Dottorato di Ricerca in Informatica
Università di Bologna, Padova

A Tuple Space Implementation for Large-Scale Infrastructures

Sirio Capizzi

March 2008

Coordinatore:	Tutore:
Prof. Özalp Babaoğlu	Prof. Paolo Ciancarini

Abstract

Coordinating activities in a distributed system is an open research topic. Several models have been proposed to achieve this purpose such as message passing, publish/subscribe, workflows or tuple spaces. We have focused on the latter model, trying to overcome some of its disadvantages. In particular we have applied spatial database techniques to tuple spaces in order to increase their performance when handling a large number of tuples. Moreover, we have studied how structured peer to peer approaches can be applied to better distribute tuples on large networks. Using some of these result, we have developed a tuple space implementation for the Globus Toolkit that can be used by Grid applications as a coordination service. The development of such a service has been quite challenging due to the limitations imposed by XML serialization that have heavily influenced its design. Nevertheless, we were able to complete its implementation and use it to implement two different types of test applications: a completely parallelizable one and a plasma simulation that is not completely parallelizable. Using this last application we have compared the performance of our service against MPI. Finally we have developed and tested a simple workflow in order to show the versatility of our service.

Acknowledgements

I would like to thank my supervisor Prof. Paolo Ciancarini and Prof. Antonio Messina for their support during the years of PhD course.
A particular thank to Prof. Thilo Kielmann and Prof. Carlos Varela for their valuable observations.

Contents

List of Figures

Chapter 1

Introduction

The Service Oriented Architecture (SOA) [52] has promoted software modularization and reuse, leading to distributed applications based on the coordination of different services that execute a specific task. In this Lego-like approach, coordinating the different services is of fundamental importance and needs a coordination model able to satisfy the requirements of distributed applications designed for SOA. Thus, there is a need to implement a coordination model able to satisfy the requirements of distributed applications designed using the SOA.

This is more evident for Grid applications that have to accomplish very complex tasks. In fact, Grid technologies are converging toward SOA, that simplifies application development and promotes sharing of computational resources.

Developing models that manage the coordination of activities in a distributed system is a very old research area. In past years several coordination models have already been proposed and today it is possible to use them in a SOA context. The most used models are publish/subscribe [53] and workflows [131]. Although both can be implemented with different features, they are the background of almost all SOA oriented frameworks.

Along with these most known models, another less widely used one has been developed: the tuple space model [63]. It manages coordination using a virtual

shared memory on which it is possible to operate using synchronous and asynchronous operations. Services that compose a distributed application can synchronize themselves writing or taking data from the tuple spaces. This model shows several interesting features:

- *Dynamic Membership*: services can join or leave the distributed application without the use of any kind of protocol and any influence on the application (if adequately developed).

- *Inherent Load Balancing*: clients can be programmed to request tasks, execute them and store the result into the space. In this way, the more powerful ones will execute more tasks thus leading to a sort of load balancing without complex support. This approach is similar to the concept of work stealing introduced by Cilk [61]

- *Declarative Approach*: there is no need to specify which host will take or write a specific data item: it is simply inserted or removed from the space using generic queries. This can simplify the development and execution of applications, abstracting them from the network topology and allowing reconfiguration of the network without changes in the application code.

- *Powerful interface*: that allows distributed applications to be rapidly developed using few simple operations. These operations also support both asynchronous and synchronous behavior.

These features can be very useful for applications as well as services development. Using tuple spaces, the coordination of activities of a high level applications can be developed in less time and with less resources, supporting at the same time every kind of distributed algorithms. Services can profit by the intrinsic features of this model like independence from the network topology and load balancing. For example, index services can be developed without worrying about load balancing that will be automatically supported by the tuple space service.

Clearly not all applications can be efficiently developed with this model. For example, applications based on the transmission of large data set should use more suited services or protocols, but the coordination of these transmissions can be achieved with the use of tuple spaces.

However, to support the high abstraction level required by this model, a carefully implementation is needed in order to avoid performance bottlenecks. In fact, many freely available and widespread tuple space implementations suffer from performance problems and this has probably limited the diffusion of the tuple space model.

The purpose of this thesis is twofold:

- to study techniques that improve the performance of tuple space implementations, maintaining its original semantics
- to demonstrate the feasibility and validity of these improving techniques developing a tuple space service and testing it with different types of applications

Thus, this thesis is organized as follows: the next three chapters describe the state of the art in Coordination, Grid technologies and P2P systems respectively. They constitute the related work of our proposal. Chapter 5 describes our proposed ideas to enhance the performance of actual tuple space implementations in both centralized and highly distributed contexts. Using some of these ideas we have developed a Grid service that implements the tuple space model as described in Chapter 6. Chapter 7 describes the experiments we have conducted in order to study the efficiency of our implementation using two different types of applications: a highly parallel one and another one with a higher communication to computation ratio that simulates a plasma. Using this last application as a benchmark, we have compared the performance of our framework against MPI [138], a typical application programming interface (API) for the message passing model. Moreover, a simple workflow has been tested in order to show the versatility of our service. Finally, Chapter 8 makes some conclusive remarks.

Chapter 2

Coordination and Tuple Spaces

2.1 Introduction

Research in the coordination field has more than 20 years of history and several paradigms and systems have been proposed. The main concept of this research field can be defined as follows:

Program = Coordination + Computation

This means that programs show two orthogonal aspects: the first one does the computational work and the second one has to do with the coordination of the efforts required to reach the final solution. Although this definition seems quite simple, it has produced several proposals that can be very different from each other. In fact, they can be classified in several ways.

If we look at the way in which coordination is inserted into the computational code, we can identify two groups of systems [11]:

- *Endogenous Systems* in which the coordination code is **not** separated from the computational one.
- *Exogenous Systems* in which the coordination code is clearly separated from the computational one.

If we analyze the way in which the coordination is achieved we have the following two groups of models [100]:

- *Data-Driven Coordination Models* in which the coordination is lead by data transformations.

- *Process-Oriented Coordination Models* in which the coordination is defined by means of the coordination patterns used by the processes.

These two types of classification are orthogonal and coexist at the same time in all coordination systems.

2.1.1 Data-Driven Coordination Models

A typical example of data-driven coordination models are tuple space systems [63] in which coordination is achieved by means of a virtual shared memory, the tuple space, used by all computational entities to store and retrieve their data. The tuple space represents a separation between time and space because the computational entities do not know either each other or the state of the entire system, but only the data that they need to process and that are gathered from the space. There are several implementations of this model that will deeply described in Section 2.2.

Another example of a data-driven coordination model is the multiset rewriting in which the coordination is accomplished defining rewriting rules on multisets (i.e. sets whose element can have multiple copies) that lead data transformations. Implementations of this models are for example GAMMA [19] that is based on a chemical metaphor (rewriting rules are repetitively applied until no suitable data are present in the multi sets), CHAM [30] or IAM [8].

Bauhaus [38] is a tuple space based model in which tuples can be multisets.

2.1.2 Process-Oriented Coordination Models

In process-oriented coordination models, the coordination is achieved defining the coordination patterns that connects the various computational entities of the systems and that can change during the time. In this type of models data have no meaning and computational entities are considered black box. Typically these models are also exogenous because the definition of coordination patterns is separated from the computational code.

A typical control driven model is for example IWIM [10] that is based on processes, ports, channels and events. Processes can be of two types: workers that do the computational work and managers that create new processes and dynamically connects them using channels. Ports are used by processes to write and read data and constitute the start and endpoints of channels. Events are used by the processes to gather information about the environment state. MANIFOLD [13] is an implementation of the IWIM model.

ConCoord [69] is a coordination language very similar to MANIFOLD.

Configuration description languages like DURRA [20], DARWIN [88] or RAPIDE [112] describe complex software by interconnecting existing components.

Reo [12] is a model similar to IWIM in which coordination patterns are described only by means of basic channel types and their composition without the need of events or the notion of manager and worker processes.

TOOLBUS [29] uses a unique communication channel to coordinate different processes.

2.1.3 Hybrid Coordination Models

Both process-oriented and data-driven coordination models have some disadvantages. Data-driven models can be inefficient or too tightly integrated with the computational code to easily support modifications in the application. On the other side process-oriented models can be too static to handle the dynamic execution of open systems.

Thus, some proposals have been made in order to merge the best features of these two models. For example ACLT [95], TuCSoN [96] and MARS [35] merge the concept of events with shared data space defining the notion of programmable medium: programmable actions are triggered by operations on the space.

IWIM-LINDA [99] or ECM [111] describe the integration of a process-based view in tuple space based systems. ECM is a general model for the coordination languages STL, STL++ and Agent&CO.

In the following sections we will describe more deeply the tuple space model, since it is the basis of our implementation. As showed by the previous general taxonomy of coordination models, there are several other models that could be also used. The main reason for the choice of the tuple space model is that it supports better open systems whose environments or requirements can change. Process-oriented models can deal with this type of systems too, but we think that the separation of time and space supported by tuple spaces is better suited for this type of systems, allowing an application to work in different environment or under different requirements without any need to rewrite or reconfigure it.

2.2 Tuple Spaces

The Tuple Space model has been proposed by Gelernter and Carriero as coordination model for distributed application [63]. It is based on the concept of a unique virtual shared memory, the tuple space, on which various hosts arranged in a cluster can operate using a small number of synchronous and asynchronous operations. These operations are:

- `out` that inserts a tuple into the space
- `in` that synchronously removes a tuple from the space matching the given template. If no such tuple is found, the application waits until a matching one is inserted into the space.

```
int x;

out("data",2,3.5);
 //this call matches the previously inserted tuple
in("data",?x,double);
//from here x == 2
```

Table 2.1: *An example of the original tuple space operations*

- `rd` that synchronously reads a tuple from the space matching the given template. Like the previous operation, if no tuple is found the application waits until a matching one is inserted.

- `inp, rdp` are the asynchronous versions of `in` and `rd` respectively. They return immediately NULL if no matching tuples are present into the space.

- `eval` that executes a process on the first available host using the passed function. This operation is used to create worker processes.

In the original model by Carriero and Gelernter, tuples are usually ordered arrays of typed values. The allowed types are the C primitive types and the pointers to arrays and structs. Templates are tuples which contain one or more wildcards used for matching other tuples. A wildcard can be a type descriptor or a variable that will contain the corresponding value after an operation call. Tuples are selected through the use of templates applying the so called associative matching: two tuples matches if they have the same length and every corresponding pair of elements has the same type or the same value. Thus, templates can be seen as filters that select the desired tuples. The previously defined operations are used inside standard C code. Since some of them are synchronous, the tuple space model can be used to synchronize the execution flow of distributed applications deployed in a cluster.

During the years the original model has been modified in several ways and many other tuple space systems have been developed. Due to their number it is difficult to described them all. Thus, we will focus only on the most important ones.

Tuple space systems can be classified in two main groups according to the way in which tuples are stored:

- **Centralized tuple space systems** in which all tuples of a space are stored on the same server.
- **Distributed tuple space systems** in which the tuples of the same space can be stored on different servers.

In the first type of systems, the centralized tuple store can become a bottleneck, whereas in the second ones load-balancing strategies can be employed but operations can be more expensive. Moreover, in the latter case, the tuple distribution can be more or less transparent to the clients.

Following this simple classification, in the next sections we analyze the most important implementations of both groups. For the sake of simplicity, we have classified Linda as a centralized tuple space system, although in some aspects it can be considered a distributed one too.

2.2.1 Centralized Tuple Space Systems

Linda

TCP Linda [148] is the last incarnation of the original system thought by Carriero and Gelernter and it is used as a cluster wide computational framework especially in the field of pharmaceutical applications (the Gaussian application family is based upon it). Respect to other cluster-based frameworks like MPI, it introduces tuple space operations as full-fledged statements of the C and Fortran programming languages and not as function calls that reside in a library. Thus, a specific compiler has to be used to detect the tuple space operations and to

define the distribution that is hard-coded into the binary files produced. Since the system is closed source, it is very difficult to exactly know the real strategies employed for the distribution but it is clear that this approach can gather more information than a normal distributed system and apply specific optimizations. In fact, with a static analyzer it is possible to consider operation usage patterns and optimize the data flow between the hosts.

The cluster organization can be seen as a master/slave model, in which the master is the node where the application starts and the slaves, that do not change during the execution, are defined by a configuration file. This structure is rigid in the sense that no fault-tolerance policy are employed and when a slave crashes the application aborts.

TSpace

TSpace [91] is a tuple space system developed by IBM and implemented in Java. The model introduced many changes to the original model of Linda. In particular the most important modifications are:

- **Multiple space handling**: TSpace can manage many tuple spaces with different names at the same time whereas Linda uses a unique space.
- **Event notifications**: clients can register themselves for receiving notifications of modifications of the space like the insertion or removal of tuples
- **Transactional support**: to guarantee ACID properties to operations between different spaces.
- **Support XML tuples**: to contain semistructured data.
- **Access control for clients**: to authenticate clients before operating on the space.

The system consists of one or more central servers where the spaces reside and clients that remotely access them.

Java Spaces

JavaSpaces [116] is a standard service of the Jini framework [117] that implements a tuple space. Like TSpaces, it supports multiple spaces, access control, events and transactions but does not support natively XML tuples. It also introduces support for the persistence of the space that is saved on the file system and can be reloaded after crashes or shutdowns of the server.

A particular characteristic of JavaSpaces is the way in which tuples are defined. In fact, usually tuples are considered ordered vectors of values but JavaSpaces see them as standard java objects with different named fields. These fields can have different types that are used for the associative mapping. It is an idea similar to named columns in database tables. In this way, the programming interface is more friendly and it is simpler to create *active tuples*, i.e. tuples with associated operations since they are full-fledged Java objects.

Other implementations

In more than twenty years several different tuple space models and implementations have been proposed in the research literature. The previous three systems are the most important due to their diffusion and characteristics. Other centralized systems are for example Tucson [96], that introduces the concept of command tuple that can modify the behavior of the space, Objective Linda [78] is probably the first proposal of an object-oriented tuple space model, Klaim [47] that introduces the concept of location for identifying tuples alongside a deep formal analysis of tuple spaces in general, and X-Mars [36] that uses XML-based tuple spaces for the coordination of agents.

2.2.2 Distributed Tuple Space Systems

GigaSpaces

GigaSpaces [139] is a commercial tuple space implementation. It is based on the JavaSpaces model and interfaces but provide higher scalability and efficiency. It is designed to be the core of a framework in which tuple spaces are used to guarantee scalability and efficiency to applications without any need to rewrite them. In fact, all main services are implemented using tuple spaces, allowing changes of distributed applications structure and scalability simply by means of configuration files. So, developers are not more bothered by scalability issues and can concentrate themselves on the development of the requested functionalities. Services implemented in this way are, for example, a JMS-based messaging service and a distributed cache.

Moreover, it is possible to deploy standard J2EE applications on the top of GigaSpaces. In fact, there is a framework, OpenSpace, that is implemented using GigaSpaces and the Spring framework, allowing standard SOA applications to use GigaSpaces's features. It can be also accessed directly using C++ and .Net.

To achieve the required scalability and performance, GigaSpaces tuple spaces can be configured in several ways and they are deployed in a SLA-driven cluster using different replication strategies.

Blossom

Blossom [121] is a high performance distributed tuple space implementation written in C++. It extensively uses the default C++ type parametrization to implement tuples and related classes. The whole system is a standard C++ library, thus no precompiler is needed in order to compile Blossom programs.

One of the most important characteristics of this tuple space implementations is its support to strongly typed tuple spaces. In fact, each tuple space has an associated tuple type: only the tuples that match this type are allowed to be inserted

into the space. In this way, the developer cannot introduce new bugs incorrectly modifying tuples.

Moreover, other advanced specification of the tuple structure can be defined for the space. For example, it is possible to assert that all tuples of a space have a constant in the first field. In this way, the space runtime can use this information to automatically hash this constant value and use the result to distribute the tuples between the various hosts composing the cluster.

Thanks to this distribution approach and the extensive use of C++ type parametrization, Blossom seems to be more efficient than SCA Linda, as reported by some tests.

Blossom implementation is based on Roots, a C++ high performance communication library developed by the same author.

Lime

Lime [101] is a tuple space implementation designed to extend the original tuple space model in order to support ad-hoc networks. The system is based on agents that can move between different hosts of the same network (logical mobility) or different networks thus modifying their topology (physical mobility). These agents are the only active entities of the system, are identified by a unique ID and can own one or more tuple spaces where the tuples are stored. This spaces can be shared with other agents on the same network and their content will be merged in order to give the agents the view upon a virtual tuple space. The merging operation is done transparently by the Lime runtime and is executed whenever an agent joins or leaves a network.

The semantics of the traditional operations is unchanged although some new operations have been defined in order to send data directly to one specific agent (location).

Moreover, the system introduces the concept of reaction to some events, like the insertion of a tuple or an agent leaving a network. Agents can register for

some of these events and execute code when they are fired. This simply represents a classical notification support with the exception that two types of events can be fired: *strong events* that are atomically fired across the entire network and *weak events* that do not follow a strict synchronization constraint. The first ones are more computationally expensive.

Lime is an example of a so called Global Virtual Data Structure, i.e. a data structure that is created merging and sharing local data owned by the participants of an ad-hoc network.

Comet

Comet [82] is the communication infrastructure of the Automate middleware (see Section 3.2.3) and represents a distributed tuple space implementation for Grid-like environments. Its architecture is based on the following layers ordered from top to bottom:

- Coordination Layer that exposes the operations and stores the tuples
- Communication Layer that implements an index for the tuples and dynamically organizes the p2p overlay network
- JXTA substrate on which the overlay network is created.

As a common p2p application, each node is responsible to store a fraction of all possible tuples that can be inserted into the space. Tuples are described using XML and are indexed using the following procedure:

- For each field a hash function is computed creating an n-dimensional point
- The n-dimensional point created is then mapped to one dimension using a Hilbert Space Filling Curve.
- Finally the tuple is forwarded to the correct node using Chord (see Section 4.1.1)

The tuple search algorithm is similar to the previous one. A template represents a hyperplane on an n-dimensional space. Using a procedure similar to the previous one it is mapped on a list of possible hosts where the corresponding tuples can be stored. This list is then searched to find the requested tuple.

This system seems an interesting solution for p2p tuple spaces but it presents many drawbacks:

- It is not freely available
- does not support common Grid standards like WSRF
- There are doubts on the performance of the range search algorithm used, that seems to be heavily based on application-level support since Chord does not provide range search capabilities.

Tota

Tota [89] is a middleware based on agents that can communicate through a distributed tuple space. The tuples are composed by data and distribution rules that define how they should be distributed to neighbors. Every agent has a runtime that receives tuples from the neighbors or propagates the tuple produced by its own agent. When a tuple arrives to a new node, the propagation rule is executed in order to define how it should be distributed. There are three types of tuples:

- *MessageTuples* that travel the entire network as a wave
- *HopTuples* whose distribution is based on the number of hops performed
- *SpaceTuples* that use some types of geographical information (e.g. GPS) to travel across the network.

SwarmLinda

SwarmLinda[41] is a biologically inspired implementation of a distributed tuple space. The model used is based on ant colonies: the tuples are the food and the

templates are the ants that try to find the requested tuples. During the search for tuples, the template releases a sort of trace on the visited nodes. This trace can be followed by next templates in order to optimize the search: if the trace of a similar template is found then it is followed, otherwise a random walk is employed. Traces have an evaporation rate that prevents ants from following old routes.

If after some time the template has not found any results, it has three choices: suicide, sleep for a random time and then restart the process or jump to another random node and continue the search.

Storage of new tuples is based on a modified version of the brood sorting algorithm [60]. A new tuple is stored in a node that has neighbors with similar tuples. If no such node has been found, a random decision is taken in order to store the tuple in the current node or to search a better one.

PeerSpace

PeerSpace [33] defines a formal tuple space coordination model that is completely decentralized and based upon p2p networks.

Each peer is identified by an id and stores a subset of the data space. Since the model is completely abstract, no restriction has been made upon the format used for the tuple. However, the chosen format should be flexible enough to represent all data required, to be readable by all peers and to be lightweight enough to reduce the communication overhead. PeerSpace identifies three kinds of data:

- *local data* that can be retrieved only using the id of the peer that owns them and disappear when the peer leaves the network
- *replicable data* that are transparently replicated between peers
- *generic data* that can be transparently moved through the network

These data types are needed in order to support both context-aware and context-transparent data. Context-transparent data are needed in order to avoid the so

called Slashdot effect [135], i.e. the slowdown caused by a high number of request for the same data item.

Three possible operations are defined in order to access data on the decentralized space. These operations can be composed in a sequential or a parallel fashion. The operations defined are:

- `write` that inserts new data into the space
- `read` that gets non-destructively data from the space according to a template
- `take` that extracts destructively the data from the space

All these operations support the three types of data previously defined. The write operation is always done on the local storage and then the replicable and generic data are diffused in the network according to some load-balancing rules. The read and take operations work using a peer horizon. Since the network can be huge, it is impossible to gain a global view on all the peers connected. For this reason only the peers reachable in a predefined number of hops are involved in the operations. This represents a relaxed semantics for consuming operations with respect to that commonly used in tuple space models, but it is needed in order to support networks with large sizes. However, this policy does not avoid important informations to be retrieved: actually they can be replicated or moved as needed.

The model defined by PeerSpace is abstract, but a sample implementation of it has been developed using JXTA.

2.2.3 Comparison Between Distributed Tuple Space Systems

As can be seen from the previous sections, there are only a few tuple space implementations that are completely distributed. This is probably due to the fact that completely distributed tuple space systems are more complex to develop than centralized ones.

Probably for this reason the scalability of these systems is quite doubtful. SwarmLinda and PeerSpace does not present any result regarding their real or simulated performance. Lime and Tota are designed for small ad-hoc networks and uses flooding-based approaches that can result in low performance. Moreover, as will be shown in Chapter 5, Lime can also suffer from low performance due to its tuple space implementation when the number of tuples is high. Comet seems to be the only implementation specifically designed for medium and large networks. Nevertheless, its scalability can be a problem too; in fact using space-filling curves to index tuples can produce a high number of requests for associative matching that can saturate the network. The presented test results do not help in removing this doubt because they have been collected on small networks of at most 50 peers. Blossom is probably the only system that provides high performance in a distributed context. Nevertheless, it is designed only for homogeneous clusters and cannot manage a large scale network composed by different sites.

