LarKC

The Large Knowledge Collider:
a platform for large scale integrated reasoning and Web-search

FP7 – 215535

D5.5.4 Report on platform validation and recommendation for the final version

Coordinator: Vassil Momtchev (ONTO)

With contributions from: Matthias Assel (HLRS), Alexey Cheptsov (HLRS), Luka Bradeško (CycEur), Christoph Fuchs (UIBK), Spyros Kotoulas (VUA), Gaston Tagni (VUA)

Quality Assessor: Irene Celino (CEFRIEL)
Quality Controller: Matthias Assel (HLRS)

Document Identifier:	LarKC/2008/D5.5.4/v1.0
Class Deliverable:	LarKC EU-IST-2008-215535
Version:	1.0
Date:	30/04/2011
State:	Ready for peer review
Distribution:	PUBLIC

EXECUTIVE SUMMARY

This deliverable aims to give an update of the definition of the validation goals and metrics for the LarKC platform, together with results and interpretations. The document does a two-fold evaluation from one side; it asses the recommendation given by its predecessor and on the other it validates the final LarKC platform architecture by discussing a range of platform features, as well as ongoing implementation work and future goals. The result is a thorough overview of platform components and behaviours, with a clear path for achieving the defined targets.

Finally, the deliverable presents recommendations for the last release of the LarKC platform that will make easier the design, testing and exploitation of various reasoning techniques to be done at vastly reduced costs.

DOCUMENT INFORMATION

IST Project Number	FP7 - 215535	**Acronym**	LarKC
Full Title	The Large Knowledge Collider: a platform for large scale integrated reasoning and Web-search		
Project URL	http://www.larkc.eu/		
Document URL			
EU Project Officer	Stefano Bertolo		

Deliverable	**Number**	D5.5.4	**Title**	Report on platform validation and recommendation for the final version
Work Package	**Number**	5	**Title**	The Collider Platform

Date of Delivery	**Contractual**	M37	**Actual**	
Status	version 1.0		final □	
Nature	prototype □ report x dissemination □			
Dissemination level	public x consortium □			

Authors (Partner)	Matthias Assel (HLRS), Barry Bishop (ONTO), Luka Bradeško (CycEur), Alexey Cheptsov (HLRS), Christoph Fuchs (UIBK), Spyros Kotoulas (VUA), Vassil Momtchev (ONTO), Gaston Tagni (VUA)		
Responsible Author	**Name**	Vassil Momtchev	**E-mail**	vassil.momtchev@ontotext.com
	Partner	Ontotext	**Phone**	

| **Abstract (for dissemination)** | This deliverable aims to give an update of the definition of the validation goals and metrics for the LarKC platform, together with results and interpretations. The document does a two-fold evaluation from one side it asses the recommendation given by its predecessor and on the other it validates the final LarKC platform architecture by discussing a range of platform features, as well as ongoing implementation work and future goals. The result is a thorough overview of platform components and behaviours, with a clear path for achieving the defined targets.

Finally, the deliverable presents recommendations for the last release of the LarKC platform that will make easier the design, testing and exploitation of various reasoning techniques to be done at vastly reduced costs. |
|---|---|
| **Keywords** | Platform, validation, data-layer, plug-in, parallel, multi-threaded, OWLIM, WebPIE, JavaGAT, JEE, Servlet |

Version Log

Issue Date	Rev. No.	Author	Change
02/01/2011	0.1	Vassil Momtchev	Initial skeleton
08/03/2011	0.2	Vassil Momtchev	First draft of the document
05/04/2011	0.3	Vassil Momtchev	Integrated all contributions
26/04/2011	0.7	Vassil Momtchev	Send for QA review
30/04/2011	1.0	Vassil Momtchev	Implemented reviewers comments

PROJECT CONSORTIUM INFORMATION

Participant's name	Partner	Contact
Semantic Technology Institute Innsbruck, Universitaet Innsbruck		Dieter Fensel, Semantic Technology Institute (STI), Universitaet Innsbruck, Innsbruck, Austria, E-mail: dieter.fensel@sti-innsbruck.at
AstraZeneca AB		Bosse Andersson, AstraZeneca Lund, Sweden Email: bo.h.andersson@astrazeneca.com
CEFRIEL - SOCIETA CONSORTILE A RESPONSABILITA LIMITATA		Emanuele Della Valle, CEFRIEL - SOCIETA CONSORTILE A RESPONSABILITA LIMITATA, Milano, Italy, Email: emanuele.dellavalle@cefriel.it
CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O.		Michael Witbrock, CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O., Ljubljana, Slovenia, Email: witbrock@cyc.com
Höchstleistungsrechenzentrum, Universitaet Stuttgart		Matthias Assel, Höchstleistungsrechenzentrum, Universitaet Stuttgart, Stuttgart, Germany, Email: assel@hlrs.de
MAX-PLANCK GESELLSCHAFT ZUR FOERDERUNG DER WISSENSCHAFTEN E.V.		Lael Schooler, Max-Planck-Institut für Bildungsforschung Berlin, Germany Email: schooler@mpib-berlin.mpg.de
Ontotext AD		Atanas Kiryakov, Ontotext Lab, Sofia, Bulgaria Email: naso@ontotext.com
SALTLUX INC.		Kono Kim, SALTLUX INC, Seoul, Korea, Email: kono@saltlux.com

SIEMENS AKTIENGESELLSCHAFT		Volker Tresp, SIEMENS AKTIENGESELLSCHAFT, Muenchen, Germany, E-mail: volker.tresp@siemens.com
THE UNIVERSITY OF SHEFFIELD		Hamish Cunningham, THE UNIVERSITY OF SHEFFIELD Sheffield, UK, Email: h.cunningham@dcs.shef.ac.uk
VRIJE UNIVERSITEIT AMSTERDAM		Frank van Harmelen, VRIJE UNIVERSITEIT AMSTERDAM, Amsterdam, Netherlands, Email: Frank.van.Harmelen@cs.vu.nl
THE INTERNATIONAL WIC INSTITUTE, BEIJING UNIVERSITY OF TECHNOLOGY		Ning Zhong, THE INTERNATIONAL WIC INSTITUTE, Mabeshi, Japan, Email: zhong@maebashi-it.ac.jp
INTERNATIONAL AGENCY FOR RESEARCH ON CANCER		Paul Brennan, INTERNATIONAL AGENCY FOR RESEARCH ON CANCER, Lyon, France, Email: brennan@iarc.fr
INFORMATION RETRIEVAL FACILITY		John Tait, INFORMATION RETRIEVAL FACILITY Vienna, Austria Email : john.tait@ir-facility.org
TECHNICAL UNIVERSITY OF CLUJNAPOCA		Prof. Dr. Eng. Sergiu Nedevschi TECHNICAL UNIVERSITY OF CLUJNAPOCA Cluj-Napoca, Romania Email: sergiu.nedevschi@cs.utcluj.ro
SOFTGRESS S.R.L.		Dr. Ioan Toma SOFTGRESS S.R.L. Cluj-Napoca, Romania Email: ioan.toma@softgress.com

TABLE OF CONTENTS

List of Figures

List of Tables

List of Acronyms

Acronym	Description
API	Application Programming Interface
CAN	Content–addressable network
COMPS	Component Superscalar
DEISA	Distributed European Infrastructure for Supercomputing Applications
DMP	Distributed Memory Parallel
EC2	Elastic Compute Cloud
EGEE	Enabling Grids for the E-Science
FF	Fact Forge
GT4	Globus Toolkit 4
HPC	High Performance Computing
JARSTOP	Java RDF Stored Procedure
JEE	Java Platform, Enterprise Edition
JavaGAT	Java Grid Application Toolkit
KB	Knowledge Base
LarKC	Large Knowledge Collider
LLD	Linked Life Data
LORAPI	Low level RDF APIs
LRMS	Local Resource Management System
MPI	Message Passing Interface
ORDI	Ontology Representation and Data Integration
OWL	Ontology Web Language
P2P	Peer-to-peer
PBS	Portable Batch System
QoS	Quality of Service
RAM	Random Access Memory
RDF	Resource Description Framework
RDFS	RDF Schema
REETON	RDF Engine Extension
SAWSDL	Semantic Annotations for WSDL
SCP	Scalable Concurrent Programming
SMP	Symmetric multi-processing
SSH	Secure Shell
WSDL	Web Services Description Language

1. Introduction

The main target of the deliverable is the validation of the final LarKC platform architecture and design, as described in D5.3.3 [3]. Basically, the aim of the deliverable is two-fold. On one hand, it evaluates the level of fulfilment of the recommendations given in D5.5.3 [6] for the new architecture and design, based on the feedback from the application community in particular. On the other hand, the document investigates the final architecture and design of the LarKC, evaluates the main platform aspects and provides recommendations for improvements of the final release of the platform (M36), based on D5.4.2 [4].

The document is built upon the outcome of the following previous deliverables:

- D5.4.2 [4] – describes the second release of the LarKC platform;
- D5.3.3 [3] – presents the final architecture and design of the LarKC platform;
- D.5.5.3 [6] – summarises the validation goals and metrics for the 1st public release of the LarKC platform, done in M24.

