LarKC

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

FP7 – 215535

D5.1 Summary of parallelisation and control approaches and their exemplary application for selected algorithms or applications

Coordinator: Georgina Gallizo (HLRS)
With contributions from: Sabine Roller, Axel Tenschert, HLRS; Michael Witbrock, CycEur; Barry Bishop, Uwe Keller, UIBK; Frank van Harmelen, Gaston Tagni, Eyal Oren,VUA

Document Identifier:	LarKC/2008/D5.1 /v1.1
Class Deliverable:	LarKC EU-IST-2008-215535
Version:	1.1
Date:	01.09.2009
State:	Revision of V1.0, submitted 29.09.2008
Distribution:	Public

EXECUTIVE SUMMARY

This deliverable lists the different techniques to be considered when designing the LarKC Collider Platform. It intends to be a guidance to help the architects and developers to choose the most appropriate solution. The platform will most probably be designed and developed as a combination of different parallelization and resources distribution techniques that will make possible the deployment of new reasoning techniques to be done at vastly reduced costs.

We will distinguish two different technology areas: Parallel computing and Distributed computing. For each of them, different models can be considered, some of them complementary, and each of them with their different advantages and drawbacks. The optimal solution can only be chosen as a result of the analysis of the concrete situation, context and requirements, which will be part of the architecture and prototype design tasks (Task 5.2 Early Release Prototype, Task 5.3 Platform Architecture and Design), in close cooperation with the plug-in development WPs (WP2, WP3 and WP4) and considering the input from the Use cases WPs (WP6 and WP7).

The application of parallelization and distribution techniques has implications for the developers of the LarKC components, which must be considered right at the beginning of the design phase.

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**	D5.1	**Title**	Summary of parallelisation and control approaches and their exemplary application for selected algorithms or applications
Work Package	**Number**	WP5	**Title**	The Collider Platform

Date of Delivery	**Contractual**	30.09.2008	**Actual**	V1.0: 29.09.2008 V1.1: 01.09.2009
Status	version 1.1		final x	
Nature	prototype □ report x dissemination □			
Dissemination level	public x consortium □			

Authors (Partner)	Georgina Gallizo, Sabine Roller, Axel Tenschert (HLRS); Michael Witbrock (CycEur); Barry Bishop, Uwe Keller (UIBK); Frank van Harmelen, Gaston Tagni, Eyal Oren (VUA)			
Responsible Author	**Name**	Georgina Gallizo	**E-mail**	Gallizo@hlrs.de
	Partner	HLRS	**Phone**	

Abstract (for dissemination)	This deliverable lists the different techniques to be considered when designing the LarKC Collider Platform. It intends to be a guidance to help the architects and developers to choose the most appropriate solution. The platform will most probably be designed and developed as a combination of different parallelization and resources distribution techniques that will make possible the deployment of new reasoning techniques to be done at vastly reduced costs. We will distinguish two different technology areas: Parallel computing and Distributed computing. For each of them, different models can be considered, some of them complementary, and each of them with their different advantages and drawbacks. The optimal solution can only be chosen as a result of the analysis of the concrete situation, context and requirements, which will be part of the architecture and prototype design tasks (Task 5.2 Early Release Prototype, Task 5.3 Platform Architecture and Design), in close cooperation with the plug-in development WPs (WP2, WP3 and WP4) and considering the input from the Use cases WPs (WP6 and WP7).

	The application of parallelization and distribution techniques has implications for the developers of the LarKC components, which must be considered right at the beginning of the design phase.
Keywords	Parallelization, Distribution, P2P, Thinking@home, Cloud computing, SOA

Version Log			
Issue Date	**Rev. No.**	**Author**	**Change**
23.04.2008	0.1	Georgina Gallizo	Skeleton draft
27.05.2008	0.2	Georgina Gallizo	First draft document
29.08.2008	0.3	Georgina Gallizo	Input from WP5 partners integrated: general comments, parallelism in Java
08.09.2008	0.4	Georgina Gallizo	Comments from VUA and UIBK
15.09.2008	0.5	Georgina Gallizo	General review, improvement of some sections with further information
17.09.2008	0.6	Georgina Gallizo	Version ready for internal quality assessment
24.09.2008	0.7	Georgina Gallizo	Version ready for Scientific coordinator approval
29.09.2008	1.0	Georgina Gallizo	Final version ready for submission
01.09.2009	1.1	Georgina Gallizo	Addressed comments from 1st LarKC Review Report: updated References section with appropriately scientific citations. Formatting general sanity check.

PROJECT CONSORTIUM INFORMATION

Participant's name	Partner	Contact
Semantic Technology Institute Innsbruck, Universitaet Innsbruck		Prof. Dr. 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		Georgina Gallizo, Höchstleistungsrechenzentrum, Universitaet Stuttgart, Stuttgart, Germany, Email: gallizo@hlrs.de
MAX-PLANCK GESELLSCHAFT ZUR FOERDERUNG DER WISSENSCHAFTEN E.V.		Dr. Lael Schooler Max-Planck-Institut für Bildungsforschung Berlin, Germany Email: schooler@mpib-berlin.mpg.de
Ontotext Lab, Sirma Group Corp		Atanas Kiryakov, Ontotext Lab, Sofia, Bulgaria Email: atanas.kiryakov@sirma.bg
SALTLUX INC.		Kono Kim, SALTLUX INC, Seoul, Korea, Email: kono@saltlux.com
SIEMENS AKTIENGESELLSCHAFT		Dr. Volker Tresp, SIEMENS AKTIENGESELLSCHAFT, Muenchen, Germany, E-mail: volker.tresp@siemens.com
THE UNIVERSITY OF SHEFFIELD		Prof. Dr. Hamish Cunningham, THE UNIVERSITY OF SHEFFIELD Sheffield, UK, Email: h.cunningham@dcs.shef.ac.uk

VRIJE UNIVERSITEIT AMSTERDAM		Prof. Dr. Frank van Harmelen, VRIJE UNIVERSITEIT AMSTERDAM, Amsterdam, Netherlands, Email: Frank.van.Harmelen@cs.vu.nl
THE INTERNATIONAL WIC INSTITUTE, BEIJING UNIVERSITY OF TECHNOLOGY		Prof. Dr. Ning Zhong, THE INTERNATIONAL WIC INSTITUTE, Mabeshi, Japan, Email: zhong@maebashi-it.ac.jp
INTERNATIONAL AGENCY FOR RESEARCH ON CANCER		Dr. Paul Brennan, INTERNATIONAL AGENCY FOR RESEARCH ON CANCER, Lyon, France, Email: brennan@iarc.fr

TABLE OF CONTENTS

List of Figures

Acronyms

Acronym	Definition
API	Application Programming Interface
BOINC	Berkeley Open Infrastructure for Network Computing
DMP	Distributed Memory Parallel
HPC	High Performance Computing
HPF	High Performance Fortran
IPL	Ibis Portability Layer
LarKC	Large Knowledge Collider
MIMD	Multiple Instruction stream, Multiple Data stream
MPI	Message Passing Interface
MTA	Multi-Threaded Architecture
NUMA	Non-Uniform Memory Access
P2P	Peer to Peer
PVM	Parallel Virtual Machine
RDMA	Remote Direct Memory Access
RMI	Remot Method Invocation
RPC	Remote Procedure Call
SETI	Search for ExtraTerrestrial Intelligence
SIMD	Single Instruction, Multiple Data
SMP	Symmetric multi-processing
SOAP	Simple Object Access Protocoll
SPMD	Single Process, Multiple Data
UMA	Uniform Memory Access
UDDI	Universal Description, Discovery and Integration
WP	Work Package
WSDL	Web Service Description Language
WSRF	Web Service Resource Framework

1. Introduction

This report lists the different techniques to be considered when designing the LarKC Collider Platform. It intends to be a guidance to help the architects and developers to choose the most appropriate solution. The platform will most probably be designed and developed as a combination of different parallelization and resources distribution techniques that will make possible the deployment of new reasoning techniques to be done at vastly reduced costs.

