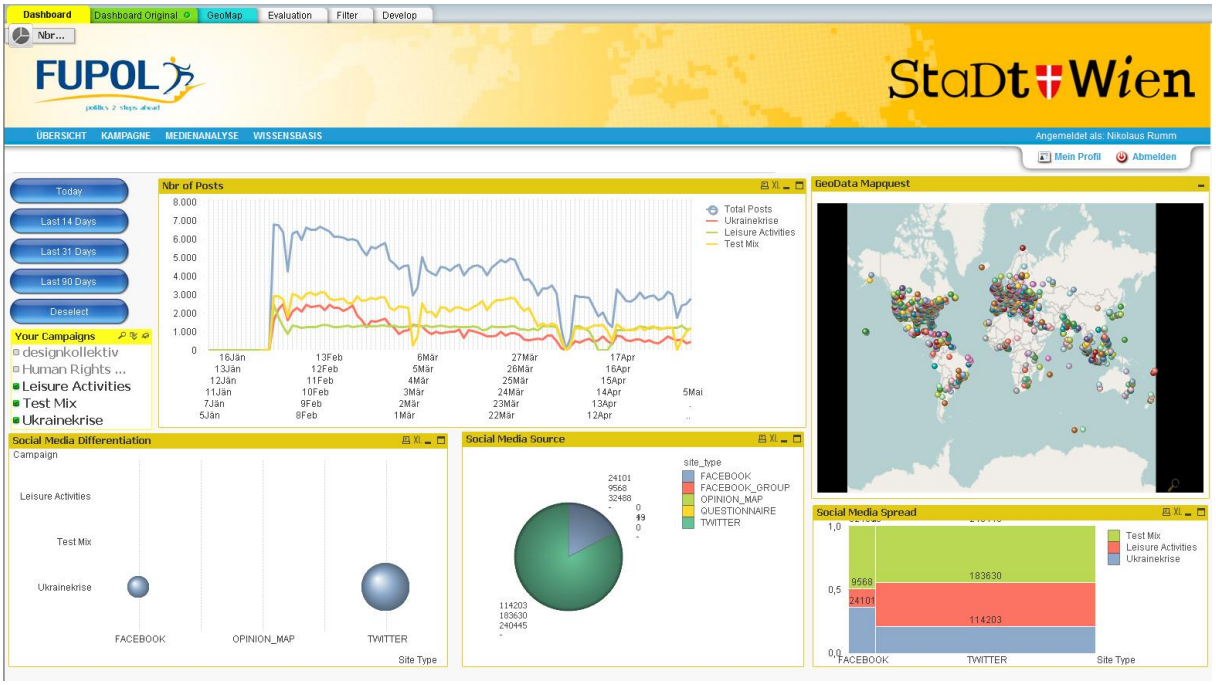


Image 1: an experimental user-defined dashboard

The approach to add a BI tool that has direct access to the database is viable because we expect future city customers (especially larger cities) to use FUPOL in an on-premise installation (most likely on SAP Hana), as most people that we talked to

regarding FUPOL operations don't feel comfortable with exposing their sensitive citizen data (though most of it is collected from public sources) in a public cloud service.
By utilizing Hana's in-memory database we'll be able to perform real-time analytics, so there's no delay between the data collection and the availability of the collected data for analysis and visualization.
Note that the self-service BI tool will only cover the analysis of mass data, so it follows a statistical approach, while the semantic visualizations are the domain of SEMAVIS.
The self-service BI tool is currently an add-on to the core platform, and we might want to offer it as an upselling option.

4.2.3 Campaign Data Browser

The campaign data browser enables the user to search for content based on specific criteria. The search criteria were extended by...

- the category – search for content that relates to one, to a selectable list or to all categories
- the user – search for content that was published by a user

Image 2: Enhancements to the campaign data browser

Note that the search for the user doesn't search for a natural person, but instead for a user account. As we don't unify users and accounts this doesn't break anonymity.

5 Deployment View

The current pilot system is equal to the proposed "demo system" and hosted on two systems:

- FUPOL's virtualized servers (Tomcat, Postgres)
- SAP's HANA cloud (Tomcat, HCP)

The pilot cities used the existing virtualized server version for most of their end-user tests, while we used the HANA version for our benchmark tests. As the move from Postgres to HANA was done in the middle of the pilot cities' tests and as we were unable to migrate all existing user data to HCP within reasonable time, we decided to continue the pilot city tests with the older Postgres-based version. Otherwise we would've had to interrupt pilot operations for several days.

the FUPOL virtualized server(s). Cloud hosting is not available yet, but we experimented with Microsoft's Azure cloud. The move from the dedicated servers into the cloud will be done in year four.

For a detailed description of the deployed system we refer to D3.2/D3.5.

6 Tests and Quality

For a list of test cases please refer to D3.5.

7 User Manual

In order to support the users in the pilot city (and to prevent resource drain caused by personal phone/email support) WP8 wrote a user manual.
This manual is available in electronic form (as a set of wiki pages or PDF) and as printed documentation.

8 Benchmark Tests

As proposed in the previous review remarks we performed intense benchmark tests on the system, both on the virtualized on-premise solution (Tomcat, Postgres), and on the cloud based solution (Tomcat, SAP Hana HCP).

8.1 Introduction and Related Work

We aim to test the performance of our Fupol core platform in the two final versions available at the end of the project:

- the on-premise Postgres-based system (called FUPOL classic)
- the SAP HANA based system that's hosted in SAP's HCP cloud (called FUPOL Hana)

The use case that stresses the system most (CPU load, database I/O) is social media processing, including the hot topic sensing functions, so the test focuses on that:

- network latency of the core platform (time required to process a posting)
- resource consumption (CPU, RAM, database I/O)
- scalability (tests still ongoing)

The following figure describes the overall test architecture with our system modeled as a black box. "System under Test" refers to the core platform (with some tests extending the system under test to the HTS module). We used partially-synthetic data [1] to generate a reproducable social media workload, e.g., Twitter or Facebook posts. The data sample was carefully selected from live data (collected by the demo system[1] that the pilot cities use until the end of August 2015), so the content of the test data has been collected from live media, while the sample that we pulled from that data was filtered based on a statistical analysis of the source data's properties.

Please note that both test environments are virtualized and thus the test results are heavily influenced by the resource balancing mechanisms that are in effect on the underlying platforms (VMWare for FUPOL classic, Hana HCP for FUPOL Hana). We tried to align the capabilities (CPU, RAM, …) of the underlying platforms as good as we could, but the results show what the system that was available for the tests is capable

[1] https://fupol-6.cellent.at/fupol-client, Accessed: 27.8.2015

of and not what it could be capable of given that we add additional resources to it. However the results show some interesting facts about the architecture and its limitations.