Apart from Blossom, none of these systems employs fault-tolerance techniques, like replication, in order to assure that a given tuple will be present with high probability into the space despite peer volatility. In this way, data can be lost independently of their importance and the standard blocking semantics of the tuple space cannot be implemented.

Finally none of the systems uses a structured p2p network approach and we have not found any example of a similar system in literature at the time of the writing of this thesis. Probably, this is due to the fact that tuple space implementations need a range query support that has been introduced in structured networks only recently.

For these reasons, we have decided to study the possibility to use structured p2p networks to implement distributed tuple spaces. The results we have obtained are described in Chapter 5.

Chapter 3

Grid Technologies

Grid technologies have been developed to support "coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations" [59]. A virtual organization is a temporary group in which preexistent resources (computational services, storage, ...) owned by different institutions are shared according to policies defined by their owners themselves. Virtual organizations and the relationships present inside them can have very different purposes, size and stability. Thus Grid technologies can be seen as middlewares that help virtual organizations to reach their goals promoting the sharing of resources, but respecting the policies defined by the various resource owners. The multi-institutional and dynamic nature of virtual organizations are main features that Grid technologies should support. In the last years several different Grid middlewares have been developed: however they share a general architecture. It consists of the following layers from the bottom to the top:

- **Fabric** that provides the resources whose sharing will be mediated by the Grid. It implements the local resource specific operations and for this reason should not be used directly by applications. Example of resources implemented by this layer are for example NFS storage clusters, access to local scheduling systems ....
- **Connectivity** that implements the basic communication and authentication protocols used for accessing the resources and the other components of the

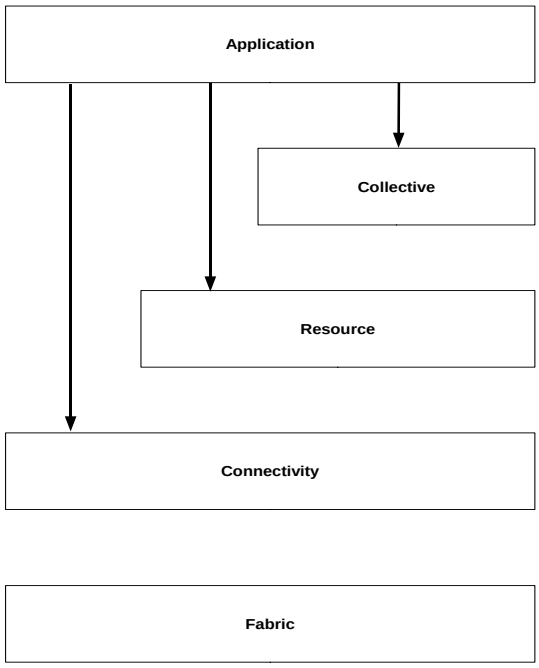

Figure 3.1*: Schema of the general Grid architecture*

Grids. It contains standard protocols like HTTP, FTP or more specific protocols like GridFTP or GSI protocols that provide Grid specific features.

- **Resource** that provides mechanisms for the initiation, monitoring, control and accounting of single fabric resources. With respect to the fabric layer, the operations provided are standard and does not depend on the specific local resource.

- **Collective** that provides services and protocols able to handle collections of resources using the Resource and Connectivity layers. Services on this layer are for example scheduling engines, workload management systems or directory services.

- **Application** is the top most layer that uses the previous ones to implement user specific tasks.

This common architecture can be more or less evident in the various systems.

Often the term Grid is used incorrectly. In fact, sometime it is used for systems whose purposes and architectures are very different from the previously defined ones, or as a synonym for High Performance Computing (HPC). To better understand these differences, Foster in [56] underlines that a Grid system is a distributed system with all the following particular characteristics:

- it coordinates resources that are not subject to centralized control.
- it uses standard, open, general-purpose protocols and interfaces.
- it delivers non-trivial qualities of service.

For example the Sun Grid Engine [147] provides non trivial quality of service but has a centralized control and, according to this definition, it is incorrect to define it with the term Grid.

3.1 Grid Middleware

The generic architecture described previously has driven the development of several different Grid middleware with different characteristics. Most of them are based on standard service-oriented middleware enhanced to support the previously described challenges.

Usually in their architectures is evident the difference between a lower and an upper layer. The lower layer can be identified with the fabric layer defined by the general Grid architecture, although sometimes it can also contains part of the connectivity layer like the implementation of high performance communication protocols. Instead, in the upper layer, it is sometimes impossible to clearly identify the connectivity, resource and collective layer. High level services are deployed on traditional application servers like JBoss and often provide Web Service interfaces. High level services are usually implemented in Java whereas the low level ones in C/C++.

Although various types of Grid applications have been developed, it is still difficult to have exact information about the coordination systems used. Nevertheless it is possible to identify three groups:

- *Legacy Applications* that are executed using metascheduler or batch systems. This kind of applications maintains its original coordination model without changes. For example, most of them use MPI as coordination infrastructure and so the ported version uses now a Grid-enabled version of it.

- *New Grid Applications* specifically created for the Grid, they use the coordination models supported by the middleware on which they are deployed. For example, this kind of applications uses workflow systems to accomplish their tasks.

- *Ported Applications*, originally developed for other distributed systems and then adapted to the Grid. Typically these applications use a hybrid approach to coordination. In fact, they are composed by old legacy components and new modules that use Grid services to allocate and schedule their tasks.

This limited survey on coordination model used by Grid applications shows an important characteristic of these types of environments: two different application visions coexist at the same time. In fact, applications can be executed on batch systems (scheduler/metascheduler) like in the 70's, or they can be composed by different Web Services like predicated by the SOA.

The first vision can be defined as "black-boxed" because the user has limited or no possibility to define how the application will be distributed on the network.

The second one is "white-boxed" because through workflows the user can define its own distribution.

Each of these visions has advantages and disadvantages: the "black-boxed" approach can deal more simply with large network, heterogeneity and long running applications, but can hardly optimize complex interactions because it does

not know the execution flow. The "white-boxed" approach instead can manage complex interaction and dynamic execution but is more complex to code on large network.

The following describes only the most important and stable Grid middleware available today. This list is not exhaustive, because new middleware are developed continuously for specific tasks.

3.1.1 Globus Toolkit

The Globus Toolkit (GT) [57] is developed at the University of Chicago and represents the de facto standard for Grid computing. It is composed by several Web Services and provides SOA functionalities to Grid applications. As stated in [32], the GT can be used in two different ways: as a standard SOA middleware or as a batch system, executing legacy jobs. The way in which the system is used depends on the specific application. The toolkit can be divided into the following main components:

- **Common Runtime** that contains common libraries for both applications and services. It implements the WSRF [144] and WSN [143] specifications on which all Web Services of the GT are based. It is implemented in Java and C. For Python and Perl are available only clients.
- **Execution Management** that provides a Web Services (GRAM) for job submission. It works as a Web Service based gateway to other cluster scheduling systems like PBS [151], Condor [119] or LSF [145]. The Globus Toolkit also provides a meta-scheduler GridWay [70] that allows the control of many GRAM enabled clusters, aggregating them in a unique virtual cluster.
- **Security** that provides communication security, authentication and authorization. It uses X.509 certificates for authentication and implements the WS-Security and WS-Trust for message level security and credential delegation respectively. It also provides a SAML implementation for describing more advanced authorization policies.

- **Data Management** that provides Web Services for data movement (RFT), replica location (RLS) and replica management (DRS). It also implements GridFTP and a common interface for database access (OGSA-DAI).

- **Information Service** that provides an index service (MDS) that is able to aggregate information from different sources and allows queries on them. Information coming from other standard Globus services are automatically collected and every Web Service can register its own information. The MDS is able to trigger operations when some of the information change.

The previously described services represent only the main components provided by the GT. Other optional tools are developed by subprojects like:

- MPIch-G2 [77] that is a Grid aware MPI [138] implementation. It allows different MPI clusters to be glued together in a unique virtual cluster like GridWay does for schedulers. It also support firewall traversal functionalities for MPI messages. Unfortunately, it is still not well integrated in the last version of the GT.

- BSP (Bulk Synchronous Parallel) Model [42] implementation provided by the subproject BSP-G. It implements a coordination model similar but simpler that MPI

- CoG Kits [124] that are deployed on top of the toolkit and provide high level interfaces that allow rapid development of Grid applications. Moreover, they abstract the services from underlying grid infrastructures, allowing the code to be more portable across different infrastructures or versions of the same infrastructure.

3.1.2 WSRF.NET

WSRF.Net [71] is an implementations of the WSRF specification using the .NET framework that aims to be interoperable with the Globus Toolkit. Its architecture is completely different with respect to Globus and the development of Web

Services is based on annotated sources. Stubs and skeletons are indeed automatically created by external programs using the annotations present in the source code. This middleware can be deployed only on Windows systems and probably on Unix flavors on which the Mono Runtime Environment is installed [1]. It offers a smaller number of services than its Globus counterpart. In particular, this project has implemented GridFTP.Net and GRAM.Net composed by clients and services compatible with the homonymous Globus services.

3.1.3 UNICORE

UNICORE [113] (UNiform Interface to COmputing REsource) is another Grid framework based on the concepts described by the Open Grid Service Architecture. It has been designed to provide a seamless, secure, and intuitive access to heterogeneous computing environments. It is composed by two different software packages: the UNICORE Client and Server Bundle.

The UNICORE Client is composed by a GUI that helps UNICORE users to prepare their applications for submission. Applications are defined using workflows, connecting together the services provided by UNICORE, or using custom scripts if needed. After having prepared the application, its whole lifecycle is automated and can proceed unattended.

To cope with software heterogeneity, application workflows are compiled in Abstract Job Objects (AJO) representing generic service functionalities that are automatically mapped to real implementations at the server-side. Moreover, each application or job group has an associated USpace that logically groups all input and output files and automatically manages the data transmission between different UNICORE sites and jobs.

The Server Bundle is installed on clusters that represent UNICORE sites hosting different types of UNICORE services. The clusters also provides several legacy applications that can be accessed through the AJO mapping. More legacy

[1] This is an idea based on our knowledge of the Mono Runtime Environment but we have not found examples of such a use in the research literature.

applications can be plugged in UNICORE servers implementing a specific wrapper in Java or Perl. There are a series of standard service hosts by the UNICORE container like for example data management, job submission, storage management and other. Other high level services can be also developed and deployed on top of the UNICORE container.

UNICORE is implemented in Java and usually uses RMI or Java Serialization for the communications. In the last version several Web Service interfaces have been developed in order to enhance interoperability with other Grid middlewares like the Globus Toolkit. Thus UNICORE sites can be accessed through the old interfaces as well as the new Web Service interfaces and this choice is done automatically by the UNICORE client when the application is prepared and submitted.

3.1.4 gLite

gLite [137] is a Grid middleware developed by the EGEE project to constitute the basis of a common European Grid infrastructure for scientific computing. It has been developed using components and experiences coming from other projects and it is the successor of the LCG-2 middleware used previously. The Grid used by the EGEE project is deployed on more than 200 sites worldwide with more than 30000 CPUs. Its main task will be to support the LHC experiment at the CERN.

The gLite's architecture is composed by the following main modules:

- **User Interface (UI)** composed by a set of tools for job submission, monitoring and data transfer used for managing application on the Grid. This module represents the access to the Grid and is usually installed on users PCs.

- **Computing Element (CE)** represents a cluster on which computations take place. Several CE are available in the network and they are composed by

a Grid Gate, that enables the access to the CE, a Local Resource Management System, that schedules the jobs like normal batch systems, and a set of Worker Nodes, that execute the jobs. Several different Local Resource Management Systems can be employed like Condor, OpenPBS, LSF, SunGrid Engine.

- **Storage Element (SE)** provides uniform access to storage resources. Each SE is managed by a Storage Resource Manager that can have different functionalities depending on the size and characteristics of the storage system that has to manage. SE are accessed via the GISFTP protocol and can be composed by different type of hardware like disk arrays or type-based mass storage systems.

- **Information Service (IS)** provides information about Grid resources and their status. All resources are described using the GLUE schema that represents a common conceptual data model for every resource on the Grid. Two different types of IS are used in gLite: MDS (provided by the Globus project), for resource discovery and to publish the resource status, and R-GMA, for accounting, monitoring and publication of user-level information.

- **Data Management** responsible for handling file replicas that can be stored at different sites. Files are identified in a position independent way by means of GUIDs or Logical File Names. Storage URLs and Transport URLs depend instead on the site where the replica is located. A mapping of these identifiers is maintained by the data management subsystem and continuously updated.

- **Workload Management System (WMS)** is responsible to choose the CEs where to submit the jobs. Suitable CEs are chosen on the basis of the requirement expressed in the job description (match-making) and on the current load of the controlled CEs. Moreover, WMS uses the Logging and Bookkeeping Service to track job status.

- **Security** responsible for enforcing and integrating the security policies of the different domains that forms virtual organizations, allowing a transparent access to resources. It is based on the GSI model that uses X.509 certificates and a public key infrastructure for identifying users and doing data cryptography.

3.1.5 GridBus

GridBus is a Java middleware developed by the University of Melbourne that implements a Grid framework compatible with other Grid systems. It consists of several high level layers and services that can be deployed on other Grid middlewares like the Globus Toolkit, UNICORE, NorduGrid or Apple's XGrid.

One of the particular characteristics of GridBus is the use of economic metaphors to model various aspect of the services like the job scheduling or the allocation of resources. They are based on the idea that resources or actions have a cost and that entities that have to handle them try to minimize the expenses respecting the original requirements. Using this model, algorithms can produce quasi-optimal allocation or scheduling policies efficiently.

The main components developed by GridBus are:

- **Alchemi** [86] is a job submission system written in .Net that can execute jobs on Windows clusters. It has been designed to support different configuration of the cluster and to automatically adjust the jobs execution based on the cluster load.

- **Gridbus Broker**, a client-side metascheduler used to execute applications.

- **Grid Workflow Engine**, an XML-based workflow engine.

- **Grid Market Directory**, an index service that also stores resource costs.

- **GridSim**, a framework for the simulation of Grid applications, scheduling policies or other type of distributed applications. It is able to simulate an

heterogeneous computer network on which Grid services and applications can be deployed to study their performance.

- **GridScape** allows rapid development of Grid access portals without any need to know web technologies.

The Gridbus Broker, Grid Workflow engine and Grid Market Directory will be discussed more deeply later in this chapter.

3.1.6 NorduGrid

NorduGrid [51] is a Grid infrastructure that extends the functionalities of the Globus Toolkit 2.4. It has been designed to be scalable, to avoid single points of failure and to meet the requirements of both users and system administrators. Although it is based on the Globus Toolkit, several new services have been developed to satisfy the previous goals. In particular the main components developed by the NorduGrid project are:

- **User Interface (UI)** is a lightweight client component that is installed on the user's machine to allow a simple access to the Grid. In particular it provides several functionalities like job submission and monitoring, resource discovery and brokering.

- **Information System**, based on the Globus Toolkit's MDS, has been modified to become more scalable and better represent status information used by the project. The Information System is composed by a distributed sets of databases and indexes arranged hierarchically in a tree with redundant paths to avoid single points of failure. The data model used is different with respect to the original one to better represent every possible type of information used in the network.

- **Computing Cluster** is the basic computing unit of NorduGrid and is composed by a cluster of worker nodes, hidden from the rest of the network, a

Grid Manager, that manages job requests coming from the Grid, and a local Information System for storing the cluster and jobs status. NorduGrid does not impose a specific configuration for clusters, rather it tries to minimize its requirements allowing the Grid Manager to be deployed on existent cluster without modifying local policies.

- **Storage Element** that is used to store the data and is eventually accessed by the Grid Manager or the UI to deal with data transfer. The protocol used is GridFTP.

- **Replica Catalog**, based on the Globus counterpart with some minor changes, is used to locate and manage replicas on the network.

In the last years some work has been done in order to provide interoperability between NorduGrid and gLite.

3.1.7 BOINC

The Berkeley Open Infrastructure for Network Computing (BOINC) [7] is a framework used to create so-called Desktop Grids, i.e. Grid systems that use normal PCs as computational resources. Volunteers donate some of their unused CPU time to one or more projects. Probably the term Grid is not well suited for this system due to the low QoS supported. The BOINC architecture is composed by two main components: a server and client module.

The server module is deployed on a mid-range server and it is used to schedule and monitor project workunits and their results. There is one server module installation for every project and all clients that participate to the project contact it to receive workunits and to submit results.

The client module is common for all projects and it is able to execute workunits coming from different projects. It is installed on the volunteer machine and executes workunits when the machine is less loaded.

This system has to deal with two major issues: cheating of results and heterogeneity. The first problem is addressed distributing redundant workunits and analyzing the returned results: the most common result is considered to be correct. To complete this analysis a minimum number of results should be returned, otherwise more workunits should be rescheduled. Heterogeneity is addressed making available different versions of the same application compiled for different architectures. Clients will automatically download and execute the application version matching their architecture.

The most famous project based on BOINC is Seti@home.

3.2 Main Research Topics

The research in Grid systems addresses different main topics regarding functional as well as architectural aspects. The main research topics are the following:

- Security
- Resource Discovery and Management
- P2P
- Mobility
- Coordination

These topics are only the most important ones of the wide Grid research field. Moreover, several topics intersect each others (like coordination and resource discovery or p2p and mobility) and can produce hybrid solutions. In every case, these research areas have produced different solutions and implementations.

3.2.1 Security

This was one of the first topic that the research in Grid systems had to address. Actually, the support to virtual organizations and their mutable nature needs a

strong access control and authorization to guarantee that only allowed persons or institutions can access the network and its services. This has lead to the definition of different security systems, like GSI [127], VOMS [4] or Shibboleth [126], that can also allow interoperability and mutual identification of credentials between different Grid middlewares.

3.2.2 Resource Discovery and Management

Grid networks are composed of different types of resources that applications can use. Thus, part of the applications task consists in identifying suitable resources and allocate them in the most cost effective way. Thus, this research area tries to solve two problems. The first one is effectively indexing the various type of resources to retrieve them in a second time. The second problem is developing systems that are able to collect information about the available resources and allocate them according to the application requirements.

The first problem has lead to the creation of different types of index services, that index resources and update their information using some sort of shared schema.

For example, the **MDS** [46] is an index service part of the Globus Toolkit. It is able to aggregate information from different sources and allows queries on them. Information coming from other standard Globus services are automatically collected and every Web Service can register its own information. The MDS is able to trigger operations when some of the information change.

Grid Market Directory [132] is an information service part of the GridBus middleware. Its main characteristic is that service providers can publish their services and related costs, allowing consumers to find those ones that satisfy their requirements with the minimum cost.

To address the second problem, resource brokers or matchmaking agents have been developed. These agents try to automatically allocate resource pools representing the best compromise between available resources, their cost and application requirements.

Gridbus Broker [122] is a metascheduler that uses an economic model to optimize the scheduling of processes and the allocation of resources. The idea is that every operation on resources (e.g. allocation) has a cost and the broker should minimize it respecting at the same time the application requirements. It is a client side application that resides on the user machine and it is compatible with different execution system like for example GRAM, UNICORE or Alchemi.

In [14] another marketmaker approach is used in order to reduce message exchanges between clients and providers. A marketmaker agent is used to hide completely the providers from the clients. Its task is to allocate resources at the minimum price and to resell them to clients. In this way clients need to know only the marketmaker agent, that is able to buy or lease larger resources from the providers thus reducing communications and optimizing allocation.

In [34] several other strategies are described in order to better support Grid scheduling. The thesis proposed by the authors is that using economic based strategies it is possible to obtain a quasi-optimal scheduling and resource allocation without a global knowledge of the entire Grid. The approach proposed in this paper has been used to develop the GridBus Broker.

DI-Gruber [50] is a completely distributed resource broker that aims to avoid the possible bottleneck represented by a central brokering service. It has been developed as an extension of the GRUBER broker deployed on the Open Science Grid. The prototype has been developed and tested on PlanetLab with a simulated network of up to 40000 nodes.

3.2.3 Peer to Peer

Since its beginning, in the Grid community is present a research field that aims to integrate p2p solutions in Grid environments in order to:

- provide more scalability
- better support fault tolerance

- simplify the system configuration

This vision has not yet led to an integration of p2p systems in current Grid projects due to great problems in security, trust and support of QoS requirements. Nevertheless, several projects have been started to integrate p2p approaches in a Grid environment. These projects follow two different research directions: enhancing singular Grid service with p2p solutions like [58, 72, 44, 17, 128] or developing a complete new middleware based on p2p paradigms. Although the first research direction is more probable to be integrated into Grids currently in production, the second one is also interesting for the challenges it poses. The main organic projects that aim to develop a complete p2p grid middleware are described in the following sections.

WSPeer

WSPeer [67] is a complete middleware based on Web Service and WSRF that supports a p2p environment. It is based on the idea that the standard client/server model used by all common Grid system is not suitable for a p2p world. Thus all peers should be considered service providers that can be contacted and discovered in a p2p fashion. WSPeer is based on the P2PS middleware [125] that creates and manages unstructured p2p networks based on superpeers. WSPeer is build on top of P2PS and implements the WSRF standard. It uses three types of communication protocols: HTTP, p2ps, a protocol defined by P2PS, and Styx, a protocol developed for the Inferno Operating System and used for its NAT-traversal capability. This middleware supports network transparency using so called Virtual Network Addresses (VNA) based on URNs. They uniquely identify peers and resources independently from the network in which they reside. According to some test results this middleware is interoperable with the GT4 when the HTTP protocol is used. This middleware is still under development.

Narada Brokering

The Narada Brokering [98] is a complete messaging middleware focused on distributed Web Service systems. It is based on JXTA and implements a publish/-subscribe model to enable communications between various distributed entities. Several policies for message exchanges can be used like reliable delivery, ordered delivery, secure delivery. Communication can be done using different protocols like UDP, TCP, HTTP, SSL and parallel TCP. The system is compatible with the JMS interfaces and implements several Web Service standards like WS-Transmit, WS-Reliability or WS-Eventing but does not still provide support for WSRF [2] at this time.