We will distinguish two different technology areas: Parallel computing and Distributed computing.

Chapter 2 gives an overview of these technologies. Chapters 3 and 4 go more into details, considering different possible models to be applied, advantages and drawbacks of each of them and the implications of choosing this concrete technology for the developing of the LarKC modules.

Chapter 5 is a conclusion section that gives some hints on how the different techniques and technologies can be combined in order to get the optimal solution for LarKC.

2. Overview of technologies to be considered

This chapter aims to give a general overview of the parallelization and control technologies to be considered within LarKC. A more detailed analysis is done in the following chapters.

Two main technology areas are distinguished for the analysis of parallelization and control approaches within LarKC:

- **Parallel computing**: within 1 platform, at least within 1 site, and 1 code or 1 coupled application like multi-physics code, but tightly coupled. Communication of data between parallel processes takes place not only at begin/end of the code, but also during execution.
- **Distributed computing**: involves several techniques including SOA, simple Client-Server architecture or thinking@home approach. It is workflow like, but a complex workflow including loops, branches, conditions, … Several individual steps might be executed in parallel, and each individual step might be executed in parallel in itself. The parallel execution of an individual step again might either parallel computing (as described above) or distributed computing.

Different terminology aspects must be considered within this document:

- The word “computing” is used even if the step is searching or other.
- The word parallel / parallelisation is in general used only for the first area (Parallel computing). The latter (Distributed computing) might also run tasks at the same time. Then the wording “simultaneous execution” or “concurrent execution” is used.
- Data base can be a data base in the sense of an SQL data base, but also any collection of data, coming from different sources and in different formats.
- Coupling: A strong coupling between tasks running at the same time means “a lot of communication” during execution of the tasks. This usually requires a good network and probably some site/hardware specific communication method, e.g. Message Passing Interface (MPI) to communicate between processors or OpenMP to communicate directly within the memory (see section 3.3 for details). In this case parallel computation should be preferred. Loose coupling means less data to be communicated during execution of the

individual step, while the amount of data might be large at the beginning or end of an individual step. In this case neither architecture is favoured at this point.

- "A lot of communication" means that communication time is noticeably large compared to the duration of the computing (or search or whatever the CPU does in between). A lot of communication does not necessarily mean large amounts of bytes to be communicated. It might also be started often, i.e. in short time intervals. In that case, latencies (i.e. time for start-up of a communication) plays an important role.

2.1. Parallel Computing

In this document, we consider that parallel computing is performed within a cluster. Clusters consist of more or less homogeneous processor and memory hardware which is connected via a high speed interconnection network. The hardware is located at one place. The parallel application uses the whole cluster or a part of it at the same time. Typically, resource allocation is done by a batch system (queuing system) in a dedicated mode. This implies the possibility that resources are not available at once but the users request has to wait until resources are free and assigned to the user by some kind of workload management software. A more detailed description of this technology can be found in chapter 3.

2.2. Distributed Computing

Distributed computing is a computing model that involves multiple computers that may be remote from each other, in order to solve a single problem.
There are many different types of distributed computing approaches and many challenges to overcome in successfully designing the appropriate solution. The main goal of a distributed computing system is to connect users and resources in a transparent, open and scalable way.
Some of the concepts related to Distributed Computing that we will consider within LarKC are briefly described in the following sections. **Note that they are not necessarily in the same conceptual level, and even some of them can be a subset of others.**

Client-Server Architecture

The client-server software architecture consists of a client system and a server system. Both systems are communicating by a computer network. The normal use of such a system is that a client is able to send a request to a server which gives response to the client. A client-server architecture is a distributed system which is practical used for web services.

P2P

Peer to peer (P2P) is a particular case of distributed computing. A P2P architecture is a distributed system in which participants rely on one another for services. Furthermore, peers in the system can elect to provide services as well as consume them. That is, a pure P2P network does not have the notion of clients or servers, but only equal peer nodes that simultaneously function as both "clients" and "servers" to the other nodes on the network. This model of network arrangement differs from the client-server model where communication is usually to and from a central server.

Thinking@home

Thinking@home is the term used to describe a distributed computing approach that consists on splitting up a program into parts that run simultaneously on multiple computers communicating over a network. It must deal with heterogeneous environments, network links of varying latencies, and unpredictable failures in the network or the computers.

Cloud computing

Cloud computing is a subset of distributed computing, where the computing (or the storage) is done somewhere "in the cloud". The architecture behind cloud computing is a massive network of "cloud of servers" interconnected and orchestrated to work together on common

problems. The main idea of Cloud Computing is that applications and data are not stored or processed on a local computer but as a cloud on a distant system, where resources are applied and managed by the cloud as needed.

Service Oriented Architecture (SOA)

SOA is a paradigm for the design and realization of a distributed architecture, where services can be distributed over a network and can be combined and reused to create business applications. Services communicate each other with standard protocols, the interface of each service is published and where a change in a service does not affect dramatically to others. Therefore, it can be said that a SOA leads to a loosely coupled architecture. A SOA can be implemented with Web services technology, in order to make functional building blocks accessible over standard Internet protocols that are independent from platforms and programming languages. These services can be new applications or just wrapped around existing legacy systems to make them SOA-enabled.

3. Parallel Computing

With the given technology a further increase in computational power is only possible by some kind of parallel hardware. Naturally, this brings up the question of how to map the application software to the physical hardware. In what follows we throw light on the current situation of both aspects.

3.1. Concepts

In general, there are several levels of parallelization, on the processor level itself as well as between processors:

The lowest level of parallelization is enabled within a processor by special instructions. They realize pipelining or vectorization of the data processing (SIMD parallelization, Single Instruction, Multiple Data). An example is the adding of two vectors of 100 numbers each. From the applications point of view only a sufficient data independency has to be realized. The rest is done by the compiler.

The next level is given by the availability of several functional units (i.e. floating point units) which can operate in parallel (MIMD parallelization, Multiple Instruction stream, Multiple Data stream).

Given these individual processors, larger units of parallel architectures can be constructed by connecting a larger number of processors with an appropriate interconnect to a larger machine.

Determined by the way the memory can be accessed, we distinguish between shared memory (intra-node parallelization) and distributed memory (inter-node parallelization). In a shared memory system, the processors are all connected to a "globally available" memory, via either a software or hardware means. The operating system usually maintains its memory coherence.

In a distributed memory system, each processor has its own individual memory location. Each processor has no direct knowledge about other processor's memory. For data to be shared, it must be passed explicitly from one processor to another as a message.

3.2. Major parallel hardware architectures

Several multi-purpose parallel hardware architectures can be distinguished, depending on the way they manage the access to the memory:

Shared Memory: Symmetric multiprocessing (SMP)

In a shared memory system, two or more identical processors are connected to a single shared main memory. In this case, all CPUs are connected to all memory banks with the same speed. It has, therefore, Uniform Memory Access (UMA). This kind of architecture is called a symmetric multi-processing (SMP) architecture.