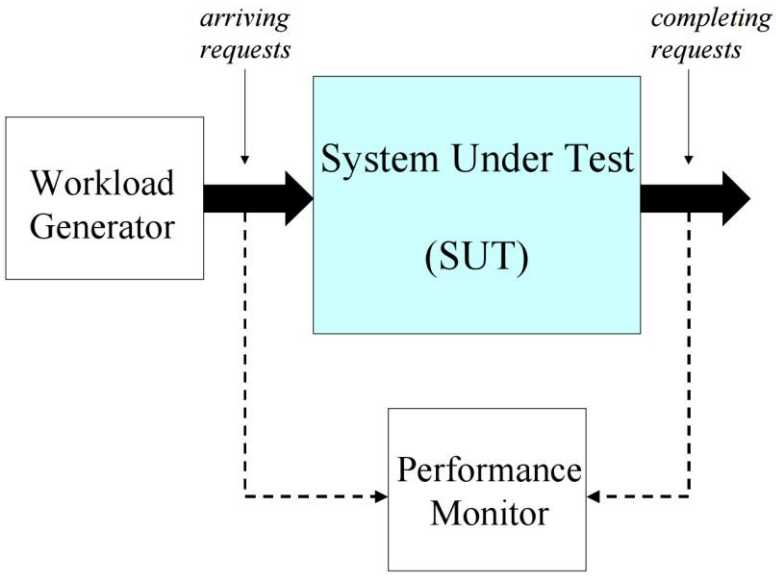


Image 2: the benchmark's design [2]

To measure test-relevant properties, we implemented aspects [3], e.g., for accepting and completing HTTP requests. An aspect is a "*simple code, that is a well-modularized crosscutting concern*" [4]. If an exception, e.g., a Time-Out, during the execution occurs the corresponding aspect rejects the measurement.

The aspects can be seen as code that's required to measure facts of the system under test. This code has to be injected into the system under test at runtime, adding complexity and additional load to it, thus influencing the results. We tried carefully to keep this influence as low as possible and assume that it adds a comparable bias to both systems under test, so we neglected it. We minimized the impact from the measurement and data generation by using a separate servlet for measurement and data generation. These servlets communicate over HTTP-requests by setting and receiving measurement relevant values from the core platform.

We used Dropwizard's metrics library[2] to summarize all measurements from the aspects at the performance monitor because it is extensible with less effort to support our own metrics and has a native integration to the core-platform.

The setting of each environment is controlled by JVM command line parameters, e.g., Xmx, and operating system (OS) dependent commands, e.g. taskset or the Hana cockpit.

[2] https://dropwizard.github.io/metrics, Accessed 24.8.2015

8.2 Benchmark Requirements

In this section we describe the qualitative and quantitative requirements of the benchmark.

The top level requirements of the benchmark test are:

1. Reproducibility: It is the "*ability of a test or experiment to be accurately reproduced, or replicated, by someone else working independently*" [5]. This is one design goal that we addressed with our benchmark [5].
2. Parameterizable: The benchmark test has to provide a basic graphical user interface (GUI) for setting benchmark related parameters, such as the number of posts or the textual representation. Hence it allows to produce different levels of workload and latency bursts.
3. Results: The measurements are generated in a transparent way. Figure 2 describes the work-flow of an aspect for a measurement. Before a method is executed, the aspect starts the time measurement by taking a timestamp down to the nano-seconds. When the method finishes, the aspect increments the number of calls and computes the consumed time of the method. Otherwise the measurement is neglected.

We divided the qualitative measurements of our benchmark into three categories, each representing one fact that we aim to measure:

- the network
- storage and disk I/O
- JVM system resources (i.e. heap memory)

For each fact our benchmark records various attributes, for example the number of connections and the throughput. The measured value is exposed on a REST endpoint and is aggregated by the metrics library. Furthermore the library monitors the JVM

related resources like consumed heap memory or relative CPU usage with respect to the given test settings.

For the other two categories we injected appropriate point cuts with our aspects, such as interfaces to 3rd-party systems or database interfaces. When these methods are called, the aspect measures its execution time and the number of the method invocations.

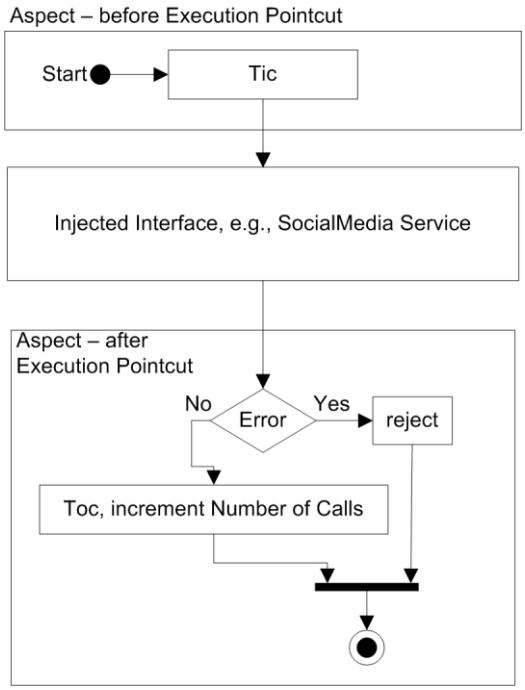


Image 3: an aspect's activity diagram

The minimum crawler timespan is 10 minutes, and for this reason we decided to aggregate all measured values to a one minute mean value (granularity of time).

Furthermore we addressed scalability requirements to avoid implementation bottlenecks (i.e. race conditions in the test data generator).

Out of scope for our benchmark are exhaustive tests, i.e., we tested the major functionalities of the isolated core platform features such as the ability of the core platform to receive, harmonize and extract features of interest, such as topics from

the social media platforms. We used as topic extraction mechanism, the ”Simple” categorizer (which is based on boolean search and exectuted directly inside the core platform) of the core platform.

8.3 Test Environment

One of the major challenges of the benchmark was to generate comparable results because we aim to compare virtualized systems (VMWare and SAP Hana HCP). A cloud system trades consistent system-wide data against scalability, as a consequence from the CAP Theorem. This theorem describes the relation between consistency, availability and partition tolerance [7, 8]. Figure 3 visualizes those relationships. RDBMS-based local systems are located in the intersection between CA whereas cloud systems are at AP.

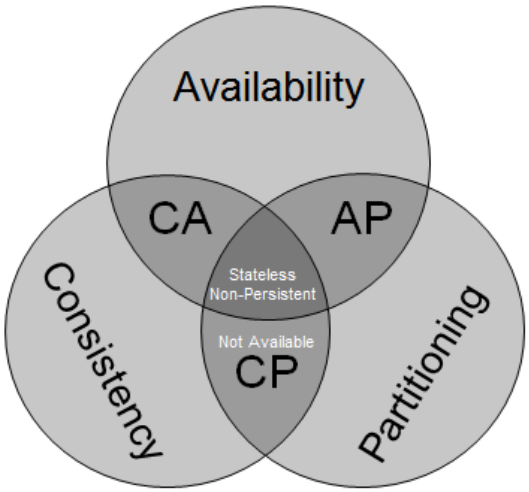


Figure 3: CAP Theorem

In the first step we generated base tests for the FUPOL classic test system and tested if they are eligible for our Hana system. Eligility means that we have more control of the FUPOL classic system than on Hana and so we can choose from a wider selection of technical solutions to implement aspects, measure facts etc. So we had to verify that all aspects and measurements that we implement for FUPOL classic are available in the Hana version. For example we still don't know the operating system that Hana uses, but we successfully measured the heap size on both systems under test.

