

UFRJ - UNIVERSIDADE FEDERAL DO RIO DE JANEIRO  
COPPE - COORDENAÇÃO DOS PROGRAMAS DE PÓSGRADUAÇÃO DE ENGENHARIA  
PESC - PROGRAMA DE ENGENHARIA DE SISTEMAS E COMPUTAÇÃO

## **PARALLEL STRATEGIES FOR PROCESSING SCIENTIFIC WORKFLOWS**

Luiz Antônio Vivacqua Corrêa Meyer  
Marta Lima de Queirós Mattoso

RIO DE JANEIRO, RJ – BRAZIL

**APRIL, 2004**

## GENERAL INDEX

---

<b>I - INTRODUCTION.....</b>	<b>5</b>
<b>II – TECHNOLOGICAL INFRASTRUCTURE.....</b>	<b>8</b>
II.1 - GRID COMPUTING.....	8
II.1.1 – <i>Globus Toolkit</i> .....	9
II.1.2 - <i>Grid Data Access</i> .....	10
II.2 – SCIENTIFIC WORKFLOWS.....	12
II.2.1 - <i>DAGMAN APPLICATIONS</i> .....	14
II.2.2 – <i>BPEL4WS</i> .....	15
II.2.3 – <i>GSFL</i> .....	16
II.2.4 – <i>DAML-S</i> .....	17
II.2.5 – <i>XScufl</i> .....	18
<b>III – RELATED WORK.....</b>	<b>21</b>
II.1 - WASA.....	22
II.2 - METEOR.....	22
II.3 - SDM.....	23
II.4 – gRNA.....	25
II.6 – SRMW.....	26
II.7 - CHIMERA.....	27
II.8 - PEGASUS.....	29
II.9 - MYGRID.....	30
II.10 - GRIDFLOW.....	30
II.11 – DISCUSSION.....	31
<b>IV – WORKFLOW PARALLEL PROCESSING.....</b>	<b>34</b>
IV.1 - WORKFLOW PARALLEL STRATEGIES.....	34
IV.2 – WORKFLOW PARALLEL DESIGN & EXECUTION.....	40
IV.3 – INITIAL HEURISTICS.....	42
<b>V – PRELIMINARY RESULTS.....</b>	<b>44</b>
V.1 – MHOLLINE WORKFLOW.....	44
V.2 PARALLEL MHOLLINE DESIGN.....	45
V.3 EXPERIMENTAL RESULTS.....	45

## Figure Index

---

Figure 1 - OGSA Grid Service [25].....	8
Figure 2 - Globus toolkit main services [26].....	9
Figure 3 - Grid file data access.....	10
Figure 4 - Grid Database Access [25] .....	11
Figure 5 - DAGMAN input file and respective Diamond DAG (adapted from [33]).....	14
Figure 6 - BPEL Document Structure (adapted from [34]) .....	15
Figure 7 – Web Services and GSFL Workflow Models [36] .....	17
Figure 8 - Process modeling Ontology [37].....	18
Figure 9 – Xscufl example (adapted from [61]).....	19
Figure 10 - WASA Architecture (adapted from [42]).....	22
Figure 11 - Meteor architecture [43].....	23
Figure 12 - Scientific Workflow Management System architecture [45] .....	24
Figure 13 - BioOpera Architecture [47] .....	25
Figure 14 – SRMW Architecture [58] .....	26
Figure 15 - VDC main components [15].....	27
Figure 16 - VDL specification and the correspondent acyclic graph (adapted from [16]) .....	28
Figure 17 - Configuration of Pegasus when driven by Chimera [50].....	29
Figure 18 - GridFlow architecture [53].....	31
Figure 19 – Inter-workflow parallelism with grouped input data distribution .....	35
Figure 20 - Inter-workflow parallelism with individual input data distribution .....	35
Figure 21 - Intra-Workflow Parallelism for a pipeline of programs (sequential pattern).....	36
Figure 22 – Intra-program parallelism for input data processing independently .....	37
Figure 23 - Intra-Program Parallelism with dataset partition .....	37
Figure 24 – The theoretical $Wf_1$ workflow.....	38
Figure 25 – Inter-workflow strategy to execute $Wf_1$ .....	38
Figure 26 – First intra-workflow strategy to execute $wf_1$ .....	39
Figure 27 - Second intra-workflow strategy to execute $wf_1$ .....	39
Figure 28 – Workflow parallel execution architecture.....	41
Figure 29 - The parallel workflow design architecture.....	41
Figure 30 - WFDB Class Diagram.....	42
Figure 31 - Parallel Workflow Execution Architecture.....	42
Figure 32 - The MholLine Workflow components .....	44
Figure 33 - Implemented Architectures .....	45
Figure 34 – Inter workflow parallelism strategy with grouped and circular distribution.....	46
Figure 35 – Intra workflow parallelism strategy and for the three executions together..	47