With the upcoming multi-core processors this concept is already realized inside the processor itself, where the memory is accessed via the memory bus. Via a crossbar it is possible to realize independent memory access for all processors inside a so called SMP node.

SMP systems allow any processor to work on any task no matter where the data for that task are located in memory; with proper operating system support, SMP systems can easily move tasks between processors to balance the workload efficiently.

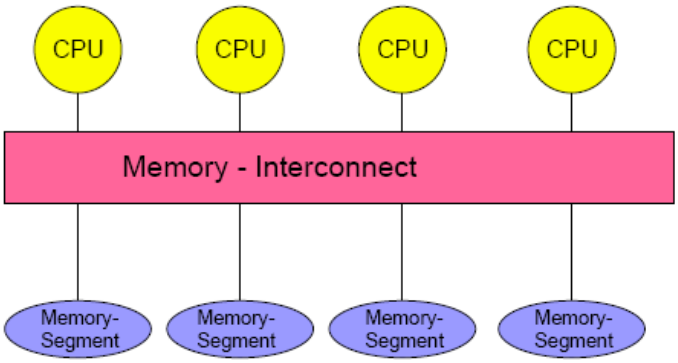

Figure 1: Symmetric multi-processing (SMP) architecture

Distributed memory: Distributed memory parallelization (DMP)

In the distributed memory architecture the processor has only direct access to its own memory and access to the memory of all other processors via an inter-node network. This is called a multi-computer with distributed memory.

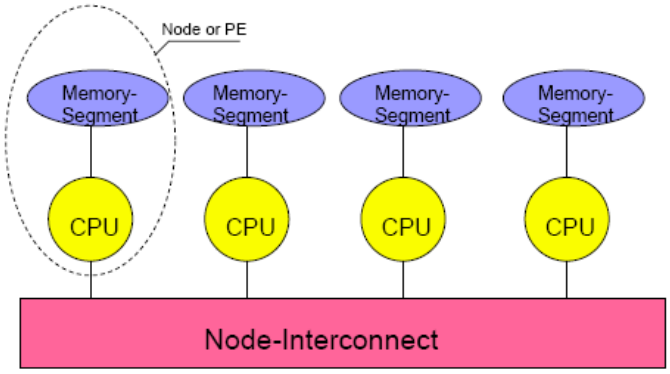

Figure 2: Distributed memory parallel architecture (DMP)

This implies that the access to the others CPUs memory is significantly slower than the access to its own one. It has, therefore, Non-Uniform Memory Access (NUMA). The cost and performance of the network depends on the number of available links between the processors. For example, for a 3-D torus node-interconnect, each processor is connected to each of its neighbours in a Cartesian topology (6 links). Often one can find a hierarchy of switches organised in a tree where the bandwidth is increasing on the higher tree levels.

Hierarchical memory systems: hybrid architecture

A hierarchical memory system is a combination of the basic architectures described above. Actually, nowadays most high performance computing (HPC) systems are clusters of SMP nodes. Most common multiprocessor systems today use a SMP architecture. In case of multi-core processors, the SMP architecture applies to the cores, treating them as separate processors:

- SMP is applied inside each node
- DMP is applied on the node interconnect

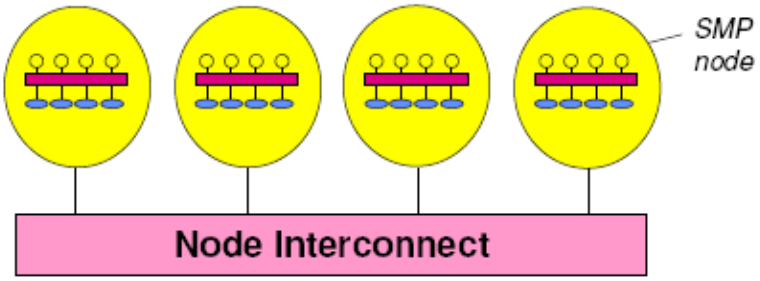

Figure 3: Hybrid architecture

Other architectures

Other kind of more complex architectures can be the following (without going into details):

- ccNUMA (cache coherent non-uniform memory access): a distributed (hybrid) architecture. It looks like one big SMP and is programmable like one big SMP. But in reality is a cluster of several small SMPs. From the programming viewpoint it allows global access with same load/store instruction as local and the parallelization can be done, e.g. with OpenMP
- ccNUMA with >500 CPUs and multi-level network: in this case, the parallelization can be performed, e.g., with Multi Level Parallelism (MLP)
- DMP with RDMA (remote direct memory access): in this case the programming is done considering global memory access with special instructions, e.g. in Co-array Fortran or in UPC (Universal Parallel C)
- MTA (multi-threaded architecture)

3.3. Parallel programming models

Keeping the given hardware in mind, the question arises how to utilize the available parallel hardware features in an application software.

The following aspects must be considered:

- Two major resources of computation are available:
  - Processor
  - Memory
- Parallelization means:
  - distributing work to processors
  - distributing data (if memory is distributed)

  and
  - synchronization of the distributed work
  - communication of remote data to local processor (if memory is distributed)

The different programming models offer a combined method for distributing the work and data to the processors and the memory, and for synchronization between the processes and for communication of data between the processes.

Distributing work must be distinguished from distributing data:

- work decomposition is based on loop decomposition (see figure below)

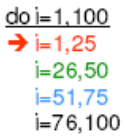

Figure 4: Work decomposition example

- In the case of data decomposition all work for a local portion of the data is done by the local processor.

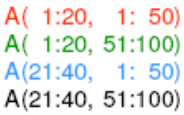

Figure 5: Data decomposition example

- A third level of decomposition is the domain decomposition: in this case decomposition of work and data is done in a higher model, e.g. in the reality.

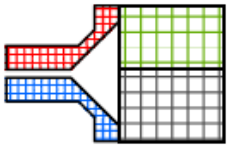

Figure 6: Domain decomposition example

As stated above, synchronization and communication are also necessary when talking about parallelization.

- Synchronization may cause idle time on some processors and overhead to execute the synchronization primitive.
- Communication is necessary on the boundaries, e.g. between domains.

High Performance Fortran (HPF)

A programming model with the focus on data parallelism is High Performance Fortran (HPF) [16]. Here, the user specifies on array level the data decomposition via directives. Communication and synchronization is implicitly done by the compiler. Due to the fact that not all compiler vendors support this model we will not further discuss this approach, but concentrate in the following on the major programming models OpenMP and MPI only.

OpenMP

OpenMP [1] is a directive based approach for shared memory architectures. The compiler translates the directives into an appropriate thread handling. Usually it defines the work decomposition on loop level. This means that the loop is divided in equal parts which are distributed to the available number of threads. Data decomposition as well as communication is not necessary as all threads have access to the local memory in the node. Synchronization is implicit, although it can be also user-defined. Outside an OpenMP region (parallel region) which is defined by appropriate directives the program is executed by the so called master thread only (serial region). Besides the directives, OpenMP consists of a few additional library routines. OpenMP is standardized and available in C/C++ and Fortran compilers.