For our Hana environment[3] we used the standard (free) developer account, which represents the lowest available size of the platform. With the available resources we're able to create and deploy Web-Archives, such as the Fupol core platform on Hana's Tomcat and adopted our database scheme accordingly. Table 3 provides a detailed view on both systems:

Property	**Fupol classic**	**Fupol Hana**
Operating System / Data Center	Ubuntu 14.04 LTS	unknown
RAM (Tomcat / Total)	1024 / 3952	1000[4]
Database	PostgreSQL 9.4	Hana DB
Processor-Type	64 Bit	64 Bit[5]
Virtualized	Yes	Yes
Harddisk (Used / Total)	7.15 GB / 51.4 GB	11.8GB[6]
Tomcat-Version	8.0.15	7
Java-Version	8; Update 25 (1.8.0_25)	7

In both systems the initial hardware settings are set to equal parameters, i.e., to one processor and 1 GB memory. To test the scalability of both systems, we adapt both settings depending on the test. Table 4 describes the hardware setting for each measurement cycle.

Settings	**Fupol classic**	**Fupol Hana**
Initial	1 CPU, 1 GB RAM	1 CPU, 1 GB RAM

[3] http://hcp.sap.com/developers.html, Accessed 24.8.2015
[4] Depends on the Scaling, controlled by appropriate JVM Setting
[5] According to their description on a single NUMA
[6] Depends on the Scaling

1st measurement	1 CPU, 2 GB RAM	1 CPU, 2 GB RAM
2nd measurement	1 CPU, 3 GB RAM	1 CPU, 3 GB RAM
3rd measurement	2 CPU, 1 GB RAM	1 CPU, 1 GB RAM
4th measurement	2 CPU, 2 GB RAM	1 CPU, 2 GB RAM
5th measurement	2 CPU, 5 GB RAM	1 CPU, 5 GB RAM

8.4 Data Generator

The data generator generates partially synthetic social media posts [1], based on a statistical evaluation of the demo system's collected live content. The reason for shaping the test data is that we don't feed the system with simple postings, but instead of that with object graphs (user, forum, links, text, reposts, …). By identifying representative user data we tried to eliminate repetitive data (i.e. retweets from power users that contain the same text over and over again), thus eliminating the effect of caching mechanisms that might differ between both systems under test.
The selected posts have frequently used words, tags, and links[7]. Furthermore, we also modeled the activity of the social network. For example our live data shows that the probability of a repost/retweet is 30% on average.

The next figure shows the evaluation of the available data from the demo system. The blue line is a histogram of the users, ordered by the number of postings that they sent (red). Power users (who generate many postings) are to the left, while casual users (few postings) are to the right.

[7] abbreviated as http://www.example{n}.com/ where n is an increasing number

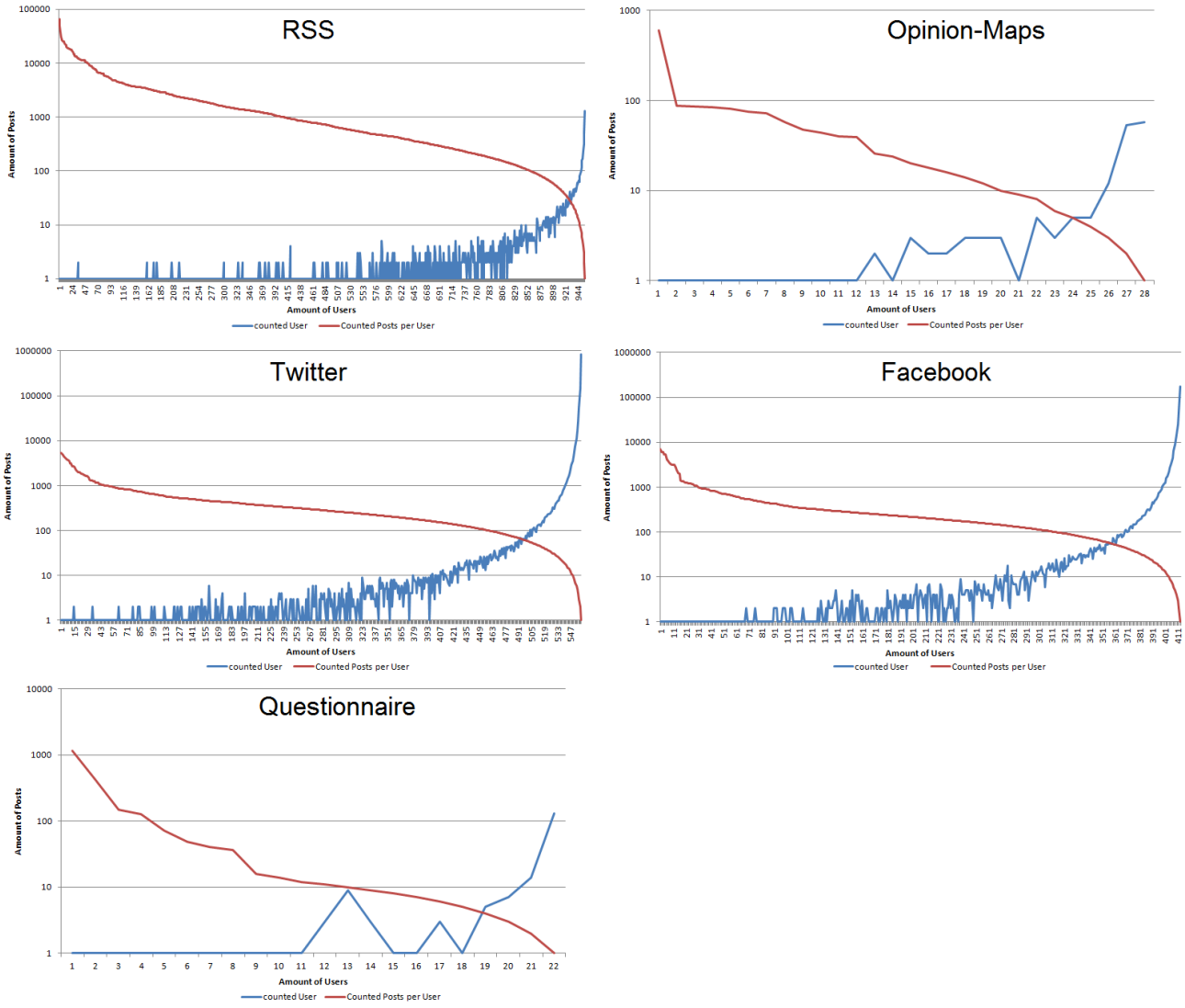


Image 4: posts per user for different media sources

For data generation purposes, we analyzed the statistical properties from each network compared to the total amount of social media data. The distribution of our data is as it is expected log normal [9]. As sampling frame we used the origin of sources. Table 6 shows the results of the analysis[8]. %GT represents the ratio of the source's data volume (number of postings) compared to the overall volume (the demo system that we used as a sampling source stored 6.6 million posts).

