Technical Report

Performance Testing of Semantic Publish/Subscribe Systems

Martin Murth [1]
Dietmar Winkler [2]
Stefan Biffl [2]
eva Kühn [1]
Thomas Moser [2]

[1] Institute of Computer Languages, Vienna University of Technology
Argentinierstrasse 8/4th floor, AT 1040 Vienna, Austria
{mm, eva}@complang.tuwien.ac.at

[2] Christian Doppler Laboratory ASE-Flex-SE
Institute of Software Technology and Interactive Systems,
Vienna University of Technology
Favoritenstrasse 9-11/188, AT 1040 Vienna, Austria
{dietmar.winkler, stefan.biffl, thomas.moser}@tuwien.ac.at

Technical Report No. IFS-QSE 10/05
Issued: August 2010

Performance Testing of Semantic Publish/Subscribe Systems

Martin Murth[1], Dietmar Winkler[2], Stefan Biffl[2], eva Kühn[1] and Thomas Moser[2]

[1] Institute of Computer Languages, Vienna University of Technology
[2] Christian Doppler Laboratory "Software Engineering Integration for Flexible Automation Systems", Vienna University of Technology
{mm,eva}@complang.tuwien.ac.at
{dietmar.winkler,stefan.biffl,thomas.moser}@tuwien.ac.at

Abstract. With the growing industrial adoption of semantic web technology, the underlying middleware systems are faced with increasing demands on dynamism and efficiency. An approach that has recently gained popularity in this field are semantic publish/subscribe mechanisms. With these mechanisms, clients can observe the knowledge currently represented in a semantic repository and they can react to changes of this knowledge. Since the currently available implementations of semantic publish/subscribe systems differ essentially with respect to performance and functional scope, it is particularly important to thoroughly evaluate which system best meets an application's requirements. However, just this evaluation often turns out to be difficult as common evaluation frameworks and methods have not yet been established for this kind of systems. In this paper, we take a first step towards such an evaluation framework by defining two performance metrics and implementing a benchmark framework for measuring these metrics. For illustrating its application, the paper further describes the use of the framework in the SENS project, where it was employed (i) to find performance bottlenecks of a lightweight semantic publish/subscribe framework, (ii) to identify potential optimization approaches, and (iii) to quantify the improvements achieved with the developed optimization mechanisms.

1 Introduction

The adoption of semantic web concepts in a wide range of application areas has led to increased requirements on the underlying technologies. Semantic repositories are not only used for one-time loading and reasoning about a knowledge base but rather for the management of highly dynamic knowledge bases. For reacting to changes of such knowledge bases, novel mechanisms are needed that allow for observing the knowledge represented in a semantic repository. An approach that is currently gaining popularity in this field are semantic publish/subscribe mechanisms. With these mechanisms, it is possible to register queries (e.g., using SPARQL [21]) on a semantic repository and to subscribe for new results of these queries. When the knowledge represented in the semantic repository is updated, the system re-evaluates the queries and notifies the subscribers about new query results.

Semantic publish/subscribe mechanisms are currently provided by several semantic repositories, each exhibiting significant differences regarding performance and functional scope (e.g., G-ToPSS [20], OPS [26], TripCom [23], SENS [16]). For the development of semantic applications, it therefore needs to be carefully evaluated which system best meets the given requirements. However, standardized methods for the evaluation of semantic publish/subscribe mechanisms are not yet available. Accounting for this, we developed the SEP-BM (Semantic Event Processing Benchmark) framework. SEP-BM defines two commonly applicable performance metrics and implements a Java framework for measuring these metrics for semantic publish/subscribe systems.

In the last years, a number of benchmark frameworks for semantic repositories have been developed (e.g., LUBM [6], UOBM [12], BSBM [1]). These benchmarks are used for the assessment of load, reasoning, and query performance of semantic repositories (e.g. Sesame [2], Jena [13] or OWLIM [8] etc.) and their reasoning systems (e.g., Pellet [24], Racer [7]). A benchmark run typically comprises the following steps: (i) a predefined benchmark data set is loaded into the semantic repository, (ii) additional information is inferred using a given benchmark ontology, and (iii) several queries are executed against the inferable information. The times required for loading the data and for executing the queries are then compared to those of other systems. Unfortunately, these frameworks do not allow for a comprehensive evaluation of semantic publish/subscribe mechanisms as they only evaluate the performance of a single load, reason, and query cycle. In contrast, semantic publish/subscribe systems repeatedly execute queries and reasoning cycles in very short time intervals, and several systems employ optimization mechanisms that require multiple query or reasoning iterations until they reach their optimal performance. The SEP-BM framework takes these factors into account by generating sequences of publication and notification operations and defining metrics via the average of the measured values.

Benchmark frameworks comparable to the SEP-BM framework have also been developed for traditional publish/subscribe systems (e.g., SPECjms2007 [22]). However, these systems are different from semantic publish/subscribe systems as they implement a pure store-and-forward approach. Information providers publish messages at the message broker, where they are forwarded to the information consumers. In contrast, subscribers of a semantic publish/subscribe system are notified about changes in a knowledge base. While these changes result from publication (or deletion) operations, the clients are not provided with the published information but with notifications about the resulting changes. Benchmarks for traditional publish/subscribe systems therefore only measure the time between the publication of a message by the information provider and the reception of the same message by the information consumer. In contrast, SEP-BM recognizes the indirect relationship of publications and notifications and measures the time between a publication and the notifications resulting from this publication.