Message Passing Interface (MPI)

The second major programming model is the message passing interface MPI [2]. Here the user holds everything in his hand except the communication over the network itself. This means that the programmer specifies how the work and the data should be distributed and how and when communication has to be done. In practice, this is realized by calling from the sequential program MPI library routines which are linked to the application. As each processor runs the same executable there has to be the possibility to choose a specific process inside the program. This is done by a unique value for every process called a rank which is returned by a library routine. The message passing (communication) itself is done via special send/receive routines. Whether synchronization is explicitly necessary depends on the kind of send/receive routines used. In practise, you will typically find a domain decomposition of the application problem. At the boundaries of the local domain the boundary data of the neighbouring domain is also available and has to be updated via communication according to the underlying algorithm if necessary. MPI is standardized and defines interfaces for C/C++ and the Fortran language.

Most of the Semantic research community use Java for the implementation of their algorithms. For this reason, implementations of programming models in Java must be also considered.

See section 3.5 for more information of available implementations of MPI in Java.

3.4. Criteria to choose a parallel programming approach for applications (general hints)

From the description of the programming models given above it becomes clear that MPI is the most general approach in the sense that an application is able to run also on distributed memory architectures. Unfortunately it is also the most time consuming one getting an application running in parallel.

On the other side OpenMP is limited to shared memory machines. If this is no restriction, OpenMP has the advantage that it can be incorporated into an existing sequential application step by step. This is not possible with an MPI program. But, regarding the performance the higher effort for an MPI program will most likely pay off by getting a better speedup in contrast to the OpenMP approach. This is due to the better data locality you can achieve by the domain decomposition of your problem. Unfortunately modern hardware is very sensitive to data locality, so that it makes sense to put effort into this.

Therefore, how much data, where that data is, how it can be accessed will be the most important design considerations, especially for a distributed model. We have the choice to send data to computation nodes or send computational tasks to nodes where the data is located.

3.5. How to handle parallelization within LarKC

The application of parallelization techniques within LarKC will strongly depend on the chosen algorithms for implementing the different plugins and the platform itself. What is clear is that the algorithms developers need to adapt / develop their software in a suitable way for its parallel execution.

As mentioned before, most of the Semantic research community use Java for the implementation of their algorithms. For this reason, implementations of parallel programming models in Java must be also considered.

Java Implementations for MPI

Currently there are several Java implementations for MPI:

- One of the first attempts was Bryan Carpenter's [3] mpiJava [4] , essentially a collection of JNI wrappers to a local C MPI library, resulting in a hybrid implementation with limited portability, which also has to be recompiled against the specific MPI library being used.
- However, this original project also defined the mpiJava API [6] (a de-facto MPI API for Java following the equivalent C++ bindings closely) which other subsequent Java MPI projects followed.
- javaMpi [5] is another wrapper around C MPI library, comparable to mpiJava. JavaMPI wrappers were automatically generated from C MPI header by a special purpose code generator. This eases the implementation work, but does not lead to a fully object-based API because it is very close to the C binding.
- An alternative API is the MPJ API [7], designed to be more object-oriented and closer to Sun Microsystems' coding conventions.
- MPJ Express [8] is an implementation of the Java bindings for the MPI standard. The system implements thread-safe communication in a Java messaging system to make it compliant with Javas threading. This library addresses contradictory issues of high-performance and portability by providing communication devices using Java NIO (pure Java) and Myrinet. It is possible for end users to switch communication protocols at runtime
- William Pugh et al. compare MPJ to the use of MPI for Fortran language [9].
- DPJ [10] and Java-DVM [11] provide class libraries to wrap data-parallel and Distributed-VM-model parallel computations respectively.

- HPJava [12] [13] is an environment for scientific and parallel programming using Java. It is based on an extended version of the Java language. One feature that HPJava adds to Java is a multi-dimensional array, or *multiarray*, with properties similar to the arrays of Fortran.

The Ibis Framework

At the time of writing this deliverable, it is being considered the use of the Ibis framework [17], and especially their communication library IPL (Ibis portability layer). Ibis is an open source Java grid software project of the Computer Systems group, which is part of the Computer Science department of the Faculty of Sciences at the Vrije Universiteit, Amsterdam, The Netherlands. IPL helps processes communicate in a grid: flexible, efficient, with high performance, and quite simple to use. Ibis was designed to be used in a multi layer system. See the picture of the design below. On top of the system are the applications. These applications can use any of the programming models present in the Ibis project. The Ibis Portability Layer (IPL) acts as a common interface for the different programming models to the bottom implementation layer. Multiple implementations are available. Some, such as TcpIbis using 100% Java code to ensure portability, and some, such as the MPI based Ibis implementation taking advantage of local high speed networks using native code.

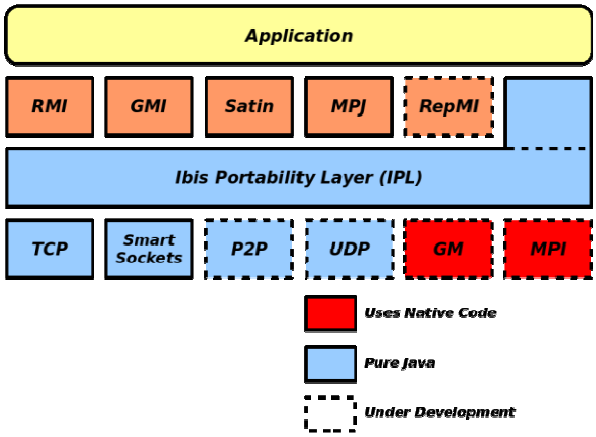

Figure 7: Ibis Portability Layer [17]

Other process communication Java Implementations

Other process communication Java implementations, which Sung Hoon Ko analyses in [14]:

- Sockets:
  - Initialization of socket communication is a complex procedure and clearly it should not be coded anew for each application program.
  - In their Life example the messages were contiguous byte vectors that could be transmitted efficiently through the *read* and *write* methods of the Java socket API. But in general the messages will have more complex types and the data may not be contiguous in memory. Using the typed primitives of the standard API may then incur extra costs of copying and type-conversion.
  - For reasons such as these, even with the simplified Java socket API, they concluded that direct socket programming will probably remain unattractive to scientific parallel programmers, even with the simplified Java socket API.
- Java Remote Method Invocation (RMI):
  - Java Remote Method Invocation (RMI), which is an object-oriented version of Remote Procedure Call (RPC), has been included with the Java Development Kit release 1.1. RMI provides a simple and powerful Java-to-Java communication model for invoking member functions on objects that exist in other Java virtual machines, exactly as if it were local objects running in the same virtual machines.
  - High performance distributed computing can be done with Java RMI, but this has a few drawbacks since RMI is designed for client-server programming in Web based systems over slow networks. In parallel programming fast RMI with low latency and

high bandwidth would be required. A better serialization would be needed, since Java's object serialization often takes 25%-50% of the time needed for a remote invocation.

- Parallel Virtual Machine (PVM):
  - Is a message-passing system that permits a heterogeneous collection of networked computing systems to be used as a single large parallel machine.
  - More focused on cluster scenarios, many cheap machines acting as one big computer vs. MPI which is focused on multi-processor machines
- Javelin:
  - The design is based on widely used components: Web browsers and the portable Java. By pointing their browser to a known URL of a broker, users automatically make their resources available to host parts of parallel computations. This is achieved by downloading and executing an applet that spawns a small daemon thread that waits and listens *for tasks from the broker. This approach makes it easy for a host to participate-all that is needed is a Java-capable web browser and the URL of the broker.*
  - It is good for very loosley-coupled parallel applications ("task parallelism") but it is less appropriate for the more tightly-coupled SPMD programming made possible by our approach.
- Java Party:
  - Preprocessor which translates Java code with an extra keyword remote *to RMI code, reducing the programming overhead. Communication costs are exactly the same as in RMI case.*

3.6. Implications for the LarKC "components" developers

Once it is decided that parallelization is going to be applied to certain algorithms, the developers need to adapt / develop their software in a suitable way for its parallel execution.