[8] Status: July 2015

	FB	Twitter	Blogspot	RSS	O-Map	Question.
% GT	20.2%	43.3%	0.04%	36.4%	0.03%	<0.01%
μ	5.263	5.528	2.898	6.570	3.045	1
σ	1.598	1.208	1.744	1.615	1.404	1
Q1	4.652	4.943	1.830	5.553	2.168	1
Q2	5.347	5.651	2.441	6.601	3.087	1
Q3	5.901	6.214	3.826	7.673	4.118	1

Table 1: statistical analysis of social media sources

For example 43.3% of our content is from Twitter.

We used the 1st and 3rd quartile to draw a representative sample from our demo system. If we use the exponential function on the values of the quartiles, we get the limitation for selecting a representative sample.

	Facebook	Twitter	Blogspot	RSS	O-Map
1 Quartile	105	140	6	258	9
2 Quartile	210	285	11	736	22
3 Quartile	365	500	46	2150	61

Table 2: sample boundaries based on the user's number of postings

Figure 8 shows the quartiles drawn in the distribution of Facebook data. The quartile represents the numbers of posts that a representative user had to post to be eligible for selection. This prevents that a topic of an opinion-leader dominates the selected data or fewer discussed topics.
We chosed a random sample[9] to determine the most frequent words and tags for our data generator.

[9] n=1000

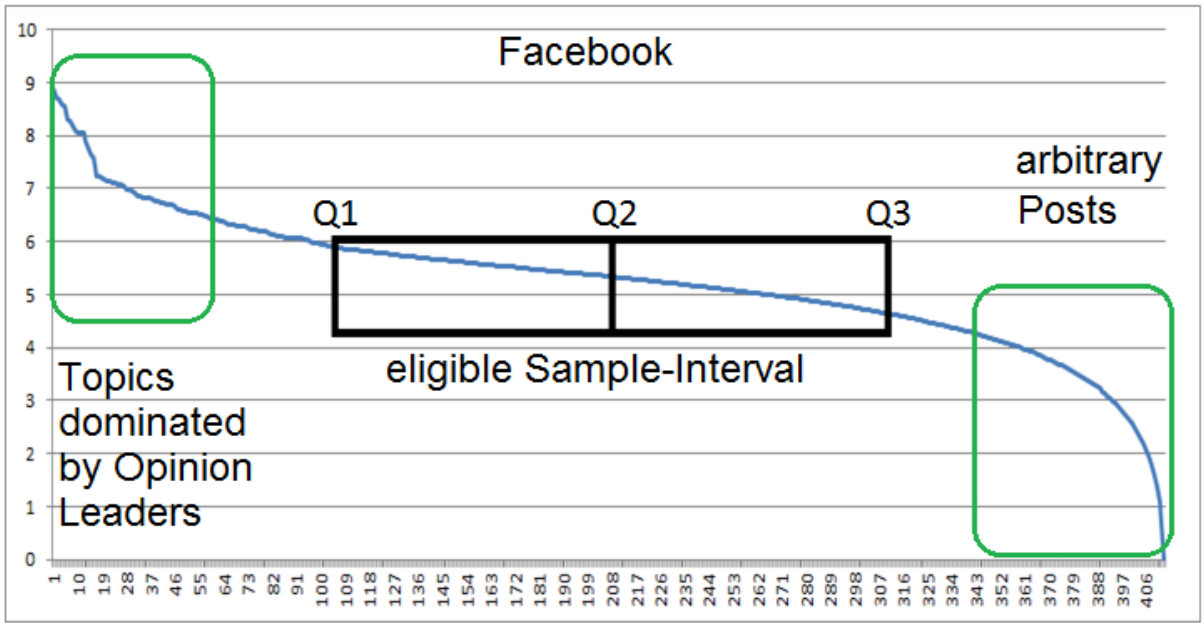


Image 5: boundaries for sampling eligible test data

These words, tags and links are used as generic posts in our data generator and allow us to scale the workload of our benchmark sample depending on the phase of the test. As a random seed generator we used the COLT library[10] with a normal distribution and a user entered seed number for the posts.

[10] http://dst.lbl.gov/ACSSoftware/colt/, Accessed 25.8.2015

8.5 Tests

Figure 9 sketches a general test run based on the requirements listed in Section 8.2 and the settings from Section 8.3. A complete test cycle performs 20 test run executions.

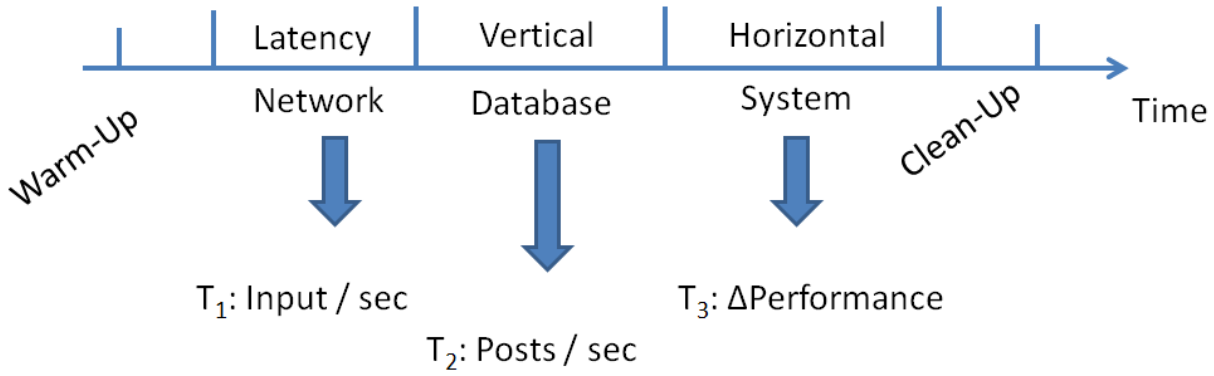


Image 6: a single test run's anatomy

Before we start our measurements we initialize the network, caching and database mechanisms of the core platform by starting pre-defined searches (warm-up)[11] [10]. When these searches finish, the benchmark starts its measurements.

First we measure the latency. *"Latency is the time delay between the moment that a stimulus occurs and the moment that its effect begins or ends*" [11]. In our benchmark it is the amount of time that is elapsed between the beginning and the end of HTTP transmissions. Additionally we measure the amount of input data that the core-platform is able to process and the number of open connections to 3rd-party systems to calculate more precise statistics, such as elapsed time per post.

When the data is collected and harmonized, we measure the resulting database traffic when the core platform saves the entities. In that case we measure throughput and

[11] We search for "example" and "Test" on Twitter, RSS and Facebook

latency properties of the database wrapper by using Hibernate's statistics plug-in[12]. However we do not distinguish between the different database entities nor do we measure all available properties, so the number is the aggregated database load.