All in all, it presents the updated definition of the validation goals and metrics for the LarKC platform, based on D5.5.3[6]. It also describes the LarKC platform subsystems evaluation, as conceptualised in D5.3.3 [3] (see Figure 1).
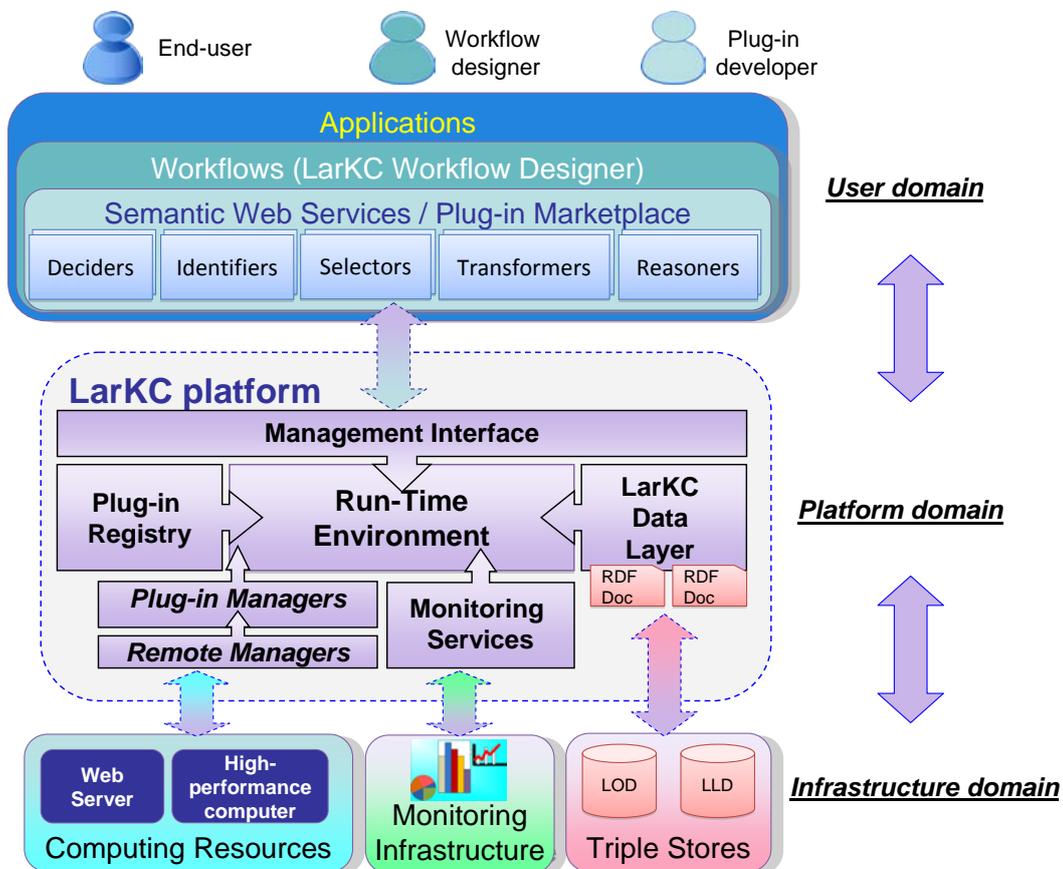
Figure 1. The final LarKC platform architecture and main subsystems

Chapter 2 targets the Data Layer, which is a core platform component that enables the exchange and persistent storage of highly heterogeneous information, represented in the RDF data format. Chapter 3 verifies and validates important recommendations concerning the LarKC Runtime Environment

(RTE), Workflow Support System and Plug-in Managers, as implemented in the second LarKC platform release [5]. Chapter 4 present a graphical tool, the LarKC Workflow Designer, aiming for easy creation, management and execution of workflows.

Finally, the document summarises all the actual and emerging requirements regarding the final LarKC platform release.

2. Data Layer

The LarKC architecture has evolved into a flexible workflow platform capable of managing complex sequences of plug-in executions. The data layer is a core platform component that enables the exchange and persistence of highly heterogeneous information represented in the RDF data format. A key critical success factor for the overall architecture is the offering of the right data layer service methods to manage the processed information in a highly efficient and flexible way.

The preceding deliverable, D5.5.3 Validation goals and metrics for the LarKC platform [6], provided many examples of data sets and benchmarks that can be used to measure the performance of the LarKC data layer. The same 'metrics' can be also used to monitor the continuing evolution of the data layer, with BigOWLIM at its core, both in terms of:

- how well-known data sets, e.g. FactForge, increase in size (and possibly complexity) and continue to be managed by the data layer;

- how its performance improves over time – it should get faster to load and reason with a fixed dataset.

2.1. Updated dataset and performance figures

Fact Forge

The latest FactForge[1] dataset statistics, formerly Linked Data Semantic Repository (LDSR), are given in Table 1. The total number of explicit statements loaded in this dataset has grown to around 1.5 billion statements, nearly four times larger than in the beginning of LarKC project in 2008.

Table 1. FactForge dataset statistics (LDSR4), December 2010

Dataset	Explicit Indexed Triples ('000)	Inferred Indexed Triples ('000)	Total # of Indexed Triples ('000)	Entities ('000 of nodes in the graph)	Inferred closure ratio
Schemata (Proton, SKOS, FOAF, RSS, DC) and ontologies (DBpedia, Geonames)	15	9	23	8	0.6
DBpedia (SKOS categories)	2,915	47,837	50,751	1,135	16.4
DBpedia (owl:sameAs)	9,123	0	9,124	12,758	0.0
NY Times	574	328	902	185	0.6
UMBEL	4,638	6,936	11,575	1,190	1.5
Lingvoj	20	182	201	18	9.2
CIA Factbook	76	4	80	24	0.1
WordNet	1,943	6,067	8,010	842	3.1
Geonames	142,011	194,191	336,202	42,738	1.4
DBpedia core	825,162	166,740	991,902	125,803	0.2
Freebase	494,344	52,411	546,754	123,511	0.1
MusicBrainz	45,492	36,572	82,064	15,585	0.8

---

[1] http://factforge.net/

Total		1,526,312	511,277	2,037,589	323,797	0.3

In order to give some idea of the progress, we have extended a small section of Table 3 from D5.5.2 [6] with the loading information for LDSR3 and LDSR4 from April 2010 and December 2010 respectively.

Table 2. Updated loading time and dataset statistics for LDSR - loading speed includes inferred statements

Engine	Dataset	Scale (mill. ex.st.)	Speed (KSt./ sec.)	Overall Complexity Rate	Loading Time (hours)	Forward-Chaining Semantics
BigOWLIM 3.1	LDSR1	357	14,167	28	7.00	OWL-Horst
BigOWLIM 3.3	*LDSR3*	*1,177*	*6,107*	*40*	*93.6*	*OWL-Horst*
BigOWLIM 3.4	*LDSR4*	*1,526*	*15,952*	*40*	*35.5*	*OWL-Horst*

The results for BigOWLIM version 3.4 show a sustained loading and inference speed of just fewer than sixteen thousand explicit and inferred statements per second. This compares favourably with a slightly lower speed for BigOWLIM 3.1, which used a much smaller dataset.

Lehigh University Benchmark (LUBM)

The Lehigh University Benchmark is often used as a performance benchmark for semantic repositories due to its semi-realistic data, inference complexity, and broad range of query tasks. LUBM (8000), which contains data for 8000 universities with over 1 billion explicit statements, represents an upper bound on scale that the world-leading semantic repository vendors are tentatively willing to provide performance statistics for. Several vendors provide loading times for their products with this dataset, although they do not provide complete information on the level of semantics applied, or thoroughness of the query evaluation process.

Nevertheless, this benchmark is often used for measuring the performance of BigOWLIM and is also used for measuring the performance of WebPIE (refer to Section 3.3 – WebPie for further details). This dataset will be used as a basis for the experiment conducted in the current chapter.

Table 3. Updated loading details for LUBM (8000) - BigOWLIM version 3.4

Engine	Dataset	Scale (mill. ex.st.)	Speed (KSt./ sec.)	Overall Complexity Rate	Loading Time (hours)	Forward-Chaining Semantics
BigOWLIM 3.1	LUBM(8k)	1,068	20,631	15	14.38	OWL-Horst
BigOWLIM 3.1	LUBM(8k)	1,068	66,196	5	4.48	none
BigOWLIM 3.3	*LUBM(8k)*	*1,068*	*21,010*	*15*	*14.12*	*OWL-Horst*
BigOWLIM 3.3	*LUBM(8k)*	*1,068*	*88,046*	*5*	*3.37*	*none*
BigOWLIM 3.4	*LUBM(8k)*	*1,068*	*21,010*	*15*	*14.04*	*OWL-Horst*
BigOWLIM 3.4	*LUBM(8k)*	*1,068*	*88,046*	*5*	*3.25*	*none*

Updated values for the loading time of LUBM (8000) with the latest version of BigOWLIM are given in Table 3. Some improvement in loading times is noticeable with both OWL-Horst and 'empty' semantics, with the corresponding improvement in loading speed.