Other considerations must be taken into account when deciding to execute a plugin within a cluster, in a parallelized way, such as security issues.

These implications will be further analysed in future deliverables of the LarKC project.

4. Distributed Computing

Different approaches to distributed computing are analysed in the following sections. The one we will finally decide for LarKC will probably be a combination of some of them. As stated previously, **note that they are not necessarily in the same conceptual level, and even some of them can be a subset of others.**

4.1. Peer to Peer

Concepts

Peer to peer (P2P) is a particular case of Distributed computing in which participants rely on one another for services. Furthermore, peers in the system can elect to provide services as well as consume them.

The simplest P2P architecture consists of two computers which are connected whereby both are acting as server and as client. Theoretically a P2P network can be developed into a large network consisting of arbitrary various computers. However, there are several general aspects regarding to P2P architectures such as scalability, robustness and security which have to be considered for the deployment of an adequate P2P network.

Different models

There are several approaches on how to deploy a P2P architecture. In general it has to be considered how much centralization and decentralization of a P2P architecture is needed. Three types of P2P architectures are presented here:

- **Hybrid P2P model**, is a mixture between the client-server architecture and the peer-to-peer architecture. In this case, a server has the indexes of the peers and the resources, just the data exchange is decentralized. That is, search is performed over a centralized directory, whilst data exchange still occurs in a P2P fashion. The hybrid model takes place for services like Napster or ICQ.

- **Super-Peer model**, which includes a server for a certain amount of peers like it is done by KaZaA. The super-peer is the node that operates both as a server to a set of clients, and as an equal in a network of super-peers.

- **Straight P2P model** which does not distinct between different steps of authorisation for the different peers because all peers have equal rights.

Furthermore, we can consider different type of nodes in our P2P architecture. Particularly we have to distinguish three types of nodes, especially if we think on distributing computational tasks such as it is done in SETI [18] or BOINC [19]:

- **Constantly Available nodes with a High Bandwidth**: They provide their computational and storage resources all the time and that are connected to a network with a high bandwidth. These nodes are fully available all the time. They dedicate a large quantity of their computational and storage resources to the tasks. However, the high-bandwidth nodes can be used to calculate or storage large amounts of data so that the data is easy to access. However, these nodes do not have the required connectivity for "real" parallel applications. Their main benefit is data exchange intensive parallel execution.

- **Constantly Available Nodes with a Low Bandwidth:** they are typically unsuitable for exchange of a large amount of data and are more practical to take over minor computational tasks. Hence, these nodes are suitable for offline computational tasks which require only very simple communication such as decision support. It is not impossible to process large amount of data with such nodes but they are not optimized for this use. If the time for data exchange is not essential they can be used to complete arbitrarily expensive computations. The general interaction of low-bandwidth nodes should be reduced to a minimum. These nodes provide their resources all the time but they can only transport a reduced amount of data.

- **Sporadically Available nodes**: they are offline most time and the data exchange can take place only to a certain level. This type of nodes are, therefore, more adequate for offline computational tasks. They are related to low-bandwidth nodes. However, as opposed to the latter, a timely response cannot be expected. We have to distinguish between two different models of sporadically available nodes:
    - nodes with strong computational resources: Nodes that provide strong computational power in an offline mode can be used to execute complex tasks that need to be performed, but are of minor priority (i.e. no one relies on them). Typically one should expect that high(er) amounts of data can be exchanged during the online time
    - nodes with idle time computational resources: Idle time CR nodes are practically of no relevance to the network. They can take over only tasks of the lowest priority or least complexity, that would otherwise reduce the computational strength of other nodes. The amount of data is of less interest.

Advantages & drawbacks of different approaches

The main advantages of a P2P architecture in general are the high availability despite peer volatility, support for heterogeneous architectures and high scalability.

However, the performance of large-scale experiments with this kind of systems is a major challenge. For this reason, an adequate P2P model must be designed.

In the case of a **Hybrid P2P** approach:

- It can often result in a better trade-off between the ease of software maintenance, scalability, and reliability. One of the key advantages of the hybrid architecture is that it makes the discovery process more simple. This enables the deployment of a large number of clients and a high degree of scalability.
- Hybrid systems have also their drawbacks, in the sense that the costs resulting from the single node housing the centralized index are very high. Unless the index is distributed across several nodes, this single node becomes a performance and scalability bottleneck. Hybrid systems are also more vulnerable to attack, as there are few highly-visible targets that would bring down the entire system if they failed.

A Hybrid P2P architecture is shown in the following figure.

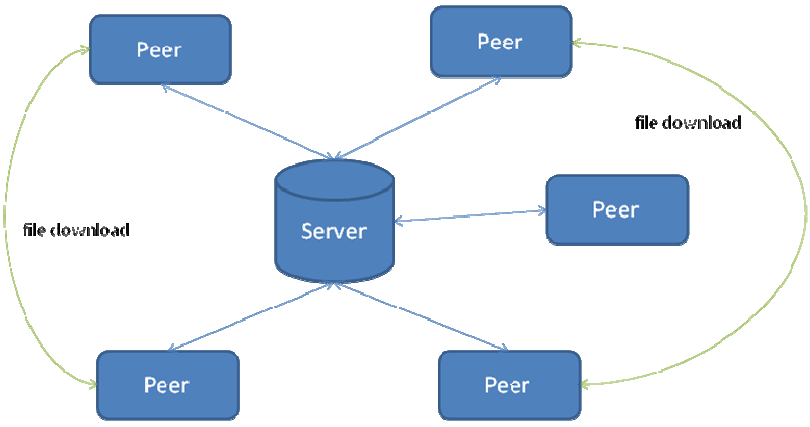

Figure 8: Hybrid P2P architecture

As shown in the figure above a hybrid architecture consists of a server which is connected with several peers. As mentioned before it is a mixture between the client-server architecture and the peer-to-peer architecture. In this case we have a server with indexes to the peers and the resources but the data exchange (here: the file download) is decentralized. Hybrid P2P architectures are known as brokered systems as well.

If we consider a **Super-Peer** architecture:

- The main advantages are, like in the previous model, the scalability and the easiness of the discovery process. Furthermore, Super-peer networks benefit from a balance between the inherent efficiency of centralized search, and the autonomy, load balancing and robustness to attacks provided by distributed search. Since there are relatively many super-peers in a system, no single super-peer need handle a very large load, nor will one peer become a bottleneck or single point of failure for the entire system. Moreover, they take advantage of the heterogeneity of capabilities (e.g., bandwidth, processing power) across peers.
- However, a super-peer may become a point of bottleneck for its clients. Therefore, the design of a super-peer network involves performance tradeoffs and questions difficult to answer, such as, ratio clients/super-peer, topology of the super-peer network, reliability of the super-peers towards their clients,...