Finally when both previous experiments finished, we measure the scalability of the environment by increasing the workload and simulating an increased user traffic.

When all tests finished, we delete the collected data from the database and restart the test again. Each test run is executed 20 times to get significant results and to rule out variations of background load or other influences that are out of our sphere of control, i.e., on Hana.

Each test execution uses one of two sets of social media sources. First, several searches to measure the behaviour of the core platform within one of the frequently used social media sources (single social media window) are performed. Then, after testing the single sources, another 13 runs contain mixes of the sources to cover a wider search scope (mixed social media window). The mixed social media windows simulate user behaviour (query mix).

8.5.1 Single Social Media Window

This test type represents a social media search for one social media platform, e.g., search for „bikes" on Twitter. The purpose of this test type is to measure the differences between the search implementations for the social media platforms.

8.5.2 Mixed Social Media Windows

This test type is a composite of the single social media window searches, e.g. a social media search for „bikes" on various platforms. The purpose of this test type is to test the data harmonizing ability (mapping the content from various sources to the same

[12] https://docs.jboss.org/hibernate/orm/3.5/api/org/hibernate/stat/Statistics.html, Accessed 25.8.2015

underlying ontology) of the core platform, because we integrate different social media data sources in one workflow.

8.6 Test Architecture

The architecture of our benchmark is divided into three components. Figure 10 illustrates the overall architecture (not shown are the parts related to caching and aggregation). As already mentioned we tried to reduce the bias that is introduced by the measurement itself, i.e. by physically seperating the system under test from the test data generator etc.

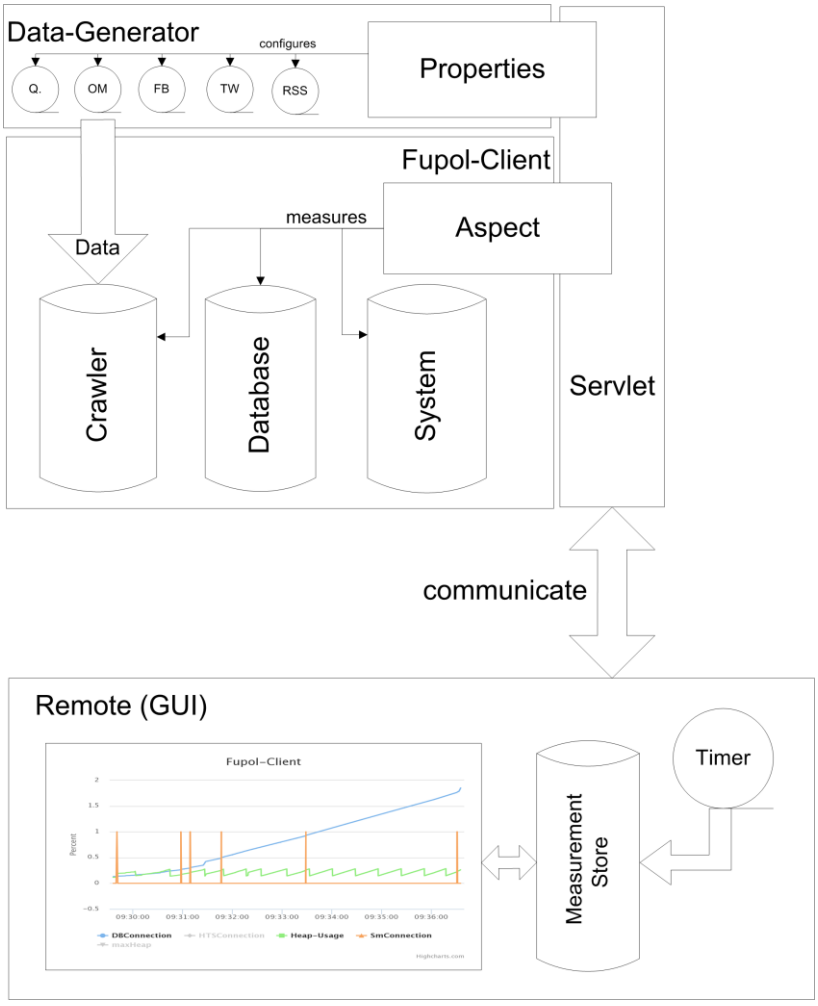


Image 7: benchmark's architecture

The GUI-servlet is located on a separate machine and posts the user entered measurement parameter to the metrics servlet. The servlet sets these parameters as

system properties and initializes the aspects. Furthermore it starts a timer that polls the measurement servlet in user defined intervals and stores the received value.
The next component, the data-generator uses the system properties to scale the test data according to the user entered values. When a social media search starts, the core platform starts a search request on the data generator instead of the corresponding social media platform[13]. One simplification of our benchmark is that it does not support timezones, i.e., all timestamps are in GMT, but we don't expect this to be of significant impact (time calculations should be in-memory and linear).

In addition to our partially synthetic data described in 8.3, we measured the network latency between the system under test and the social media sites (without the data generator tool) to estimate the external processing time per post. The external time is the duration of the core platform's request to a medium (i.e. Facebook) including the transmission of the results. The overall processing time of the core platform is the sum of the external processing time and the internal processing time.

We use the external processing time to calculate the theoretical network-latency[14] of our core platform. As search terms we used the top 10 search terms proposed by Goolge[15] fed them into the platform as search targets on social media platforms, i.e. Facebook, Twitter. For RSS-feeds we searched appropriate feeds, that discuss or blog about these topics. For more details please refer to the appendix.
We excluded opinion maps and questionnaries from these measurements, because those sources are directly provided by users to the core platform and therefore they have no dependence on 3*rd*-party platforms. Furthermore the observed data volume and frequency is neglectable as compared to the (social) media sources.

[13] This is parametrized by using appropriate Spring-Profiles
[14] Formula: $n_{DataGenerator} * AVG(t_{post})$
[15] http://www.google.com/trends/hottrends, Accessed 21.8.2015

8.6.1 Verification

We verified the measurements of our benchmark by comparing the values, i.e., consumed JVM heap memory with measured values from Java JVisualVM. JVisualVM is the Java analytic toolkit for system resources that is supported by Oracle and included in each Java installation [12]. Figure 11 shows a comparison of measured memory consumption between our benchmark and JVisualVM.

Image 8: result validation by comparing the measured heap usage with the result that another tool (JVisualVM) observes

In the image above JVisualVM uses the local time (GMT+2h) whereas our benchmark uses GMT.

Furthermore we validated the number of network connections by using Wireshark[16], a frequently used network analyzing tool by taking network snapshots during a test cycle [13].

[16] https://www.wireshark.org, Accessed 26.8.2015

8.7 Results

8.7.1 Processing Facebook posts

The figure describes the results of our Facebook latency tests. We divided the time spent per search by the number of returned posts. As input data, we used our generic Facebook posts from the data generator.