2.2. Community benchmarking activities

The Berlin SPARQL Benchmark (BSBM)[2] has been extended to version 3 with two new scenarios. What was BSBM v1 and v2 is now the Explore scenario in version 3, which also includes an Explore-and-Update scenario and a business intelligence scenario. The new scenarios make use of SPARQL 1.1 features, which provide a data manipulation language, i.e. insert and delete, as well as aggregate functions, e.g. COUNT(), SUM(), AVG().

The most recent BSBM test run was conducted in February 2011[3], by the BSBM team at the Freie Universität, Berlin. The stores under test were: 4store[4], BigData[5], BigOWLIM[6], Jena/TDB[7] and Virtuoso[8]. BigOWLIM was the top performer on:

- loading time - BigOWLIM loads the 100M and the 200M data sets almost twice as fast as the next best product;
- query performance – BigOWLIM is among those repositories that can handle update and multi-client query tasks.

The business intelligence scenario caused many problems for the stores tested, so it was decided not to publish the results of this round.

BigOWLIM did very well in this benchmark and was able to provide the necessary functionality for all published scenarios (BigData and Virtuoso were not able to participate in the Explore and update scenario, 4store was not able to participate in the Multi-Client, Explore use case). BigOWLIM was by far the fastest at loading and inferencing, but did not do so well at query performance, indicating where future optimisation work should concentrate.

2.3. Moving computation closer to the data – BigOWLIM plug-ins

As discussed in LarKC deliverable 5.3.3 [4], BigOWLIM now includes a plug-in API. The intention is to allow plug-in components to be created *at the data level* that can alter or enhance storage, reasoning and query answering. Several plug-ins have been created that extend BigOWLIM and these fall in to several categories as shown in Table 4 (this list is not exhaustive).

Table 4. A list of BigOWLIM plug-in components, current and planned.

Category	Plug-ins
New data access method	Node Search – pattern matching over tokenised literals and URIs
	RDF Search – Lucene –based full-text search over RDF Molecules
Ranking and Selection	RDF Priming – selection technique based on activation spreading
	RDF Rank – allow ordering of results by popularity or 'interconnectedness'
Indexing/query optimisation	Geo-spatial – extensions for efficient storage and more expressive

---

[2] http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/V3/spec/index.html

[3] http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/results/V6/index.html

[4] http://4store.org/

[5] http://www.systap.com/bigdata.htm

[6] http://www.ontotext.com/owlim/

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

[8] http://virtuoso.openlinksw.com/

	query constraints
	Temporal – extensions for efficient interval algebra (planned)
	XPath – integrate XPath sub-queries in to SPARQL that are evaluated over rdf:XMLLiteral values (planned)

Some plug-ins provide functionalities that go beyond RDF/RDFS/OWL storage and reasoning. Examples of these are:

- the text search plug-in - allows full-text queries to be embedded into SPARQL.
- RDF Rank - exposes the interconnectedness of an entity (cf. Google PageRank) via a special system predicate that can be used to order query results with a SPARQL ORDER BY clause.
- RDF Priming - allows the priming of a graph by setting activation values for certain nodes and propagating these values through the graph until they fall below some specified threshold.

See LarKC deliverable D2.4.2 [2] for details.

Many of these techniques provide mechanisms that were not present in an RDF/OWL setting beforehand and therefore would not allow a direct comparison of the utility of this approach. However, a more recent BigOWLIM component that does allow a direct measurement of the efficiency of implementing *on top of the data layer* compared to an implementation directly *in the data layer* is the geo-spatial plug-in. The remainder of this section discusses the functionality provided by this BigOWLIM plug-in and a comparison of the performance of answering geo-spatial queries with and without the plug-in.

The geo-spatial plug-in is designed to facilitate the formulation of queries with geo-spatial constraints, and to provide the necessary indexing components, to make the processing of geo-spatial data as efficient as possible. Specifically, this plug-in supports the 2-dimensional geo-spatial data that uses the WGS84 Geo Positioning RDF vocabulary (World Geodetic System 1984). A special index is created for this data based on an R-Tree [9], which permits the efficient evaluation of special query forms and extension functions that allow to find locations that are:

- within a certain distance of a point, i.e. within the specified circle on the surface of the sphere (Earth), using the nearby(...) construction;
- within rectangles and polygons, where the vertices are defined using spherical polar coordinates, using the within(...) construction.

The WGS84 ontology contains several classes and predicates:

Table 5. The WGS84 Ontology

Element	Description
SpatialThing	The class representing anything with spatial extent, i.e. size, shape or position.
Point	The class representing a point (relative to Earth), defined using latitude, longitude (and altitude). subClassOf http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing
location	The relation between a thing and where it is. range SpatialThing subPropertyOf http://xmlns.com/foaf/0.1/based_near
lat	The WGS84 latitude of a SpatialThing (decimal degrees). domain http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing
long	The WGS84 longitude of a SpatialThing (decimal degrees). domain http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing

lat_long	A comma-separated representation of a latitude, longitude coordinate.
alt	The WGS84 altitude of a SpatialThing (decimal meters above the local reference ellipsoid). domain http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing

The special syntax for querying geo-spatial data works with the SPARQL's RDF Collections syntax[9]. This syntax uses round brackets as a shorthand for the statements, connecting a list of values using `rdf:first` and `rdf:rest` predicates, by terminating `rdf:nil`. Statement patterns that use one of the special geo-spatial predicates are intercepted by the geo-spatial plug-in and interpreted differently. The following special syntax is supported when evaluating SPARQL queries (all descriptions have the namespace `omgeo: <http://www.ontotext.com/owlim/geo#>`):

?point omgeo:nearby(?lat ?long ?distance)

This statement pattern will be evaluated as true, if the following constraints are valid:

- ?point geo:lat ?plat .
- ?point geo:long ?plong .
- Shortest great circle distance from ?plat, ?plong to ?lat, ?long <= ?distance

?point omgeo:within(?lat$_1$?long$_1$?lat$_2$?long$_2$)

This statement pattern is used to test/find points that lie within the rectangle specified by the diagonally opposite corners ?lat1 ?long1 and ?lat2 ?long2.

?point omgeo:within(?lat$_1$?long$_1$ ... ?lat$_n$?long$_n$)

This statement pattern is used to test/find points that lie within the polygon, whose vertices are specified by three or more latitude/longitude pairs.

double omgeo:distance(?lat$_1$, ?long$_1$, ?lat$_2$, ?long$_2$)

This SPARQL extension function computes the distance between two points in kilometres and can be used in FILTER and ORDER BY clauses.

In order to measure the performance improvement when using this plug-in with geo-spatial data, a BigOWLIM repository was loaded with the GeoNames[10] dataset (72,748,078 statements). Then, a pair of parameterised queries was executed - one that makes use of the geo-spatial syntax and index, and another that does not. The first query uses FILTER clauses to properly locate the 2D points:

```
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX gn: <http://www.geonames.org/ontology#>
SELECT * WHERE {
    ?link geo:lat ?lat.
    ?link geo:long ?long.
    ?link gn:name ?name .
    FILTER(xsd:float(?lat)  >= i ) .
    FILTER(xsd:float(?lat)  <= i+1 ) .
```

---

[9] http://www.w3.org/TR/rdf-sparql-query/#collections

[10] http://www.geonames.org/

```
        FILTER(xsd:float(?long) >= j ) .
        FILTER(xsd:float(?long) <= j+2 ) .
    }
```

The second query uses the special 'within' syntax:

```
    PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
    PREFIX gn: <http://www.geonames.org/ontology#>
    PREFIX omgeo: <http://www.ontotext.com/owlim/geo#>
    SELECT *
    WHERE {
        ?link omgeo:within( "i" "j" "i+1" "j+2" ) .
        ?link gn:name ?name .
        ?link geo:lat ?lat .
        ?link geo:long ?long .
    }
```

For both queries, the parameters i, j are varied as follows:

- i   from -90 to 90 step 20

- j   from 0 to 180 step 20

leading to one hundred variations of both queries, 45 of which return one or more results. The total execution times for both query forms are given in Table 6.

Table 6. Results of comparing equivalent queries that either use or do not use the geo-spatial extension component

Query type	Total execution time (100 queries)
Filter (no geo-spatial extensions)	3712431ms
Within (uses geo-spatial extensions)	7320ms

Both query runs were conducted on an initialised repository in order to avoid the effects of caching. As can be seen from Table 6, when using the geo-spatial constraint and corresponding index, there is an approximately five hundred-fold improvement in performance.

2.4.   *Summary of the changing performance benchmark landscape*

The intensive activity in the Linked Open Data community helps to foster projects such as FF and LLD and this can be seen in their ever growing size. Engineering advances manage to keep up with the growing amount of data and its relative complexity.

LUBM continues to be a very common benchmark and it is encouraging to see that the performance of BigOWLIM, and hence the LarKC data layer, is slowly improving the processing performance associated with this benchmark.

Encouraging is also the continued input from the community regarding independent benchmarking activities. Hopefully this trend will continue with all improved datasets and methodologies.

Finally, the metrics set out in D5.5.2 [6] still hold and the performance of the LarKC data layer continues in this direction. Let's hope that a 'quantum leap' towards these goals is possible, as discussed in chapter 5.