The paper is structured as follows: Section 2 defines two metrics for determining a semantic publish/subscribe system's responsiveness and overall rate of processed operations. An overview of the SEP-BM framework implementation is provided in Section 3. Section 4 describes the application of the framework in the SENS project [16], illustrating its practical use for the identification of performance bottlenecks and the development of optimization mechanisms. Finally, conclusions and future work are presented in Section 5.

2 Benchmark Metrics

During the development of several semantic applications (e.g. [3][4][5][10][17]), two main performance-critical areas have been identified: responsiveness and overall rate of processed operations (cf. [15]). We therefore defined two metrics covering these two areas: the metric *notification time* as a measure for responsiveness and the metric *publication throughput* as a measure for the overall rate of processed operations.

2.1 Notification Time

The user-perceived responsiveness of a system corresponds to the time between the triggering of a particular application activity and the expected application feedback. If an application activity involves the interaction with a semantic publish/subscribe system, this includes (i) the creation of a request, (ii) sending the request to the semantic publish/subscribe system, (iii) processing the request, (iv) sending the generated notifications back to the client, and (v) providing the according user feedback (e.g., displaying the received information). While the first and the last step are carried out by the developed application, steps (ii)-(iv) are taken over by the semantic publish/subscribe system and are covered by the following definition of notification time:

Definition (Notification Time): *The* notification time δ(notify) *of a notification defines the time interval between the invocation of the operation that first allowed for the detection of an event and the generation of the notification that describes this event.*

Hence, the notification time describes the interval between the first point in time a notification could (theoretically) have been generated and the point in time it was really generated

For determining the responsiveness of an semantic publish/subscribe system, we further extend the above definition and define the *average notification time δ(S)* for a semantic publish/subscribe system *S* as the average of all notification times of notifications that were generated during a given time interval.

Fig. 1 illustrates the determination of notification times. It shows an example of a sequence of publication requests, the resulting notifications, and the corresponding notifications times. The bars show the system's processing phases which are triggered when new information is published.

In the example, certain processing phases last longer than others. This might, for instance, result from different sizes of the published graph data structures or from the application of certain optimization mechanisms. As a consequence, also the notification times of the resulting notifications range from $\delta(notify_{2.1}) = 40ms$ to $\delta(notify_{3.1}) = 160ms$. In total, the three single notification times of the investigated time interval lead to an average notification time of $\delta(S) = 100ms$.

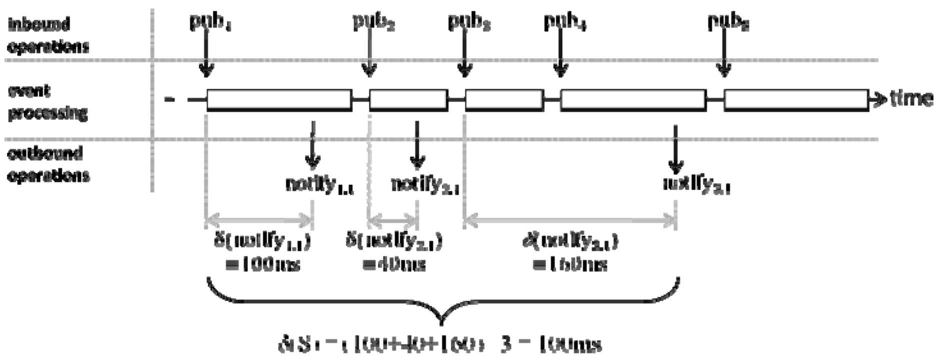

Fig. 1. Notification times and average notification time[1]

2.2 Publication Throughput

Throughput is commonly understood as the rate of requests that a system can successfully process within a given time interval. In semantic publish/subscribe systems, the vast majority of processed operations are publication operations (compared to the deletion of knowledge). Hence, the rate of processed publication operations is considered the most important indicator for the throughput of a semantic publish/subscribe system. We therefore defined the metric *publication throughput* for measuring this rate of processed publication operations as follows:

Definition (Publication Throughput). *The* publication throughput μ(S) *of a semantic publish/subscribe system S defines the number of publish requests that can be completely processed during a given time interval. A publish request is thereby only considered to be completely processed, if all resulting notifications have been generated during this time interval.*

The metric publication throughput therefore describes the number of publish requests for which all possible notifications have been generated in a given time interval.

In most application scenarios, semantic publish/subscribe systems receive publish requests in strongly varying time intervals. In some cases, the system needs to process multiple requests concurrently; in other cases, the system is in idle mode when a new request is received. When multiple requests are processed at the same time, usually quite high publication throughput rates can be achieved. RDF graphs published within a short time interval can be consolidated to one big graph and the semantic publish/subscribe system can evaluate all requests within one single reasoning and querying run. With this approach, the publication throughput can be increased almost linearly (with respect to the number of concurrently processed request) for up to 100 graphs in many of the implementations we investigated (e.g., [9][16]). As a consequence, the worst-case

[1] Notifications are given with two subscript numbers. The first one referring to the publication operation that triggered the notification and the second one to distinguish notifications that were triggered by the same publication operation.

publication is to be expected when no two publish requests can be processed at the same time. In order to obtain an indicator for such worst-case publication throughput rates, we propose to measure the publication of graphs in a sequential order, so that a new graph is published immediately after the completion of the preceding processing phase. Such non-overlapping sequences of publication operations also often occur in real applications, e.g., when semantic publish/subscribe systems are used to implement interaction patterns like request/response (two clients alternately publishing new information in response to received notifications).