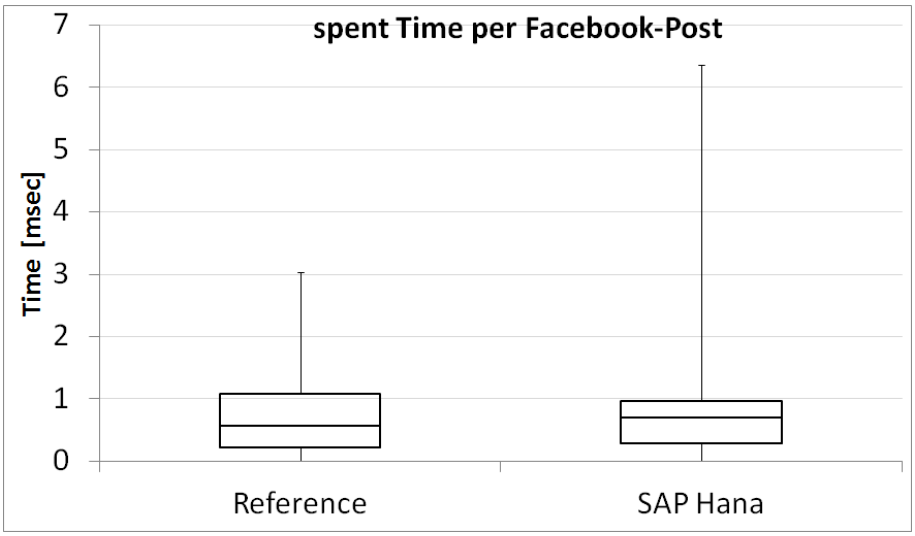


Image 9: average Facebook processing latency

The test shows that the SAP Hana cloud is faster than the reference system.

8.7.2 Processing Twitter posts

The following figure describes the results of our Twitter latency tests. Again, we divided the time spent per search by the number of returned posts. As input data, we used our generic Twitter posts from the data generator.

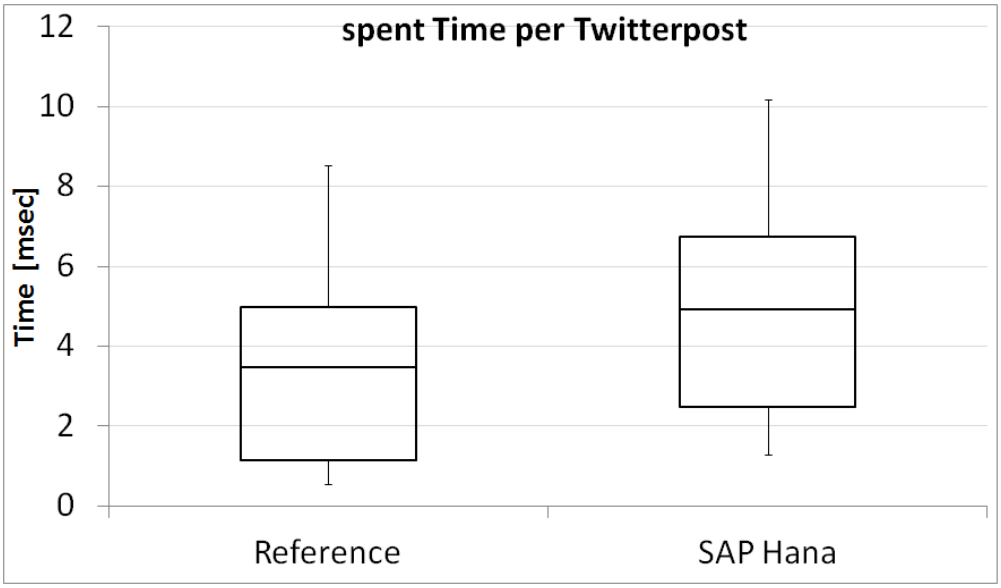


Image 10: average Twitter processing latency

The test shows that the reference system is faster than the SAP Hana cloud.

8.7.3 Processing RSS posts

The next figure describes the results of our RSS latency tests. Again, we divided the time spent per search by the number of returned posts. As input data, we used our generic RSS posts from the data generator.

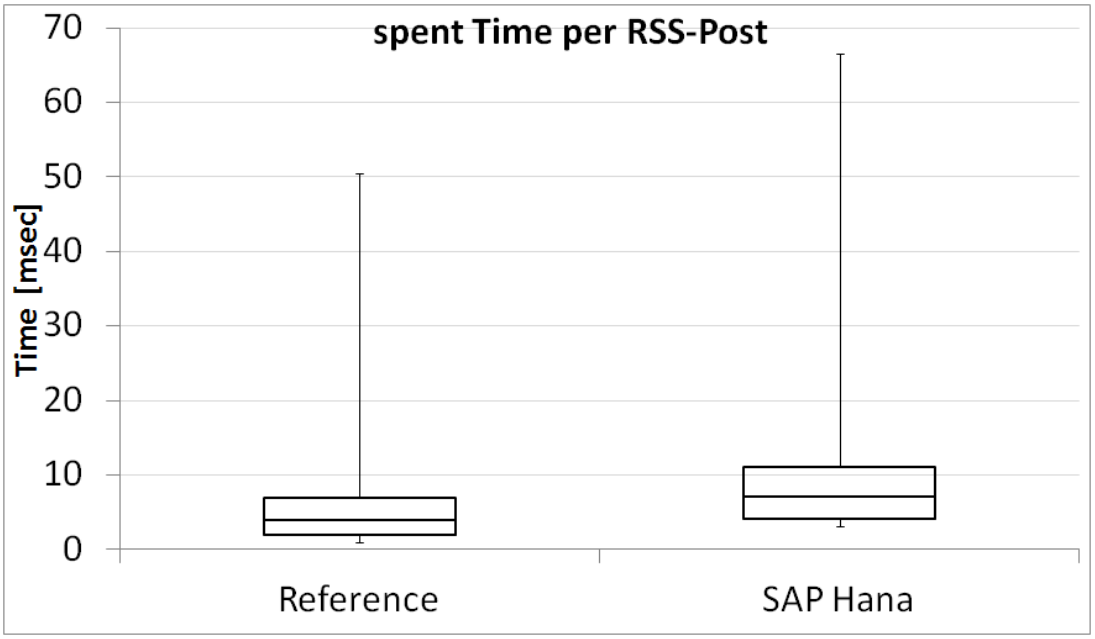


Image 11: average RSS processing latency

The test shows that the reference system is faster than the SAP Hana cloud.