3. Workflow support system

The LarKC Runtime Environment (RTE) and Plug-in Managers are key platform components, responsible for instantiating and invoking the workflow descriptions and the assembly of LarKC applications. The two components are responsible for the workflow branching, conditional loops, splits and merges of the information flow by coupling various RDF plug-in input and output. This chapter performs a functional validation of the benefits to support this feature and verifies the correct implementation by performing a series of tests.

3.1. Control Flow

Recommendation 1: *The software components (plug-ins or separate utilities) for splitting and merging a stream of RDF statement data should be implemented for the next public release and verified through inclusion in to more than one working pipelines that demonstrate a measurable performance increase over current alternative approaches.*

Although multi-threading of particular plug-in algorithms is not foreseen as a direct feature provided by the Platform, the realisation of this parallelisation technique shall be applied for the main plug-ins (i.e., those being developed by WP2, 3 and 4) and the best practices will be provided as a feedback to all plug-in developers.

After the introduction of the workflow description ontology in version 2.0 of the LarKC platform, it became possible to define graph-like workflows that allow a more flexible way of connecting plug-ins. Previous platform versions were limited to "pipeline"-like workflows, which only allowed the sequential execution of one plug-in after another.

The enhanced workflow mechanisms inherently offer the possibility to divide the data flow by connecting one plug-in to multiple successor plug-ins ("split"), effectively enabling two or more plug-in instances to run in parallel. Further, a plug-in can have multiple predecessors, allowing the combination of multiple data flows ("merge").

To provide a functional evaluation, a plug-in has been implemented to simulate a computational intense operation. The internal operation of the plug-in takes exactly 2s, effectively blocking the execution of any following plug-ins for this amount of time. As specified in the LarKC plug-in interface, this operation has been implemented in the `invokeInternal` method. No other methods of the plug-in are essential for this functional evaluation, since initialisation and shutdown (represented by the methods `initialiseInternal` and `shutdownInternal`) are not called during the execution, but on workflow initialisation and workflow destruction respectively.

Two workflows based on the plug-in described above have been designed.

Serial workflow

First, a serial workflow has been designed that declares five plug-in instances (all of the same type) and connects them in a sequential fashion. Listing 1 shows the full workflow description in N3.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix larkc: <http://larkc.eu/schema#> .


# Define all plug-in instances
_:plugin1 a <urn:eu.larkc.plugin.example.IdlePlugin> .
_:plugin2 a <urn:eu.larkc.plugin.example.IdlePlugin> .
_:plugin3 a <urn:eu.larkc.plugin.example.IdlePlugin> .
_:plugin4 a <urn:eu.larkc.plugin.example.IdlePlugin> .
_:plugin5 a <urn:eu.larkc.plugin.example.IdlePlugin> .
```

```
# Connect the plug-ins
_:plugin1 larkc:connectsTo _:plugin2 .
_:plugin2 larkc:connectsTo _:plugin3 .
_:plugin3 larkc:connectsTo _:plugin4 .
_:plugin4 larkc:connectsTo _:plugin5 .


# Define a path to set the input and output of the workflow
_:path a larkc:Path .
_:path larkc:hasInput _:plugin1 .
_:path larkc:hasOutput _:plugin5 .


# Connect an endpoint to the path
_:ep a <urn:eu.larkc.endpoint.test> .
_:ep larkc:links _:path .
```

Listing 1. Workflow descirption in N3 describing a serial connection between five plug-in instances

Figure 2 shows a graphical representation of the depicted workflow. Note that the path and endpoint described in the workflow are omitted in the graphic, since they are of no vital importance for this evaluation.
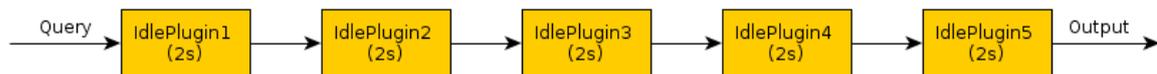

Figure 2 Graphical representation of the sequential workflow

Parallel workflow

The second workflow consists of the same type of plug-ins, but is connected in a different manner. We utilise the *splits* and *merges* to execute three plug-in instances in parallel. Note that the splits and merges are not defined explicitly in the workflow description. The plug-in managers (see package `eu.larkc.core.pluginManager`) will forward the output of a plug-in to all its successors, effectively *splitting* the data flow by duplicating the data stream. Furthermore, the plug-in managers will gather and merge RDF triples from all predecessors of a plug-in, *merging* the data flow by combining data from multiple plug-in instances. Note that no partitioning of the data is done by the plug-in managers. Instead, it can be achieved by implementing a plug-in that acts as a filter.

Figure 3 shows a graphical representation of the parallel workflow. For the full workflow description of the parallel workflow see Listing 2.
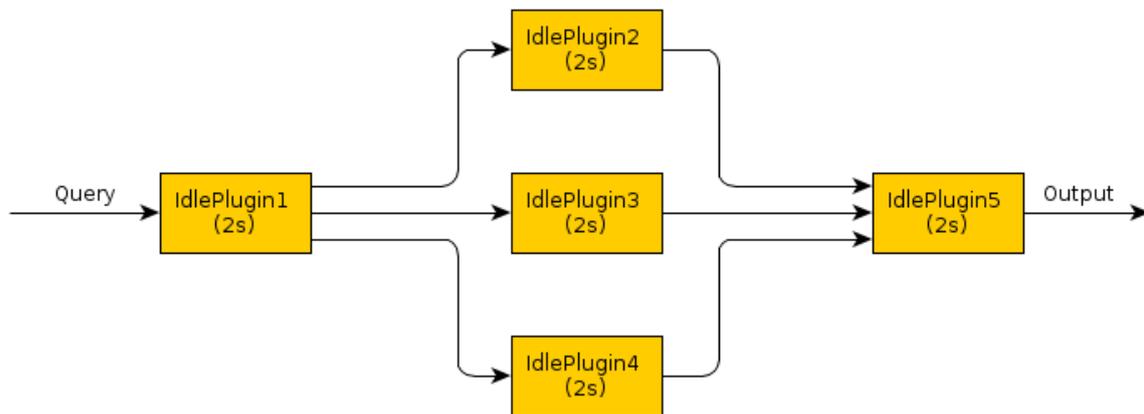
Figure 3. Represenation of the parallel workflow

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix larkc: <http://larkc.eu/schema#> .


# Define two plug-ins
_:plugin1 a <urn:eu.larkc.plugin.example.IdlePlugin> .
_:plugin2 a <urn:eu.larkc.plugin.example.IdlePlugin> .
_:plugin3 a <urn:eu.larkc.plugin.example.IdlePlugin> .
_:plugin4 a <urn:eu.larkc.plugin.example.IdlePlugin> .
_:plugin5 a <urn:eu.larkc.plugin.example.IdlePlugin> .


# Connect the plug-ins
_:plugin1 larkc:connectsTo _:plugin2 .
_:plugin1 larkc:connectsTo _:plugin3 .
_:plugin1 larkc:connectsTo _:plugin4 .
_:plugin2 larkc:connectsTo _:plugin5 .
_:plugin3 larkc:connectsTo _:plugin5 .
_:plugin4 larkc:connectsTo _:plugin5 .


# Define a path to set the input and output of the workflow
_:path a larkc:Path .
_:path larkc:hasInput _:plugin1 .
_:path larkc:hasOutput _:plugin5 .


# Connect an endpoint to the path
_:ep a <urn:eu.larkc.endpoint.test> .
_:ep larkc:links _:path .
```

Listing 2. Workflow description in N3 describing a parallel worlkflow consisting of five plug-in instances

Merging the data flow is done in a straight-forward manner, as explained above. However, when a plug-in has multiple inputs, the plug-in manager needs to know when the plug-in can be invoked. The current implementation utilises the `larkc:hasInputBehaviour` parameter, which is an integer

value representing how many inputs are needed for the plug-in to fire. If no `larkc:hasInputBehaviour` parameter is given the plug-in will fire as soon as one input is ready. In the parallel workflow example described above a blocking behaviour can be achieved by describing an input behaviour to wait for three inputs. Listing 3 shows how the example from Listing 2 can be extended to guarantee that all plug-ins before the last one have finished execution. In this particular example the execution time is indifferent from the input behaviour, since all plug-in instances, which are executed in parallel, have the same execution time.

```
...


# Define an input behaviour for plug-in instance nr. 5
_:plugin5 larkc:hasInputBehaviour 3 .


...
```

Listing 3. Specification of non-default input behaviour for plug-in instance 5 of the parallel workflow

Functional evaluation

To get reliable measurements each workflow has been initialised and executed five times. Since this functional evaluation is in regard to splits and merges of the data flow, as well as parallel execution of multiple plug-in instances, the measurement that is of most relevance is the execution time. Table 7 shows the measurements taken of all ten successive runs.