Fig. 2 shows a semantic publish/subscribe system that is processing such a sequence of publications. The figure depicts the generated notifications and shows the resulting publication throughput.

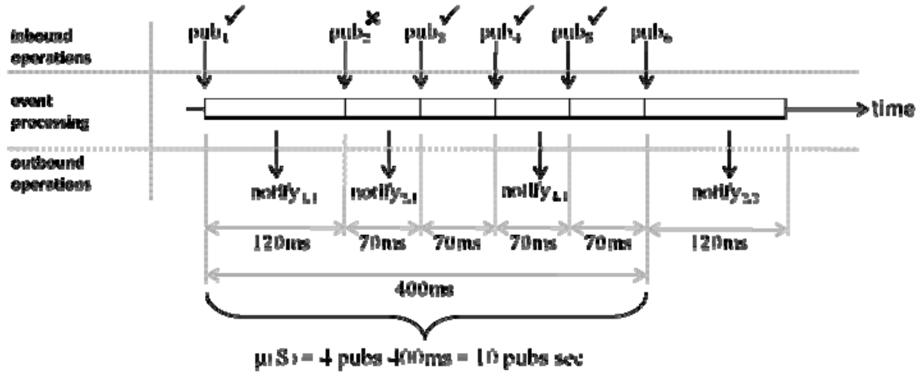

Fig. 2. Publication throughput

Here, it is assumed that the semantic publish/subscribe system employs an optimization mechanism that allows for more efficient generation of notifications but which may also sometimes miss an event (such mechanisms are used in, e.g., [15]). Depending on whether or not a notification is generated by the optimization mechanisms, the according processing phases exhibit processing times of 70ms and 120ms, respectively. For determining the publication throughput, only those publish requests are considered, for which all notifications have been generated and sent to the subscribers within the investigated time interval. This requirement ensures that the processing of publication operations (persisting, reasoning, querying, generation of notifications) is included in the metric. In the example, the requests for which all notifications were generated within this time interval are *pub*$_1$, *pub*$_3$, *pub*$_4$, and *pub*$_5$. As a second notification for *pub*$_2$ is generated in a later processing phase, this request is not taken into account. In total, 4 publish requests were completely processed in the investigated time interval of 400ms, resulting in a publication throughput of *μ(S) =10 pubs/sec*.

3 SEP-BM Benchmark Framework

For measuring the performance of semantic publish/subscribe systems with respect to the defined metrics, we developed the SEP-BM (Semantic Event Processing Benchmark) framework, consisting of a benchmark base configuration and a Java benchmark framework.

3.1 Benchmark Base Configuration

The benchmark base configuration comprises a benchmark data set, an ontology that is used to infer new information, and 20 query definitions for subscriptions.

The developed benchmark configuration is based on the LUBM Benchmark [6], the most widely used benchmark for the evaluation of semantic repositories (e.g. in [19][6]). The LUBM Benchmark was conceived for the assessment of load, reasoning, and query performance of semantic repositories. It therefore creates a benchmark data set, loads it into the semantic repository, infers additional information using the LUBM ontology, and executes 14 queries on this data.

The LUBM benchmark data set describes master and reference data of universities (e.g., statements about professors, students, courses, publications, etc.) and can be generated in different sizes. The 14 benchmark queries are defined in such a way that they allow drawing conclusions about certain performance characteristics of semantic repositories. This way, the performance impact of queries with different *selectivity, query complexity, reasoning complexity*, *numbers of variables*, etc. can be investigated in detail. A comprehensive characterization of these queries is provided in [6].

In first experiments, we created a benchmark for semantic event processing mechanisms with the data and queries defined by the LUBM benchmark framework (see [14]). However, it turned out that the LUBM-based benchmark setting did not address several performance-relevant areas of semantic publish/subscribe systems in enough detail:

- The entire benchmark data set is generated by a data generator component and does not contain any user-provided data. E.g., the name of the resource *u1d1:Professor1* in LUBM is “Professor1”. Although some applications, e.g., planning or coordination systems, might exclusively work with generated data, most applications typically employ both generated and user-provided data.
- About half of the LUBM queries can typically be evaluated very efficiently even with bigger benchmark data sets (i.e., <10ms for a data set size of ~1 million RDF statements). For such queries, it is difficult to analyze efficiency of semantic publish/subscribe systems, as the measurement inaccuracy is almost as high as the actual notification time.
- Several types of queries that are relevant for a comprehensive evaluation of semantic publish/subscribe systems are not covered by the LUBM queries (e.g., queries with FILTER or UNION operators to constrain or extend results sets).

In order to better cover the above points, two extensions for the LUBM setting have been defined: an *Extended Data Set* that defines additional, realistic benchmark data and an *Extended Query Package* consisting of 6 additional benchmark queries.

Extended Data Set

The *Extended Data Set* defines additional realistic data for several parts of the benchmark data set which are extracted from DBPedia[2] [11]. The internet data repository DBPedia contains structured information from Wikipedia in the form of RDF graphs and is used to generate, e.g., first names and surnames for all persons, names for universities, real email addresses, telephone numbers, and titles for publications. As not only existing data is replaced but also new data is added, the data set of a SEP-BM benchmark setting is bigger than the according LUBM data set. For example, a SEP-BM benchmark data set that is generated from LUBM(2)[3] defines 287 897 explicit RDF statements, while the original LUBM(2) data set defines only 230 531 statements.