*Table Index*

---

<i>Table 1 – Basic workflow control patterns [54].....</i>	<i>13</i>
<i>Table 2 – Primitive activities.....</i>	<i>16</i>
<i>Table 3 – Structure activities.....</i>	<i>16</i>
<i>Table 4 – Workflow Projects General Aspects.....</i>	<i>32</i>
<i>Table 5 – Estimated time to process <math>Wf_1</math> in a homogeneous environment.....</i>	<i>39</i>
<i>Table 6 - Estimated time to process <math>Wf_1</math> in a heterogeneous environment.....</i>	<i>40</i>
<i>Table 7 – Workflow patterns and respective heuristics.....</i>	<i>43</i>
<i>Table 8 - Schedule for 2004.....</i>	<b><i>Erro! Indicador não definido.</i></b>
<i>Table 9 - Schedule for 2005.....</i>	<b><i>Erro! Indicador não definido.</i></b>
<i>Table 10- Schedule for 2006.....</i>	<b><i>Erro! Indicador não definido.</i></b>

# I - INTRODUCTION

---

Scientific resources like programs and data present characteristics of distribution, heterogeneity and in some cases huge volume of information. Multiple sites [1, 2, 3, 4, 5, 6] make their data and programs available through the use of customized interfaces allowing scientists to submit their work. One alternative to this remote use is downloading the data and the programs and installing them in their own site. Frequently, scientists need to combine these resources to perform a higher-level function. In these cases, programs and data are composed in an execution chain such that the output of a program execution can be used as the input of another. As observed by Deelman et al [7], scientific communities like physicists, astronomers and biologists are no longer developing applications as monolithic codes. Instead, standalone components are being combined to process data. In this scenario, scientific applications can be viewed as scientific workflows.

Scientific programs often generate and process large datasets [52]. Therefore, one problem that can arise when running scientific workflows is the time needed to accomplish their execution. The execution of some components and consequently the entire workflow can be a time consuming task. Thus, in order to enhance the performance of scientific workflows parallelism can be exploited.

One possible way to exploit parallelism is to use a parallel machine. However, this solution presents some drawbacks like the high costs for hardware and software acquisition. Another possibility is to use a distributed environment like a PC cluster, which can provide computation performance equivalent to a parallel machine but with much lower costs. A third alternative can be the use of Computational Grids [14]. Grids are emerging as platforms for higher performance and for integration of networked resources. A Grid can be defined as a virtual environment where distributed and heterogeneous resources, owned by independent organizations, can be shared and aggregated to form a virtual computer.

One typical scenario in scientific applications is having a set of input data to be processed by a workflow. Thus, one question that can be made is how to process this scientific workflow in parallel. The adoption of the traditional parallel techniques to scientific workflows may not apply. Firstly many scientific workflow components are legacy programs. Therefore, it is not possible to modify their code and they have to be treated like black box components. Secondly, although applying data parallelism for each program individually is straightforward, this may not lead to the best solution. There are many alternatives to execute a scientific workflow in parallel because programs and data can be distributed among the nodes in many ways. Choosing the best strategy for parallel execution is difficult because this choice must consider:

- The dependencies among the components;
- The unbalanced execution time of the programs;
- The different size of datasets;

- The computational resources available.