A Super-Peer architecture is shown in the following figure:

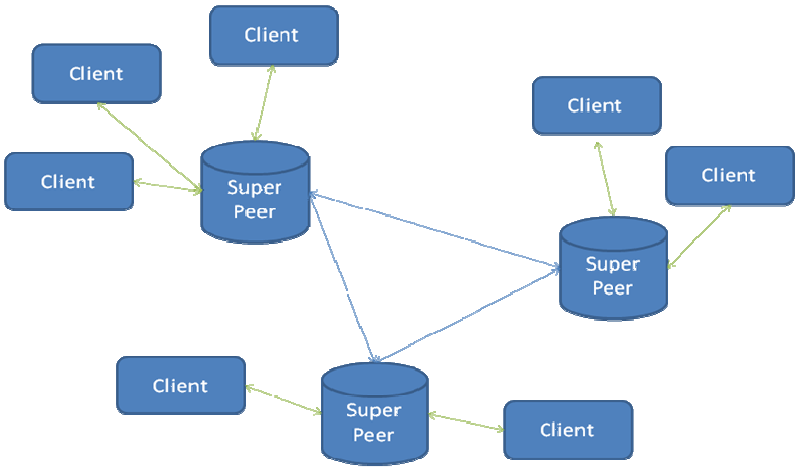

Figure 9: Super-Peer architecture

As shown in the figure a super-peer architecture consists of a set of super peers which are connected to the other super peers as well as to a set of clients. As mentioned before a super peer is a node that operates both as a server to a set of clients and as an equal in a network of super-peers.

Regarding the **straight P2P** approach:

- Pure P2P systems tend to be inefficient in different aspects such as:
  - Search protocol, that usually flood the network with query messages
  - Bottlenecks, due to the limited capabilities of some peers
- An efficient P2P system must be designed taking advantage of the peers heterogeneity, assigning greater responsibility to those who are more capable of handling it.

A straight P2P architecture is shown in the following figure.

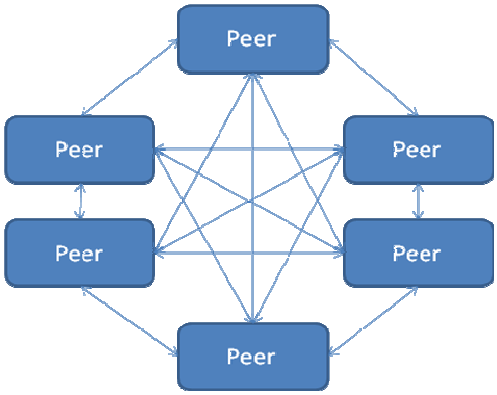

Figure 10: Straight P2P architecture

As shown in the figure a straight P2P architecture consists of a set of peers which are connected to all other peers. As mentioned before a straight P2P architecture does not distinct between different steps of authorisation for the different peers because all peers have equal rights. This approach is a rudimentary step of developing a more specified P2P architecture or to get a simple approach.

When thinking about the different types of nodes (high-bandwidth, low-bandwidth, sporadically available), the amount of data transferred between peers must be considered.

A general problem of P2P architectures is the persistent growing of data and applications that compromise the data coherence and the reliability of the system:

- With non-contractually committed parties in a P2P network, reliability and coherence is not given

- Reliability needs to be improved through data replication, but in this case data coherence becomes an issue
- Updates / changes in data need to be distributed across the network, via:
    - handshake protocols for ensuring transfer
    - high management overhead

4.2. Thinking@home

Concepts

Thinking@home is the term used to describe a distributed computing approach that consists on splitting up a program into parts that run simultaneously on multiple computers communicating over a network. It must deal with heterogeneous environments, network links of varying latencies, and unpredictable failures in the network or the computers.

The Berkeley Open Infrastructure for Network Computing (BOINC) platform [19] is currently the most popular volunteer-based distributed computing platform, originally developed to support the SETI@home project (Searching for Extra-Terrestrial Intelligence) [18]. In essence BOINC is software that can use the unused CPU cycles on a computer to do scientific computing. BOINC consists of a server system and client software that communicate with each other to distribute, process, and return work units.

Different models

The idea of Thinking@home is strongly related with a P2P architecture. In this case, we have to consider sporadically available P2P nodes with both strong computational resources and idle time computational resources, due to the originally “volunteer” nature of the nodes.

Advantages & drawbacks of different approaches

Since Thinking@home is a particular case of a P2P architecture, major advantages are also the support for heterogeneous architectures and high scalability.

However, due to the “volunteer” nature and the non-predictable performance of the available resources within the nodes the architecture design of this kind of systems is a major challenge in order to get the major possible reliability.

4.3. Cloud computing

Concepts

Cloud computing is a subset of distributed computing, where the computing (or the storage) is done somewhere “in the cloud”. The architecture behind cloud computing is a massive network of "cloud of servers" interconnected and orchestrated to work together on common problems. The main idea of Cloud Computing is that applications and data are not stored or processed on a local computer but as a cloud on a distant system, where resources are applied and managed by the cloud as needed.

The cloud itself is some kind of management which refers the application of a user to a service provider. Hereby, a main objective is that the user does not have to care about things such as finding an adequate service provider. The only thing the user has to do is to select the required service and put his data in the cloud. This request gets passed to the system management which finds a suitable service provider and adequate resources.

Cloud Computing is a Distributed Computing architecture, and a particular case of P2P architecture where the user does not run his applications and the required hardware on his own but it is done by a provider in the cloud.

The cloud computing "revolution" is being driven by providers including Amazon, Google, Salesforce and Yahoo! as well as traditional vendors including Hewlett Packard, IBM, Intel and Microsoft and

adopted by users from individuals through large enterprises [23]. For cloud computing solutions, see section 4.5

Different models

Different types of "clouds" can be considered, the main are the following:

- Cloud storage, where data is stored on multiple virtual servers, rather than being hosted on dedicated servers. Usually, hosting companies operate large data centers; and people who require their data to be hosted buy or lease storage capacity from them and use it for their storage needs. The data center operators, in the background, virtualize the resources according to the requirements of the customer and expose them as virtual servers, which the customers can themselves manage. Physically, the resource may span across multiple servers
- Cloud services, that are normally exposed as web services, offered in the form of a cloud computing architecture.
- Mixed model, combining both previous approaches

Advantages & drawbacks of different approaches

- One of the main advantages of the Cloud Computing is the availability and reliability. If an individual server in the cloud fails, the remaining servers quickly respond by taking over the workload.
- On the other hand, trust and security may become a major concern for the user of the cloud, as the data storage and the data processing is done elsewhere in the "cloud". Therefore, a clear trust and security policy must be established.

Cloud Computing can be deployed as a complement of a Thinking@home architecture where, additionally to the "volunteers@home", there exist one or more "clouds" of powerful and reliable servers hosted in remote locations.

4.4. SOA

Concepts

SOA [27] is a paradigm for the design and realization of a distributed architecture, where services can be distributed over a network and can be combined and reused to create business applications. Therefore, it can be said that a SOA leads to a loosely coupled architecture.

When we talk about SOA, we usually refer to three major concepts:

- Services: pieces of self-contained business functionalities
- Interoperability: between distributed systems, using different platforms and technologies
- Loosely coupling: that is, reducing system dependencies

When we talk about Service Oriented Architectures we talk about a set of loosely coupled and standard-based business aligned services that provide a high level of flexibility in responsiveness to business opportunities. The architecture style defining a SOA describes a set of patterns and guidelines for creating such an architecture. However, simply adopting SOA alone does not guarantee a successful work and in some cases a SOA approach should not be adopted at all.

As mentioned before the Services in an SOA are loosely coupled but they are re-usable as well. For the development phase of a SOA this means that it becomes possible to re-use existing service components and even re-use updated service components of existing services.