Extended Query Package

For the evaluation of further characteristics of semantic publish/subscribe systems, 6 additional queries have been defined with a focus on particular performance-relevant aspects of a system (queries Q15-Q20):

- **Queries Q15 & Q16** return results that are combinations of generated and user-provided data. These queries are used to analyze the impact of less homogenous data on the performance of the event processing mechanisms.
- **Queries Q17 & Q18** return results that contain variable bindings which are not contained in one single published graph. These queries are used to investigate the performance of the event processing mechanism when notifications need to be inferred from multiple graph publications.
- **Queries Q19 & Q20** employ more complex WHERE clauses than those used in the LUBM benchmark in order to constrain/extend the queries' result sets. These queries are used to examine how the size of a result set influences the performance of the event processing mechanism.

The query statements and a more detailed description of their characteristics are provided in Appendix A1.

3.2 Benchmark Framework

Fig. 3 shows the developed benchmark framework, consisting of two main applications: the *Configuration Generator*, which is used to create a benchmark configuration

[2] http://wiki.dbpedia.org

[3] The number in brackets defines the number of described universities and is therefore taken as reference number for the size of the used benchmark data set.

description, and the *Benchmark Runner*, which executes the benchmark using this configuration.

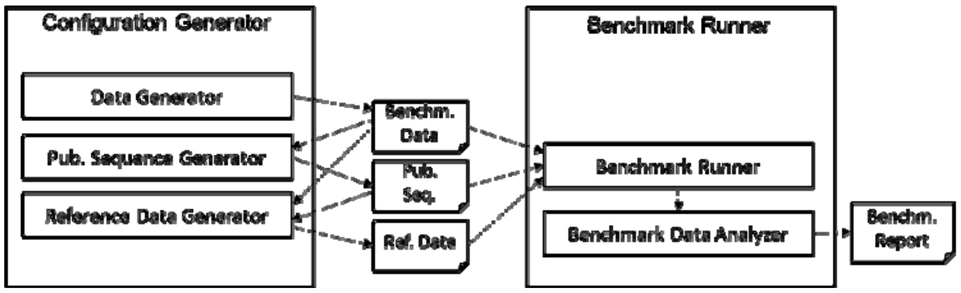

Fig. 3. Benchmark framework components

Configuration Generator

For generating a benchmark configuration description, the *Data Generator* is first employed to generate the benchmark base configuration described in the previous section.

The *Publication Sequence Generator* then creates a description of a sequence of publication operations for a concrete benchmark run. It therefore loads the benchmark data and executes all queries of the Extended Query Package. It then creates single graphs by selecting the concise bounded descriptions[4] of the resources described by the query results and generates the publication sequence by randomly selecting an equal number of graphs for each query.

For the correct measurement of notification times, it is further necessary to know which published graph allows for the generation of which notifications. The *Reference Data Generator* creates a mapping between notifications and publications describing this relationship. It loads the benchmark data, registers all queries, subscribes to these queries, and publishes the graphs of the publication sequence. The benchmark framework thereby always waits with the invocation of the next publish operation until the system has completed the processing of the previous publish operation. No optimization mechanisms are employed and it is therefore guaranteed that the publication operation that first allowed for the generation of a notification is always the one that was invoked immediately before the reception of a notification. Hence, it is trivial for the Reference Data Generator to create the according mapping.

Benchmark Runner

The *Benchmark Runner* executes the actual benchmark. It therefore publishes all benchmark data at the semantic publish/subscribe system to be benchmarked, only omitting those statements contained in the publication sequence. Then it publishes the

[4] Concise bounded description (CBD) [25]: Given a particular node (starting node) in an RDF graph, the concise bounded description of the resource denoted by this node comprises all those statements of the RDF graph where the subject of the statement is the starting node.

graphs of the publication sequence, also each time waiting until the previous iteration has completed, and it records the send times of the publish requests and the reception times of all received notifications.

When the benchmark run is finished, the *Benchmark Data Analyzer* determines the notification times and publication throughput. For a particular notification, the corresponding notification time is determined by calculating the time between the reception of the notification and the publication from which this notification resulted from (as described by the mapping in the reference data file).

For determining the publication throughput, the Benchmark Data Analyzer relates the number of publish operations to the overall benchmark time. Furthermore, it also checks whether all notifications have been generated during the benchmark run by comparing the set of received notifications with those described in the reference data file.

The obtained benchmark results are written to a report file, also containing detailed information about the benchmark setting, number of received notifications, average notification time and sequential publication throughput.

4 Benchmarking a Semantic publish/subscribe system

In the SENS (Semantic Event Notification System) project [15], a framework is developed that adds publish/subscribe functionality to semantic repositories. In this project, the SEP-BM framework was used to find performance bottlenecks and to identify potential areas of optimization.

4.1 First SENS Prototype

In the first prototype of SENS, a very simple event detection approach was chosen, which is also implemented in a number of other systems with semantic publish/subscribe mechanisms (e.g. in TripCom [18] and G-ToPSS [20]). Each time the knowledge base is modified, SENS re-evaluates all registered subscriptions and compares the returned result sets with the results sets of the previous evaluation. It generates notifications for those results not contained in the result sets before and sends them to the subscribers.