The choice of best execution alternative is even more difficult in a Grid than in a parallel machine or a PC cluster due the heterogeneity and the dynamic nature of the Grid environment. Although services mechanisms for information, security, resource management and data management are already available in a Grid environment, according to Foster et al [18], choosing the best strategy for a workflow execution in a Grid is a challenging research area. Decisions to replicate procedures and datasets have to be taken either on demand or pre-staging in order to provide better performance. However, as pointed out by Blythe et al [20], finding an optimal allocation of processors for tasks in a workflow is NP-hard and tools must focus on finding reasonable heuristics or on identifying families of problems that can be solved efficiently.

Besides defining a parallel strategy for a workflow execution, there is also a need to manage the parallel execution of the workflow, to perform the initial distribution of the work among the nodes and also to collect and re-distribute the partials results in order to have the workflow processed.

The complex aspects involved in a workflow parallel processing point to the need to develop tools to provide users with a workflow parallel design and a workflow parallel execution. The development of such tools can make scientists independent from parallel processing specialists in order to have their scientific workflows executed with better performance.

Many initiatives to enhance the performance of scientific applications can be found in the literature. In the bioinformatics area for example, there are several studies [10, 11, 12, 13, 41] addressing the parallel execution of bioinformatics programs in PC cluster and Grid environments using data and program parallelism techniques. However, these works deal with the parallelism of individual programs. There are also many projects [42, 43, 45, 46] that enable the design and execution of scientific workflows, but with no exploitation of parallel processing. Workflow parallel processing is showed in Chimera [18] and GridFlow works [53]. However, in [18] details regarding how the workflow parallel design was achieved are not given, and in Gridflow, parallelism seems to be achieved only if a workflow component is already a parallel program.

Scientific workflows can benefit from both data and program parallelism. The combined use of data and program parallelism can improve the execution of: multiple instances of a workflow, one instance of a workflow and a single component of a workflow. However, it is mandatory to have an adequate workflow execution strategy.

Database Management Systems (DBMS) have been using parallel processing to achieve better performance in their operations. The parallel hardware allows DBMS to use inter-query parallelism, intra-query parallelism and intra-operation parallelism. In the former case, many queries can be executed at the same time, each by one respective processor. In the intra-query parallelism, also called “pipelined”, the output of one operator is streamed into the input of another operator so both operators can work in parallel within the same query or, if two operators from the same query are independent

they can also execute in parallel. Finally, by partitioning the table among multiple processors, one single operation from a query can execute in parallel, with each process running the same operation but working on a specific part of the table. This partitioned data with parallel execution characterizes the intra-operation or partitioned parallelism [19].

In the same way, a parallel hardware can be used to gain better performance to execute scientific workflows. Like a relational query, formed by a set of operators that can communicate their results, a workflow is composed by a chain of programs, in general processing the antecessor results. So an analogy to parallel processing in DBMS can be done, and three strategies of parallelism can be accomplished when executing workflows: “inter-workflow”, “intra-workflow” and “intra-program” parallelism.

The main goal of our work is to provide a software layer that can propose and execute a parallel plan for a scientific workflow. We intend to do experiments with different forms of parallel strategies, in order to propose heuristics to automate the design and execution of scientific workflows. The idea is to provide non-specialists in parallel processing a parallel solution to execute their scientific workflow.

The rest of this work is organized as follows. Section two provides some background about the technologies regarding Grid computing and scientific workflows. Section 3 discusses the main aspects of projects that deal with definition and execution of scientific applications in Cluster, Grid and Web environments. The fourth section characterizes workflow parallel processing and presents guidelines of heuristics to execute workflow components. The fifth section presents experimental results obtained for MholLine [21], a structured genomic workflow used in our experiments. Finally, section 6 shows the planning of this work for the next two years.