Table 7. Initialisation and execution times of five successive runs for sequential and parallel workflow graphs

Run	Workflow	Initialization time (seconds)	**Execution time (seconds)**
1	Sequential	1,214	**10,016**
2	Sequential	1,602	**10,063**
3	Sequential	1,628	**10,072**
4	Sequential	2,023	**10,112**
5	Sequential	1,854	**10,051**
1	Parallel	0,616	**6,018**
2	Parallel	0,520	**6,048**
3	Parallel	0,569	**6,037**
4	Parallel	0,530	**6,033**
5	Parallel	0,556	**6,018**

Table 8 shows the median initialisation and execution time for each workflow. As expected we can see an increased execution time of 4 seconds, which translates to a linear speedup since three plug-in instances are run in parallel.

Table 8. Median of the five initialisation and execution times for sequential and parallel workflows

Workflow	Median initialisation time (seconds)	**Median execution time (seconds)**
Sequential	1,628	**10,063**

Parallel	0,616	**6,033**

3.2. Multi-threaded support

Recommendation 2: *The next public release of the LarKC platform should include multi-threaded support functions for plug-ins such that common thread management and synchronisation behaviour is not implemented multiple-times within each plug-in.*

As demonstrated in the previous version of this deliverable D6.6 [8] and in D6.7 [9] respectively, workflow branching (i.e., parallel execution of multiple plug-in instances) greatly improved the execution time of particular plug-ins and thus the overall workflow performance.

Recapitulating workflow branching, the main principle of this concept consists in applying multi-threading techniques to different parts of the LarKC Platform code, in particular to the Plug-in Managers, to process "splittable" inputs concurrently and combine the individual results afterwards (i.e., data sets must be independently processable from each other).

In order to make this functionality available to any plug-in, which can operate on separate data junks simultaneously, the latest LarKC Platform architecture has been extended to support such a feature out of the box without requiring any specific knowledge of implementing multi-threaded code, as well as the necessity to re-implement such a feature every time.

Therefore, to implicitly use the platform's capability for running multiple plug-in instances concurrently, two minimal functional requirements have to be considered. First of all, plug-in developers should carefully describe the functionalities of their plug-ins (i.e. provide the correct annotation of the plug-in's input and output behaviour), and secondly, workflow designers have to specify inside the workflow description that a particular plug-in's input "is splittable" and hence able to process the data junks in parallel. More details of the design and implementation can be found in [4].

3.3. Parallel and distributed execution

Recommendation 3: *The basic plug-in and workflow communication patterns should be identified together with an appropriate corresponding parallelisation strategy that ensures the best performance and scalability for each of the identified patterns.*

Plug-in parallelisation (supported by the distributed execution techniques on the workflow level) is key enabler of the large-scale reasoning applications. There are several parallelisation strategies and technologies considered in LarKC, based on the many year experience of the project partners: multi-threading, MPI, MapReduce, CUDA.

Multi-threading is a thread-based parallelisation approach, whereby parallel processing is achieved by means of forking and joining two or more concurrently running tasks, e.g. each processing a separate *Statement* from the given *SetOfStatements*. In case of multi-threading, the RAM space is shared among every spawned thread, which makes the parallel implementation of the code extremely easy for the plug-in developer. However, all the threads only exist in the context of a single process. This basically means that the scalability of this approach (in terms of the maximal number of parallel threads) is limited to the boundaries of the shared-memory compute architectures (normally, not more as 32 threads can be efficiently supported). However the following approaches (MPI and MapReduce) enable overcoming this limitation as support the processes-based parallelisation, i.e. execution on distributed parallel systems such as a cluster of workstation or a HPC.

MPI

MPI stands for Message Passing Interface. This is a process-based technique, whereby processes communicate by means of messages transmitted between (a so-called "point-to-point" communication) or among the nodes (involving several or even all processes, a so called "collective" communication). Normally, one process is executed on a single computing node. If any of the
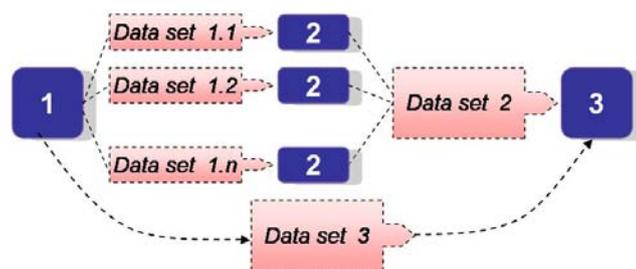
processes needs to send/receive data to/from other processes, it should call a corresponding MPI communication function. Both point-to-point and collective communications available for MPI processes are documented in the MPI standard[11].

MapReduce is also a process-based parallelisation technique, complementary to MPI. In contrast to MPI, MapReduce has a strong focus on the data-based parallelisation, introducing a two-step parallel processing schema[12]. The first step, i.e. *Map*, is done by the master process, which takes the input data set, partitions it up into smaller sub-problems, and distributes those to worker processes. On the second step, i.e. *Reduce*, the output from each process is collected by the master and combined in some way to answer the original problem.
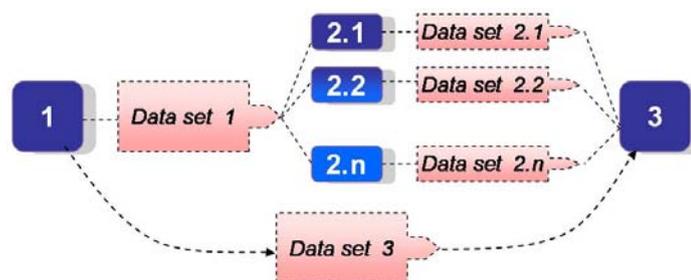
A good alternative to both MPI and MapReduce techniques is CUDA (stands for Compute Unified Device Architecture). CUDA enables NVIDIA GPUs not only for the traditional graphic rendering, but also for the highly-scalable and vector-based data processing.

In the current recommendation, we were requested to analyse patterns of the LarKC applications based on the code-data dependencies and specify the parallelisation strategies to be the most efficiently used for each of those patterns. This was to a large extent done in deliverable D5.3.2 [3]. Summarising the outcomes of the work done in D5.3.2, the following three main patterns were identified (Figure 4):

- Single Code Multiple Data (SCMD) - the data can be partitioned up into unrelated subparts (Figure 4a); the same computation operation is performed on those subparts.

- Multiple Code Single Data (MCSD without conveyer dependencies) – the data is not partitionable and several different operations are performed on this data (Figure 4b).

- Multiple Code Multiple Data (MCMD) – the data is partitionable and different operations are performed on every subset (Figure 4c).
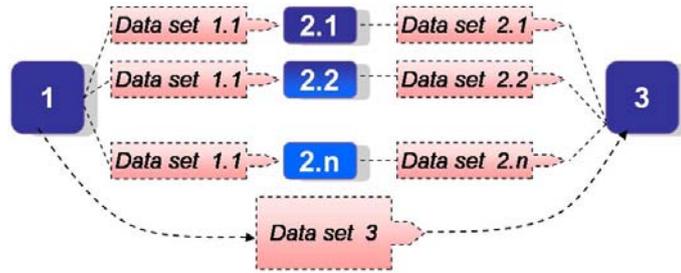

a) SCMD configuration

b) MCSD configuration

---

[11] http://www.mcs.anl.gov/research/projects/mpi/mpi-standard/mpi-report-1.1/mpi-report.htm

[12] http://en.wikipedia.org/wiki/MapReduce

c) MCMD configuration

Figure 4. Typical code-data dependencies in LarKC plug-ins.

It is recommended to apply multithreading for each of the patterns, as most of the current compute architectures support this technology very efficiently. Moreover, if the plug-in is executed on the dedicated high performance system, e.g. a cluster of CPU/GPU compute nodes, the following recommendations can be done for each of the patterns:

- SCMD – MPI, MapReduce
- MCSD – MPI, CUDA
- MCMD – MPI, MapReduce, to less extent CUDA

Recommendation 4: *The next public release of the LarKC platform should include support functionality for the distribution and parallel execution of plug-ins and verified by their use in more than one working pipelines that demonstrate a measurable performance increase over current alternative approaches.*

Although, the new architecture design has been done quite recently, there is one SCMD workflow (Semantic Space Search by means of Random Indexing[13]) implemented with MPI, and two MCMD workflows (WebPIE and Semantic Space Generation by means of Random Indexing) implemented with MapReduce.

The performance of the RandomIndexingTransformer plug-in is evaluated in D2.5.2 [10]. The parallelisation approach is based on non-overlapping data domain decomposition, implemented with MPI. Performance was evaluated for the four semantic space configurations defined in [10] and described in Table 9. The use cases were executed on the Nehalem14 cluster of HLRS, which compute nodes are equipped with Intel Xeon 2.8 GHz processors, interconnected with Infiniband. Configuration of 1, 2, 4, 8, and 16 compute nodes, each equipped with 12 GB of shared RAM space, were benchmarked. The MPI environment was served by mpiJava installed on top of Open MPI 1.4.1.

Table 9. **Characteristics of the benchmarked semantic spaces.**

Semantic Space	Nr. of vectors	Disk size, GB	Description
LLD1	0,064 M	0,082	Subset of LLD
LLD2	0,5 M	0,65	Subset of LLD
Wiki1	1 M (low density, terms only)	1,6	Term set from 1M most central Wikipedia articles
Wiki2	1 M (high density, entire documents)	16	Document set from 1M most central Wikipedia articles

The quantitative evaluation was focused on the total execution time, the time for loading the vector space from the file on the disk, the duration of the search operation, and the overhead of the inter-node MPI communication. The obtained benchmarking results are collected in Table 10.