8.7.4 Scaling

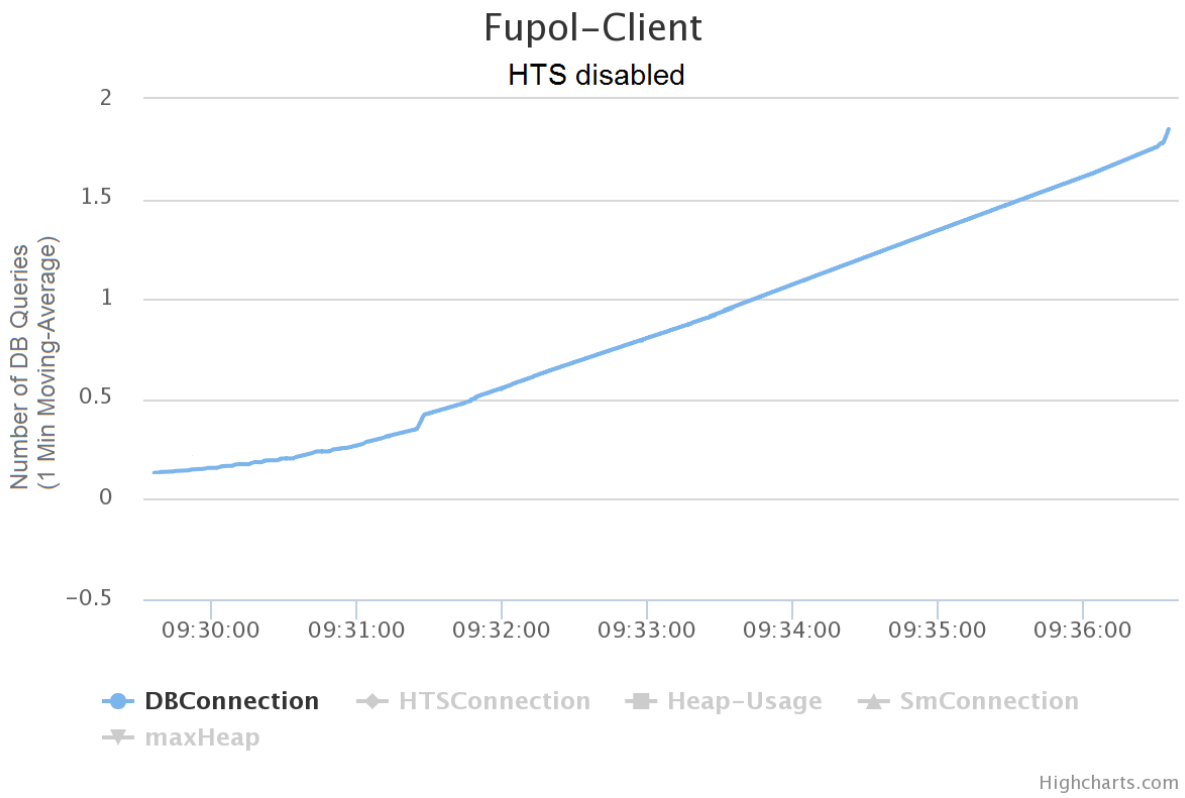
Increasing the amount of the hardware resources is done by using SAP's HCP scaling. Currently SAP supports two types of Scaling:

1. Scale Up or vertical scaling
   When a system is scaled up, the amount of available system resources is increased, e.g., a database size is increased.
2. Scale Out or horizontal scaling
   When a system is scaled out, resource nodes are added. In that case, the system splits the occurring traffic equally between all available nodes.

Färber [14] et al. stated that the Hana database can not be compared to a RDBMS. Despite these assumptions we still tried to compare our reference system to Hana DB. The scalability tests for the cloud are currently work in progress, due to the high complexity of Hana deployments.

8.7.5 Impact of using Hot Topic Sensing on the database

The next figure compares the number of database queries from a campaign with disabled hot topic sensing (HTS) and the same campaign with HTS enabled.
HTS can be enabled/disabled by the user. The consequence is that the categorization algorithm performs (local) binary search instead of (remote) NMF categorization.



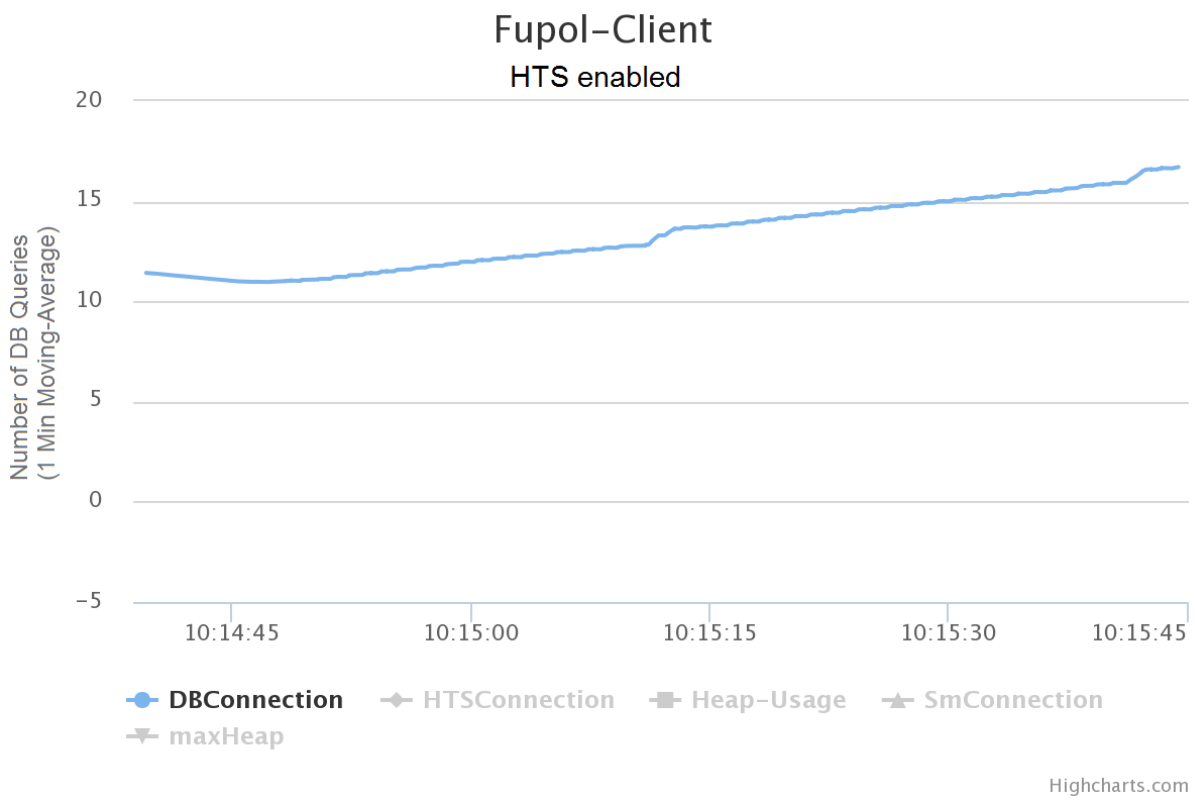


Image 12: number of queries with enabled and disabled HTS

When we disabled HTS, the number of database calls decreased to 10% as compared to enabled HTS. This shows the overhead related to synchronizing both databases (one in the core platform and one in the HTS server – a consequence of the loose coupling based on SOA). The high database utilization results from applying the categorization matrix on the content.