## II – TECHNOLOGICAL INFRASTRUCTURE

### II.1 - GRID COMPUTING

Grid computing has been described using an analogy to a power grid. When we turn on some electrical device, we do not know where the electrical power comes from. The local utility company provides the interface into a complex network of generators and power sources providing us with an acceptable quality of service for our energy demands [22]. The vision of Grid computing is similar. Diverse geographically distributed computing resources can be aggregated to form a virtual computer, but this resources are not visible to the user just as the consumer of electric power does not know how their electricity is being generated. Grid technologies and infrastructures support the sharing and coordinated use of these resources.

The Global Grid Forum (GGF) [23], is a community forum of researchers and practitioners. According to its home page, GGF aims “to promote and support the development, deployment and implementation of Grid technologies and applications via the creation and documentation of ‘best practices’ – technical specifications, user experiences and implementation guidelines”. GGF contains several area groups and, within these areas, working groups dealing with a particular Grid-related problem. The current areas are information services, security, scheduling and management, performance, architecture, data, and applications and models.

One GGF specification is the Open Grid Service Architecture (OGSA) [24]. The core of OGSA architecture is the Grid Service [25], which may be a computational resource, storage resource, program or database. A Grid service is defined as a Web Service that provides a set of well-defined interfaces and that follows specific conventions. The interfaces address discovery, dynamic service creation, lifetime management, notification and manageability. The conventions address naming and upgrade ability.

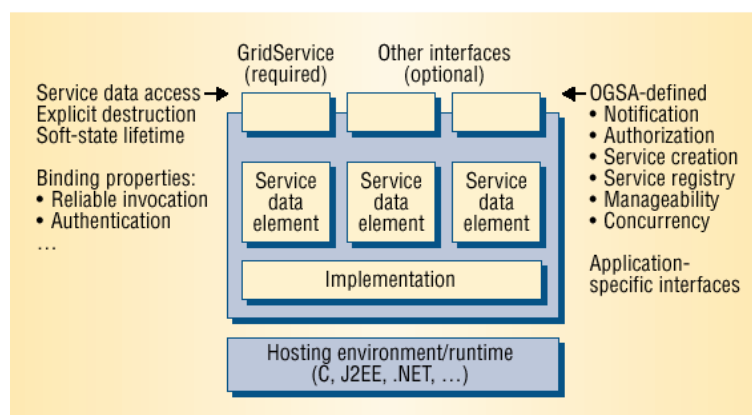


Figure 1 - OGSA Grid Service [25]



Figure 1 shows on the top a set of interfaces defining a Grid service. The mandatory *GridService* interface defines an operation *FindServiceData*, for querying and retrieving service data. Associated with each grid service interface is a set of service data elements, which provide a standard representation for information about Grid service instances. Finally, a Grid Service can be implemented in a variety of ways and host it in different environments. Nothing in grid service specification imposes how Grid Services are written, what operating system they run on, what languages they use or the programming model they conform to.

### II.1.1 – Globus Toolkit

The Globus alliance [26] is a joint effort of researchers and developers from around the world to develop the fundamental technologies needed to build computational Grids. The alliance provides software tools that make it easier to build computational Grids and Grid-based applications. These tools are called the Globus Toolkit. The composition of the Globus Toolkit can be pictured as the following three pillars showed in figure2.

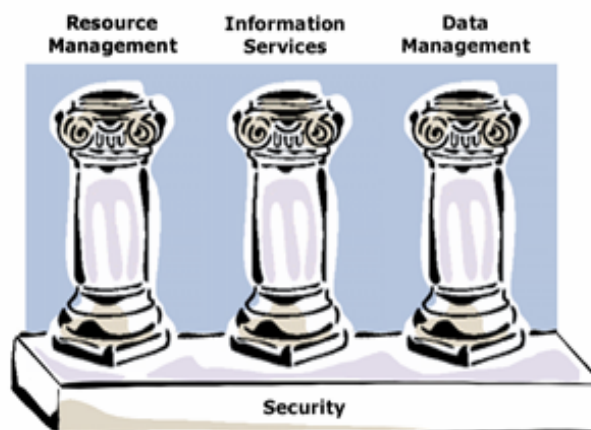


Figure 2 - Globus toolkit main services [26]