---

[13] http://wiki.larkc.eu/LarkcProject/statisticalSemantics/parallelisation

[14] http://www.hlrs.de/systems/platforms/nec-nehalem-cluster/

Table 10. Evaluation results on the Nehalem cluster.

Semantic Space	Nr. of compute nodes (processes)	Time, s.				Speed-up
		Loading	Search	MPI comm.	Total	
LLD1	1	-[15]	-	-	2	1
LLD1	2	0,75	0,5	0,04	1,61	1,25
LLD1	4	0,4	0,4	0,05	1,2	1,7
LLD1	8	0,23	0,32	0,1	1,01	1,98
LLD1	16	0,17	0,29	0,16	0,94	2,13
LLD2	1	12	6	-	19,5	1
LLD2	2	4	3,3	0,03	7,9	2,47
LLD2	4	2,4	1,8	0,23	4,6	4,24
LLD2	8	1,2	1	0,16	2,9	6,72
LLD2	16	0,6	0,7	0,2	2	9,75
Wiki1	1	18	4	-	22	1
Wiki1	2	8,9	3,8	1	13,3	1,65
Wiki1	4	4,6	2	0,08	7,4	2,97
Wiki1	8	2,3	1,3	0,23	4,4	5
Wiki1	16	1,2	0,75	0,52	2,8	7,86
Wiki2	1	309	83	-	395	1
Wiki2	2	59	27	0,58	88	4,5
Wiki2	4	35	13	16	59,1	6,7
Wiki2	8	20	8	4	32,2	12,3
Wiki2	16	10	3,7	0,16	14,6	27

The evaluation results show that the MPI-parallelised version of RandomIndexing Transformer scales well on the parallel architecture for the use cases of any complexity, varying from the sparse term vectors (LLD1) to large data sets containing a million of documents (Wiki2), despite the increasing communication overhead. In the best case, the performance speed-up achieved by the parallelised algorithm was approximately 27 times.

The pure MPI approach described above, whereby a single MPI process runs on a compute node, does not allow for a shared-memory architecture of the parallel compute node. However, most of the modern parallel and cluster systems are built upon the hybrid architecture, i.e. when a compute node offers multi-core CPUs. Therefore, the performance of the "pure MPI" communication pattern was not optimal due to the only one core of a multi-core CPU involved in the computation (see Figure 5).

---
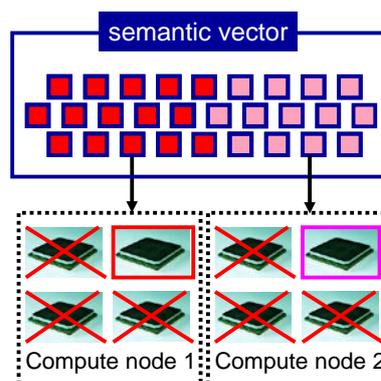
[15] This configuration was not tested due to the RAM limitation on the single node

Figure 5. Pure MPI execution on a multi-core compute node.

A straightforward approach to overcome this limitation with the previously described implementation is to start several MPI processes per compute node. Performance characteristics, when overloading the compute node with several MPI processes, are collected in Table 11. The advantage of this is that it does not require any changes in the previously described algorithm.

Table 11. Performance with multiple MPI processes on two compute nodes (for the Wiki1 vector space)

Number of MPI processes per node	Time, s.	
	Search	*MPI comm.*
1	3,8	0,42
2	2	0,07
4	1,1	0,2
8	1	0,6

The evaluation reveals that running several MPI processes on a multi-core compute node improves the performance by approx. 3,8 times for the tested use case. However, the communication between the MPI processes is still performed via the network interconnect, which is a bottleneck for the processes running in the same shared-memory space. Moreover, when running several MPI instances and hence multiple Java Virtual Machines per compute node, the memory requirements grow accordingly, which does not allow this method to be applied for the vector spaces of large dimensionality.

These limitations can be avoided by applying multi-threading techniques inside an MPI process, ensuring the optimal resource utilisation and performance. This approach is considered a hybrid communication pattern. According to this technique, only one MPI process is running per compute node, spawning multiple threads each of them running on a separate core (Figure 6). While the threads communicate via the shared memory, whenever access to the data of another thread is needed, the communication between the nodes is performed by means of the MPI. Thread spawning can be efficiently performed by means of a thread pool, the elements of which are mapped to the CPU cores.
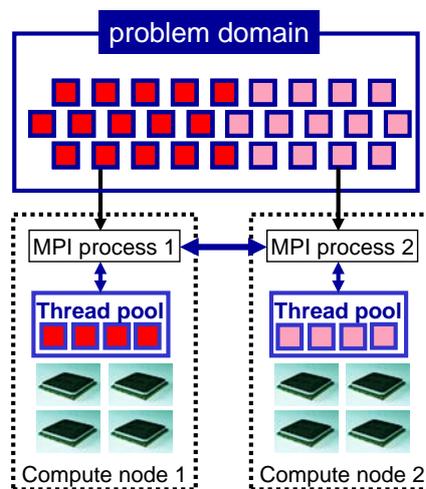

Figure 6. Hybrid MPI+JavaThreads communication pattern.

The performance characteristics obtained with the MPI-JavaThreads technique are collected in Table 12.

Table 12. Performance with hybrid MPI-JavaThreads communication pattern on two compute nodes (for the Wiki1 vector space)

Number of Java Threads per node	Time, s.	
	Search	*MPI comm.*
1	3,8	0,03
2	1,8	0,03
4	1	0,03
8	0,73	0,03

Tests with the Wiki1 use case reveal that the hybrid realisation allows the tested Java application to achieve better performance as in case of the pure MPI one. Applying this pattern for the most complex data set we had (Wiki2), the maximum speed-up of 33.5 was achieved (see Figure 7), which is around 24% more efficient as the MPI-only realisation.
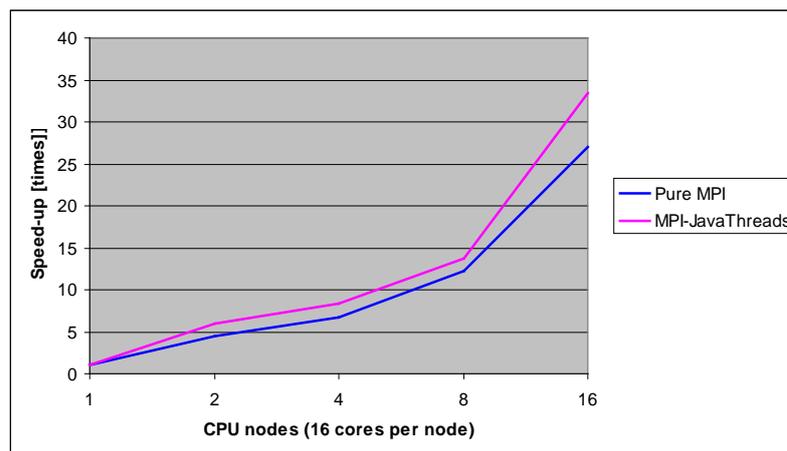

Figure 7. Comparison of pure MPI and MPI-JavaThreads communication pattern performance (for the Wiki2 vector space