AutoMate

AutoMate[2] is an experimental framework for autonomous services on the Grid. It uses algorithms inspired by biological systems and its architecture is composed by the following layers:

- *Accord Programming Layer* that extends existing distributed programming models and frameworks to support autonomic elements.
- *Rudder Coordination Layer* that provides a coordination framework and an agent-based deductive engine to support autonomic behaviors. The coordination framework is based on a distributed p2p tuple space implementation, Comet[82], described in Section 2.2.2.
- *Meteor/Pawn Middleware Layer* that provides a content-based middleware with support for content-based routing, discovery and associative messaging.
- *Sesame Access Management Layer* that provides access control and dynamic context-aware control.

[2] According to some announcements, support for WSRF will be implemented in a future release

Although the very interesting features of this system and the great number of related publications, Automate is not freely available. Moreover, it seems not to support legacy applications and actual Web Service standards like WSRF.

SP2A

SP2A [6] is another p2p middleware based on Web Services. It is developed using JXTA and allows resources to be semantically annotated with OWL-S ontologies. The serialization support is relatively simpler than that provided by the previous middlewares and does not support the WSRF standard.

In [40] a p2p application specific scheduling algorithm is proposed. The aim of this paper is to propose a model to define p2p scheduling for generic application. The model uses two different types of agent that define two overlay networks. The first type is represented by computational agents that execute tasks. They are ordered in a tree like structure and know their parent and their children. The second type of agents is represented by distribution agents that distribute data in order to transmit it to the best computational nodes according to an application metric (e.g. computational speed). At the application start the distribution node create a torus with the k best nodes chosen between n. These nodes execute the tasks and the torus is continuously updated in order to contain always the k best nodes.

Finally, in [39] an organic approach to desktop Grid is proposed. Actual desktop Grid middlewares like BOINC have a centralized nature. This paper describes how a completely distributed desktop middleware can be implemented using a biologically inspired approach. Tasks and subtasks of the computation are organized in a tree like structure, that is continuously updated according to the computational speed of the agents and the link status between them. Only a part of the children of an agent are used for the computation: they represent the best ones according to a specific metric. In such a way completely decentralized and scalable desktop Grids can be created.

3.2.4 Mobility

Often Grid middleware need to be employed in situation in which the network is also composed by mobile devices or sensors that transmit data to services for their analysis. Similar situations are, for example, seismological analysis or disaster management. Thus, this research tries to define models and services that integrates mobile aspects in Grid middlewares.

Mobile Grid is a research area of the Grid community that aims to develop models and middlewares that allow mobile devices (PDAs, sensors...) to access resources on a Grid infrastructures or to be part of a Grid-based applications. Scenarios in which this approach can be useful are, for example, geological monitoring, crisis management or traffic monitoring. This is a relatively new research area in the Grid community and practical results are sometimes still missing. For example, both the projects Akogrimo[76], financially supported by the European Community, and K*Grid[142], financially supported by the South Korean government, aim both to provide middlewares and standards for mobile Grids.

The ISAM[129] project proposes a pervasive computing environment that integrates three concepts: context-awareness, mobility, and Grid computing. The applications developed upon it can show adaptive behaviors but, for the mobility, only wireless network infrastructures are supported. Ad hoc networks are not taken in consideration.

Another project, MoGrid[49], has developed a complete p2p mobile Grid infrastructure. It is composed by two different softwares: MoCA, a middleware for building context-sensitive mobile applications, and InterGridade, a Grid middleware supported by the Brazilian government.

Kurkovsky and others in [81] propose a distributed problem solving environment based on mobile devices that is integrated into a Grid infrastructure.

3.2.5 Coordination

Coordinating the tasks composing an application is of a fundamental importance when the application components should operate on different network and services spread worldwide. For this reason services and coordination models have been used in order to simplify the development and deployment of Grid applications.

Most of the Grid middleware implement publish/subscribe services or message passing interfaces like MPI. However, these coordination model can hardly handle complex interactions or execution on different sites. For this reason, workflow engines have been employed in order to simplify the development of complex application. Some workflow engine are part of the Grid middleware (e.g. UNICORE), whereas others are available as third part services. The most important workflow engine are the followings.

The **Karajan Workflow Engine** [152] is a workflow engine for the Globus Toolkit. It uses a custom XML language for describing workflows that is based on GridAnt, a make-like application of Grid middleware.

The **Grid Workflow Engine** [130] uses a simple XML-based workflow language to schedule processes on Grid environments. IBM TSpaces is used to implement the workflow engine. Moreover, this system employs an optimization model based on economic metaphors.

Pegasus [48] is another workflow engine for Grid systems. Its main characteristic is the ability to define reusable abstract workflows that are automatically transformed in concrete workflows on the basis of the available resources. This automatic transformation employs AI planning techniques to avoid manual intervention.

Kepler [84] uses another approach to define and schedule workflow tasks. Actually, it is based on a generalization of the Actors model [3] which defines two types of entities: actors that are responsible for the computations and directors that dynamically define the connections between actors and their execution order.

Finally **Triana** [120] is a problem solving environment that uses workflows to represent applications. Users are able to define the distribution policy of the tasks composing their workflows and to change the workflow structure at runtime. It does not depend on a specific midlleware: Triana workflows can be executed on standard Grid middleware like the Globus Toolkit as well as on a JXTA network.

The systems described so far are well tested and successfully used in many applications. Nevertheless, in the literature other prototypes are described which manage workflows using different strategies.

In [15] a description of an intelligent Grid environment is presented. The approach used defines an intelligent broker based on agents: they autonomously create workflows based on an abstract description of the job submitted by the user. Ontologies defining services and tasks help agents in creating workflows and executing them. Moreover, it is possible to detect faults during the execution and automatically redefine the workflow.

Another proposal is presented in [21] and uses a process language inspired by the π-calculus. Agents execute workflows described in this language. The system allows the use of so called "coordination templates", i.e. workflows not completely defined that represent generic coordination models. They can be useful in order to modularize the coordination, since the workflows employed can be categorized into few models. Agents can exchange these templates and instantiate them according to the job parameters. It is an approach similar to Model Driven Architecture (MDA) or Problem Solving Model (PSM).

An interesting proposal is described in [43]. This paper describes a simulation framework based on user-defined component and rules. The simulation is decomposed in components that are dynamically loaded by a specific service at runtime. The components are executed in parallel and at the end of the computation the next steps are defined by some user defined rules that are fetched from a central repository. This approach is similar to workflows but with some differences. First of all the components are deployed at runtime and are simpler to develop than standard services. In this way the Grid environment is transparent

from the components point of view, allowing a simple porting to other environments. Moreover, the rules can simplify the development of fault-tolerant applications implicitly defining a dynamic workflow. Unfortunately the way in which a network-wide synchronization is established is not clear and the centralized architecture of the information system is a possible bottleneck.

In [92] an interpreted language (Alua) is used in order to interactively run and coordinate distributed applications. Although an interactive environment can be useful for small applications or rapid development, it is questionable if it can be useful for long-running applications too. However, the interpreted language has a syntax simpler than a classical XML-based workflow language.

Another approach is described in [18]. It uses a chemical programming paradigm in order to accomplish coordination. In fact, the semantics of this programming language implicitly defines coordination. Unfortunately the work described is only a proposal and does not seem to have any real implementation.

Our proposal can be inserted in this research topic.

Chapter 4

P2P Systems

In this chapter we will describe the state of the art of p2p systems. Some of the systems presented in the following sections are on the basis of our proposal. In particular, p2p protocols supporting range queries will be used by our prototype as described in the next chapter.

P2P systems are distributed systems with the following particular characteristics:

- **No Different Roles**, nodes in a p2p system do not show different roles like client or server: all peers are considered equal and are supposed to implement the same behavior or interfaces. Sometimes, some peers are more ”important” than others since they are more stable or have more computational power. In this case, they are called ”Super Peers” but they behave exactly like normal peers.

- **High Volatility**, peers can join or leave the network at any time and without any previous notification. The support to this behavior leads to a high tolerance to faults.

- **High Scalability**, the network size can scale up to millions of nodes. Thus, all decisions should be taken locally without the presence of a centralized authority.

These characteristics, that apply in different ways to the various p2p implementations, make this kind of distributed system very useful in situations where fault tolerance and high scalability are needed.

The most important operation in p2p systems is the discovery of the best peer (or peers) that can store or owns a specific resource. Although this operation is also present in other "traditional" distributed systems, in p2p systems it becomes the most important one because it must take into account the particular characteristics of these systems. Since a centralized authority cannot be present, the decision on how to choose the best peer is taken locally with a limited amount of information. According to the way in which this operation is accomplished, p2p systems can be classified into two main groups:

- **Unstructured Networks** in which peers do not know the resources stored by their neighbors. Thus, routing paths are not previously known and have to be defined locally collecting information along the way from neighbors. The algorithms used for these types of systems are based on flooding-like models sometimes also known as percolation, gossip or epidemic. Systems that implement similar models are for example JXTA [115], Gnutella [108] and eMule [80].

- **Structured Networks** in which peers know the possible resources stored by the neighbors. In this way, the search can be sped up avoiding to query peers that cannot store the requested resource. The allocation policy is defined by the algorithm and usually based on identifier equality (Uniform routing) or other type of relations (Non-uniform routing).

  These systems can be grouped in other two main subgroups according to the model employed to insert or find a resource in the network:

  - **Distributed Hash Tables (DHT)** that create the abstraction of a hash table for managing the resources in the network.
  - **Tree-based structures** that provide the abstraction of a tree arranging the peers accordingly.

In the following sections n will indicate the number of peers composing the network.

4.1 Distributed Hash Tables

One of the most important model of structured networks are the so called Distributed Hash Tables (DHT), that create a logical structure implementing an exact matching behavior. Peers and resources are addressed with the same type of identifier: given a resource id, the peer with the most similar id is responsible for storing that resource. In this way, a routing algorithm can be implicitly defined forwarding the messages to the peer that is closer to the destination. However, to reach good performance the routing process should be as quick as possible requiring a low number of hops between nodes. Usually DHTs arrange their nodes in a way that requires $O(\log(n))$ messages to identify suitable peer. The most of them uses skip lists in order to reach such performance.

Skip Lists are sorted lists in which the i-th node has $\lfloor \log_m(n) \rfloor$ pointers to nodes that are at distance of $1, m^1, m^2, \ldots, m^{\log_m(n)-1}$ from it. In this way, a search algorithm has a complexity of $O(\log_m(n))$ instead of being linear. Unfortunately, creating and maintaining such a data structure is extremely expensive and it cannot be employed in a distributed system since it would require too many messages. Therefore, DHTs use probabilistic skip lists, that are constructed only with the available nodes trying to respect the properties of exact skip lists, and tend to them when the number of elements tends to infinity.

As said before these types of p2p systems originally provided only the support for exact-matching queries. Clearly this type of queries is not suitable for some applications like for example distributed databases or computation processing. For this reason in the last years there was a great effort for integrating support for range queries in this type of systems. In the following section we will discuss the original DHT systems as well as the most important extensions that introduce the support for range queries.

4.1.1 Original DHT Models

CAN

CAN [106] arranges nodes in a k-dimensional toroidal space: each node represents a k-dimensional point that is responsible for a specific hyper-volume. All resources, whose id falls into this volume, are stored by its owner. Identifiers in CAN are computed using a SHA1 hash function that produces a sequence of 160 bits that is split according to the number of dimensions. When a new node joins the network, the volume in which it falls, that was owned by another peer, is divided in two along one dimension and the new peer becomes the owner of the new volume. Resources that were managed by the old peer and that fall in the new volume migrate to the new node. In this way, each node knows at least $2 * k$ neighbors. Messages are forwarded to the peer nearest to the destination according to the Euclidean distance.

The routing cost of CAN is $O(k/4 * n^{1/k})$, thus its performance increases with the number of dimensions, being relatively poor for low dimensions. Therefore, to reduce communication costs, some enhancements of CAN use several instances of this protocol at the same time with an increasing number of dimensions.

Chord

Chord [114] uses a ring like structure in order to arrange their nodes. Each node has an id that is computed using a SHA1 hash function and it is positioned in the ring according to its natural ordering. Each node has pointers to its predecessor and to $O(\log(n))$ successors at exponentially increasing distance. A resource is managed by the node that has an id greater or equal to it. When a new node joins the network it is inserted in the ring according to its id and the links of its predecessor are updated to take into account its presence.

The routing cost of Chord is $O(\log(n))$

Pastry

Pastry [110] is a DHT implementation intended to be the communication core of a decentralized service-oriented framework: some services, such as the publish/subscribe implementation Scribe, have been already implemented.

Pastry uses prefixes in order to arrange its nodes. Identifiers in Pastry are obtained using an MD5 hash function that produces sequences of 128 bits. They are interpreted as binary integers with a base of b bits (usually 4). The base b is a configuration parameter that should be tuned in order to get the best performance. Each node in Pastry maintains two tables: the routing table and the leaf table. The routing table is composed by $128 \div (b-1)$ entries containing the known peers whose id shares the first i digits with the actual id and differs in the $i+1$-th digit. Each entry has a limited number of registered peers (defined by another configuration parameter) that are chosen taking into account the real network topology: only the peers with the minimum ping delay or with the minimum distance in hops are inserted in the routing table. Moreover, they are ordered using a LIFO policy to take into account the fact that "oldest" nodes are more probably to remain active than "newer". The excluded nodes are inserted in the so called leaf table that is used in case of failures or lack of information. In order to provide better fault tolerance, resources can be replicated among more peers.

A request is forwarded to the peer that shares the longest prefix with it or to the nearest one, according to the Euclidean distance between their ids, in the case of multiple candidates or lack of information. Using this algorithm the routing cost of Pastry is $O(\log_b(n))$.

Tapestry

Tapestry [133] is another DHT implementation based on prefixed similar to Pastry. It uses a dynamic version of the Plaxton algorithm [103] and, unlike Pastry, uses a 160 bit space for identifiers and introduces the concept of object pointers for replicas: instead of replicating the whole object on different peers, pointers to

that object are replicated whereas the original object remains on the initial host.

Kademlia

Kademlia [90] uses a tree topology to order the nodes. Its identifiers are 160 bit long and their binary representation defines their position in a virtual binary tree. The distance between two nodes is defined as the XOR composition of their respective identifiers. Each node has a routing table composed by 160 entries (k-buckets), that maintain nodes with distances between 2^i and $2^i + 1$. These k-buckets can store at most k nodes (usually 20) that are arranged in a least recently seen fashion: the most visited nodes are at the end of the list.

The routing information is updated at every operation getting new information from the messages that arrive at the nodes. In this way, it is possible to discover new peers and arrange them using the described policy. Moreover, ping-like messages that probe the network topology are no longer necessary, simplifying the protocol.

A message is forwarded to the nearest peer using the XOR based metric and requires $O(\log(n))$ steps in order to reach its destination.

4.1.2 Support to Range Queries in DHTs

The DHT systems described so far implement only exact matching for discovering peers or resources. Although this is enough for most cases, some applications like distributed databases or processing need more complex queries like range-query or k-nearest-neighbor. Therefore some work has been done in order to enrich DHTs with these types of queries.

Protocols that support complex queries have to take in consideration the following aspects in order to obtain good performance:

- **Locality**: similar data should be memorized in adjacent nodes; in this way, it is simpler to discover all the resources contained in a range

- **Load Balancing**: data access and distribution among nodes should be the most uniform possible
- **Limited Metadata**: to drive the queries. The minimum number of additional metadata should be used; theoretically, no metadata should be needed at all.

The proposed systems use different strategies in order to introduce support for more complex queries:

- Additional software layers over existing DHT implementations
- Enhancing existent protocols better exploiting their characteristics
- Developing completely new protocols that natively support complex queries

In the first approach a preprocessing phase is employed in order to map values and range queries to the available peers. For example, space-filling curves can be used to map multi-dimensional values or ranges into one-dimensional ones. Then, a standard DHT implementation is used to store the values and querying for results. In this way, the DHT protocol is not modified but the performance can suffer due to an increased number of queries (a range can be split into several exact match queries) or a low hit ratio since the queries cannot identify all candidates. This approach has been used for example by SCARP [62]. Other similar solutions are described in [118], in which a distributed quadtree is created on standard Chord nodes, and in [134], that uses a balanced binary segment tree upon OpenDHT [107] nodes and supports cover queries.

In the second approach the protocol is enriched with a support to range queries, exploiting some of its intrinsic characteristics. In this case the protocol is modified only partially because it already creates overlay networks that potentially allow complex queries. Examples of this approach are CAN or Chord: the overlay networks they create can intrinsically support, for example, range queries but these have not been considered in the original versions of both protocols.

Finally, in the third approach completely new protocols are developed with the specific goal to support complex queries from the beginning.

In the following section we will present the most important works following the last two approaches.

4.1.3 Enhancing of Original DHTs

CAN Extension

Extended-CAN [9] is an example of enhancing an existent protocol in order to support range queries. It follows the same idea of the original CAN introducing some limited modifications. For example, data are distributed in the network using a space filling curve that maps them to a d-dimension space. In this way, using the locality propriety of some space filling curve, similar data are stored by adjacent nodes. The range query processing can be accomplished using two different strategies:

- **Controlled Flooding** requests are sent to neighbor nodes that intersect the query: duplicate requests can arrive at the same node.
- **Directed Controlled Flooding** requires that the first node that intersects the query generate two ”request weaves”: the first one toward nodes that intersect the query with greater intervals, the second one toward the remaining nodes that intersect the query with smaller intervals.

A strategy adopted to obtain a good load balance is to randomly perturbate the data identifiers modifying their less significant digits. In this way, they can be distributed more uniformly but the exact match search can become impossible.

Chord Extension

Abdallah and Le [1] describe a protocol that extends Chord allowing range queries. The modifications of the protocol are based on two aspects: the hashing function and the load balancing.

The first modification proposed by the authors is the substitution of a standard hash function with any other function that maps values to nodes preserving their natural order. In this way, values are assigned to subsequent nodes and not scrambled over the network.

Unfortunately, preserving the natural order of values can lead to load imbalances in some nodes when the data distribution is not uniform. To avoid this case, a careful load balance algorithm has been proposed. The idea is that nodes can change the value of their identifiers moving along the ring, in order to take the responsibility of more data and reducing the load.

If a node is overloaded, it can move toward its successor (increasing its identifiers) or its predecessor can move toward it: since identifiers have changed, data should be reassigned to the nodes thus reducing the load. If neither the predecessor nor the successor can accept new data, another unloaded and newly-joined node is found that "jumps" to a place on the ring where it can split the distribution peak. Unloaded nodes map themselves using the DHT infrastructure and thus are found with $O(\log(n))$ messages.

To know if they are unbalanced or not, nodes should know the average load balance of the entire network. To do so, they use epidemic or gossip style aggregation functions that exponentially converge to the right result.

Given the described order-preserving mapping function and load-balancing algorithm, the range query is executed in the following way: first the node maintaining the lower bound of the range is found using the standard exact matching procedure; then, the request is propagated to its successors until the upper bound is reached. In this way, a range request is achieved in $O(\log(n) + m)$ messages where m is the number of nodes contained in the range.

HotRoD

HotRoD [102] is a DHT based framework aiming to introduce support for range queries. It can be imposed on existing DHT implementation with minor changes.

The two key features that it implements are the followings: a locality-preserving hashing function and a replication-based load balancing scheme.

HotRoD can use any existent DHT implementations in order to work but it was developed using Chord. Data are considered to be database tuples of a k-attribute relation $R(A_1, \ldots, A_k)$. To store a tuple in the network at most k order-preserving hash function are used (one for each attribute) plus another one calculated on the primary key of the relation. Thus, each insertion requires $O((k+1) * \log(n))$ messages. The hash functions are not specified but the most suitable ones can be used according to the attribute ranges.

However, this indexing does not reduce the load of data present in the network: a load-balancing scheme is needed. Unlike the majority of the works that use migration to avoid imbalances, HotRoD employs replication. This is due to the fact that migration can reduce the load only in terms of number of resources stored in a singular node, but it does not do anything in term of popularity of a specific resource. Migrating a popular resource to another node does not solve the problem, instead distributing similar requests among different replicas can reduce the average number of requests processed by each node.

To obtain this result HotRoD uses a so called multi rotation hashing that can calculate the identifier of the i-th replica of the given attribute. In practice the hash of replicas is augmented by a value that rotates in specific ranges defined by the maximum number of replicas. In this way, replicas can be stored on different nodes that compose a virtual ring.

Replicas are automatically generated by nodes that at random intervals check if they are too heavily loaded and decide to create new replicas of their data. To do this however, they need to know how many replicas of a specific data item are presented in the network. These values are also indexed and replicated by the network.

The maximum number of replicas is a configuration parameter: high value of this parameter corresponds to more uniform distribution of load but at the cost of a high overhead.

The range search proceeds in the following way: first a randomly selected replica of the lower bound of the range is identified. Then the request proceeds to the highest bound identifying randomly at each step an available replica of the actual value. So the overall complexity of a range search is $O(\log(n)) + m)$ where m are the number of values contained in the range.

Prefix Hash Trees

Prefix Hash Trees [105] try to store data that share a common prefix on the same node using a standard DHT implementation. The idea behind this protocol is that a data item should be stored on the node whose id match the hash value computed on its prefix. In this way, all data that share a common prefix are stored on the same node and can be retrieved directly. When the number of data that share a common prefix on a node reach a configuration limit, they are moved to different hosts that are responsible for a longer prefix. The old host maintains information about the moved data allowing range queries to retrieve them.

4.1.4 DHTs with Native Support to Range Queries

GosSkip

GosSkip [64] is a general framework that creates a p2p structured network allowing range searches and probably other types of complex queries.

With respect to other systems, in GosSkip node identifiers are not restricted to bit sequences but can be every kind of data upon which it is possible to define a deterministic total ordering. For example, identifiers can be represented by strings or topic filters allowing GosSkip to be used as the basis of a highly scalable publish/subscribe system.

Each node has a skip list that is used to store nodes at increasing distance preceding or following the actual node according to the defined order. The respective distance of nodes defines its level: direct neighbors are at level 0, nodes a distance 2 are at level 1 and so on. A message sent to a node of level i becomes

a message of level i. To maintain these skip lists piggyback messages are used: every message has a variable number of additional entries that store information about known nodes at the message level. These entries are stored in the various nodes that the messages reach and then attached to the following ones enriched with the actual node information. Information contained in the entries is used to update the skip list, an approach similar to Kademlia.

Once the network is set up, range queries are forwarded to the first known node that is inside the given range. Then they are forwarded to other higher and lower level nodes that are inside the same range.

SCRAP MURK

SCARP and MURK are two p2p protocols described in [62] that allow the execution of range queries. They are based on different architectures and provide different performance.