To evaluate the performance of this approach, SENS was benchmarked with SEP-BM. For this purpose, a simple adapter had to be implemented, which maps the SENS interface to the six interface operations used by SEP-BM (*publish, delete, createChannel*[5]*, deleteChannel, subscribe, unsubscribe*). As SEP-BM is based on the widely supported Sesame semantic web framework [2], such adapters can usually be implemented with a few lines of code. The benchmark was then executed with a publication sequence of 300 graphs and the 20 queries registered at SENS (C1-C20). The used benchmark data set was generated from the LUBM(10) data set and comprised 1.6 million explicit RDF statements. Using the LUBM ontology, the reasoning engine can infer up to 0.8 million additional implicit statements, resulting in a knowledge base of 2.4 million statements in total.

[5] In SENS, subscriptions are defined as “channels”.

Fig. 4 shows the average notification times measured for SENS with this benchmark configuration[6].

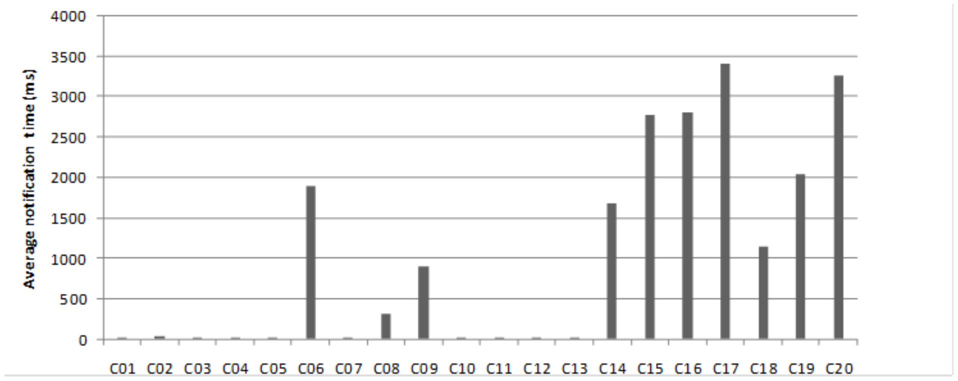

Fig. 4. Notification times of first SENS prototype

While 10 subscriptions could be processed very efficiently, essential delays were observed for the other 10 subscriptions (316-3407ms). For applications that need to provide immediate user-feedback, notification times of 2 seconds and above would probably not be acceptable (C14-C17, C19, C20).

Fig. 5 further shows the publication throughput rates of the first SENS prototype. For better illustration of the critical throughput rates, those rates greater than 25pubs/sec are not shown in the figure (marked with ">25").

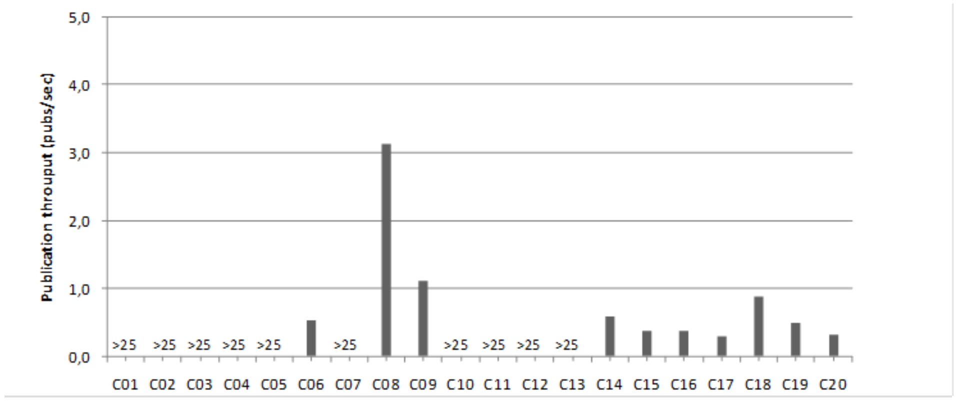

Fig. 5. Publication throughput of first SENS prototype

[6] All benchmarks were run on a Pentium Core2 Duo 3GHz, 4GB RAM Windows Vista PC. For the benchmark runs, server and client were run on the same physical node, so that the communication overhead was minimal and did not have a significant impact on the benchmark results.

Not surprisingly, those channels with low notification times also exhibit high publication throughput rates. The efficient evaluation of the respective queries leads to good results for both metrics.

Interpretation of Benchmark Results

Considering the characteristics of the queries (Appendix A1 and [6]) used in those subscriptions (i.e. channels) with high notification times and low publication throughput, it turns out that at least one of the following two properties applies to each of these queries:

- **Complex reasoning or constraint evaluation**: The execution of the query requires running complex reasoning tasks or involves the processing of expensive constraint evaluations (e.g. string based matching in C19) or join operations (e.g., joins on large data sets in C09).
- **Large input and low selectivity**: Queries with large input and low selectivity lead to large result sets. For the detection of new results, the current result set needs to be compared to the previous result set after each publication operation. Hence, very large result sets may cause significant delays during publication processing.

Based on these observations, a new optimization approach was developed in the SENS project, trying to improve the performance of subscriptions with the above properties. This optimization approach and its evaluation using SEP-BM are described in the following section.

4.2 SENS with Heuristics-based Optimization Algorithms

The developed heuristic-based optimization mechanisms are based on the following observation: In many cases, the publication of *similar* graphs leads to the generation of similar notifications. For example, the publication of new information about papers often leads to new notifications of subscriptions that report updates of university professors. Hence, detecting such similarities will allow for "guessing" which notifications will result from a new publication. Testing whether such guessed notifications are really valid notifications can be implemented efficiently and takes significantly less time than the full evaluation of a subscription. Thus, the generated notifications can be sent to the subscribers earlier than in the first prototype of SENS.