Different models

"The SOA model" does not exist itself, but the concrete SOA solution must be designed for every concrete situation, considering the specific context and requirements.

However, there are certain principles that specify a SOA. Besides reusability it is important to consider interoperability to ensure a interaction between the different modules of the architecture.

Furthermore, common standards as well as industry specified standards should be abided regarding to the special aim of the architecture. Further principles are Service Identification, Categorization, Provisioning / Delivery and Monitoring and Tracking. A basic SOA might look as shown in the following figure .

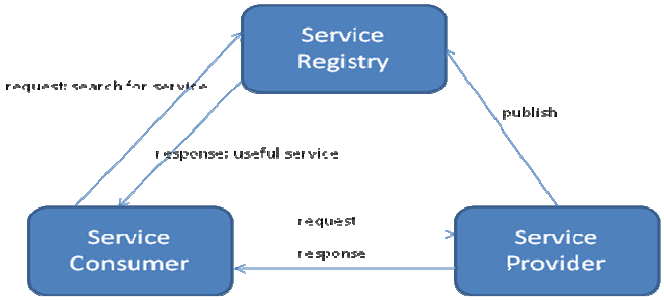

Figure 11: Basic SOA concept

The presented SOA architecture includes a service registry which stores indexes of services, a service consumer who requests a service and a service provider who gives response to the consumer. Furthermoe, the service provider publishes his service to the registry to enable the service consumer to find his service via the registry. It is obvious that the detailed architecture depends much on the requirements of a service and what the architecture should be about. Hence, there are further principles of architecture regarding design and service definition that depends much on the specific theme and design of a service. A main point of developing a SOA is to build an architecture which is strongly related to a specified service.

In general a SOA consists of a Service Provider, a Service Broker and a Service Requestor. The role of the Service Provider is to create a Web Service and mostly to publish an interface as well as store information in a Service Registry. The Service Provider has to decide about how to handle the Service and what aspects are more important and which aspects are not really relevant to the Service. Some possible aspects are the trade-off between security and availability and the decision which services should be exposed. The service Provider has always to consider what requirements a Service Requestor has to comply to make use of the service. Other important aspects are the agreements between Service Provider and Service Requestor which clarify the conditions for using the service. Furthermore, when developing a SOA the Service Broker has to be considered. The Service Broker is nothing else than the Service Registry which is responsible for providing the interface of the service and for implementation access information for the Service Requestor. Hence, the Service Requestor also called the Service Client locates entries in the Service Registry to invoke a suitable service. If we simplify this process we can say that we have a Service Provider, a Service Client and a Registry which are interacting.

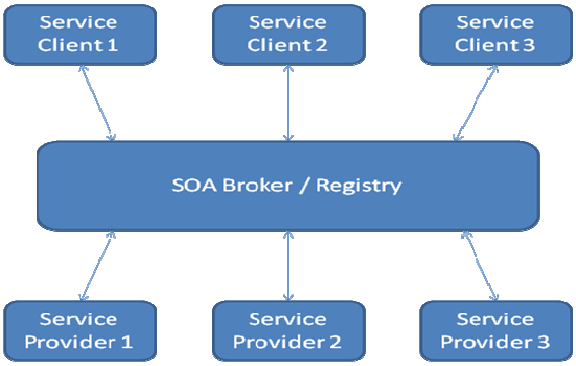

Figure 12: SOA using Service Broker approach

Web Services are one possible way of realizing the technical aspects of SOA. These services can be new applications or just wrapped around existing legacy systems to make them SOA-enabled. However, common technologies for developing web services are WSRF, SOAP, UDDI and WSDL. Furthermore, when using these technologies XML is a basic technology for developing web services this way.

When developing web services two instances can be identified, a service consumer and a service provider. The service consumer is able to request the service provider regarding to the service which is supported. UDDI can be used to register existing services for a service broker. To describe a service WSDL is used and to ensure the communication between service consumer and service provider SOAP is an adequate technology. To demonstrate the use of the mentioned technologies the basic SOA example is presented with some modifications.

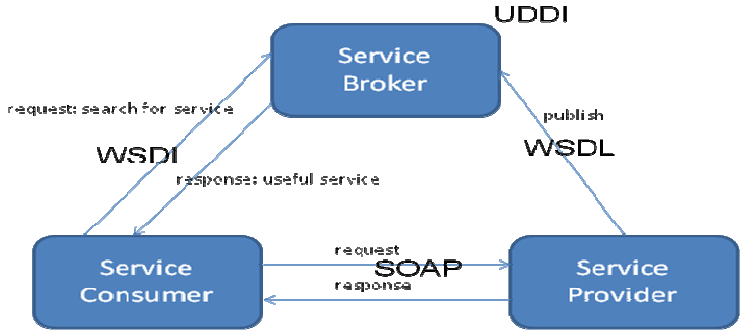

Figure 13: Basic SOA, possible implementation

Advantages & drawbacks of different approaches

- One of the major advantages of a SOA is flexibility. Business functionalities are modelled as independent services that can even be re-used for different applications, including the possibility to update "legacy" services to make them SOA-enabled
- The main drawback of SOA is that it is not a concrete tool that can be bought, but a paradigm that leads to a concrete architecture, applied to the concrete situation, context and requirements. Therefore, there is not a magic recipe to apply SOA, but the appropriate design must be applied to the concrete circumstances and this is a major challenge of the use of SOA.
- Loosely coupling introduces complexity to the system with regards to design, development and maintenance
- Aspects such as performance must be considered when designing a SOA. E.g. Web Services may not be the perfect solution when high performance is required
- The fact of using "legacy" services into the SOA leads to issues such as stability and backward compatibility

When we design the LarKC Collider Platform with a SOA approach, LarKC "plug-ins" should be designed as loosely coupled services that interoperate with each other through the Platform and also with the Platform itself.

The use of Web Services may be considered as the optimal solution for those services where the high performance is not the major requirement. However, for components such as the storage, where the reduced access time is a major requirement, a different technology may be a better choice.

4.5. How to handle distribution within LarKC

The application of distribution techniques within LarKC will strongly depend on the chosen algorithms for implementing the different plug-ins and the architecture of the platform itself. The algorithms developers will need to adapt / develop their software in a suitable way depending on the chosen techniques and architecture.

Distributed Implementation Frameworks

There are several distributed implementation frameworks or middleware available which should be considered in order to choose the most adequate(s) for the LarKC solution:

- **BOINC** is the short term for Berkely Open Infrastructure for Network Computing [15]. It is a platform for volunteer computing and desktop Grid Computing. It includes features such as autonomy of the project, volunteer flexibility, a flexible application framework, a digital

signature based on public–key encryption, server performance and scalability, source code availability, support of large data, multiple participant platforms, an open extensible software architecture and volunteer features. BOINC is designed to provide applications with large computation requirements and/or storage requirements. At this it becomes necessary that the applicaition is divisible into a large number of independent jobs. Otherwise the use of BOINC does not affect the application in a useful way. In order to achieve lot of participants for an project which share their computational power with the application public appeal is very important. However, the input and output data are sent via common commercial internet connections. Through this, the data have to meet the requirements of such internet connections to ensure an effectively processing. If the data per day and per CPU gets too much, it may be cheaper to use in-house cluster computing rather than volunteer computing.