SCARP uses a two step insertion operation in order to identify the node where to store a given resource. In the first step a space-filling curve is used to map multi-dimensional values to one-dimensional ones. In the second phase these values are stored to nodes in a way that preserve their order. In this way however, the value distribution between nodes can become unbalanced, so in the case of overloaded nodes a migration policy is used in order to split data across neighbors.

Range queries in SCARP are executed in two phases too. In the first one from a multi-dimensional range query a series of one-dimensional ranges are obtained according to the specified space-filling curve. In the second phase matching nodes are queried in order to get the values.

MURK uses an approach similar to CAN but with two great differences:

- In CAN, when a node joins the network the virtual space is equally divided, in MURK the load is equally divided so that the ”old” and the ”new” node have the same number of resources.

- CAN uses hash functions to map nodes whereas MURK uses a space-filling curve.

This structure does not guarantee load-balancing during the evolution of the network so the same load-balancing scheme of SCARP is used.

The message routing depends on the MURK network topology chosen. In fact, three possible topologies can be used:

- The standard CAN topology
- The standard CAN topology enhanced with $2 * \log(n)$ skip pointers for each node chosen at random
- The standard CAN topology enhanced with $2 * \log(n)$ skip pointers chosen using the distance between their partition center as metric

To verify the performance of range queries the authors have conducted two types of test to study locality, i.e. the average number of nodes across which the answer to a given query is stored, and the routing costs, i.e. the average number of messages exchanged between nodes in order to route a given query. These tests have been repeated varying the number of dimensions, the network size and the selectivity of the query. According to obtained results it seems that MURK outperforms SCARP especially when enhanced topology is used. This is probably due to the fact that a range query can be mapped to a great number of unidimensional ranges thus requiring many requests.

NRTree

NRTree [83] defines a super peer based network that is arranged in a way allowing the execution of range queries. The n-dimensional space is subdivided into hypercubes each of them managed by a specific super peer. Super peers are arranged in a way similar to a n-dimensional CAN, thus requests are forwarded to the super peers responsible for the hypercubes that intersect them. Each super peer is responsible for a variable number of peers that help it to store the

resources. To index the resources, each peer and super peer uses a distributed R-Tree [65], a spatial data structure that can map hypercubes and hyperpoints to resources in an efficient way. The super peer and the peers owns only a part of the entire R-Tree of the region they are responsible for: the super peer maintains the top of the tree whereas the other peers manage a subtree according to their position in the n-dimensional space.

Since R-Trees support range search, such requests are forwarded to the super peers that then forward them to the adequate peers according to their identifiers. If the number of peers is high and that of super peers low the efficiency of the range search is high since it can proceed through the R-Trees.

There are two problems that this protocol has to address: the load balancing between nodes inside the same region and the election of the super peer in case of failures.

4.2 Tree-based P2P structures

Tree-based structures are another example of structured p2p networks that natively support range queries. In contrast to DHTs they create tree-like overlay networks that can be searched in order to find the desired resources. Tree-based structures have been developed for better supporting distributed databases and their query processing that heavily uses range queries. Moreover, they appeared some years after the first DHT implementations because a tree-based overlay is more complex to maintain due to its intrinsic fault-proneness and load-imbalance. In fact, the closer a node is to the root the more important it is and can be overloaded by queries. Moreover, if a node fails it is impossible to reach its children. To avoid this type of problems the tree-based structures should carefully implements load-balancing strategies and create redundant paths to connect nodes and avoid network splits. The most important tree-based systems are described in the following sections.

BATON

BATON [74] was probably the first p2p systems that employed a tree overlay network. There were some previous proposals but they did not resolve all the performance and fault-tolerance issues of a such structure.

BATON organizes the peers in a balanced binary tree structure according to their identifier. Every node maintains a range of values and a set of links to other nodes with the associated range. To avoid network splits redundant paths should be created and for this reason every node maintains pointers to its children, its parent, its left and right siblings at exponential increasing distance and its left and right adjacent nodes, i.e. the nodes that are the predecessor and the successor of the current node in an in-order traversal of the tree.

A new peer joins the network contacting a participant and then this operation proceeds in two phases: in the first one the right parent of the new peer is found using the adjacent nodes, in the second one siblings are contacted to update their routing tables. All this process requires $O(\log(n))$ messages.

Soft leaves and node failures are handled using in part the same algorithm. When a node wants to gracefully leave the network, it is responsible to identify another node that can take care of its children. To do this it uses siblings and/or adjacent nodes. After having successfully found a new adequate node it can disconnect from the network. A node failure is handled in a similar way: when a node discovers that it is unreachable it contacts its parent that from now on is responsible to contact other nodes to update their routing table. These operations also require $O(\log(n))$ messages.

To maintain the balanced structure after joins or leaves, network restructuring is needed. It is done using node rotations in a way similar to that employed by AVL trees requiring additional $O(\log(n))$ messages.

Queries are forwarded through the nodes using the links and the associated ranges. In the exact matching query the search stops when it arrives to the peer storing the range that contains the value requiring $O(\log(n))$ messages. In the range query after having found the first node intersecting with the range, the

search proceeds through the children and or siblings requiring $O(\log(n) + m)$ messages when m nodes intersect the ranges.

The insertion of new data into the overlay network is accomplished in three steps: first an exact matching query is done in order to find the appropriate node, then the data is inserted and finally the neighbors are updated when the range of the node changes.

Beside the balancing of the structure, this system requires a balancing of the data too. It is based on migration of part of the range to other nodes in the neighborhood. To avoid too many message exchanges internal nodes balance their load with adjacent nodes whereas leaf nodes can leave and rejoin the network becoming the children of a less loaded node. This operation can require a network restructuring.

BATON*

BATON* [73] is an enhancement of BATON that uses a generic tree based overlay instead of a binary tree. In BATON* each node has m children and the left and right routing tables are based on skip lists storing sibling nodes at exponentially increasing distance on the same level. An old node can only accept a new node if it has full routing tables and does not have m children. Otherwise, it has to forward the request. A node can leave the overlay network only if the latter does not become unbalanced, otherwise it should find a replacement. The overall cost for the insertion or deletion of a node is $O(m \log_m(n))$, whereas the cost for an equality search is $O(\log_m(n))$.

BATON* supports natively only one-dimensional queries, for supporting multi-dimensional queries different indexes are employed. In particular the indexes are of two types: indexes for singular attributes that are the most used or indexes for a group of attributes that are less used. To map attributes on a single index, an Hilbert Space Filling Curve is used. When a multi-dimensional query is processed it is split in different subqueries according to the indexes that are then

executed in parallel. At the end an intersection of the results is done. Heuristics are employed to improve performance in query processing.

VBI-tree

The Virtual Binary Index Tree (VBI-Tree) [75] is an abstract data structure build on top of an overlay framework. It was inspired by the BATON structure and can support any kind of hierarchical tree indexing structures. Every node can be an internal or a data node and each peer store an internal and a data node (its left adjacent in the tree traversal).

Internal nodes maintain links to its children, parent left and right siblings and left and right adjacent nodes. Moreover, they keep track of the regions covered by each of their ancestors and the heights of the subtrees rooted in them. Data nodes do not contain routing information since their identity is implicitly established by the algorithm.

The join operation is similar to BATON and it is composed by two phases: the first one identifies the correct peer to join with using the BATON algorithm whereas the second one splits the corresponding data node in an internal node and other two data nodes. The new internal node and its left child will be managed by the new peer whereas the right child will be managed by another adequate node according to the actual traversal order. After these steps the ancestors are updated if required. The total number of messages required for this operation is $O(\log(n))$. When a peer wants to leave, it should find a replacement for itself in order to avoid tree imbalance. This operation also requires $O(\log(n))$ messages. The fault tolerance and load balancing support is the same used in BATON.

Given the previously described overlay network, it is possible to define a multi-dimensional index upon it in a way similar to other tree based indexing structures like R-Trees. Every internal node maintains a region that contains those of its children. In order to reduce the number of updates in the children nodes two heuristics are employed:

- Updates are propagated only when the network is not busy

- Internal nodes can maintain so called ”discrete data” (i.e. data that do not fall in any of the children regions) that avoid children regions to be enlarged. Regions are modified and updates are propagated only when the number of ”discrete data items” exceeds a specific threshold.

The exact and range queries proceed through the ancestor, descendant and neighbors. Heuristics are used in order to avoid that the queries proceed too far up in the ascendent line and to take into account ”discrete data”.

LOT

LOT [5] is a tree-based overlay network connecting peers arranged in clusters. It has been designed to exploit the topology of today's large-scale networks, composed by thousand of peers grouped in a small number of highly-connected clusters, deployed in very distant geographical locations in the world.

Every peer maintains an internal state that is then aggregated to obtain the final overall result. This state can be for example the result of a partial computation or a SQL query. To reach good performance, the cluster based topology and a virtual tree structure are exploited. In fact, the peers inside the same cluster represent a super leaf that is a part of the virtual tree connecting the various clusters. Inside the super leaf, communication is accomplished using gossip-style messages because this scheme allows fast data interchange between all peers.

Every super leaf uses a subset of their peers to emulate its own ancestors in order to be fault tolerant. These peers can emulate the state of one or more ancestors depending on the number of the members of the cluster. Since these emulation peers are present in every cluster, it is possible to deal with faults and avoid network split. However, to maintain a correct state, the emulation nodes should be able to deterministically calculate it from its children and to continuously execute a maintenance algorithm synchronized with the other clusters. This maintenance algorithm proceeds in rounds from the bottom to the top in order to obtain the correct states at every level of the tree.

To join a new node contacts another one already part of the LOT network. This replies with a list of emulation nodes that can be the ancestors of the new node. The new node selects the nearest of them and then iterates this process until the best node is found. This process however can create unbalanced trees, for this reason when a node has too many children it splits itself and notifies this modification during the maintenance algorithm so that other nodes can know it.

Chapter 5

A New Approach for Tuple Spaces Implementation

5.1 Introduction

In this chapter we will discuss an approach that improves the performance of tuple spaces. In fact, many widespread tuple space implementations suffer from low performance when a high number of tuples is stored into the space. This is mainly caused by inefficient implementations of its operations: in many cases simple lists or arrays are used to index tuples. Although this approach can be satisfied when a low number of tuples is stored, it becomes a bottleneck when the tuple number increases. Therefore, more efficient approaches to index tuples should be used. However, these indexes should support the particular nature of tuples that can show very different structures: they can be arrays or objects, like in JavaSpaces, and have different types and size. These differences are difficult to handle and lists represent the simplest solution.

Notwithstanding these complex requirements, we think that it is possible to use more efficient indexing schemes to increase the performance of tuple space implementations. For this reason, we have developed an approach that is composed by two phases. In the first one, tuples or templates are mapped to a n-dimensional space applying hash functions to every of their fields or attributes.

In the second phase, the produced mapping is used to operate on a data structure that indexes the original tuples and allows fast insertion, removal and lookup.

These two phases are loosely coupled and can be developed separately until they respect some requirements. In particular, the hashing functions should produce keys that are selective enough to avoid that too many values are mapped with the same key. On the other side the indexing data structure should efficiently support insertion, removal and range queries, because these last one are needed to implement take and write operations.

Indexes that support the previous requirements are not so common but good candidates can be found in the area of spatial databases as will be discussed in Section 5.2. The use of such data structures allow the creation of highly efficient centralized spaces as will be shown by our tests.

As described in Chapter 4, in the research literature are also present some examples of distributed indexes, that support exactly the requirements we have defined. Thus, using them it is possible to implement p2p tuple spaces, as will be described in Section 5.2.

Thus, this approach show its versatility allowing the development of different flavours of tuple spaces following the same scheme. Although we have implemented this approach in our Grid-based service, it is completely general and can be developed using other technologies.

5.2 Centralized Tuple Space implementation using Spatial Indexes

As described previously our approach is composed by two phases. In the first one, that can be called ”mapping” phase, tuples or templates are mapped to a n-dimensional space, where n is the number of fields or attributes.

The Cartesian space used can be subset of $\mathbb{R}^n$ or $\mathbb{N}^n$, according to the way in which tuple field values are hashed. In our implementation, we have employed

the space $[0, 1] \subset \mathbb{R}^n$ for approximation reasons. The mapping we have used presents some differences between tuples and templates.

For tuples, we use an hash function for every field. It produces keys in the interval $[0, 1]$ and it is chosen according to the field type.

For templates we create hypercubes defined by two n-dimensional points: the lower left and upper right point. Every empty field produces two possible value 0 or 1 whereas non empty field are hashed like in tuples. Thus, every tuple contained in the produced hypercube satisfies the template.

After having mapped tuples to a n-dimensional space, we can use spatial indexes to efficiently handle them. Spatial indexes have been developed for spatial databases and Geographical Information Systems (GIS) and support efficiently the requirement previously defined.

The mapping between tuples and n-dimensional points is not new and it has been already proposed by Comet and in [94]. Nevertheless, such approaches do not use spatial indexes.

5.2.1 Overview of Spatial Indexes

There are many examples of multi-dimensional space indexes that can be used to implement our space. According to their structure and characteristics, they can be grouped in the following way:

- **Tree-based** that use tree-like structures to index points and can be organized in two other subgroups:
  - **Point-Oriented** that can index only n-dimensional points
  - **Shape-Oriented** that can index n-dimensional shapes too, like hyper-rectangles.
- **Partition Strategies** that try to calculate once a key from the spatial position and index it using traditional data structures like B-Trees.

The first spatial index structures that have been proposed are the point-oriented ones.

KDTrees [26] index hyper-points using an unbalanced binary tree subdividing recursively the space in two subspaces, according to the point position. At each tree level the split axis is rotated in order to have a more uniform subdivision of the space.

QuadTrees [54] recursively split a n-dimensional space in $2n$ regular hyper-cubes independently from the point position. The point will be stored by the nearest node.

These data structures have two disadvantages: they are not balanced and can index points only. More recently shape-oriented tree-based strategies have been proposed that, with respect to the previous ones, use general balanced trees and can also index shapes. The most important of these structures is the R-Tree [65]. It indexes shapes using a B-tree like structure in which keys are hyper-rectangles that completely contain the corresponding subtree. Insertion, removal and queries are done comparing geometrically the intersection of the keys.

When a node become full, a split is needed to maintain the tree balanced and optimize future queries. Different strategies have been employed for this operation leading to various flavors of R-Tree. The original R-Tree implementation tries to find two partitions of the children that cover the minimum volume. The R*Tree [22] enhances this heuristics taking into account the perimeter too. The X-Tree [28] uses the R*Tree strategy but allows nodes to be not split if the best partition is not good enough according to a configuration parameter. This strategy was developed in order to better support high-dimensional data that were handled not suitably by the R*Tree. Another important R-Tree like scheme is the M-Tree[45], that is designed to be a generic tree-based indexing scheme that reduces the number of geometrical comparisons needed for its operations, thus improving the overall performance. One of the most relevant disadvantages of these methods is the computational cost needed to visit the tree. In fact, to select the correct subtree it is necessary to calculate the intersection between the query

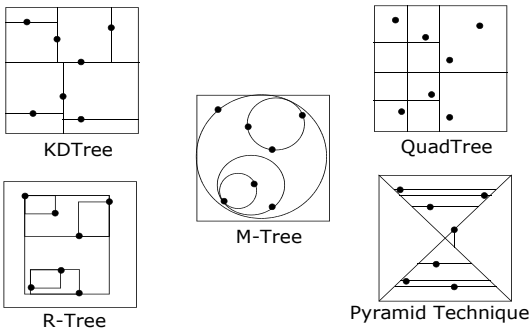

Figure 5.1: *The different spatial partitions defined by some of the spatial indexes described in this thesis.*

and each key. This can be very expensive especially for high-dimensional data. Moreover, the heuristics employed by split operations sometimes can fail in finding the optimal split. To overcome these problems, space partitioning strategies have been proposed. Basically these strategies try to compute a unique key once for each element and use it to index the point. Since keys are integer or floating point values, geometrical comparisons are not more needed and the computational cost is lower. Moreover, they can be indexed using more common data structures like traditional B-Trees.

The most important partitioning schemes are the Pyramid Technique [27], the Γ-partitioning and the Θ-partitioning [97]. The Pyramid Technique subdivides the space in $2n$ pyramids. So each point can fall only in one pyramid and its distance from the basis of the pyramid determine its key. So using the pair `<pyramid number, point height>` it is possible to index the points using a B-Tree. The problem of this approach is that the partitioning is not injective and for some highly-skewed data the performance can be low as stated in [85]. However, in normal cases this approach can have very good performance. Another partitioning scheme that overcome these limitations is the Γ-partitioning. It recursively subdivides the space into m Γ shaped subspaces each containing two regions. Every n-dimensional point is completely identified by the number of the region that contains it and its distance form the region base. Using this key it can

be stored by a normal B-Tree. The number of regions used is a parameter defined manually and does not depend on the number of dimensions. The Θ-patitioning is similar with the difference that the regions are hypercubes or hyperrectangles contained in each other.

A disadvantages of these space partitioning schemes is that they can hardly index shapes.

5.2.2 Experimental Results

Due to the great number of different spatial indexes available, we have decided to implement our centralized prototype using two different types of spatial indexes: the X-Tree, because it is optimized for high-dimensional data, and the Pyramid Technique. In this way, we are able to compare the behavior and the characteristics of these two types of indexes and to choose the most suitable for our purpose. For example the X-Tree, along with all the other shape-oriented indexes, can be used to implement efficient notification systems for tuple spaces too. In fact, they can be used to retrieve a tuple given a template, as well as to retrieve a template given a tuple. If a template represents a subscription, this feature can be used for identifying efficiently the destination of the notification.

Both indexes have been implemented in Java from scratch since we can only find C or C++ based implementations or Java implementations not adequate for our purposes. This means that their code is not well optimized and, with some tuning, these implementations can be more efficient.

To better understand the behavior of our prototypes, we have compared them with the most important tuple space systems available today. We have conducted the following test: inserting an increasing number of tuples into the space and calculating the average time required to write and take them. The test has been conducted on a 3GHz Core Duo PC with 1GByte of RAM. The results are reported in Figure 5.2 and 5.3. To better show the differences between the various implementations, we have used a logarithmic scale.

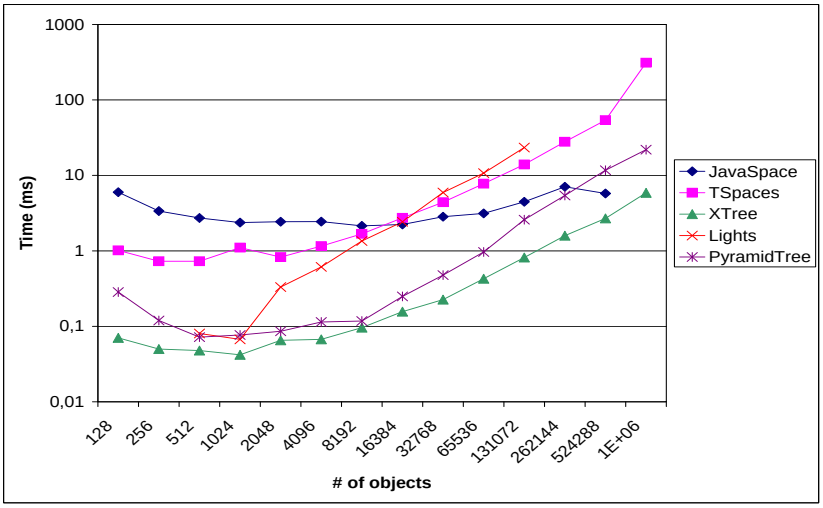

Figure 5.2*: Average time spent for take operations*

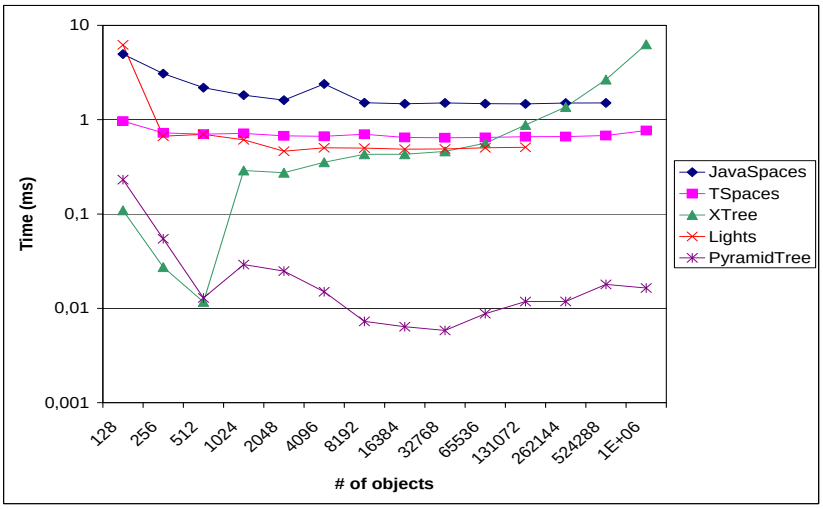

Figure 5.3*: Average time spent for write operations*

Unfortunately, we could not collect the same amount of data for every implementation because some of them were unable to handle huge spaces and crashed. For example JavaSpaces could not finish the test with 2^{20} objects even if the JVM can use up to 2GB of heap memory [1].

Nevertheless, the collected data show an interesting trend of our two implementations: both of them perform very well resulting often as the fastest with respect to all the other implementations. In fact, the X-Tree based implementation is the fastest of all in taking tuples from the space independently from the size whereas the pyramid based implementation is the second best and overcomes the JavaSpaces implementation for spaces with a size less than 2^{19}.

The X-Tree strategy performs better in take operations rather than in write ones: this is due to the higher number of operations needed to find the correct position inside the tree. In fact, it has to check if hyperrectangles intersect each other, whereas the Pyramid technique uses B-Trees that needs only integer comparisons, and the other implementations use lists or arrays that need a constant number of operation to insert a data item.

Taking in consideration both take and write time, the pyramid based implementation is the best for medium/large spaces whereas the X-Tree based one performs better for very large spaces.

5.3 Distributed Tuple Space Implementation using Structured P2P Network

In this section we will describe how to use our spatial-based approach to develop distributed tuple spaces upon structured p2p networks. Exploiting the following technique, we aim to make distributed tuple spaces more efficient and fault-tolerant.