MapReduce

WebPIE (Web-scale Parallel Inference Engine) is a MapReduce distributed RDFS/OWL inference engine built on top of Hadoop framework. Detailed evaluation experiments for WebPIE are presented in this paragraph. The figures below summarise our findings concerning the performance of our system for a varying number of nodes. Figure 9 shows the runtime for calculating the closure of LUBM together with the (theoretical) linear speed-up and Figure 10 shows the scaled speed-up. The latter is defined as speed-up/# Nodes, and is an indicator for the efficiency of our system, given additional computational nodes. A scaled speed-up of 1 means that given twice the number of nodes the system will perform twice as fast. We note that, for 16 nodes or less, our system has superliner speed-up, meaning that for double the number of nodes, our system performs better than twice as fast. This is attributed to the fact that, with a larger number of nodes, we can store a larger part of the input in memory, which dramatically increases performance.
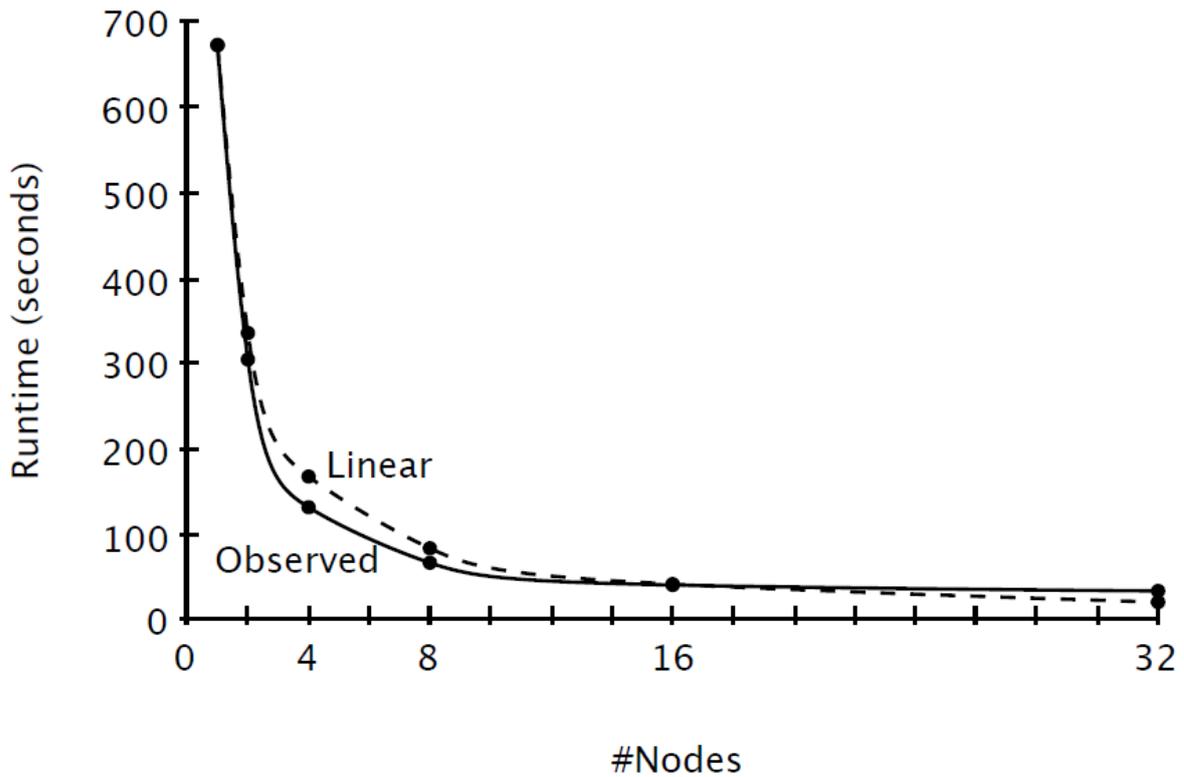
Figure 8. WebPIE scalability and the number of nodes

In Figure 9 we also observe that, for 32 nodes, the scaled speed-up decreases drastically. This is because of the time required by the platform to start a job. Regardless of the number of nodes, calculating the closure of LUBM will always take more than approximately 25 minutes (a limitation imposed by the Hadoop platform).

Figure 10 shows the runtime for input size up to 100 billion triples on 64 nodes. Here, we see a similar situation as before: for small data sizes our throughput is reduced, since the runtime is dominated by platform overhead. Starting from 5 billion triples, the throughput improves dramatically, as the platform overhead is amortised over the longer runtime. For input larger than 10 billion triples, we benefit less from keeping large portions of the data in memory, and the throughput slightly decreases.
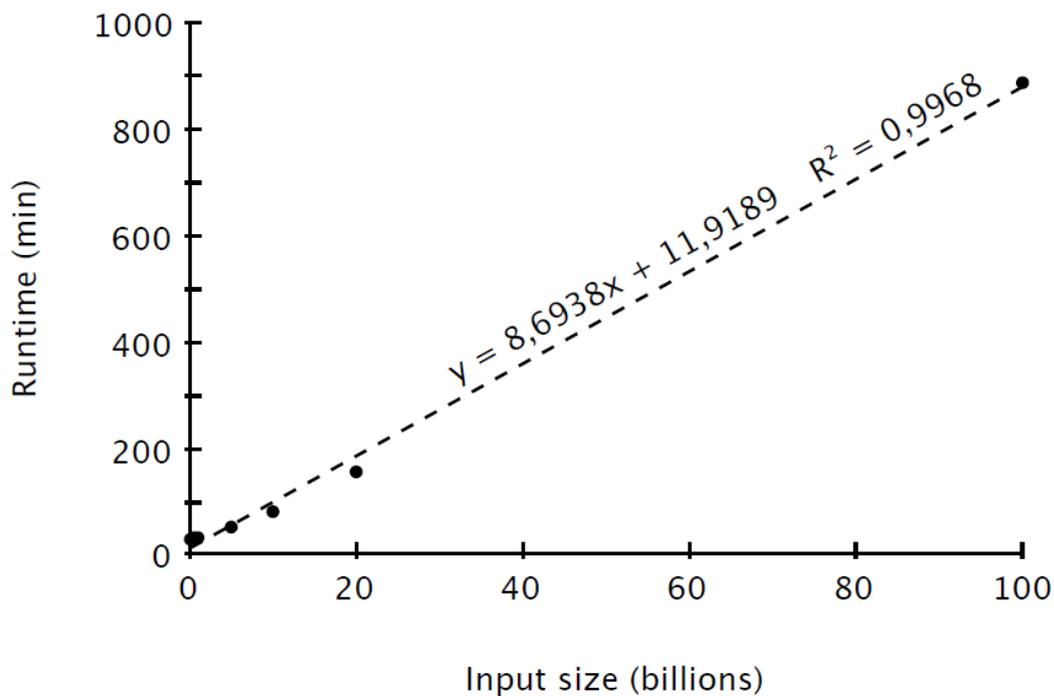
Figure 9. WebPIE scale-up versus size

Recommendation 5: *The next public release of the LarKC platform should include an initialisation step that can materialise all implicit statements from any given standard rule-set by utilising a cluster of machines. The effectiveness of this technique will be verified by comparison with the initialisation time of a single machine, where a considerable improvement is expected.*

The distributed materialisation of implicit statements is key functionality supported by the LarKC platform that enables the processing of practically unlimited scale of RDF data compared with the scale of the existing RDF databases. Therefore, in order to support more usage scenario the WebPIE work is integrated as a platform workflow that enables the further selection, filtering or processing of the inferred statements before to actually load them in the Data Layer. The WebPIE workflow in Figure 8 how parallelisation can be exploited to increase performance of any LarKC based application. In this workflow, we want to answer queries, under inference, over data in a (large) file. To this end, we are using a series of plug-in to perform materialisation, and we pre-process the query to extract only the relevant statement patterns from this file. We focus on two elements:

a) We are using WebPIE to accelerate reasoning by using parallel hardware (as shown by the plug-in with the relevant annotations). In this manner, we are offloading computation from the data layer to a map-reduce based parallel implementation.

b) Parallel selection. The Filtering plug-in, also implemented in MapReduce, allows very fast selection of the statements that might contribute to an answer, before they are loaded into the data layer
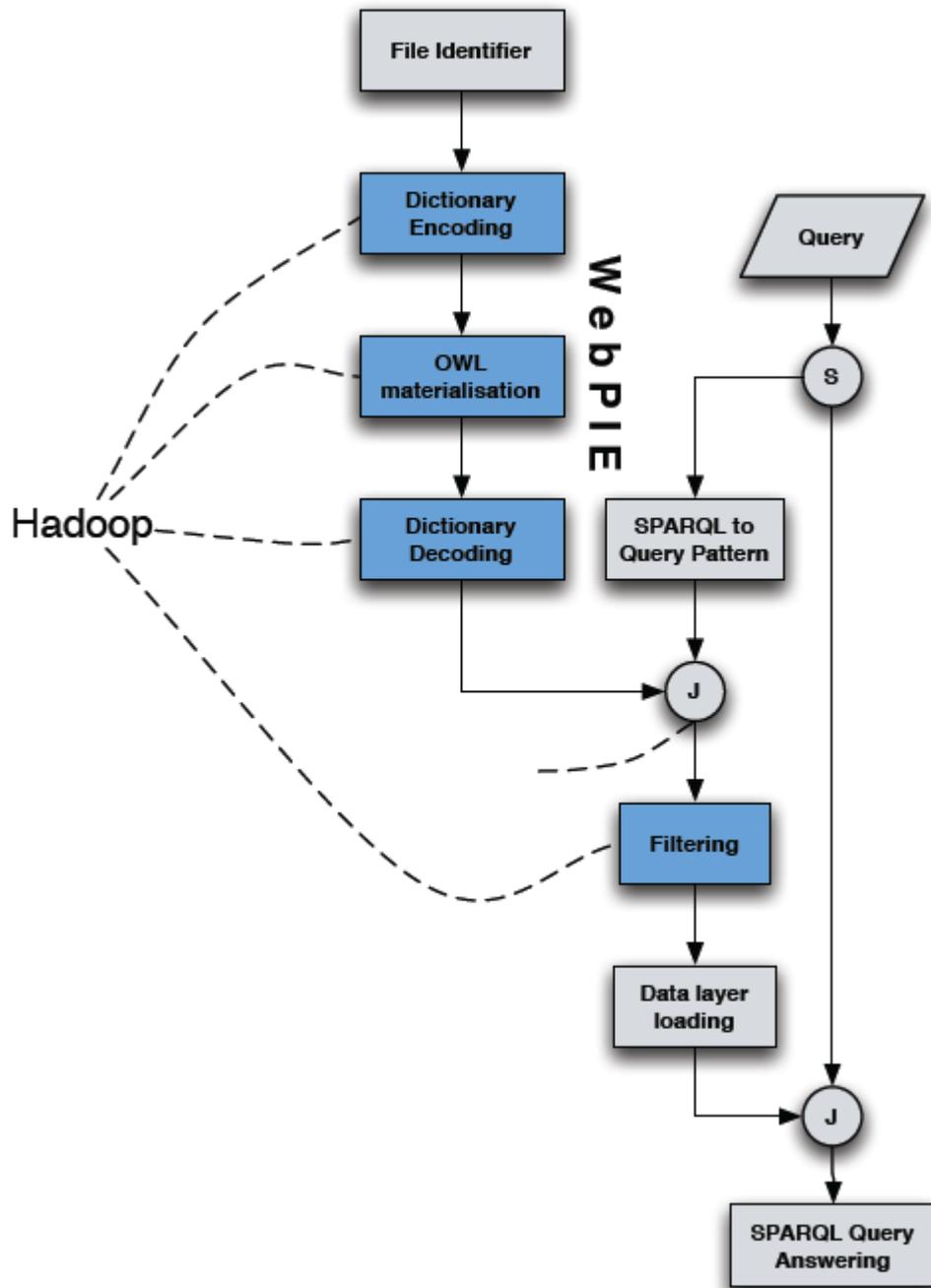
Figure 10. The WebPIE integration into a LarKC workflow.