The same mechanisms can also be employed to improve the publication throughput of the system. SENS therefore records how many of the notifications are already generated by the heuristic algorithms and how many of them are only generated during the full evaluation of a subscription. If the rate of heuristic-generated notifications is higher than 80%, then SENS will not evaluate each subscription immediately after a publication operation. It rather relies on the heuristics to generate notifications and only evaluates the subscriptions in certain time intervals. Consequently, more publications can be processed in the same time.

Fig. 6 shows the notification times for SENS without heuristic and for SENS employing two heuristic optimization algorithms [15]: The StringRS Heuristic defines similarity via the textual representation of the published graphs and notifications, and the GraphSS Heuristic defines similarity via the structure of the published graphs.

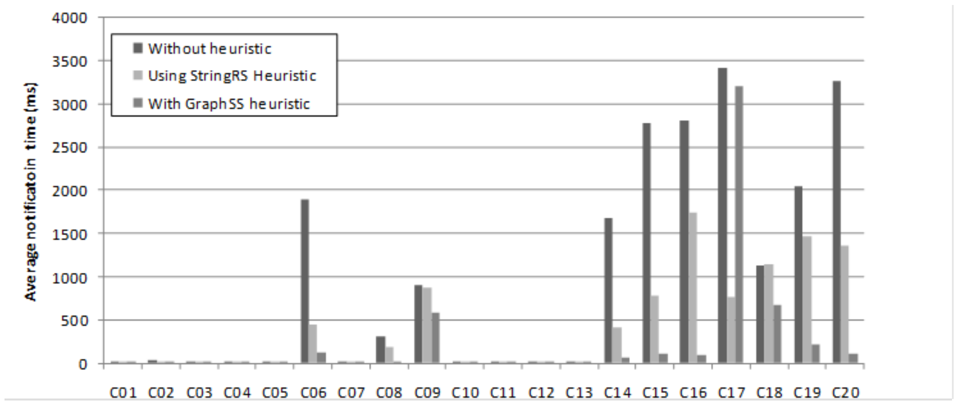

Fig. 6. Notification times of SENS with heuristic optimization

Employing the StringRS Heuristic, good results could be achieved for those scenarios in which mainly generated data was published to the semantic publish/subscribe system. With the GraphSS Heuristic, even better notification times could be achieved. The only exception is C17, where the generation of notifications is only possible based on the textual similarity of graphs and notifications. In summary, the average notification time of all notifications generated during the benchmark could be reduced from 1013ms to 461ms with the StringRS Heuristic and to 260ms with the GraphSS Heuristic.

Fig. 7 further shows the publication throughput achieved by employing heuristic optimization.

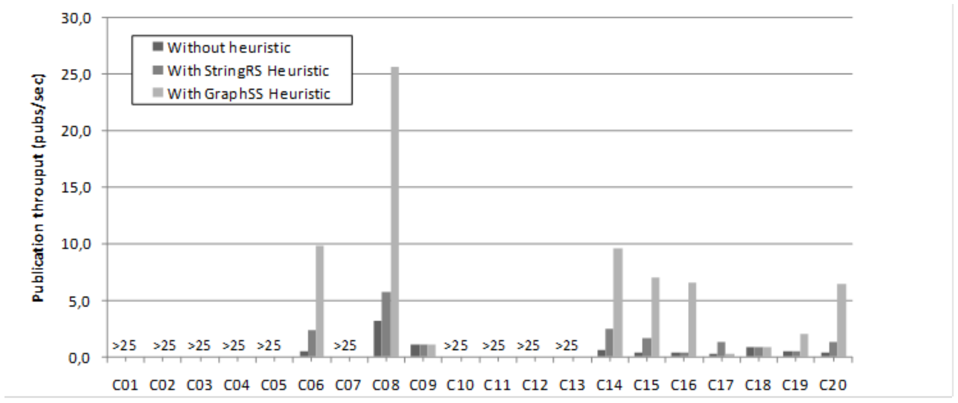

Fig. 7. Publication throughput of SENS with heuristic optimization

The average publication throughput rates of the 10 channels with low publication throughput could be increased by a factor of 1.76 using the StringRS Heuristic and by a factor of 5.9 with the GraphSS Heuristic.

Interpretation of Benchmark Results

By relating the benchmark results to the characteristics of the subscription's query statements, several conclusions about the application of heuristics-based optimization mechanisms can be drawn. First of all, the following properties turn out to represent general prerequisites for the successful employment of heuristic optimization:

P1. Recurring events: A subscription reports a series of events (i.e., it is not only used to detect a one-time event).
P2. Small number of variables: A small number of variables (ideally ≤ 3) are used in a subscription's query statement.
P3. Variable binding values are contained in a single graph: All (or at least most) variable binding values of a notification are contained in a single published graph.
P4. High number of notifications: A subscription generates notifications very frequently.

Furthermore, the following characteristics had positive effects on the application of the StringRS Heuristic:

P5. Graphs of equal size: Graphs leading to notifications of the same subscriptions are frequently of the same size (i.e. consist of the same number of string blocks).
P6. Equal substrings in graphs: Graphs leading to notifications of the same subscription contain a high number of equal substrings.
P7. Equal substrings in notifications: Notifications of the same subscription contain a high number of equal substrings.