[1] It was useless employing more memory because the physical RAM of the computers was 1GB and thus we were already using part of the swap partition, slowing down the computation and probably obtaining not comparable results

Although some p2p tuple space implementations are already present in the literature, none of them seem to use directly p2p structured networks as communication layer but they prefer to use unstructured network approaches. This is probably due to the fact that the structured p2p network should support range queries in order to implement the take operation, but this is an uncommon feature. In fact, the implementation of an efficient range query support for structured p2p network is an open research topic as described in the previous chapter.

To develop our p2p tuple space we have followed the same approach described in Section 5.2, changing the data structure used for storing tuples in the second phase. In fact, it has been substituted with a p2p network implementation that stores the tuples and supports range-queries. Generally speaking, each peer in the network is responsible for a non overlapping part of the space and the tuples it contains. Thus, tuple space operations are implemented using those defined by the p2p network, for example take and read operations are based on range-queries.

The way in which tuples are mapped to a n-dimensional space is the same used for centralized tuple spaces.

Similarly to the previous prototype, also in this case there are many suitable p2p systems. For this reason we have chosen two different network structures: a CAN-based one and a tree-based one.

5.3.1 CAN-based Implementation

The first p2p protocol we have taken into account was CAN, because it is relatively simple to implement and is based on the subdivision of the space into a set of hypercubes.

Following our original idea, tuples are stored by the peers responsible for the space partitions in which their n-dimensional mappings fail. Probably the only problem is how to define the take operation. In fact, although CAN uses hypercubes to index resources, it was original designed only for exact-matching queries but soon some proposals have been made to support range queries too.

The most of them are based on neighbors flooding: the query is forwarded to all neighbors whose hypercubes intersect the range. For our implementation we have chosen a simple flooding based strategy without the use of additional links between nodes.

To support different tuple lengths it is necessary to maintain several instances of the protocol at the same time: one for every dimension. This is computational expensive but the protocol does not allow Cartesian spaces with a different number of dimensions to be used at the same time.

Although designed to cover a Cartesian space, this protocol shows lower performance compared with others. In fact, it needs $O(\sqrt[d]{n})$ messages for an exact-matching query and the nodes routing table is not bounded.

5.3.2 Tree-based Implementation

To overcome the disadvantages of the CAN-based prototype we have tested another p2p protocol that uses a VBI tree. It has been designed to handle multidimensional data using an approach similar to standard spatial trees. It creates and maintains a binary tree, whose nodes contain ranges of data. It has been designed to be general, i.e. the protocol handles only the general binary structure whereas the mapping to it can vary and depends on the index used. Thus, this protocol is based on two layers: a lower one that maintains the binary tree and does not change and an upper one that depends on the index used. This is a very useful feature because it allows to create different indexes, implementing only a small module. Moreover, the mapping is not so complex because the different spatial indexes arrange data in similar ways.

This protocol is more efficient with respect to the previous one because it needs $O(\log(n))$ messages for a join operation or an exact-matching query. Moreover, it does not need to maintain different network instances for every dimension because the tree structure will always be the same. It is sufficient that nodes maintain a tuple storage for every dimension and the corresponding ranges.

Unfortunately, maintaining the tree balanced and correctly updating the tables needs more messages than CAN and this number can be very high in case of highly skewed data distributions.

5.3.3 Experimental Results

To compare these two possible protocols, we have implemented them using the Peersim [136] simulator, a centralized p2p system simulator written in Java allowing rapid development of p2p protocols. Using it we have conducted two different types of tests. The first one measures the efficiency of the two protocols in the case of write or exact-matching take operations: it generates a random topology and measures the diameter of the overlay network created by the protocols. We have executed it increasing the size of the network and repeating it 1000 times for each network size. The results of this test are shown in Figure 5.4.

The second test measures the efficiency of take operations: it generates a random topology and executes 1000 random queries calculating the number of visited nodes. This test was also repeated 1000 times for each network size and its results are shown in Figure 5.5. Since in CAN the number of nodes intersecting with the queries depends on the dimension of the space used, to better compare the two approaches we have used an index equal to the ratio of visited nodes to intersecting nodes. Values nearer to 1 mean fewer messages wasted to accomplish the queries.

As shown by the graphs the VBI based protocol outperforms the CAN based one in both write and take performance.

5.4 Conclusion

In the previous sections we have proposed a generic approach to tuple space implementation, able to enhance their performance and to reduce the disadvantages

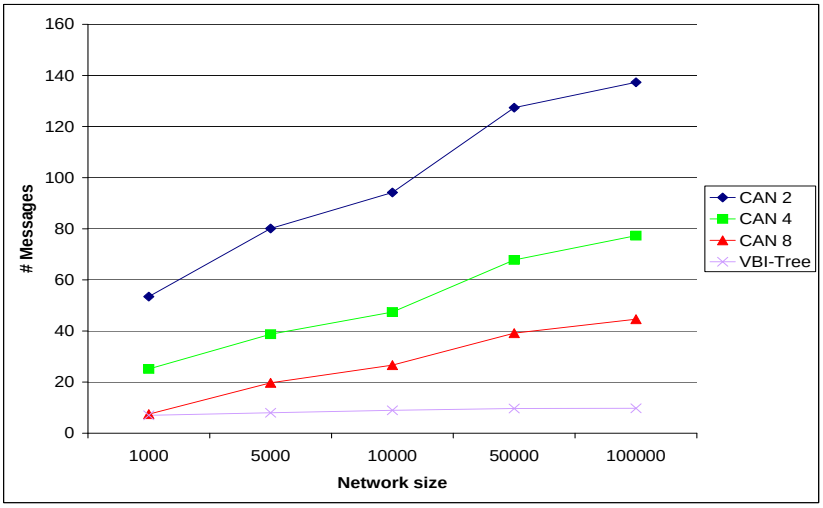

Figure 5.4: *Average of the maximum number of messages needed for write operations increasing the size of the network and the number of dimensions*

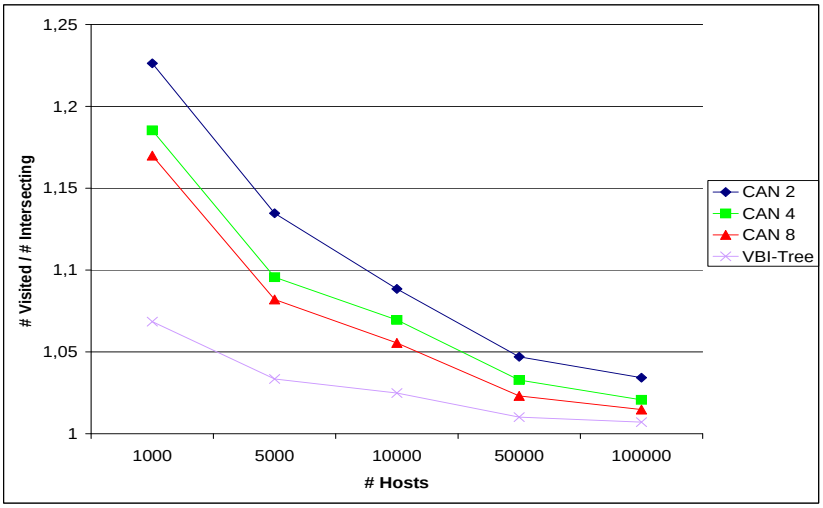

Figure 5.5: *Average of the ratio of visited nodes to intersecting nodes for take operations increasing the size of the network and the number of dimensions*

that this kind of systems can show in particular circumstances. Although this approach has been originally developed in order to better adapt tuple space systems to Grid infrastructures, it is general and can be used by other technology too.

Our proposal can be considered a novelty because, in the field of tuple space implementations, there are no other works that use the same techniques.

The approach we propose is composed by two phases: a first one that maps tuples and templates to a n-dimensional space and a second one in which the tuples are stored by an efficient spatial index. In this way, we can use the data structures and algorithms developed for spatial databases to index tuples. Since these last techniques seem to be very efficient, we can profit by them to enhance the tuple space performance. This is evident from the tests we have conducted comparing prototypes developed using our approach with other publicly available tuple space implementations. They have proved to be the best for medium-large spaces and at least the second best after JavaSpaces for larger spaces. Due to these good results, we have decided to implement both strategies in our Grid-based service, allowing applications to choose the most suitable storage.

Moreover, our approach is very versatile. In fact, it can be employed not only for centralized tuple spaces, but also to develop distributed ones. It is sufficient to substitute the spatial index in the second phase with a p2p network and the space can become distributed. Clearly the p2p network used should support the same requirements of spatial indexes, in particular it should be able to handle range-queries. Unfortunately, only a few p2p systems are adequate for this purpose and we have employed two of them in our prototype: CAN and VBI-tree. CAN-based spaces are simpler to implement because they represent a ”natural” mapping but have poor performance. VBI-based spaces have better performance but are more difficult to implement due to their tree structure that is more complex to be maintained.

Although we have obtained good results with our p2p based tuple space implementation, for the moment we have decided not to include it in our Grid service due to the small network we can use as a testbed. It is too small to take real

advantages from the p2p approach. Nevertheless the highly modularized architecture we have designed for our service allows its inclusion at a later time when more resources become available.

Chapter 6

Grinda

6.1 Introduction

After the definition and implementation of a generic tuple space infrastructure in the previous chapter, in this one we will show how this tuple space implementation can be used to implement a Grid service. We have named this tuple space system Grinda (Grid + Linda). It is designed to be deployed on the Globus Toolkit and to serve as coordination framework for applications deployed on the same infrastructure. Grinda is not intended to be a workflow engine or a batch system, although it can be also used in this way, it is a coordination service and thus applications should be coded using its API and should not use domain specific languages like BPEL4WS [141]. Nevertheless, applications can exploit interesting features like independence from network topology and the possibility to choose between different tuple space implementations, adapting the framework to the application requirements.

Since Grid environments have particular requirements we have introduced new features in our system especially in the abstract model and in the implementation of the distributed and local tuple spaces. In fact, a major characteristic of our implementation is the efficient indexing of tuples.

The sections of this chapter are organized as follows: Section 6.2 describes the tuple space model we have used comparing it with the most common ones. In

Section 6.3 we describe the overall architecture of our system whereas in Section 6.4 we explain some of the major implementation details we have employed.

6.2 Model

The tuple space model we have employed in our system is inspired by JavaSpaces. In Grinda tuples are no longer ordered arrays like in the original Linda system or in TSpaces, but every class type is potentially a tuple.

We have decided to use a JavaSpaces-like approach for two reasons:

- it simplifies application development because it does not need additional code to map data to tuples. Moreover, tuples in Linda often require additional labels or values to be correctly taken from the space. This is fault-prone since an error in setting these objects could led to malfunctions.
- it maps better into XML messages used by the Globus Toolkit for the communication and requires less effort for the serialization.

To support this model we have redefined the standard associative matching in the following way:

> A tuple t of type τ matches another one t' of type τ' if $\tau = \tau'$ or τ is a super type of τ' and every fields it contains is null or matches the corresponding one in t'.

This new definition of the associative matching is more powerful with respect to the original one. In fact, it is able to support objects and inheritance too. Every object can be used as tuple and it does not impose the use of specific interface like JavaSpaces does. This avoids the proliferation of wrapper types used only for the communication and allows every legacy types to be used directly.

Templates are defined using null values: a null attribute matches every corresponding attribute value. This certainly prevents the use of null value in tuples but simplify their definition since no special value has to be used in fields.

Moreover, this definition takes into account class inheritance simplifying the development of applications. In fact, it is possible, for example, to define a generic task type and other more specific task subtypes that inherit from the generic one implementing more specific behaviors (a similar example has been coded in Grinda). In this way, the system promotes modularization because only specific behaviors should be coded, whereas generic one can be reused where possible. This is a great improvement with respect to original tuple space systems because without this possibility a modification in the tuple structure leads to a modification of every take operation in the application. In Grinda this is not more the case.

The operations implemented follow the naming convention of JavaSpaces: `out` becomes `write`, `in` `take` and `rd` `read`. Moreover, we have implemented the following "collective" operations:

- `writeAll` that atomically writes an array of tuples into the space
- `takeAll` and `readAll` that take and read all tuples matching a given template respectively. If no matching tuple is present into the space the operations wait until at least a matching one is inserted.
- `takeAllp` and `readAllp` the not blocking versions of the previous operations.

In table 6.1 is reported an example code of Grinda operations usage. It has to be noted that to correctly support this model, the system should be able to automatically serialize every type in the corresponding XML representation. Details on how this is achieved are described in Section 6.4.

6.3 Architecture

The architecture of the Grinda framework is based on the WSRF specification and is composed of two main modules: a client-side and a server-side one. The first

```
class SimpleTuple {
  public SimpleTuple(Integer field1, Double field2) {
    this.f1=field1;
    this.f2=field2;
  }

  public Integer f1;
  public Double f2;
}

class DerivedTuple extends SimpleTuple {
  public SimpleTuple(Integer field1, Double field2,
              Integer field3) {
    super(field1,field2);
    this.f3=field3;
  }

  public Integer f3;
}

static void main() {
  TupleSpace ts=new TupleSpace("test")
  DerivedTuple t=new DerivedTuple(1,3.5,4);
  ts.write(t);
  //a template matching every simple
  //or derived tuple with attribute f1 == 1
  SimpleTuple st=new SimpleTuple(1,null);
  SimpleTuple r=ts.take(st); //r == t
}
```

Table 6.1: *Example code of Grinda usage in Java*

one is used to hide the serialization and deserialization of data and to implement the synchronous behavior of some operations. The second module represents the service that has to be deployed in the Globus Toolkit or a compatible application server like Tomcat.

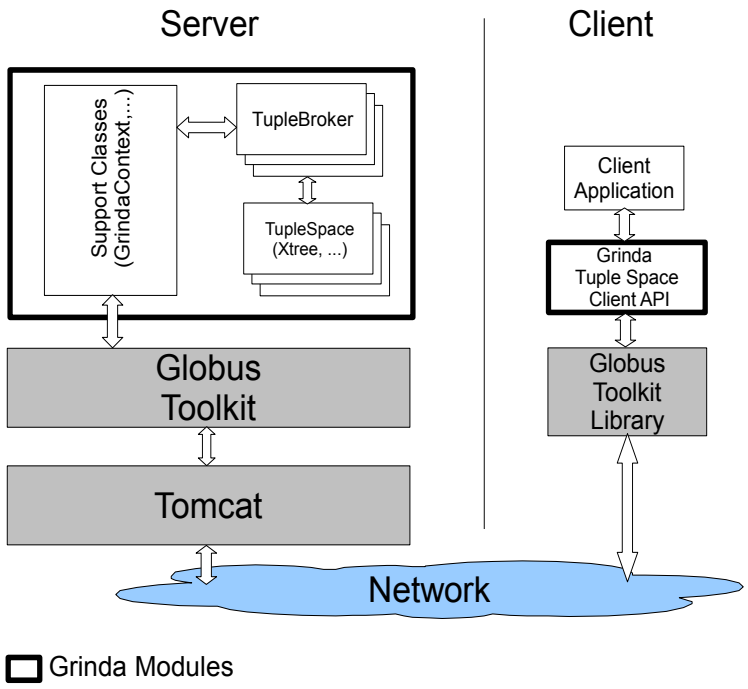

Figure 6.1: *Diagram representing the general architecture of Grinda modules*

6.3.1 The Client-side Module

As described before, the main purpose of this module is to hide the details of the communication with the server, in order to simplify the development of applications based on Grinda. It contains three categories of objects: the objects used directly by the applications, those involved in the serialization and deserialization of the data and finally those employed to mimic the synchronous operations.

The most important class in the client-side module is the `TupleSpace` one, that creates the connection with the server and transmits the operation requests

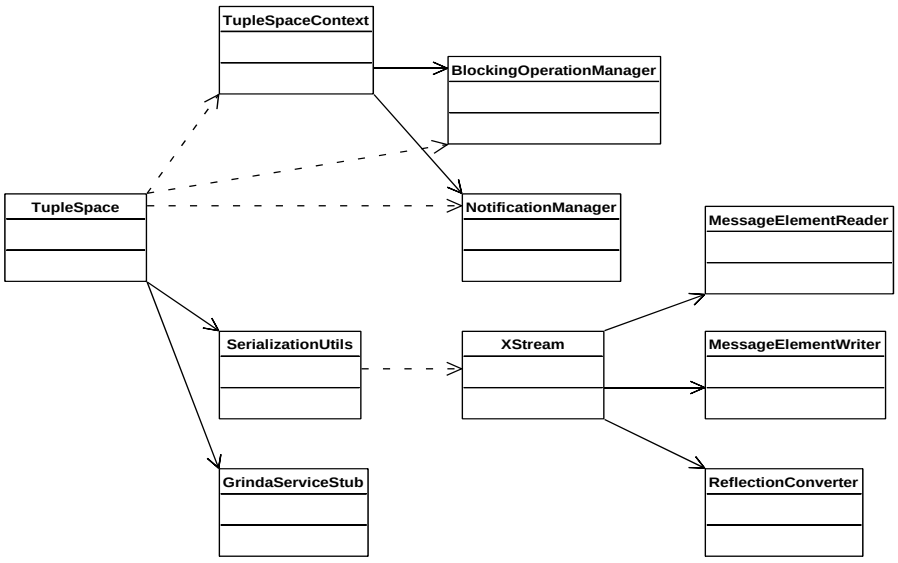

Figure 6.2*: UML class diagram representing the architecture of the client-side module*

through the `GrindaServiceStub`. It accomplishes its task using `TupleSpaceContext` that maintains the configuration and initializes the system at the startup managing the threads needed for the blocking operations and the notifications (`BlockingOperationManager` and `NotificationManager` respectively). `SerializationUtils` is responsible for marshaling and unmarshaling custom data types using the XStream library [155] as described in Section 6.4.2.

Another important task of `TupleSpaceContext` is to set up the XStream library with some of our classes: `MessageElementReader` and `MessageElementWriter` read and write objects produced by the `GrindaServiceStub` for generic type encoding whereas `ReflectionConverter` encodes and decodes custom data types following our rules.

6.3.2 The Server-side Module

The server-side module contains the logic responsible to store the tuples and to implements the tuple space operations..

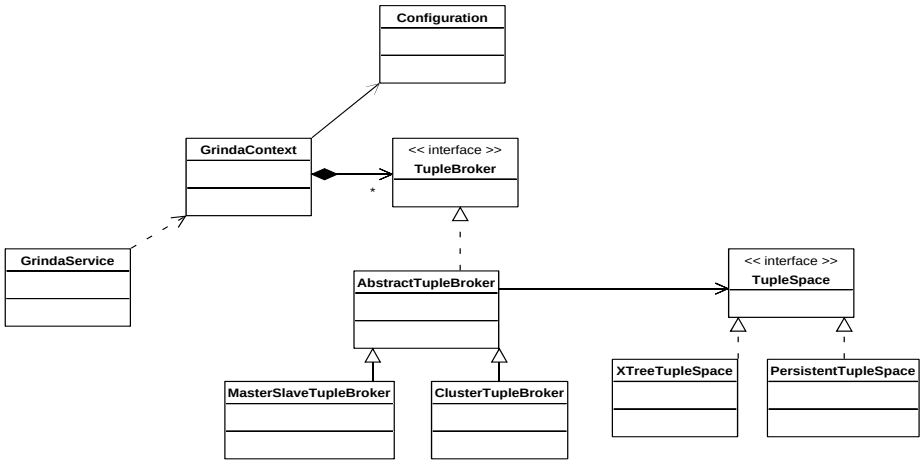

Figure 6.3*: UML class diagram representing the architecture of the server-side module*

The main class used by the server-side module is `GrindaContext`; it is implemented as a `ResourceHome` and for this reason its state survives to the web service calls. It stores the various `TupleBroker` instances active in the system along with configurations and a thread for the asynchronous sending of messages. This last one is used for communications with clients or other services in the case of distributed spaces.

The `TupleBroker` interfaces and its subclasses represent the business logic of the spaces active in the service. They contain all information needed to execute operations on both centralized and distributed spaces. In this way, the tuple space implementation is more modular and new implementations can be deployed into the server without the need to modify the service. In fact, the desired `TupleBroker` implementation is defined by a server-side configuration file and it is loaded when the service container starts up.

There is a hierarchy of `TupleBroker`: `AbstractTupleBroker` is an abstract class used for simplifying the development of new brokers, `MasterSlaveTupleBroker` and `ClusteredTupleBroker` are a centralized and a dis-

tributed tuple space implementation. The latter can be used to create spaces whose tuples are shared between different services that constitute a highly connected component. In this case, a spatial indexing strategy is used to identify the service (or services) that can contain a matching tuple.

Using this approach, it would be possible in the future to implement the VBI-tree strategy discussed in the previous chapter.

`TupleBrokers` uses various other classes to accomplish its tasks. Apart the `GrindaContext` they can use two different types of tuple space: transient and persistent one. Transient tuple spaces are based on spatial indexing of tuples whereas persistent ones use XML databases. Another tuple storage is the `Register` that maintains pending request of blocking operations.

Finally the `GrindaService` class wraps the service interface and calls methods of `GrindaContext` to obtain `TupleBrokers` and executes their operations.

As can be guessed from this simple architectural description, the server-side module can also be though of as a factory that creates and maintains new spaces. According to this view, it could be implemented following the factory design pattern of the WSRF framework. In fact, this was our first choice, but unfortunately the resulting performance was 50% slower than those showed by a similar implementation that does not use the factory pattern of the WSRF (see Figure 6.4). This was probably due to an inefficient implementation of SOAP headers (used to store the resource ID) analysis and resource storage. For this reason our implementation does not use the ”standard” factory pattern.

To implement the synchronous behavior of some tuple space operations, like take or read, we have used the WSN specification transforming them in asynchronous ones. In fact, it is impossible to implement directly synchronous operations using only web services interactions: socket timeouts and a correct network programming won't allow it. So we need a client-side thread and a register that maintains and handles synchronous operation requests.