4. Graphical front-ends for workflow construction

Since introducing the Management Interface of the LarKC platform, it became possible to construct convenient end-user graphical interfaces, which facilitate the creation, submission and execution of user workflows. One of those tools, the LarKC Workflow Designer, is about to be integrated in the oncoming release of the platform. The tool allows the end-users (workflow designers) to abstract from the details how the LarKC's Management Interface works and to concentrate on the workflow creation and execution. An example of the Workflow Designer interface is shown in Figure 11.
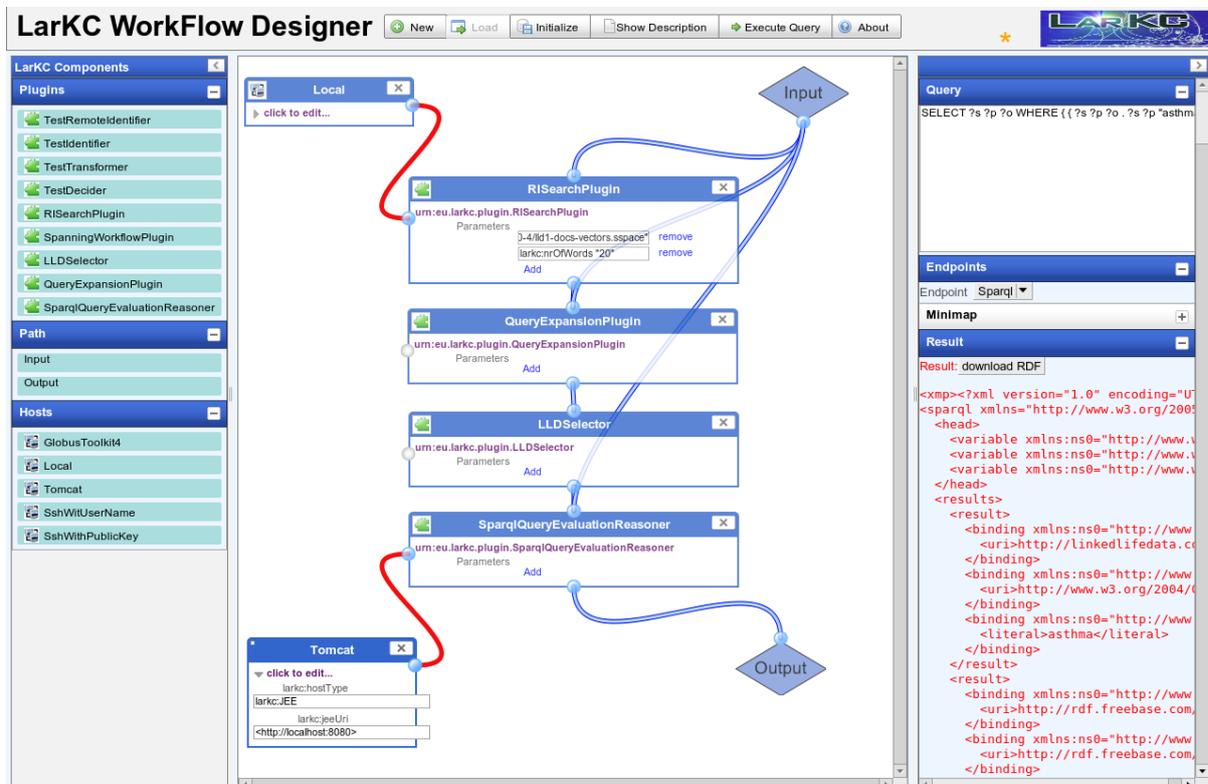

Figure 11. Workflow Designer GUI, view from a web browser window

The distinctive features of Workflow Designer are the following:

- Developed as a web application, deployed in Tomcat or Eclipse; the user only needs a web browser to access the application.
- Connects to the LarKC platform through the Management Interface.
- Retrieves a list of plug-ins as well as resource templates from the platform when starting the GUI.
- Powerful graphical interface that enables easy workflow construction by drag-and-drop of plug-ins and adding/editing connections between them.
- Specification of the plug-in parameters (if required).
- Easy specification of the plug-in deployment host, by drag-and-drop of the available host templates.
- One-click workflow submission to the platform.
- Once-click query execution by the workflow.
- Immediate visibility of results as well as possibilities to store the results (data) locally.

5. Recommendation for the final release

This chapter is a logic continuation of section 8 of D5.5.3 [6], which describes important recommendation for the 2.0 platform release, and presents new emerging ones that have to be taken into consideration and gives final conclusions for the high-level direction for the final LarKC release and the steps beyond the project end.

Recommendation 6: Support of parallel applications when executing on HPC

As the LarKC applications have become matured enough, one of the main weak points of the previous realisation turns to be a lack of the supporting mechanisms for executing parallelized applications (both with MPI and Hadoop). On one hand, when accessing the high performance resources, such operations as advanced resource reservation, job creation, and monitoring are of big importance.

On the other hand, the complexity of performing those tasks should be kept hidden from the end-users as much as possible. Such support should be provided for all of the currently accepted access mechanisms (SSH, GT4, etc.). Our suggestion is to implement a dedicated plug-in manager that will take care of all issues arising around parallelised applications.

Recommendation 7: Support for Pluggable endpoints

While developing and integrating specific LarKC plug-ins and LarKC applications in many cases it became obvious that the SPARQL endpoint is not the best option to interact with the system. This already happened in the first LarKC version with anytime behaviour interaction, which was the reason that we introduced the option to implement additional endpoints inside LarKC. Often it appears that the plug-ins which are solving some specific problem, would need the specific endpoint. The problem is that the LarKC platform currently can only load plug-ins and thus, the endpoints must be part of the main platform. A Solution for this would be that the endpoints would become a special type of LarKC plug-ins, which would allow the developers to make their own way to interact with the system. For example, currently we have SPARQL, push, anytime and one or two project specific endpoints, which all live inside of the platform, where only SPARQL and Anytime endpoints were the part of originally designed system.

Recommendation 8: Develop better community support materials, advanced documentation and extend the number of support plug-in in the marketplace

The LarKC platform attracted the attention of many users representing a wide mix of different communities spread across the world. The platform release 2.0 was downloaded more than 600 times in the first four months of 2011[16]. Based on the user feedback so far we identified that the most desired platform feature is better platform documentation that will increase 1) the number of the users who are using it in a of the shelf manner; 2) the existing workflows and use case scenarios where LarKC can be applied; 3) the number of contributed plug-ins by an external to the consortium partners. The LarKC plug-in market-place is an excellent starting point, which needs an improvement with respect to the supported plug-in by the latest platform release and the existing documentation that will ensure sustainability after the project end.

---

[16] http://sourceforge.net/projects/larkc/files/Release-2.0/stats/timeline?dates=2011-01-01+to+2011-04-29

6. References

[1] Peikov et al, D2.4.2 Approximate Activation Spreading, March 2010

[2] Gallizo et al., D5.3.2 Overall LarKC architecture and design v1, September 2009

[3] Assel et al., D5.3.3 Final LarKC architecture and design, March 2011

[4] Assel et al, D5.4.2 Second Release of the LarKC Platform, December 2010

[5] Kiryakov et al., D5.5.2 Validation goals and metrics for the LarKC platform, August 2009

[6] Momtchev et al., D5.5.3 Report on platform validation and recommendation for next version, July 2010

[7] Dell'Aglio et al., D6.6 2nd periodic report on data and performances, September 2009

[8] Celino et al., D6.7 3rd periodic report on data and performances, July 2010

[9] Lars Arge, Mark de Berg, Herman J. Haverkort, Ke Yi: The Priority R-Tree: A Practically Efficient and Worst-Case Optimal RTree, SIGMOD 2004, Research Session 8: Indexing and Tuning, June 13–18, Paris, France

[10] Damljanovic et al., D2.5.2 Geometrical semantics components v2 and D2.2.2 Baseline components v2, March 2011