- **Jxta [20]** defines a set of protocols that can be used to construct peer-to-peer systems using any of the centralized, brokered and decentralized approaches but its main aim is to facilitate the creation of decentralized systems. Jxta's goal is to develop basic building blocks and services to enable P2P applications for interested groups of peers [22]. It is designed to be independent of programming language, system platform and network platform, it attempts to provide a common language that all peers can use to talk each other. The Jxta middleware is a set of open, generalized P2P protocols that allow any connected device on the network to communicate and collaborate.

- **WCF** (Windows Communication Foundation, formerly code-named "Indigo") [21] is a set of .NET technologies for building and running connected systems. It is a way of communications infrastructure built around the Web services architecture. Therefore, it can be said that WCF is a SOA to support distributed computing where services are requested by consumers. Due to the fact that WCF is part of the .NET Framework applications can be developed in any programming language that can target the .NET runtime.

- **The Ibis Framework** [17], described in section 3.5, is also a framework to support distributed implementations.

Regarding cloud computing, different solutions and implementation frameworks can be found in the market, among them:

- **Amazon Web Services (AWS)** [24]: Amazon Web Services provides developers with direct access to Amazon's technology platform. Developers can build on Amazon's suite of web services to enable and enhance their applications.

- **Google App Engine** [25]: Google has recently launched a beta version of its Google App Engine. Google App Engine allows to run developer´s web applications on Google's infrastructure. It gives developers access to the same building blocks that Google uses for its own applications. App Engine provides a runtime environment that uses the Python programming language. Other programming languages and runtime environment configurations are being considered for future releases. Furthermore, the developer does not have to care about servers to maintain or storage to scale, as the distributed web server grows with the traffic and the distributed datastore grows with the data, transparently to the application owners.

- **IBM´s Blue Cloud** [26] is a (commercial solution) series of cloud computing offerings that allows corporate data centers to operate more like the Internet by enabling computing across a distributed, globally accessible fabric of resources, rather than on local machines or remote server farms.

As mentioned before, one of the possibilities to implement and deploy a SOA is using Web Services technologies. At the core of the Web services model is the notion of a service, which can be described, discovered and invoked using standard XML technologies such as SOAP, WSDL and UDDI. Conventionally, Web services are described by a WSDL document, advertised and discovered using a UDDI server and invoked with a message conforming to the SOAP specification. They are all open standards that have gained enormous support from thousands of companies and have been adopted by several communities, including the OGF [28], an open community committed to driving the rapid evolution and adoption of applied distributed computing. [22]

5. How to combine the different techniques?

This section aims to give some general hints on how the different techniques and technologies can be combined in order to get the optimal solution for LarKC. It is just a list of the possibilities that we should consider, but not a final choice. The decision on the concrete solution will be part of the architecture and prototype design tasks (Task 5.2 Early Release Prototype, Task 5.3 Platform Architecture and Design), in close cooperation with the plug-in development WPs (WP2, WP3 and WP4) and considering the input from the Use cases WPs (WP6 and WP7).

Parallelization

a. Parallelization to be applied to the algorithms that constitute the plug-ins (inside the plug-in). Execution of the parallel tasks within a single cluster. This case will be chosen with the different parts of the algorithm require frequent communication between each other.
b. Parallelization to be applied to the pipeline algorithm. To be analysed. In this case, several plug-ins could be executed in parallel during the same pipeline execution. For example, when the pipeline decides that some preliminary intermediate results obtained from one plug-in are enough for start executing the next step in the pipeline, while the first one is still obtaining further results. In this case, parallelization techniques such as MPI can be applied for the communication between the plugins.

Distribution

a. Distribution to be applied within a single plug.in: Here there are different possibilities
    i. Distribution of storage
    ii. Distribution of computing tasks to remote nodes, in a P2P way. For this kind of distribution, similar techniques to the parallelization case must be applied to the algorithms. In this case, it should be avoided a (frequent) communication between the different parts of the algorithm, as the peers may be located in remote locations one from each other.
b. Distribution to be applied between plug-ins:
    i. Follow SOA approach, with Web Services implementation. Plugins are modelled as “services”, with communication via web service interface to the platform.
    ii. Other kind of distributed approach between plug-ins, different than Web Services implementation: Communication between the platform and the plug-ins may be via Java RMI or some other RPC mechanism

6. References

[1] http://www.openmp.org/
[2] http://www.mpi-forum.org/
[3] Mark Baker, Bryan Carpenter, Sung Hoon Ko, and Xinying Li. mpiJava: A Java interface to MPI. Presented at First UK Workshop on Java for High Performance Network Computing, Europar 1998
[4] http://www.hpjava.org/mpiJava.html
[5] S. Mintchev. Writing Programs in JavaMPI. TR MAN-CSPE-02, Univ. of Westminster, London, UK, 1997
[6] Bryan Carpenter, Geoffrey Fox, Sung-Hoon Ko and Sang Lim. mpiJava 1.2: API Specification. October 1999

[7] Bryan Carpenter, Vladimir Getov, Glenn Judd, Tony Skjellum and Geoffrey Fox. MPJ: MPI-like Message Passing for Java. *Concurrency: Practice and Experience*, Volume 12, Number 11. September 2000
[8] Mark Baker, Bryan Carpenter, and Aamir Shafi. MPJ Express: Towards Thread Safe Java HPC, Submitted to the IEEE International Conference on Cluster Computing (Cluster 2006), Barcelona, Spain, 25-28 September, 2006, http://mpj-express.org/#
[9] William Pugh and Jaime Spacco. MPJava: High-Performance Message Passing in Java using Java.nio
[10] V. Ivannikov, S. Gaissaryan, M. Domrachev, V. Etch and N. Shtaltovnaya. 1997. DPJ: Java Class Library for Development of Data- Parallel Programs. Institute for Systems Programming, Russian Academy of Sciences, http://www.ispras.ru/~dpj/
[11] V. Ivannikov, S. Gaissaryan, M. Domrachev, O. Samovarov. DVM Object Model and its Implementation in Java Environment
[12] http://www.hpjava.org/index.html
[13] Han-Ku Lee, Bryan Carpenter, Geoffrey Fox, and Sang Boem Lim. HPJava: Programming Support for High-Performance Grid-Enabled Applications. To appear, *International Journal of Parallel Algorithms and Applications*.
[14] Sung Hoon Ko, *Object Based Message Passing in High Performance Computing using Java*. Syracuse University, 2000, http://www.hpjava.org/theses/shko/thesis_paper/thesis_paper.html
[15] http://boinc.berkeley.edu/trac/wiki/BoincIntro
[16] High Performance Fortran, http://hpff.rice.edu/
[17] http://www.cs.vu.nl/ibis/
[18] http://setiathome.berkeley.edu/
[19] http://boinc.berkeley.edu/
[20] https://jxta.dev.java.net/
[21] http://msdn.microsoft.com/en-us/netframework/aa663324.aspx
[22] Ian J. Taylor, From P2P to Web Services and Grids – Peers in a Client/Server World, Springer-Verlag London Limited, 2005
[23] http://en.wikipedia.org/wiki/Cloud_computing
[24] http://www.amazon.com/AWS-home-page-Money/b?ie=UTF8&node=3435361
[25] http://code.google.com/appengine/docs/whatisgoogleappengine.html
[26] Press Release: Armonk, N.Y. and Shanghai, China - 15 Nov 2007. IBM Introduces Ready-to-Use Cloud Computing, http://www-03.ibm.com/press/us/en/pressrelease/22613.wss
[27] Nicolai M. Josuttis, SOA in practice – The art of distributed system design, O´Reilly, 2007
[28] http://www.ogf.org/