The simplest case is when a matching tuple is already present into the space. In this case the `TupleSpace` class connects directly to the service that passes

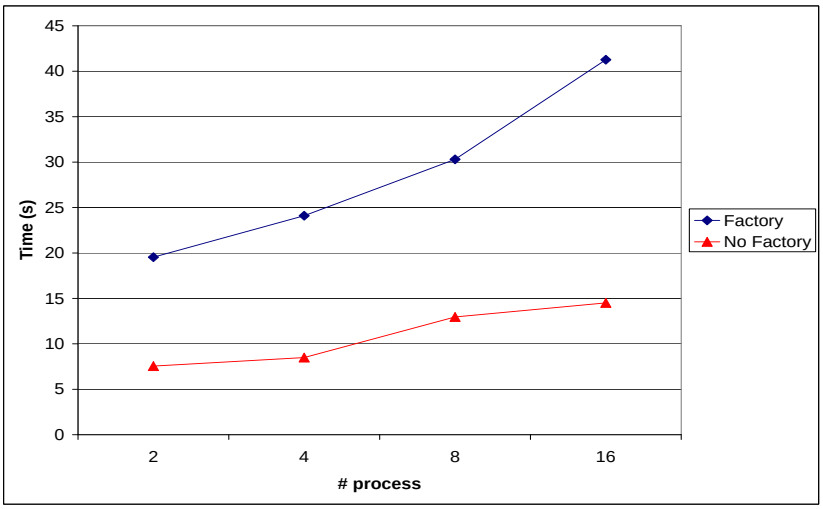

Figure 6.4: *Performance of the same application using Grinda with or without the factory pattern. It is clear the difference in performance.*

the request to the `TupleBroker` implementation responsible for the space that finally returns the result.

Instead, if no matching tuple is found, the `TupleBroker` registers the unfulfilled requests and returns an empty response. The `TupleSpace` recognizes the empty response, registers it in the `BlockingOperationManager` and blocks the application. When, after some time, a matching tuple is inserted into the space, the `TupleBroker` recognizes that it can fulfill a pending request, removes the pending request and sends the response directly to the client-side `BlockingOperationManager` that unblocks the application and passes it the result.

The `BlockingOperationManager` exposes a standard `NotificationConsumer` interface allowing to be compatible with other WSN implementation. This can be useful for implementing the client-side module since there exist various WSN implementations in different programming languages like C, C++, C#, Python, Perl or Java.

Another possible architecture for this behavior could be based on WSN topics and the relative automatic notification mechanism. The Grinda service could expose a standard `NotificationProducer` interface allowing direct registra-

tion to topics each of which representing a space. However, this architecture has not been chosen because it would be more difficult to code the interaction between `TupleBroker` instances and the registration service. In fact, the latter does not implement an API for supporting custom registration mechanisms efficiently. Moreover, it would require to develop support for a new `TopicExpressionType` (or at least a `QueryExpressionType`).

6.4 Implementation

The Grinda server-side module is implemented in Java using Web Services and the WSRF framework provided by the Globus Toolkit. It has to be deployed into the Globus Toolkit container or a compatible servlet container like Tomcat.

The client-side module is designed to be loosely-coupled with the service allowing the use of other libraries or programming languages to be better integrated with the application. In fact, we have developed two different client-side modules: one in Java and the other in C++. Both share a similar architecture with some difference that we will discuss in this section. With these two modules it is possible for new and legacy applications to interact with the tuple space.

Using other programming languages like C# or Python, it would also be possible to implement client-side modules. However, due to the lack of time and resources we have not developed such specific frontends but we hope to do it in the future.

The implementation of Grinda show various interesting aspects. The most important of them are the tuple space implementation, the transparent serialization of tuples, the notification mechanism and the implementation of the C++ client-side module. They will be analyzed in details in the following sections.

6.4.1 Tuple Space Implementation

Tuple Spaces have been designed to be modular and to share the same interface allowing different implementations to be developed. In fact, using the configuration file it is possible to specify the tuple space implementation to use: it will be automatically instantiated at runtime.

We have implemented two different types of tuple space: a transient and a persistent one. The first type uses the techniques described in the previous chapter and is present in two different flavors: X-Tree or Pyramid-Technique based. Each of these flavors can be chosen independently and loaded at runtime. In this way, the space is more efficient, reducing the computational cost of operations as demonstrated before.

Unfortunately, transient spaces lose their content on system crashes or service shutdowns. For this reason we have developed a persistent tuple space based on XML database. Write operations store the XML description of the data coming from the clients into the databases, take operations are accomplished using XPath queries on these data. As database implementation, we have used the Oracle XML Berkeley DB because according to [87] seems to have the best performance. An implementation based on Xindice would be useful, because this XML database is used by many services of the Globus Toolkit. Unfortunately, for lack of time we have still not implemented it.

6.4.2 Serialization of Tuples

One of the most important problems we have faced during the development of our service was the handling of different data types in tuples. In fact, web services developers usually have to deal with predefined data structures that are serialized (or deserialized) using the stubs created automatically by tools on the basis of WSDL definitions. Even when an element with the XML Schema type `any` is part of input or output of a web service operation, serializers can be called at client or server-side to obtain the corresponding object in order to manipu-

late it with the used programming language. This can be seen as a ”tyranny of serializers” that forces all data types transmitted to have a corresponding serializer/deserializer pair at both client and service side.

This approach is used because hard-coding the serialization is more efficient than other methods like reflection, but for our service we need something more flexible for the following reasons:

- Developers should be able to use their own custom data types in a simple way without the need to create stubs or coding serializers. This avoid the proliferation of data types whose unique purpose is to encapsulate legacy data: why do not transmit them directly? The program complexity and the development efforts would be reduced.

- Third part developers have no control on the service and thus they cannot deploy new serializers for data types not yet supported. This is a great problem especially for our tuple space implementation that is unable to know a priori what types will be transmitted to the service.

To support this feature we need a tuple model that can be simply converted to XML and vice versa. The simplest for this purpose was the JavaSpaces one because it allow a direct translation as we will describe.

However, to do a such operation we need a library that allows the direct translation of data into XML and we have chosen the XStream library [155]. This library take in input a Java object and serialize it in XML using its own internal structure without the definition of any kind of specific serializer. The library uses the Java reflection API to obtain the object fields and translates them into XML elements: field names become tag names and their values are recursively converted in textual form. The root element name is equal to the class name.

In this way, only a serializer would be enough theoretically to marshal every possible type and we do not need to create new serializers for every type we use. However, some types still need specific serializers but the XStream library

```
class Person {
  String firstName="John";
  String lastName="Doe";
  Date dateOfBirth=
    new Date("01/01/1970");
  int age="37";
```

```
<Person>
  <firstName>John</firstName>
  <lastName>Doe</lastName>
  <dateOfBirth>
    01/01/1970
  </dateOfBirth>
  <age>37</age>
</Person>
```

Table 6.2: *Example of XStream serialization*

provide a simple plugin system and specific serializers for many standard Java types.

An important problem is how these types are serialized. In fact, some data types, especially arrays, can be inefficiently serialized if only plain XML is used, and this can lead to very low performance during the transmission. This is a known problem of SOAP (and XML protocols in general) and some specifications have been proposed to resolve it. The most used of them is SOAP with Attachment [153], although it will be probably abandoned in favor of a more efficient one as MTOM [154]. They define binary encodings to transmit data inside a SOAP Envelope.

Unfortunately, with very small arrays a similar encoding will provide only an excessive overhead, whereas in other case saving the data in a file and downloading only pieces of them using another binary protocol like GridFTP will be better.

Thus, we have to offer to the developer the possibility to choose between different transport mechanisms. For this reason our serialization support allows various types of array serializers to be selected at runtime. Until now, we have implemented array serializers using plain XML, base64 encoding, SOAP Attachment and files. This last one saves the data in a file and sends only a link to it,

allowing the receiver to download the data and reducing the server load. For the moment, we have not implemented a serializer for MTOM because the current version of the Globus Toolkit does not support it.

In order to allow receivers to correctly deserialize the data, a specific attribute `enc` is added to the element representing the encoding used.

Serializing data in a efficient way, like that described above, leads to the impossibility to analyze each element during the matching. In fact, encoded elements should not be matched because this operation will be expensive and the encoded version could not respect the natural equality semantics. For these reasons, the XML attribute `nocheck` is used. It prevents the matching function to take into account this element. The matching will be then computed on the remaining elements.

A similar approach has been used to implement the matching against supertypes. In fact, in the root element the attribute `implements` is present that contains the list of classes the current type extends (interfaces have not be used because they cannot be passed to the space). So, it is possible to checks supertypes in order to compute the matching.

Using this approach, it is possible to transmit every type with the minimum effort for the developer. Apart from simplifying the code avoiding the proliferation of "wrapper types", it enforces better modularization. In fact, since every type can be serialized, it is possible to use directly part of the business logic of the application allowing methods to operate directly on serialized data. So, methods can be directly invoked on received objects and other support classes are no more needed. In this way, a result similar to Java Serialization is obtained: a data is transmitted and immediately one of its method can be called. This approach does not support code mobility because the class definition is not transmitted and should be present on every tuple space client. However, it is simpler to code than the standard interaction with stubs.

6.4.3 Notification

As in JavaSpaces we have developed a support to event notification. A listener can register itself for receiving notifications of tuples inserted or removed from the space through the use of templates. Since the notification is tightly-coupled with the distribution strategy used, we have decided that `TupleBroker` should handle it too. Similar to operations, registration requests are forwarded to the most suitable `TupleBroker` that will handle them.

Unfortunately, this approach is not compatible with the WS-BasicNotification specification because it is impossible to return immediately the `EndpointReference` of the resource that represent the subscription. In fact, apart from centralized spaces, it is not known a priori and must be discovered every time. Moreover, we need a way to specify a template to match specific tuples. According to the standard, this could be done using the `Precondition` field of the `Subscribe` message but it is not taken into consideration by the Globus Toolkit implementation, that has been revealed to be a simple topic based notification system without the possibility to specify any filters on outgoing notification messages or registrations. The lack of features is too great to use this implementation in our service.

Even the use of another specification, the WS-BrokeredNotification, cannot help. It is not implemented by the Globus Toolkit and its use could lead to the creation of a chain of brokers each responsible for the subscription of the other. So a notification message needs to travel through the entire chain before arriving at the correct destination enormously increasing the communication overhead.

For these reasons, we have decided to implement our own Web Service interface for notification. In fact, WSN specifications have showed to be inadequate to handle our distributed architecture and their implementation in the Globus Toolkits is not complete and lacks many features.

6.4.4 C++ Client

The development of the C++ client has been quite challenging. In fact, we wanted to maintain the possibility of an automatic type serialization. Unfortunately C++ does not implement a standard reflection API like Java or C# and RTTI is not powerful enough to support the automatic serialization of custom types. There are several projects that try to support reflection mechanisms in C++ like [109], [146] or [149] but most of them are only prototypes and not widely adopted. For this reason we have used the Qt4 Toolkit [150], one of the most important toolkits for the development of portable applications used in many projects like for example KDE. In fact, it provides an interesting feature: the Meta Object System. This allows to have type information about classes developed using the toolkit and to instantiate them by name at runtime. Although less evolved than the Java and C# counterparts, it is still very useful for our purpose.

In our Java implementation we have used the reflection API to collect the class fields and transform them into XML elements. We have done the same in the C++ client except from the fact that we have used `QProperties`. They are managed by the Qt runtime and can be written or read by name. So custom objects can be serialized automatically reading their properties and transform them in XML elements with the same name. To better accomplish this process properties should be defined using the `QVariant` class that allows primitive types like `int` or `float` to be treated like objects. The only inconvenience of this approach is that custom objects and their properties should be manually defined by the developers using some macros.

The Qt framework does not implement an API for SOAP or XML-RPC messaging and so we have used the gSOAP 2.7.9 library [140], a little and embeddable library that allows to develop web services in both C and C++. It can generate client or server-side stubs and provides a simple HTTP server that we have used to implement asynchronous operations. Unfortunately, it has still some bugs and we must find some workarounds for them.

The gSOAP library is also the basis of the C implementation of the Globus

Toolkit. We have not used it because it is designed only for the C programming language and has no support for C++. Moreover, it is more difficult to use and configure and requires more resources.

Chapter 7

Experimental Results

7.1 Introduction

After having successfully implemented our service, we have developed a series of tests in order to verify the behavior of our system. We have measured two different aspects: the latency of the system and the scalability, when a distributed application is deployed on a network. We have tested scalability using two different types of applications, which have completely different parallelization strategies. Moreover we have developed and tested a simple workflow in order to show the versatility of our service and a possible further use of it.

The first type of application tries to guess a hashed password using a brute force attack. It represents a highly parallel application that requires very few communications during its execution. Thus, the communication overhead should be very limited.

The second test application is a plasma simulation. The algorithm used is not completely parallel and thus the communication overhead should be greater. We have implemented this application to analyze the behavior of our system in the case of not completely parallelizable applications. Moreover, since this test application derived from an MPI-based code we have been able to compare the performance of our framework with MPI too.

Using these two different types of applications we gained more information on the behavior of our system under different circumstances.

All our experiments have been performed on the same testbed: a 100Mbps Ethernet LAN composed by Core Duo PCs equipped with Ubuntu Linux 7.04. This network was not dedicated because it is part of the students laboratories, but was the only choice we have to collect a medium-large number of hosts.

7.2 Latency Tests

The first test we have conducted on our implementation is calculating its latency. By latency we mean the time required to accomplish a tuple space operation. Given a number of slaves, the test consist in calculating the average time spent for taking/writing the same tuple from/in a space. The average has been obtained from 1000 repeated tests. As shown in Figure 7.1, the latency is not heavily

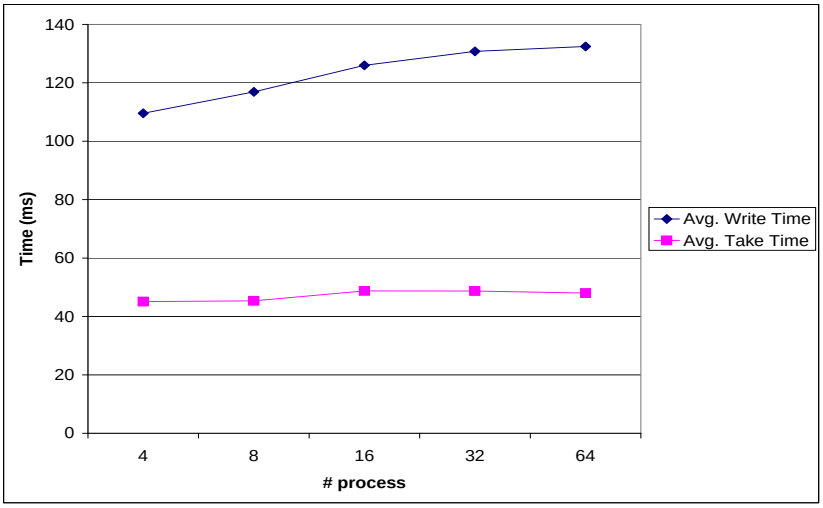

Figure 7.1: *Latency Test Results*

influenced by the number of hosts. Take operations seem to be absolutely independent of the network size whereas write operations increase of a small factor when the size grows (∼20% with an increased size of 1600%).

Another interesting aspect is that the graph shows that write operations require more time than take ones. Probably this is due to a higher overhead in instantiating all the objects required for storing tuples.

7.3 Scalability Test

As described before the scalability test has been conducted on two different types of applications. In both cases we have calculated if the application speedup increases when adding new hosts.

The first application tested was the highly parallel one. It simulates a brute force attack on an hashed password using a master/slave strategy. At the beginning the master randomly generates a password and computes its SHA1 hash. Then it writes all tasks to the tuple space. Each of them contains the hashed password along with the interval of the strings to check. The slaves continuously loop through the following steps:

1. take a task from the tuple space
2. generate the requested strings
3. calculate their SHA1 hash
4. compare them with the one sent by the master
5. write a partial result to the space

At the end the master collects all the partial results and returns the password.

It is clear that this simple application is completely parallel and uses the implicit load balancing of the tuple space. In fact, since all the tasks are written to the space at the beginning, the faster slaves can collect and execute a higher number of tasks with respect to the slower ones, thus increasing the overall utilization of the computers. Moreover, the slaves should not wait until the master finish to write all the tasks, they start immediately when a new task becomes

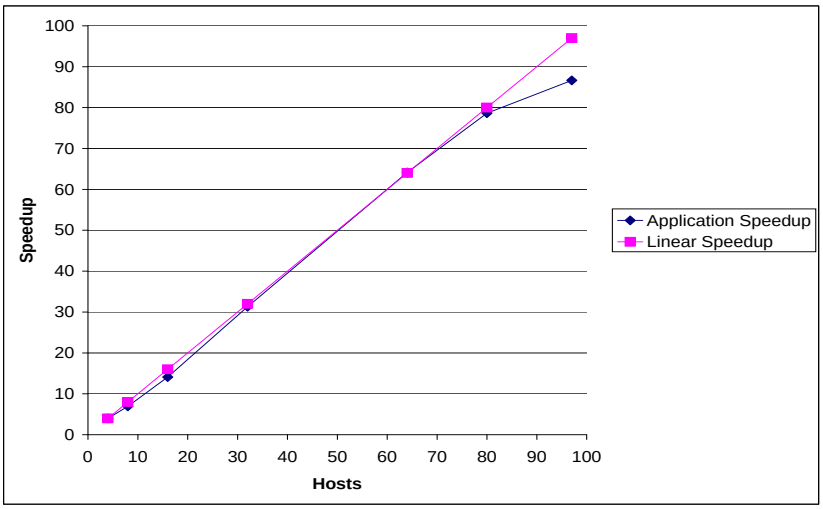

Figure 7.2*: Speedup of the highly parallel test application*

available. Finally the tasks themselves are very small thus reducing the communication overhead. As shown by the test results in Figure 7.2, the speedup of this application seems to grow very well compared to the linear one.

The second application we have used to test our framework is a plasma simulation. In the following sections we briefly describe the physical background of the model and then the results we have obtained in the test.

7.3.1 Plasma Simulation

A plasma is a hot fully ionized gas which may be regarded as a collection of N_i positive ions and N_e negative electrons interacting through their mutual electric ($\mathbf{E}$) and magnetic ($\mathbf{B}$) fields.

The force experienced by a charge q moving with velocity $\mathbf{v}$ in electromagnetic fields is (here and in the following boldface variables indicate vector-valued quantities)

$$\mathbf{F} = q\mathbf{E} + q\mathbf{v} \times \mathbf{B}, \tag{7.1}$$

and the motion of the charge is determined by the Newton law

$$\mathbf{F} = m\frac{\mathrm{d}\,\mathbf{v}}{\mathrm{d}\,t}. \tag{7.2}$$

The electric and magnetic fields are related to the charge and current density $(\rho, \mathbf{j})$ by the Maxwell equations:

$$\nabla\cdot\mathbf{E} = \frac{\rho}{\epsilon_0}, \tag{7.3}$$

$$\nabla\cdot\mathbf{B} = 0, \tag{7.4}$$

$$\nabla\times\mathbf{E} = -\frac{\partial\mathbf{B}}{\partial t}, \tag{7.5}$$

$$\nabla\times\mathbf{B} = \mu_0\mathbf{j} + \frac{1}{c^2}\frac{\partial\mathbf{E}}{\partial t}, \tag{7.6}$$

where ϵ_0, μ_0 and $c = (\epsilon_0\mu_0)^{1/2}$ are respectively the permittivity, the magnetic permeability and the light velocity in free space.

Numerical method

For plasma simulation in three dimensions, the particle-in-cell (PIC) method has the longest history (it has been introduced since early sixties for fluid dynamics [66]) and has been proved useful for application to a variety of plasma phenomena (see for a general discussion [31] and [68]). In this method the plasma volume (V) containing $N = N_i + N_e$ particles each with mass m_n charge q_n and velocity $\mathbf{v}_n$ $(n = 1, \ldots, N)$, is sampled in cells through a mesh with three step size, not necessarily equal, in each dimension, Δ_1, Δ_2 and Δ_3. Each mesh cell is then an hexahedral element with volume $\Delta_1\Delta_2\Delta_3$ and contains a large number of ions and electrons.

The standard simulation with PIC method proceeds iterating the following steps

- compute the particle parameters on the grid from the particle position
- solve the field equations on the grid to obtain the force field on the mesh

- compute the particle parameters at the particle position by back interpolation from grid point values
- advance the particles in time integrating the motion equation for velocities and positions

The first and third step are essentially an interpolation algorithm, which can be defined scatter and gather phase, following the tradition. These steps are the most compute-intensive of the simulation.

The algorithm of the second step depends on the structure of the Maxwell equations which better approximate the problem under study. Accordingly, 4-th order Runge-Kutta method for ordinary differential equations (ODE) or several methods to solve the Poisson equation apply.

The last step could be realized with a symplectic integration method as the Verlet method [123] or the 4-th order Runge-Kutta, when the magnetic field is relevant.

Interpolation

The scatter of particle parameters as, for instance, the charge is realized with a convolution between the particle charge density (δ_3 is the Dirac 3-dimensional delta function),

$$\rho_p(\mathbf{x}) = q \sum_{i=1}^{N} \delta_3(\mathbf{x} - \mathbf{x}_i), \tag{7.7}$$

and a cloud density profile, $W(\mathbf{x})$, for each particle, i.e. ($\Delta = \Delta_1 = \Delta_2 = \Delta_3$)

$$\rho(\mathbf{x}) = \frac{1}{\Delta^3} \int \rho_p(\mathbf{y}) W(\mathbf{x} - \mathbf{y}) \, \mathrm{d}\, \mathbf{y}. \tag{7.8}$$

The sampling of density on the mesh point is realized then as (the vector of indices $\mathbf{n} \equiv (n_1, n_2, n_3)$ identify the mesh cell)

$$\rho(\mathbf{x_n}) = \frac{1}{\Delta^3} \sum_{i} W(\mathbf{x} - \mathbf{x_n}). \tag{7.9}$$

The effect of the convolution process is to smooth the density and hence the related physical parameters (as field and force) on the cloud size and to limit the spatial resolution of the code. This behavior suggests that a good solution could be to reduce the overlap of grid points by the cloud density profile and to increase the number of mesh grid points to reduce the density smoothing and the space resolution, respectively.

However, the higher the number of points covered by the cloud density profile, the higher the regularity of the function: in one dimension, the continuity of the function is granted for linear interpolation involving two points, while to get continuity of the profile derivative up to second order requires quadratic interpolation involving 3 near points; in three-dimensional space this involves 8 mesh points and 27, respectively. In the third step of the iterative process, when the back interpolation of the force

$$\mathbf{F}_i = \sum_{\mathbf{n}} W(\mathbf{x}_i - \mathbf{x_n})\mathbf{F}(\mathbf{x_n}) \tag{7.10}$$

is required with the same density W profile to avoid self-forces, higher accuracy in force computation is controlled by higher regularity of the mesh profile function.