Finally, the subsequent characteristics had positive effects on the application of the GraphSS Heuristic:

P8. Same "main" resource of graphs: Graphs leading to notifications of the same subscription frequently describe the same resource type (e.g., publications about papers lead to notifications about professors).
P9. Similar structure of graphs: The structural representations of the published graphs are similar (i.e. they consist of the same RDF property values).
P10. Binding value distribution: If a subscription generates notifications with binding values coming from different publications, better results are achieved if publications lead to at most one notification for this subscription.

5 Conclusion

In this paper we presented SEP-BM, a benchmark framework for semantic publish/subscribe systems. We described a benchmark data set and a number of query statements for this framework and defined two concrete metrics to be measured during a benchmark run. The metric *notification time* is used to evaluate the responsiveness of a semantic publish/subscribe system and the metric *publication throughput* is used to evaluate the overall rate of operations that can be processed in a given time interval.

The application of the benchmark framework in the SENS project allowed us to identify performance-critical use case scenarios and pointed out where optimization mechanisms could help to improve the performance. Based on the evaluation of the benchmark results, a heuristics-based optimization mechanism was developed that tries to take advantage of those properties that originally led to performance problems: (i) the need for complex reasoning and querying tasks during the evaluation of subscriptions and (ii) large input sets and low selectivity of the subscription's query statements. Running the same benchmarks with SENS using the developed optimization mechanisms showed that most of the critical application scenarios could be handled much more efficiently. In summery, the employment of SEP-BM allowed us to (i) identify potential performance bottlenecks of the investigated semantic publish/subscribe system, (ii) gain insights about the causes of these performance bottlenecks and point out directions for possible optimizations, and (iii) quantify the concrete performance improvements when applying optimization mechanisms.

Future work will include the development of a systematic characterization of the benchmark subscription queries. So far, conclusions about certain performance properties could only be provided by a step-by-step interpretation of the textual descriptions of these queries. Here, a precise characterization of the queries with respect to a defined set of criteria will allow for a more systematic and automated evaluation of semantic publish/subscribe systems.

6 References

[1] Christian Bizer and Andreas Schultz. The Berlin SPARQL benchmark. *International Journal on Semantic Web and Information Systems*, 5(2):1–24, 2009.

[2] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF Schema. In *Proceedings of the First Int'l Semantic Web Conference (ISWC)*, volume 2342, pages 54–68, 2002.

[3] Dario Cerizza, Emanuele Della Valle, doug foxvog, Reto Krummenacher, and Martin Murth. Towards European patient summaries based on triple space computing. In *Proceedings of the First European Conference on eHealth (ECEH)*, volume 91, pages 143–154, 2006.

[4] David de Francisco Marcos, Marta de Francisco, Noelia Perez, and German Toro del Valle. Triplespaces as a semantic middleware for telecommunication services development. In *Proccedings of International Symposium on Distributed Computing and Artificial Intelligence*, volume 50 of *Advances in Soft Computing*, pages 309–318. Springer, 2008.

[5] David de Francisco Marcos, Lyndon J. B. Nixon, and German Toro del Valle. Towards a multimedia content marketplace implementation based on triplespaces. In *Proceedings of the 7th International Semantic Web Conference*, volume 5318 of *Lecture Notes in Computer Science*, pages 875–888. Springer, 2008.

[6] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3(2-3):158–182, 2005.

[7] Volker Haarslev and Ralf Möller. RACER system description. In *Int. Joint Conference on Automated Reasoning (IJCAR)*, pages 701–705, Siena, Italy, June 18-23 2001. Springer-Verlag.

[8] Atanas Kiryakov, Damyan Ognyanov, and Dimitar Manov. OWLIM - a pragmatic semantic repository for OWL. In *Proceedings of International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2005), WISE 2005*, volume 3807 of *Lecture Notes in Computer Science*, pages 182–192. Springer, 2005.

[9] Reto Krummenacher, Elena Simperl, Daniel Martin, Michael Lafite, Christian Schreiber, Alessio Carenini, and David de Francisco. Towards a scalable triple space v2.0. TripCom Research Deliverable D6.5v2, 2009. Available at: http://www.tripcom.org/docs/del/-D6.5v2.pdf.

[10] Reto Krummenacher, Thomas Strand, and Dieter Fensel. Triple spaces for an ubiquitous web of services. In *W3C Workshop on the Ubiquitous Web, Tokyo, Japan*, 2006.

[11] Jens Lehmann, Chris Bizer, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165, 2009.

[12] Li Ma, Yang Yang, Zhaoming Qiu, GuoTong Xie, Yue Pan, and Shengping Liu. Towards a complete OWL ontology benchmark. In *Proceedings of the 3rd European Semantic Web Conference*, volume 4011 of *Lecture Notes in Computer Science*, pages 125–139. Springer, 2006.

[13] Brian McBride. Jena: Implementing the RDF model and syntax specification. In *Proceedings of the 2. Int. Workshop on the Semantic Web (SemWeb)*, volume 40 of *CEUR-WS*, Hongkong, China, 2001.

[14] Martin Murth and eva Kühn. A semantic event notification service for knowledge-driven coordination. In *Proceedings of the 1st International Workshop on Emergent Semantics and Cooperation in Open Systems (ESTEEM), Italy, Rome*, 2008. In cooperation with the 2nd International Conference on Distributed Event-Based Systems (DEBS08).

[15] Martin Murth and eva Kühn. A heuristics framework for semantic subscription processing. In *Proceedings of the 6th Annual European Semantic Web Conference (ESWC)*, pages 96–110, June 2009.