Security is the foundation common to all three pillars. The first pillar of the Globus Toolkit provides *Resource Management*, which involves the allocation of Grid resources. It includes such packages as the Globus Resource Allocation Manager (GRAM) and Globus Access to Secondary Storage (GASS). The second pillar of the Globus Toolkit is for *Information Services*, which provide information about Grid resources. The third pillar of the Globus Toolkit is for *Data Management*, which involves the ability to access and manage data in a Grid environment. This involves such utilities as GridFTP and the Reliable File Transfer (RFT) service, which are used to move files between Grid-enabled devices.

## II.1.2 - Grid Data Access

Data, for many kinds of grid applications, are stored in flat files. In order to provide better performance for the user community, these files are replicated among different sites. One typical scenario depicted for this kind of data access is shown in figure 3. An application wanting to access some specific information looks for it in a metadata service that returns the logical name of the file containing the desired data. Having the knowledge of this logical name, the application can interact with a replication service that is responsible for maintaining a replica catalog. The replica catalog associates a logical file name to its physical file names. At this point, the application can access directly the file or requests a replica selection service to choose the most appropriate replica to be accessed. At last, the file should be transferred using some type of transport protocol.

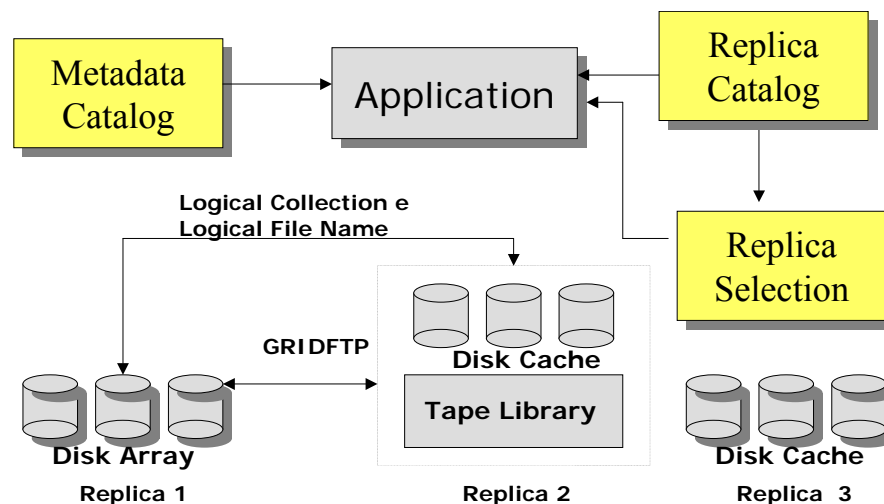


Figure 3 - Grid file data access

Although efficient for applications that need to process an entire file, this approach seems not to be adequate in situations where there are relationships between records stored in different files or for applications that only need to process a subset of the records. In the first case, if only one of these files is transmitted, retrieving the associated records may not be possible, since the associated records were not transferred too. In the second case, transmission of the whole dataset instead of the part of it can generate performance troubles.

Although most scientific applications that use grids store their data in files, there are also many other e-Science projects with an urgent need for the interconnection of pre-existing and independent databases. The GGF Data Access and Integration Services (DAIS-WG) working group, seeks to promote standards for the development of grid database services, focusing principally on providing consistent access to existing, autonomously managed databases. The OGSA-DAI project [27] implements an

architecture based on Grid services for generic data access. Three kinds of Grid services are implemented: Grid Data Service Registry (GDSR), Grid Data Service Factory (GDSF) and Grid Data Service (GDS). GDSR maintains a collection of Grid Service Handles for a set of GDSF and GDS. GDS provides the point of access to data sources and can be persistent or transient. GDSF is a specialized factory service, which can create new GDS.

Figure 4 illustrates a situation in which an application wants to discover, acquire and employ remote services to create a new database using data from a number of online databases.