Therefore the choice of the profile function is a compromise between accuracy in force computation at particle position, smoothing of the density field and computational load.

Parallelization

PIC codes are extremely compute-intensive in the simulation of interesting physical problems and require parallel computing. The parallelization of a PIC code is not an easy task. Principal factors that hinder the straightforward parallelization are

- the presence of particle and mesh which prevents the application of one only optimal method of parallelization

- the back and forth interpolation between grid and particles every timestep
- the movement of particles which can produce load unbalance
- the possible generation of strong inhomogeneity in particle distribution which require an adaptive spatial domain decomposition

Some of this issues have been successful addressed by different groups who aimed at obtaining either efficient algorithms for at least a class of problems or some general system. PICARD [37], QUICKSILVER [104], VORPAL [93] and a PPM library [16] are some of the results.

PICARD and VORPAL have been developed mainly for PIC simulation with load balancing and domain decomposition. The first implements partially some of the extensive taxonomy of different parallelization models obtained through an abstraction of grid and particle boxes, the second applies to plasma models which includes both particle-in-cell and fluid models, allowing laser-plasma interaction, and adopt the object oriented paradigm using C++. Many of the features of VORPAL are also present in the Fortran90 codes OSIRIS [55].

QUICKSILVER applies to multiblock codes and obtains rather good efficiency (about 90%) for parallelization when the distribution is uniform, and a poor one (about 60%) when dealing with irregular distributions.

Recently, Sbalzarini *et al.* [16] have produced a portable library developed in standard languages (Fortran 90 and C) and libraries (MPI) which allows also the treatment of particle-particle interaction. This method is actually an implementation of a so-called particle-particle-mesh (PPM, see for details [68]) and the authors claim a remarkable level of efficiency for a variety of problems.

All these codes can run on mainframe or on distributed environment. Finally, ALaDyn [23, 24, 25] is a code for plasma simulation which applies PIC method and has been developed by a group of the University of Bologna. Different order of interpolation are possible in this code and the computation of field and the update of particle position and velocity is performed with a 4th order Runge-

Kutta algorithm. The parallelization strategy used is present in the taxonomy analyzed by [37] and it is realized by means of MPI library.

7.3.2 Plasma Test Results

To obtain more information about the scalability of our framework we have used the ALaDyn code as another test application. Since it is written in C using MPI we have ported it to Grinda substituting only the MPI calls enabling the communications through the C++ client. The rest of the algorithm remained untouched. For the tests we have used a 250x250x800 computational mesh with 10^7 particles.

The changes we have made in the ALaDyn code are related to the MPI function calls. In particular the code uses four MPI functions: MPI_broadcast, MPI_send, MPI_receve and MPI_sendreceive. The first one is used for the initial configurations whereas the latter ones are used by workers to inform each others of the new migrated particles they have to handle.

The MPI functions have been translated to Grinda primitives as described by Table 7.1. The translation depends on the role that workers have in the communication. Actually, initiators of the communication should use primitives different from receivers. Moreover, the data transmitted through the tuple space should contains so called labels that help in disambiguating similar data items. In our case MPI process identifiers and custom data definition label have been used for this purpose.

MPI Function	**Grinda Primitive**	
	Initiator	**Receiver**
MPI_bcast	write	read
MPI_send	write	write
MPI_receive	take	take
MPI_sendreceive	write; take	write; take

Table 7.1: *Translation of the MPI functions to Grinda primitives*

At this point we were able to compare our framework with MPI using the plasma simulation application as benchmark. In fact, we have deployed the original application on the same network along with the open source MPICH 1.2.7 library. As showed by Figure 7.3 the original MPI-based application and our

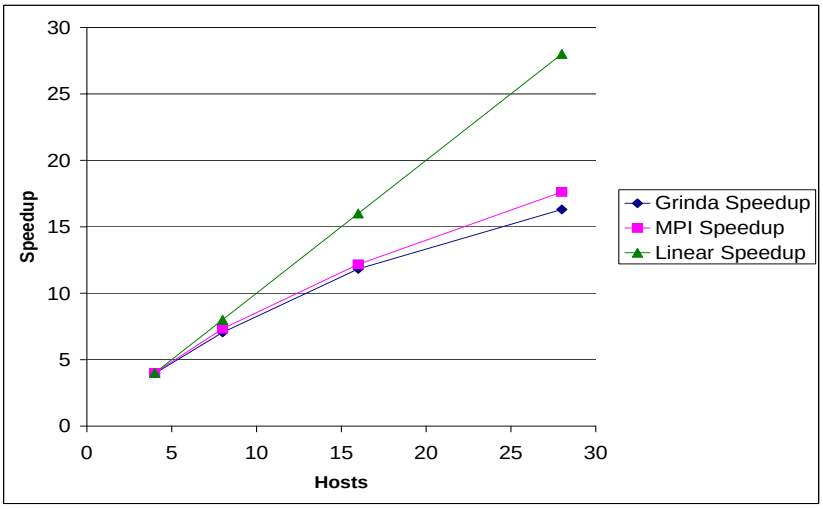

Figure 7.3: *Speedups of the Grinda-based test application and the original MPI implementation.*

code have almost equivalent speedups. The difference seems to have essentially a statistical meaning.

It should be said also that we did not change the parallelization algorithm of AlaDyn. Other parallelization models described in the [37] taxonomy could improve the performance of the application, exploiting better Grinda characteristics. In fact, the implemented algorithm does not allow any type of load-balancing, statically assigning the tasks to each node at the beginning. This can reduce the overall performance of our system, which could also benefit from other types of algorithm requiring a smaller number of communications.

Moreover, the application class of AlaDyn is probably not well suited for our service, because it can require the transmission of large amount of data that the tuple space is not meant to handle. There are several other protocols designed specifically for this purpose. Grinda is not a data transmission system like

GridFTP but a high level coordination service able to handle the dependencies between the various computing modules composing a distributed applications. We have implemented this second test and compared it to MPI only to demonstrate the adaptability of our service to different circumstances even not the best suitable for our infrastructure.

Finally it has to be noted that these results are only preliminary because more in-depth tests are needed in order to completely understand the behavior of our system.

7.3.3 Workflow Example

To show the versatility of our system, we have also implemented and tested a simple workflow. The execution model we have used is based on a generic representation of the various tasks composing the workflow. Actually, every task type we have used is a subclass of `grinda.client.Task`. They will be written into the space and executed by a variable number of hosts. Since the description is generic, the slaves do not need to know which tasks they are going to execute, they simply take from the space the first instance of `grinda.client.Task` available. Task inputs are stored inside the task class itself and, after the execution, tasks can produce other tasks or other type of data as output. In every case they will be written into the space.

According to this approach, the tuple space is used as the basis for a workflow engine, which can take advantages from the features of the tuple space model. Actually, to increase the application performance, it is sufficient to add more slaves allowing more tasks to be executed at the same time.

The workflow we have used is depicted in Figure 7.4. It is composed by more than 250 tasks: the most of them can be executed in parallel apart the central one that is unique and have to wait for the previous task to be executed. To test our workflow, we have executed it on the usual test network with an increasing number of hosts. Each of them is responsible for executing a variable number of tasks according to its load. Thus this number is not defined in advance and depends

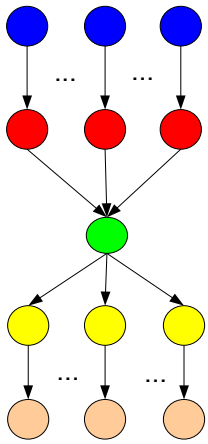

Figure 7.4: *The workflow used as test*

on the load of the single machines. For each network size, the average time spent for different runs of the test application has been calculated. The experimental results are showed in Figure 7.5. As shown by the experimental results, the ap-

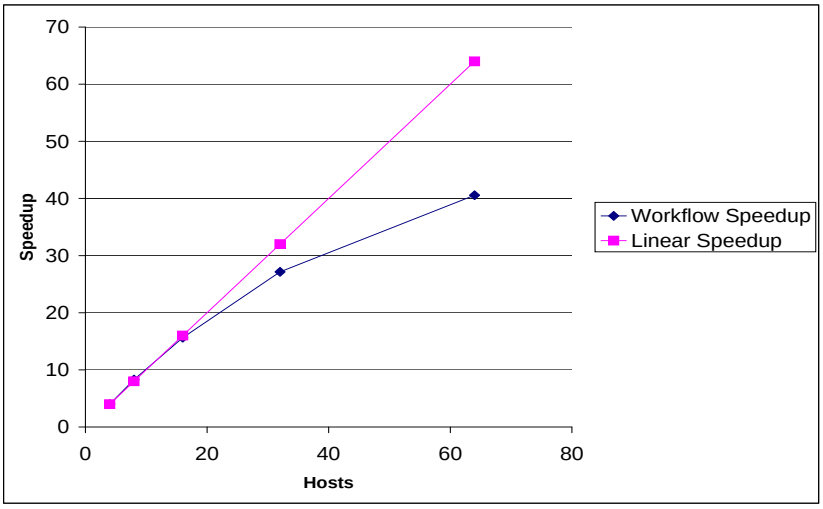

Figure 7.5: *Speedup of the tested workflow at different network sizes*

plication scales quite well at the beginning, reducing its performance for network with more than 16 hosts. This is an expected results for two reasons: firstly it is the usual trends of distributed application, secondly the workflow we have used

is not completely parallelizable and thus the maximum speedup we can obtain is less than that of a completely parallel application. Nevertheless, we think that this workflow has performed quite well.

Finally, we have to state that the workflow support we have implemented is only a prototype. Actually, it lacks of many features like a module able to automatically transform a graphical or textual representation of the workflow in a sequence of tuple space operations. However, these characteristics are out of the scope of this thesis and probably will be more deeply analyzed in a future work.

Chapter 8

Conclusion and Future Development

The Service Oriented Architecture (SOA) has promoted the modularization and reuse leading to distributed applications based on the coordination of different services executing a specific job. In this Lego-like approach the models that manage the coordination are of a fundamental importance. Several coordination models have been proposed in the literature during the years like message passing, publish/subscribe, workflows or tuple spaces. Some of them are more suitable to support the SOA model like workflows or publish/subscribe, other can be still used for this purpose but provide very few features like message passing. Finally some of them, like tuple spaces, have been used not often due to their limited diffusion.

In this thesis we have showed how to use one of these seldom used models, tuple spaces, to coordinate activities in SOA.

However, before developing a complete service, we have analyzed the advantages and disadvantages of actual tuple space implementations and proposed some techniques in order to gain better performance. In fact, many available implementations can become bottlenecks in some situations. Our proposals try to avoid these problems and focus on two different aspects: the tuple storage and distributed tuple spaces.

To increase the performance of tuple storages we have proposed the use of spatial indexing. This approach seems to have not been used up to now. In the

literature are described many spatial indexing strategies very different from each other, for this reason we have employed two of them to compare their behavior: X-Tree and the Pyramid Technique. Both implementations haven resulted the fastest ones compared to almost all other widespread implementations, especially for medium large storages.

Another aspect of tuple space implementations that we have investigated is the distribution of tuples on a network. Many distributed tuple spaces are not able to support large networks well. To overcome this limitation we have used structured p2p protocols to speedup the operations. Since take operations require a range search support to be accomplished, we have used two particular p2p protocols: the VBI-tree and Extended-CAN. As before we have compared the results of both approaches and we have showed that VBI-tree based tuple spaces perform very well compared to Extended-CAN ones. This approach represent a novelty too, because no other proposal has been made in order to use this type of structured p2p protocols for tuple spaces.

After the study of these techniques, that are general and can be implemented on various technologies, we have used some of them to design and implement a Grid service for the tuple space model using the Globus Toolkit. Its development posed challenges in different aspects.

First of all the choice of the tuple space model to employ was a key point because XML serialization poses very strong limits. For example the original array-like tuple model has been revealed inadequate due to serialization problems and complex use. At the end a model similar to JavaSpaces was chosen because it support better XML messages.

The use of this model has contributed to resolve another problem, probably the worst of all. It was caused by the architecture of the Globus Toolkit (and Apache Axis in particular) that has shown to be unable to support custom data types not associated with a serializer. This is a great problem for tuple spaces because the possibility to use every possible data type in tuples is very important. Nevertheless, we have solved this problem using smart serialization at the

client-side whereas services maintain the tuple content in an tree-like form without serialization. This solution has been very powerful since it allows every possible Java data type to be serialized (even those present in the standard library) and simplify the service that does not need to deserialize the tuples. Clearly this approach produces higher overhead with respect to the standard one but was the only possible choice to create a usable system.

During the development of our service we have also found some limits of the Globus Toolkit implementation, especially in the modules that support the WSN notification. They prevented us from implementing advanced notification mechanisms reusing the WSN support. Moreover, the factory based development strategy has revealed to be slower although more logic and modularizable from an architectural point of view.

After having successfully completed the implementation of our service, we have tested them in order to analyze its behavior. In particular, we have studied the latency of the system and its scalability. The latency has resulted stable and independent from the number of clients used although higher with respect to binary communications. This means that our framework can suffer from a high overhead in some circumstances especially when many communications are needed. For this reason we have analyzed its scalability using two different types of applications: a high parallelizable one, that simulates brute-force attacks to a password, and a plasma simulation, that instead requires more communications and is not completely parallelizable. Both applications have showed a scalable behavior even if the scalability of the second one was limited by the system overhead.

The second application was simply ported from a MPI-based original code and this fact gave us the possibility to compare our framework with MPI, a typical message passing model. The test results show that our application has performance comparable with the MPI-based one. Nevertheless these tests demonstrate that our framework can be efficiently used by real applications. It should be also outlined that the Grinda API is quite simple and powerful allowing developers

to write distributed applications in a more efficient way.

The tests presented in this thesis have not been conducted on a well-established testbed as PlanetLab [79], which is used to test other types of distributed applications. This is due to the fact that there is still not a consensus on a common testbed for Grid applications and we are unable to use production Grids due to our limited resources.

Even if we have succeeded in developing a working framework, it can be still extended in some ways. Aspects that can be enhanced are the support for security and the transactional behavior of the tuple space.

At the moment, our service can be secured through the GSI infrastructure provided by the Globus Toolkit. This means that an XML security descriptor can be used to define authorization and transmission requirements for the service. Moreover, through another configuration file it is possible to tell the client how to connect to a secure service. In every case the security provided by the GSI infrastructure will regard the whole service and not singular spaces or even tuples. A more granular approach to security could be a challenge since it should take into account performance too.

The transactional behavior is another aspect that has not been taken into account by our framework for two reasons: there is not consensus on a transaction model for Grid applications and there is not a widely adopted standard for managing Web Service transactions. We will study this problem more in-depth, analyzing possible solutions.

Finally, in order to increase the usefulness of our framework we think to port it to a more widespread infrastructure like Axis 2. In this way, it could be used by regular Web Services and we hope that this will avoid some of the limitations we have found.

References

[1] M. Abdallah and H. C. Le. Scalable Range Query Processing for Large-Scale Distributed Database Applications. In *IASTED PDCS*, pages 433–439, Phoenix, USA, 2005.

[2] M. Agarwal, V. Bhat, H. Liu, V. Matossian, V. Putty, C. Schmidt, G. Zhang, L. Zhen, M. Parashar, B. Khargharia, and S. Hariri. AutoMate: Enabling Autonomic Applications on the Grid. In *Autonomic Computing Workshop*, pages 48– 57, Seattle, USA, 2003.

[3] G. Agha and C. Hewitt. Actors: A conceptual foundation for concurrent object-oriented programming. In B. Shriver and P. Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 49–74. MIT Press, 1987.

[4] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, A. Gianoli, K. Lörentey, and F. Spataro. VOMS, an Authorization System for Virtual Organizations. In F. F. Rivera, M. Bubak, A. G. Tato, and R. Doallo, editors, *European Across Grids Conference*, volume 2970 of *LNCS*, pages 33–40, Santiago de Compostela,Spain, 2003. Springer Verlag.

[5] A. Allavena, Q. Wang, I. Ilyas, and S. Keshav. LOT: A Robust Overlay for Distributed Range Query Processing. Technical report, University of Waterloo, 2006.

[6] M. Amoretti, F. Zanichelli, and G. Conte. SP2A: A Service-oriented Framework for P2P-based Grids. In *3rd International Workshop on Middleware for Grid Computing (MGC05)*, pages 1–6, Grenoble, France, 2005. ACM Press.

[7] D. P. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Pittsburgh, USA, 2004. IEEE Computer Society.

[8] J. Andreoli, P. Ciancarini, and R. Pareschi. Interaction Abstract Machines. In G. Agha, P. Wegner, and A. Yonezawa, editors, *Trends in Object-Based Concurrent Computing*, pages 257–280. MIT Press, 1993.

[9] A. Andrzejak and Z. Xu. Scalable, Efficient Range Queries for Grid Information Services. In *P2P '02: Proceedings of the Second International Conference on Peer-to-Peer Computing*, page 33, Washington DC, USA, 2002. IEEE Computer Society.

[10] F. Arbab. The IWIM Model for Coordination of Concurrent Activities. In P. Ciancarini and C. Hankin, editors, *COORDINATION '96: Proceedings of the First International Conference on Coordination Languages and Models*, LNCS, pages 34–56, London, UK, 1996. Springer Verlag.

[11] F. Arbab. What Do You Mean, Coordination? Issue of the Bulletin of the Dutch Association for Theoretical Computer Science (NVTI), 1998.

[12] F. Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.

[13] F. Arbab, I. Herman, and P. Spilling. An overview of Manifold and its implementation. *Concurrency: Practice and Experience*, 5(1):23–70, 1993.

[14] X. Bai, K. Sivoncik, D. Turgut, and L. Bölöni. Grid Coordination with Marketmaker Agents. *International Journal of Computational Intelligence*, 3(2):153–160, 2006.

[15] X. Bai, G. Wang, Y. Ji, G. M. Marinescu, D. C. Marinescu, and L. Bölöni. Coordination in intelligent grid environments. *Proceedings of the IEEE*, 93(3):613–630, 2004.

[16] I. F. Balzarini, J. H. Walther, M. Bergdorf, S. E. Hieber, E. M. Kotsalis, and P. Koumoutsakos. PPM - A highly efficient parallel particle-mesh library for simulation of continuum systems. *Journal of Computational Physics*, 215:448–473, 2006.

[17] S. Banarjee, S. Basu, S. Garg, S. Lee, P. Mullan, and P. Sharma. Scalable Grid Service Discovery Based on UDDI. In *Proc. of 3rd International Workshop on Middleware for Grid Computing (MGC05), in conjunction with the 6th International Middleware Conference*, pages 1–6, Grenoble, France, 2005. ACM Press.

[18] J.-P. Banatre, P. Fradet, and Y. Radenac. Towards Chemical Coordination for Grids. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 445–446, Dijon, France, 2006. ACM Press.

[19] J.-P. Banâtre and D. L. Métayer. Programming by multiset transformation. *Commununications of the ACM*, 36(1):98–111, 1993.

[20] M. Barbacci, C. Weinstock, D. Doubleday, M. Gardner, and R. Lichota. Durra: a structure description language for developing distributed applications. *Software Engineering Journal*, 8(2):83–94, 1993.

[21] A. Barker. Agent-Based Service Coordination For the Grid. In *IAT '05: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 611–614, Washington, DC, USA, 2005. IEEE Computer Society.

[22] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In H. Garcia-Molina and H. V. Jagadish, editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, USA, 1990. ACM Press.

[23] C. Benedetti, P. Londrillo, L. Rossi, and G.Turchetti. Numerical investi-

gation of Maxwell-Vlasov equations. Part I: basic physics and algoritms. *Comm. in Nonlinear Science and Numerical Simulations*, 13(1):204–208, 2008.

[24] C. Benedetti, P. Londrillo, L. Rossi, and G.Turchetti. Numerical investigation of Maxwell-Vlasov equations. Part II: preliminary tests and validation. *Comm. in Nonlinear Science and Numerical Simulations*, 13(1):209–214, 2008.

[25] C. Benedetti, P. Londrillo, A. Sgattoni, and G.Turchetti. ALaDyn: a high accuracy PIC code for the Maxwell-Vlasov equations. submitted to IEEE-Transactions on Plasma Science.

[26] J. L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM*, 18(9):509–517, 1975.

[27] S. Berchtold, C. Böhm, and H.-P. Kriegal. The Pyramid-Technique: Towards Breaking the Curse of Dimensionality. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 142–153, Seattle, USA, 1998. ACM Press.

[28] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree : An Index Structure for High-Dimensional Data. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases*, pages 28–39, Mumbai, India, 1996. Morgan Kaufmann.

[29] J. A. Bergstra and P. Klint. The TOOLBUS Coordination Architecture. In P. Ciancarini and C. Hankin, editors, *COORDINATION '96: Proceedings of the First International Conference on Coordination Languages and Models*, LNCS, pages 75–88, Cesena, Italy, 1996. Springer Verlag.

[30] G. Berry and G. Boudol. The chemical abstract machine. In *POPL '90: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 81–94, San Francisco, USA, 1990. ACM Press.

[31] C. K. Birdsall and A. B. Langdon. *Plasma Physics*. McGraw-Hill, New York, 1985.

[32] P. Brebner and W. Emmerich. Two Ways to Grid: The Contribution of Open Grid Services Architecture (OGSA) Mechanisms to Service-Centric and Resource-Centric Lifecycles. *Journal of Grid Computing*, 4(1):115–131, 2006.

[33] N. Busi, A. Montresor, and G. Zavattaro. Data-Driven Coordination in Peer-to-Peer Information Systems. *International Journal of Cooperative Information Systems*, 13(1):63–89, 2004.

[34] R. Buyya, D. Abramson, and S. Venugopal. The Grid Economy. *Proceedings of the IEEE*, 93:698–714, 2005.

[35] G. Cabri, L. Leonardi, and F. Zambonelli. Reactive Tuple Spaces for Mobile Agent Coordination. In *MA '98: Proceedings of the Second International Workshop on Mobile Agents*, pages 237–248, London, UK, 1999. Springer Verlag.

[36] G. Cabri, L. Leonardi, and F. Zambonelli. XML Dataspaces for Mobile Agent Coordination. In *In Proceedings of the 2000 ACM Symposium on Applied Computing*, volume 1, pages 181–188, Como, Italy, 2000.