[16] Martin Murth and eva Kühn. Knowledge-based coordination with a reliable semantic subscription mechanism. In *Proceedings of the 24th ACM Symposium of Applied Computing (SAC) - Special Track on Coordination Models, Languages and Applications*, pages 1374–1380. ACM, 2009.

[17] Lyndon J. B. Nixon, Dario Cerizza, Emanuele Della Valle, Elena Paslaru Bontas Simperl, and Reto Krummenacher. Enabling collaborative eHealth through triplespace computing. In *Proceedings of the 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 80–85. IEEE Computer Society, 2007.

[18] Lyndon J.B. Nixon, Daniel Martin, Daniel Wutke, Martin Murth, Elena Simperl, Reto Krummenacher, Brahmananda Sapkota, Zhangbing Zhou, Hans Moritsch, Christian Schreiber, Omair Shafiq, German Toro del Valle, Davide Cerri, and Vassil Momtchev. Platform API specification for interaction between all components. TripCom Research Deliverable D6.3, 2009. Available at: http://www.tripcom.org/docs/del/D6.3.pdf.

[19] Ontotext. *SwiftOWLIM Version 2.9.1 System Documentation*, 2007. Available at: http://-www.ontotext.com/owlim/OWLIMSysDoc.pdf.

[20] Milenko Petrovic, Haifeng Liu, and Hans-Arno Jacobsen. G-ToPSS: Fast filtering of graph-based metadata. In *Proceedings of the 14th International World Wide Web Conference (WWW)*, pages 539–547. ACM, 2005.

[21] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, January 2008. Available at: http://www.w3.org/TR/rdf-sparql-query/.

[22] Kai Sachs, Samuel Kounev, Jean Bacon, and Alejandro Buchmann. Performance evaluation of message-oriented middleware using the SPECjms2007 benchmark. *Performance Evaluation*, 66(8):410–434, 2009.

[23] Elena Paslaru Bontas Simperl, Reto Krummenacher, and Lyndon J. B. Nixon. A coordination model for triplespace computing. In *Proceedings of the 9th International Conference on Coordination Models and Languages*, volume 4467 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2007.

[24] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

[25] Patrick Stickler. CBD - Concise Bounded Description. W3C Member Submission, Juni 2005. Available at: http://www.w3.org/Submission/CBD/.

[26] Jinling Wang, Beihong Jin, and Jing Li. An ontology-based publish/subscribe system. In *Proceedings of 5th International Middleware Conference*, pages 232–253, 2004.

Appendix A1 – Extended Query Package

The following prefix is used for all queries.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX uni: <http://www.lehigh.edu/~zhp2/2004/0401/univ-
bench.owl#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX u0d0: <http://www.Department0.University0.edu/>
```

The SEP-BM Extended Query Package consists of the 14 LUBM queries [6] and the following six additional queries:

Query Q15

```
PREFIX …
SELECT ?S ?N
WHERE {
  ?S rdf:type uni:Student.
  ?S uni:name ?N.
}
```

Query Q15 extends query Q06 of the LUBM Benchmark by also retrieving the surname of a student, not only the key URI. Hence, the query result set also contains data generated by the Data Enricher and is therefore much less homogenous than the result set of query Q06.

Query Q16

```
PREFIX …
SELECT ?X ?N ?E
WHERE {
  ?X rdf:type uni:Student.
  ?X foaf:mbox ?E.
}
```

Query Q16 is a modification of query Q15; it retrieves the email address instead of the name of the student. As opposed to the students' names, the email addresses also contain characters that are recognized as delimiter characters by the StringSS Heuristic.

Query Q17

```
PREFIX …
SELECT ?S ?U
  WHERE {
  ?S rdf:type uni:Student.
  ?S uni:memberOf ?D.
  ?D uni:subOrganizationOf ?U.
}
```

Query 17 returns all students together with their department and university. Here, heuristic-based notification generation is more difficult as graphs about students only contain a reference to the student's department but not to its university. However, the university may be derived from the URI string of the department (e.g., the department "http://www.complang.tuwien.ac.at" belongs to the university "http://www.tuwien.ac.at").

Query Q18

```
PREFIX …
SELECT ?S ?C ?N
WHERE {
  ?S rdf:type uni:GraduateStudent.
  ?S uni:takesCourse ?C.
  ?C uni:name ?N.
}
```

Query Q18 retrieves, similar to Q17, results with variable binding values that need to be inferred from multiple published graphs. However, the heuristics-based generation of notifications for a channel with this query is even more difficult, since none of the variables directly contains information about any other variable.

Query Q19

```
PREFIX …
SELECT ?S
WHERE {
  ?S rdf:type uni:Student.
  ?S foaf:surname ?N.
  FILTER regex(?N, \"^(A|B|C|D)\")
}
```

Query Q19 employs a FILTER expression for defining an additional constraint for returned results. Consequently, the processing effort during the query evaluation phase is increased although the result set will be smaller than without using the FILTER expression.

Query Q20

```
PREFIX …
SELECT ?P ?N
WHERE {
  {?P rdf:type uni:Student.
   ?P uni:name ?N. }
  UNION
  {?P rdf:type uni:Employee.
   ?P uni:name ?N. }
}
```

Query Q20 combines the results of two queries by means of the UNION operator in the WHERE clause. This leads to a larger result set and increases the variability of the graphs that lead to notifications for the corresponding subscriptions.