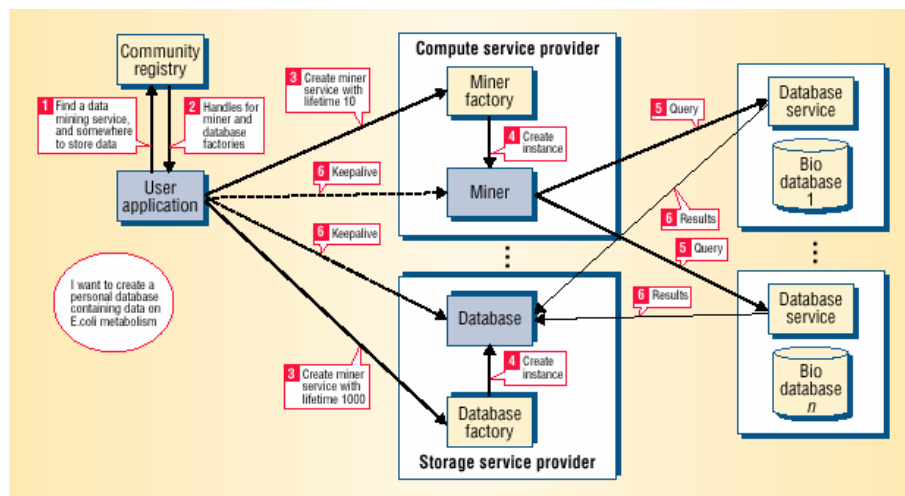


Figure 4 - Grid Database Access [25]

Initially, the application contacts the registry service to identify service providers who can provide the required data and storage capabilities. The registry returns handles identifying a miner factory and a database factory maintained by service providers that meet application requirements. In the third step, the application issues requests to both factories specifying details such as the operation to be performed, the form of the database to be created to hold results and initial lifetimes for the two new service instances. Then, the two new service instances are created. The miner service initiates queries against appropriate remote databases and results are returned to the newly created database.

There are many other works focusing the problem of grid data access. The Storage Resource Broker (SRB)[28] is a client-server based middle-ware implemented at SDSC to provide uniform access interface to different types of storage devices. It employs a DBMS to store all its metadata and provides access to data stored in many types of file systems and DBMSs. It also provides capabilities to store replicas of data. The Spitfire project [29], developed in the context of the European Data Grid Project (EDG), provides a uniform way to access many Relational DBMSs through standard Grid protocols and Grid interfaces. The Spitfire middleware mediates between a RDBMS and a Grid client,

converting HTTP requests made by the client into JDBC requests to the DBMS, and mapping the results from the DBMS into an XML output to the client. Work is in progress to evolve Spitfire towards a Grid web service.

## II.2 – SCIENTIFIC WORKFLOWS

According to The Workflow Management Coalition (WFMC) [30], *workflow* is “the computerized facilitation or automation of a business process, in whole or in part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules”. A *Workflow Management System* (WfMS) is “ a system that completely defines, manages and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic. It consists of a modelling as well as a runtime component”.

Although the above definitions make reference to “business process”, workflow is not only employed by business applications. Scientists also use workflows in order to build their scientific applications. However, there are differences between business and scientific workflows. Medeiros et al [42] identified that in a scientific environment, scientists will typically specify their workflows themselves, while in a business environment, a system administrator is commonly responsible for this task. Another characteristic of scientific workflows pointed in their work is the need to trace workflow executions. A scientist may need to reuse a workflow in order to reproduce results. The operations a user performs on given data must be recorded in order to provide scientists with the benefits of successful and unsuccessful workflows.

The characteristics, requirements and differences between business and scientific workflows are still being discussed. In two recent events, the Scientific Data Management Framework Workshop [59] and the e-Science Workflow Services [60], scientific workflow issues like workflow representation, parallelism, service composition, separate runtime & language, service description, mapping to resources, relation to distributed data queries, implementation, optimization, and the relation to business workflow languages, among others were debated. The goal of these two international events was to bring together researchers in these fields to discuss position papers about the projects being developed, the current state of the art, and to identify requirements for scientific data management and for scientific workflows.