[37] E. A. Carmona and L. J. Chandler. On parallel PIC versatility and the structure of parallel PIC approaches. *Concurrency: Pratice and Experience*, 9(12):1377–1405, 1997.

[38] N. Carriero, D. Gelernter, and L. D. Zuck. Bauhaus linda. In P. Ciancarini, O. Nierstrasz, and A. Yonezawa, editors, *ECOOP '94: Selected papers from the ECOOP'94 Workshop on Models and Languages for Coordination of Parallelism and Distribution, Object-Based Models and Languages for Concurrent Systems*, volume 924 of *LNCS*, pages 66–76, London, UK, 1995. Springer Verlag.

[39] A. J. Chakravarti, G. Baumgartner, , and M. Lauria. The Organic Grid: Self-Organizing Computation on a Peer-to-Peer Network. *IEEE Transactions on Systems, Man, and Cybernetics*, 35:373–384, 2005.

[40] A. J. Chakravarti, G. Baumgartner, and M. Lauria. Application-Specific Scheduling for the Organic Grid. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 146–155, Washington, DC, USA, 2004. IEEE Computer Society.

[41] A. Charles, R. Menezes, and R. Tolksdorf. On the implementation of SwarmLinda. In *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference*, pages 297–298, Huntsville, USA, 2004. ACM Press.

[42] T. Cheatham, A. F. Fahmy, D. C. Stefanescu, and L. G. Valiant. Bulk synchronous parallel computing-a paradigm for transportable software. In *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS'95)*, pages 268–275, Honolulu, USA, 1995.

[43] X. Chen, W. Cai, S. J. Turner, and Y. Wang. SOAr-DSGrid: Service-Oriented Architecture for Distributed Simulation on the Grid. In *PADS '06: Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*, pages 65–73, Washington, DC, USA, 2006. IEEE Computer Society.

[44] A. L. Chervenak and M. Cai. Applying Peer-to-Peer Techniques to Grid Replica Location Services. *Journal of Grid Computing*, 4(1):49–69, 2006.

[45] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 426–435, San Francisco, USA, 1997. Morgan Kaufmann Publishers Inc.

[46] K. Czajkowski, C. Kesselman, S. Fitzgerald, and I. T. Foster. Grid Information Services for Distributed Resource Sharing. In *10th IEEE Inter-*

national Symposium on High Performance Distributed Computing (HPDC-10 2001), pages 181–194, San Francisco, USA, 2001. IEEE Computer Society.

[47] R. de Nicola, G. L. Ferrari, and R. Pugliese. KLAIM: A Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engeenering*, 24(5):315–330, 1998.

[48] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. C. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[49] L. dos S. Lima, A. T. A. Gomes, A. Ziviani, M. Endler, L. F. G. Soares, and B. Schulze. Peer-to-peer resource discovery in mobile Grids. In *MGC '05: Proceedings of the 3rd international workshop on Middleware for grid computing*, pages 1–6, Grenoble, France, 2005. ACM Press.

[50] C. Dumitrescu, I. Raicu, and I. Foster. DI-GRUBER: A Distributed Approach to Grid Resource Brokering. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 38, Washington, DC, USA, 2005. IEEE Computer Society.

[51] P. Eerola, T. Ekelof, M. Ellert, J. R. Hansen, A. Konstantinov, B. Konya, J. L. Nielsen, F. Ould-Saada, O. Smirnova, and A. Waananen. The NorduGrid architecture and tools. In *Computing in High Energy and Nuclear Physics*, La Jolla, USA, 2003.

[52] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 2005.

[53] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[54] R. A. Finkel and J. L. Bentley. Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Informatica*, 4:1–9, 1974.

[55] R. A. Fonseca, L. O. Silva, R. G. Hemker, F. S. Tsung, V. K. Decik, W. Lu, C. Ren, W. B. Mori, S. Deng, S. Lee, T. Katsouleas, and J. C. Adams. OSIRIS: a three-dimensional, fully realistic, particle in cell code for modeling plasma based accelerators. In P. M. A. Sloot, C. J. K. Tan, J. J. Dongarra, and A. G. Hoekstra, editors, *Computational Science - ICCS 2002*, volume 2331 of *LNCS*, pages 342–351, Berlin, Germany, 2002. Springer Verlag.

[56] I. Foster. What is the Grid: A three Point Checklist. *Grid Today*, 2002.

[57] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In H. Jin, D. Reed, and W. Jiang, editors, *IFIP International Conference on Network and Parallel Computing*, volume 3779 of *LNCS*, pages 2–13, Beijing, China, 2005. Springer Verlag.

[58] I. Foster and A. Iamnitchi. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In G. Goos, J. Hartmanis, J. van Leeuwen, and L. Kucera, editors, *Proc. of 8th International IFIP/IEEE Symposium on Integrated Management (IM)*, volume 2573 of *LNCS*, pages 118–128, Colorado Springs, CO, USA, March 2003.

[59] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journale of High Performance Computing Application*, 15(3):200–222, 2001.

[60] N. Franks and A. Sendova-Franks. Brood sorting by ants: Distributing the workload over the work-surface. *Behavioral Ecology and Sociobiology*, 30:109–123, 1992.

[61] M. Frigo, C. E. Leiserson, and K. H. Randall. The implementation of the Cilk-5 multithreaded language. In *Proceedings of the ACM SIGPLAN '98 Conference on Programming Language Design and Implementation*, volume 33, pages 212–223, Montreal, Quebec, Canada, June 1998.

[62] P. Ganesan, B. Yang, and H. Garcia-Molina. One Torus to Rule them All: Multi-Dimensional Queries in P2P Systems. In *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases*, pages 19–24, Paris, France, 2004. ACM Press.

[63] D. Gelernter and N. Carriero. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[64] R. Guerraoui, S. Handurukande, K. Huguenin, A.-M. Kermarrec, F. Le Fessant, and E. Riviere. GosSkip, an Efficient, Fault-Tolerant and Self Organizing Overlay Using Gossip-based Construction and Skip-Lists principles. In *6th IEEE International Conference on Peer-to-Peer Computing*, pages 12–22, Cambridge, UK, 2006. IEEE Computer Society.

[65] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In B. Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57, Boston, USA, 1984. ACM Press.

[66] F. H. Harlow. Particle-in-cell computing method for fluid dynamics. *Methods Computational Physics*, 3:319–343, 1964.

[67] A. Harrison and I. Taylor. The Web Services Resource Framework in a Peer-to-Peer Context. *Journal of Grid Computing*, 4(4):425–445, 2006.

[68] R. W. Hockney and J. W. Eastwood. *Computer simulation using particles*. McGraw-Hill, New York, 1981.

[69] A. A. Holzbacher. A Software Environment for Concurrent Coordinated Programming. In P. Ciancarini and C. Hankin, editors, *COORDINATION '96: Proceedings of the First International Conference on Coordination Languages and Models*, volume 1061 of *LNCS*, pages 249–266, Cesena, Italy, 1996. Springer Verlag.

[70] E. Huedo, R. Montero, and I. Llorente. A Framework for Adaptive Execution on Grids. *Journal of Software - Practice and Experience*, 34:631–651, 2004.

[71] M. Humphrey and G. Wasson. Architectural Foundations of WSRF.NET. *International Journal of Web Services Research*, 2(2):83–97, 2005.

[72] A. Iamnitchi and I. Foster. A Peer-to-Peer Approach to Resource Location in Grid Environments. In *11th IEEE International Symposium on High Performance Distributed Computing*, page 419, Edinburgh, UK, August 2002.

[73] H. V. Jagadish, B. C. Ooi, K.-L. Tan, Q. H. Vu, and R. Zhang. Speeding up search in peer-to-peer networks with a multi-way tree structure. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 1–12, Chicago, USA, 2006. ACM Press.

[74] H. V. Jagadish, B. C. Ooi, and Q. H. Vu. BATON: A Balanced Tree Structure for Peer-to-Peer Networks. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 661–672, Trondheim, Norway, 2005. VLDB Endowment.

[75] H. V. Jagadish, B. C. Ooi, Q. H. Vu, R. Zhang, and A. Zhou. VBI-Tree: A Peer-to-Peer Framework for Supporting Multi-Dimensional Indexing Schemes. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, page 34, Washington, DC, USA, 2006. IEEE Computer Society.

[76] J. Jähnert, A. Cuevas, J. I. Moreno, V. A. Villagrá, S. Wesner, V. Olmedo, and H. Einsiedler. The "akogrimo" way towards an extended ims architecture. In *Proceedings of ICIN*, Bordeaux, France, Oct 2007.

[77] N. T. Karonis, B. Toonen, and I. Foster. MPICH-G2: a Grid-enabled implementation of the Message Passing Interface. *J. Parallel Distrib. Comput.*, 63(5):551–563, 2003.

[78] T. Kielmann. Object-Oriented Distributed Programming with Objective Linda. In *First International Workshop on High Speed Networks and Open Distributed Platforms*, St. Petersburg, Russia, June 1995.

[79] A. Klingaman, M. Huang, S. Muir, and L. Peterson. PlanetLab Core Specification 4.0. Technical Report PDN–06–032, PlanetLab Consortium, June 2006.

[80] Y. Kulbak and D. Bickson. The eMule Protocol Specification, January 20, 2005.

[81] S. Kurkovsky, Bhagyavati, A. Ray, and M. Yang. Modeling a Grid-Based Problem-Solving Environment for Mobile Devices. In *IEEE International Conference on Information Technology: Coding and Computing (ITCC-04)*, page 135, Singapore, 2004. IEEE Computer Society.

[82] Z. Li and M. Parashar. Comet: A Scalable Coordination Space for Decentralized Distributed Environments. In *HOT-P2P '05: Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems*, pages 104–112, Washington, DC, USA, 2005. IEEE Computer Society.

[83] B. Liu, W.-C. Lee, and D. L. Lee. Supporting Complex Multi-Dimensional Queries in P2P Systems. In *ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 155–164, Washington, DC, USA, 2005. IEEE Computer Society.

[84] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[85] J. Lukaszuk and R. Orlandic. On accessing data in high-dimensional spaces: A comparative study of three space partitioning strategies. *Journal of Systems and Software*, 73(1):147–157, 2004.

[86] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal. Alchemi: A .NET-based Enterprise Grid Computing System. In *International Conference on Internet Computing*, pages 269–278, Las Vegas, USA, 2005.

[87] N. Mabanza, J. Chadwick, and G. Rao. Performance evaluation of Open Source Native XML databases - A Case Study. In *The 8th International Conference on Advanced Communication Technology (ICACT 2006)*, volume 3, pages 1861 – 1865, Phoenix Park, KR, 2006. IEEE Computer Society.

[88] J. Magee, N. Dulay, and J. Kramer. Structuring parallel and distributed programs. *Software Engineering Journal*, 8(2):73–82, Mar 1993.

[89] M. Mamei, F. Zambonelli, and L. Leonardi. Tuples On The Air: A Middleware for Context-Aware Computing in Dynamic Networks. In *2nd International Workshop on Mobile Computing Middleware at the 23rd International Conference on Distributed Computing Systems (ICDCS)*, pages 342–347, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[90] P. Maymounkov and D. Maziéres. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In P. Druschel, F. Kaashoek, and A. Rowstron, editors, *First International Workshop on Peer-to-Peer Systems (IPTPS '01)*, volume 2429 of *LNCS*, pages 53–65, London, UK, 2002. Springer Verlag.

[91] S. W. McLaughry and P. Wycko. T spaces: The next wave. In *HICSS '99: Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences-*, volume 8, page 8037, Honululu, USA, 1999. IEEE Computer Society.

[92] A. Milanes, N. Rodriguez, and B. Schulze. Managing jobs with an interpreted language for dynamic adaptation. In *MGC '05: Proceedings of the 3rd international workshop on Middleware for grid computing*, pages 1–6, Grenoble, France, 2005. ACM Press.

[93] C. Nieter and J. R. Cary. VORPAL: a versatile plasma simulation code. *Journal of Computational Physics*, 196:448–473, 2004.

[94] P. Obreiter and G. Gräf. Towards scalability in tuple spaces. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 344–350, Madrid, Spain, 2002. ACM Press.

[95] A. Omicini, E. Denti, and A. Natali. Agent Coordination and Control through Logic Theories. In M. Gori and G. Soda, editors, *AI*IA '95: Proceedings of the 4th Congress of the Italian Association for Artificial Intelligence on Topics in Artificial Intelligence*, volume 992 of *LNCS*, pages 439–450, Florence, Italy, 1995. Springer Verlag.

[96] A. Omicini and F. Zambonelli. Tuple centres for the coordination of Internet agents. In *SAC '99: Proceedings of the 1999 ACM symposium on Applied computing*, pages 183–190, San Antonio, USA, 1999. ACM Press.

[97] R. Orlandic and J. Lukaszuk. A Class of Region-preserving Space Transformations for Indexing High-dimensional Data. *Journal of Computer Science*, 1(1):89–97, 2005.

[98] S. Pallickara and G. Fox. Naradabrokering: A distributed middleware framework and architecture for enabling durable peer-to-peer grids. In M. Endler and D. C. Schmidt, editors, *ACM/IFIP/USENIX International Middleware Conference*, volume 2672 of *LNCS*, pages 41–61, Rio de Janeiro, Brazil, 2003. Springer Verlag.

[99] G. A. Papadopoulos and F. Arbab. Control-Driven Coordination Programming in Shared Dataspace. In V. Malyshkin, editor, *PaCT '97: Proceedings of the 4th International Conference on Parallel Computing Technologies*, volume 1277 of *LNCS*, pages 247–261, Yaroslav, Russia, 1997. Springer Verlag.

[100] G. A. Papadopoulos and F. Arbab. Coordination models and languages. Technical report, CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands, The Netherlands, 1998.

[101] G. P. Picco, A. L. Murphy, and G.-C. Roman. LIME: Linda Meets Mobility. In *International Conference on Software Engineering*, pages 368–377, Los Angeles, USA, 1999. ACM Press.

[102] T. Pitoura, N. Ntarmos, and P. Triantafillou. Replication, Load Balancing and Efficient Range Query Processing in DHTs. In Y. Ioannidis, M. H. Scholl, J. W. Schmidt, F. Matthes, M. Hatzopoulos, K. Boehm, A. Kemper, T. Grust, and C. Boehm, editors, *10th International Conference on Extending Database Technology (EDBT06)*, volume 3896 of *LNCS*, pages 131–148, Munich, Germany, 2006. Springer Verlag.

[103] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. Technical report, University of Texas at Austin, Austin, TX, USA, 1997.

[104] S. J. Plimpton, D. B. Seidel, M. F. Pasik, R. S. Coats, and G. R. Montry. A load-balancing algorithm for a parallel elecromagnetic particle-in-cell code. *Computational Physics Communications*, 152:227–241, 2003.

[105] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Brief announcement: prefix hash tree. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 368–368, St. John's, Canada, 2004. ACM Press.

[106] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, San Diego, USA, 2001. ACM Press.

[107] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: a public DHT service and its uses. *SIGCOMM Computer Communication Review*, 35(4):73–84, 2005.

[108] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design. *IEEE Internet Computing Journal*, 6(1), Jan./Feb. 2002.

[109] S. Roiser. The SEAL C++ Reflection System. In *Computing in High Energy and Nuclear Physics (CHEP '04)*, Interlaken, Switzerland, 2004.

[110] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In R. Guerraoui, editor, *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, volume 2218 of *LNCS*, pages 329–350, Heidelberg, Germany, Nov. 2001. Springer Verlag.

[111] M. Schumacher, F. Chantemargue, and B. Hirsbrunner. The STL++ Coordination Language: A Base for Implementing Distributed Multi-agent Applications. In P. Ciancarini and A. L. Wolf, editors, *COORDINATION '99: Proceedings of the Third International Conference on Coordination Languages and Models*, volume 1594 of *LNCS*, pages 399–414, Amsterdam, The Netherlands, 1999. Springer Verlag.

[112] M. Shaw, R. DeLine, D. Klein, T. Ross, D. Young, and G. Zelesnik. Abstractions for software architecture and tools to support them. *Transactions on Software Engineering*, 21(4):314–335, Apr 1995.

[113] D. Snelling. Unicore and the open grid services architecture. *Grid Computing*, pages 701–712, 2003.

[114] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, San Diego, USA, 2001. ACM Press.

[115] Sun Microsystem. *Project JXTA: A Technology Overview*, 2002.

[116] Sun Microsystem. JavaSpaces Specifications, 2005.

[117] Sun Microsystem. Jini Technology Starter Kit Specifications v2.1, 2005.

[118] E. Tanin, A. Harwood, and H. Samet. Using a distributed quadtree index in peer-to-peer networks. *The VLDB Journal*, 16(2):165–178, 2007.

[119] T. Tannenbaum, D. Wright, K. Miller, and M. Livny. Condor – A Distributed Job Scheduler. In T. Sterling, editor, *Beowulf Cluster Computing with Linux*, pages 307–350. MIT Press, October 2001.

[120] I. J. Taylor, I. Wang, M. S. Shields, and S. Majithia. Distributed computing with Triana on the Grid. *Concurrency - Practice and Experience*, 17(9):1197–1214, 2005.

[121] R. van der Goot. *High Performance Linda using a Class Library*. PhD thesis, Erasmus University Rotterdam, 2001.

[122] S. Venugopal, R. Buyya, and L. Winton. A grid service broker for scheduling distributed data-oriented applications on global grids. In *MGC '04: Proceedings of the 2nd workshop on Middleware for grid computing*, pages 75–80, Toronto, Canada, 2004. ACM Press.

[123] L. Verlet. Computer "experiments" on classical fluids. I. Thermodinamical properties of Lennard-Jones molecule. *Physical Review*, 159:98–103, 1967.

[124] G. von Laszewski, I. Foster, J. Gawor, W. Smith, and S. Tuecke. CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids. In *ACM Java Grande 2000 Conference*, pages 97–106, San Francisco, CA, 3-5 June 2000.

[125] I. Wang. P2PS (Peer-to-Peer Simplified). In *3th Annual Mardi Gras Conference Frontiers of Grid Applications and Technologies*, page 5459, Baton Rouge, USA, 2005. Louisiana State University.

[126] V. Welch, T. Barton, K. Keahey, and F. Siebenlist. Attributes, Anonymity, and Access: Shibboleth and Globus Integration to Facilitate Grid Collaboration. In *4th Annual PKI R&D Workshop*, Gaithersburg, USA, 2005.

[127] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. Security for Grid Services. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, page 48, Washington, DC, USA, 2003. IEEE Computer Society.

[128] Q. Xia, R. Yang, W. Wang, and D. Yang. Fully Decentralized DHT based Approach to Grid Service Discovery using Overlay Networks. In *Proc. of 5th International Conference on Computer and Information Technology (CIT 05)*, pages 1140–1045, Shanghai, China, 2005. IEEE Computer Society.

[129] A. Yamin, J. Barbosa, I. Augustin, L. Silva, R. Real, C. Geyer, and G. Cavalheiro. Towards Merging Context-Aware, Mobile and Grid Computing. *International Journal of High Performance Computing Applications (JHPCA)*, 17(2):191–203, 2003.

[130] J. Yu and R. Buyya. A Novel Architecture for Realizing Grid Workflow using Tuple Spaces. In *5th International Workshop on Grid Computing (GRID 2004)*, pages 119–128, Pittsburgh USA, 2004. IEEE Computer Society.

[131] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec.*, 34(3):44–49, 2005.

[132] J. Yu, S. Venugopal, and R. Buyya. A Market-Oriented Grid Directory Service for Publication and Discovery of Grid Service Providers and their Services. *The Journal of Supercomputing*, 36(1):17–31, 2006.

[133] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical report, University of California at Berkeley, Berkeley, CA, USA, 2001.

[134] C. Zheng, G. Shen, S. Li, and S. Shenker. Distributed segment tree: Support of range query and cover query over dht. In *5th International Workshop on Peer-to-Peer Systems*, Santa Barbara, USA, 2006.

Sites

[135] S. Adler. The slashdot effect – an analysis of three Internet publications. http://ssadler.phy.bnl.gov/adler/SDE/SlashDotEffect.html, 1999.

[136] BISON Project. Peersim 1.0.2. http://peersim.sourceforge.org, 2007.

[137] EGEE. gLite – Lightweight Middleware for Grid Computing. http://glite.web.cern.ch/glite/, 2006.

[138] M. Forum. Message Passing Interface Specification 1.1. http://www.mpi-forum.org/docs/mpi1-report.pdf, 1995.

[139] GigaSpaces Technologies Ltd. Space-Based Architecture and The End of Tier-based Computing. http://www.gigaspaces.com, 2006.

[140] gSOAP Library. http://www.cs.fsu.edu/~engelen/soap.html, 2007.

[141] IBM, BEA Systems, Microsoft, SAP AG, and Siebel Systems. Business Process Execution Language for Web Services version 1.1 Specification. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf, 2003.

[142] K*Grid Project. http://gridcenter.or.kr, 2005.

[143] OASIS Open. Web Services Base Notification Specification 1.3. http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf, 2006.

[144] OASIS Open. Web Services Resource Specification 1.2. http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf, 2006.

[145] Platform Computing Corporation. Platform LSF Specification. www.platform.com, 2007.

[146] G. D. Reis. Compile Time Reflection for C++. http://gcc.gnu.org/projects/cxx-reflection/, 2007.

[147] Sun Microsystem. Sun Grid Engine. http://www.sun.com/software/gridware/, 2007.

[148] TCP Linda. http://www.lindaspaces.com, 2006.

[149] W. L. Tharaka Devadithya, Kenneth Chiu. XCppRefl Project. http://www.extreme.indiana.edu/reflcpp/, 2007.

[150] Trolltech Corporation. Qt 4. http://www.trolltech.com, 2007.

[151] Veridian Systems, Inc. OpenPBS v2.3: The Portable Batch System Software. www.OpenPBS.org, 2007.

[152] G. von Laszewski and M. Hategan. Grid Workflow - An Integrated Approach. http://www.mcs.anl.gov/~gregor/papers/vonLaszewski-workflow-draft.pdf, 2005.

[153] W3C. SOAP Messages with Attachments. http://www.w3.org/TR/SOAP-attachments, 2000.

[154] W3C. SOAP Message Transmission Optimization Mechanism. http://www.w3.org/TR/soap12-mtom/, 2005.

[155] XStream Library. http://xstream.codehaus.org, 2007.