This is a clear indication that the current SOA-based HTS implementation will benefit from Hana's in memory database.

8.8 Appendix A: Facebook latency tests

This section contains relevant search settings for reproducing the latency tests. These test are used to determine the average time for receiving a single post.

		Reference System		SAP Hana	
Meas.	Wall-URL	Number Posts	receive Time	Number Posts	receive Time
1	https://www.facebook.com/cocacolaaustria	115	55,46	94	59,26
2	https://www.facebook.com/Marveldeutschland	619	55,27	616	84,11
3	https://www.facebook.com/MicrosoftAT	82	42,33	80	43,54
4	https://www.facebook.com/BicycleNetwork	85	34,18	85	56,46
5	https://www.facebook.com/MeganFox	337	125	337	121,90
6	https://www.facebook.com/ja20com	51	36,55	25	56,58
7	https://www.facebook.com/candycrushsaga	614	123,13	594	124,08
8	https://www.facebook.com/deeznutshardcore	29	65,11	114	19,53
9	https://www.facebook.com/kelvinbenjaminofficial	46	11,48	144	59,55
10	https://www.facebook.com/TilSchweiger	32	31,20	41	41,99
11	https://www.facebook.com/Unitymedia	106	14,51	46	25,97
12	https://www.facebook.com/zoobeauval	72	61,23	61	43,16
13	https://www.facebook.com/fcbarcelona	648	89,85	646	91,42
14	https://www.facebook.com/pages/Tropical-Storm-Karen-Updates/658313490853652	12	29,51	12	67,92
15	https://www.facebook.com/amnestyglobal	26	22,26	85	48,46

The mean-value for processing a generated Facebook post in mili-seconds:

Reference / Hana = 0,713 / 0,937

8.9 Appendix B: Twitter latency tests

This section contains relevant search settings for reproducing the latency tests. These test are used to determine the average time for receiving a single post.

		Reference System		SAP Hana	
Measurement	Search-Term	Number Posts	receive Time	Number Posts	receive Time
2	test	48	55,53	43	161,90
3	example	50	29,91	50	159,13
4	example	50	31,96	50	132,69
5	bicycle	50	51,56	49	212,69
6	whatsapp	50	31,65	50	83,31
7	Yvonne Craig	47	135,78	49	182,59
8	Josh Duggar	50	191,52	50	249,80
9	Megan Fox	50	208,13	50	211,59
10	Candy Crush	49	130,57	50	130,29
11	Deez Nuts	50	203,91	50	212,60
12	Kelvin Benjamin	50	148,52	47	201,14
13	Tropical Storm Danny	50	252,75	50	262,55
14	asylum	50	192,07	50	183,40
15	Ukraine	50	162,12	50	201,16
16	Egon Bahr	50	139,87	50	108,46
17	Internet Störung	50	84,11	50	190,37
18	Til Schweiger	50	186,64	50	124,42
19	Perrie Edwards	49	162,63	50	224,86
20	Zoo de Beauval	50	97,33	50	60,26
21	Valencia Monaco	50	201,91	50	105,35

The mean-value for processing a generated Twitter post in mili-seconds:

Reference / Hana = 2,715 / 3,446

8.10 Appendix C: RSS latency tests

This section contains relevant search settings for reproducing the latency tests.

These test are used to determine the average time for receiving a single post.

		Reference System		SAP Hana
Meas.	Wall-URL	Number Posts	receive Time	Numb Posts
1	http://boss.blogs.nytimes.com/feed/	1	46,67	
2	http://www.theengineer.co.uk/XmlServers/navsectionRSS.aspx?navsectioncode=186	64	70,74	
3	http://feeds.reuters.com/reuters/businessNews?format=xml	20	81,48	
4	http://feeds.feedburner.com/techcrunch/social?format=xml	20	55,57	
5	http://www.technologyreview.com/topnews.rss	20	54,12	
6	http://www.wired.com/category/business/feed/	9	30,45	
7	http://www.tagesschau.de/xml/rss2	39	58,52	
8	http://rss.dw.de/atom/rss-en-all	75	131,85	
9	http://feeds.feedburner.com/RssAllSections	25	123,30	
10	http://croatia.hr/en-GB/RSS/News.feed	2	48,18	
11	http://www.unhcr.org/cgi-bin/texis/vtx/page/rss.xml?coi=HRV	20	38,50	
12	http://www.france24.com/en/top-stories/rss	16	32,94	
13	http://fh.oxfordjournals.org/rss/current.xml	26	140,24	

The mean-value for processing a generated RSS post in mili-seconds:

Reference / Hana = 7,875 / 11,379

9 References

[1] R. D. Fricker, Introduction to Statistical Methods for Biosurveillance:With an Emphasis on Syndromic Surveillance, Cambridge UniversityPress, 2013.

[2] D. Menasce, et al., TPC-W: A benchmark for e-commerce, Internet Computing, IEEE 6 (3) (2002) 83–87.

[3] T. Elrad, R. E. Filman, A. Bader, Aspect-oriented programming: Introduction,Communications of the ACM 44 (10) (2001) 29–32.

[4] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier,J. Irwin, Aspect-oriented programming, Springer, 1997.

[5] P. Vandewalle, J. Kova˘cevi´c, M. Vetterli, Reproducible research in signal processing, Signal Processing Magazine, IEEE 26 (3) (2009) 37–47.

[6] L. Lamport, Proving the correctness of multiprocess programs, Software Engineering, IEEE Transactions on (2) (1977) 125–143.

[7] E. Brewer, CAP twelve years later: How the" rules" have changed, Computer 45 (2) (2012) 23–29.

[8] R. Ramakrishnan, Cap and cloud data management, Computer (2) (2011) 43–49.

[9] E. Agichtein, C. Castillo, D. Donato, A. Gionis, G. Mishne, Finding highquality content in social media, in: Proceedings of the 2008 International Conference on Web Search and Data Mining, ACM, 183–194, 2008.

[10] Y. Guo, Z. Pan, J. Heflin, LUBM: A benchmark for OWL knowledge base systems,Web Semantics: Science, Services and Agents on theWorld Wide Web 3 (2) (2005) 158–182.

[11] A. H. Ghamarian, S. Stuijk, T. Basten, M. Geilen, B. D. Theelen, Latency minimization for synchronous data flow graphs, in: Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on, IEEE, 189–196, 2007.

[12] M. T. Minella, Scaling and Tuning, in: Pro Spring Batch, Springer, 387– 445, 2011.

[13] A. Orebaugh, G. Ramirez, J. Beale,Wireshark & Ethereal network protocol analyzer toolkit, Syngress, 2006.

[14] F. Färber, S. K. Cha, J. Primsch, C. Bornh¨ovd, S. Sigg, W. Lehner, SAP HANA database: data management for modern business applications, ACM Sigmod Record 40 (4) (2012) 45–51.

[15] C. Binnig, D. Kossmann, T. Kraska, S. Loesing, How is the weather tomorrow?: towards a benchmark for the cloud, in: Proceedings of the Second Inter