As pointed out in these events, scientific workflows are more data-flow oriented while business workflows are more control-flow oriented. Business workflows require the coordination of a number of small messages and document exchanges. In scientific workflows no documents undergo modifications. Instead, often a dataset is obtained via analysis of another dataset. Business workflows need complex control flow, but they are not data-intensive pipelines. On the other side, scientific workflows must deal with the heterogeneity, complexity, volume, and physical distribution of scientific data. In addition to these data problems, scientific workflows, in particular bioinformatics






workflows, often deal with legacy or third-party programs, which are also heterogeneous, and with no source code available.

The specification of a scientific workflow is often done through the use of some customized graphical interface, allowing the scientist to select and to compose the workflow components, or by the use of a workflow language. The design of the workflow is expressed in terms of an *abstract workflow* and in terms of a *concrete workflow*. A concrete workflow defines the application by program executables and data files. A *workflow engine* is a software that executes a concrete workflow. By the use of different semantic levels, abstracts workflows can specify the components in a way that is abstracted from the syntactic details of data formats or invocation mechanisms. For example, in an abstract bioinformatics workflow, a user would specify a component that can perform a gene sequence homology task and when generating a concrete workflow an executable program that performs such task would be assigned. In another semantic level, still abstract, a user would specify the components of a workflow by their logical names like Blast-p and PDB. However, in both cases, there is a need to have catalogs or repositories in order to map the abstract components of a workflow into the concrete ones, for example NCBI Blast-p and NCBI PDB version 5.2.3. Anyway, in the context of this work, we will assume the following definitions:

- *Abstract workflow* is a set of programs and datasets expressed by their logical names along their dependencies among each other. We can have several semantic levels of abstract workflows.
- *Concrete workflow* is a set of program executables and data files expressed by their physical name, that is, their full location. A logical name does not characterize a concrete workflow, since a program or dataset can be assigned to multiple physical names. This is the lowest level of a workflow definition, prior to its execution.

Workflows have their components modeled according to a number of patterns. These patterns address distinct workflow functionalities. According to Aalst et al [54], almost every workflow language supports the basic control patterns listed bellow.

**Table 1 – Basic workflow control patterns [54]**

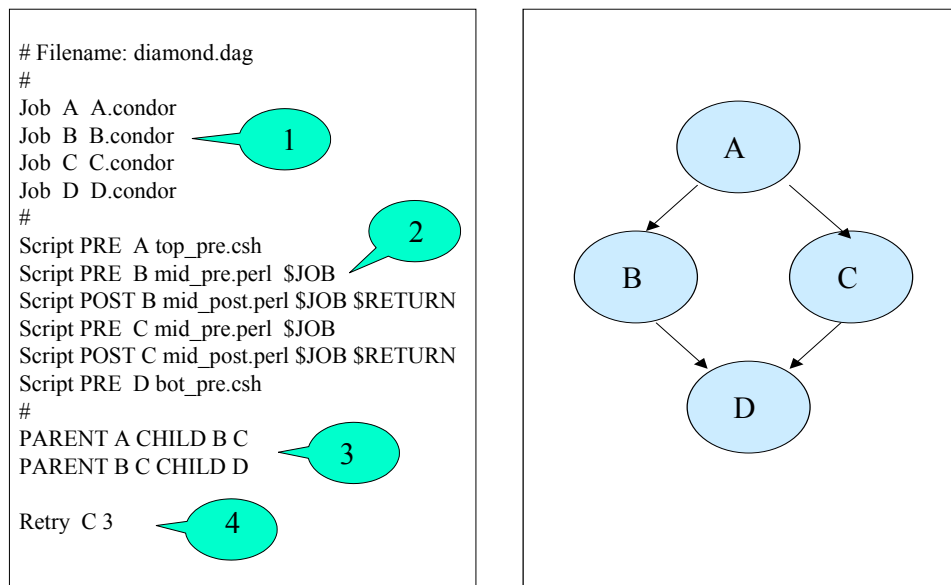
PATTERN	NAME	DESCRIPTION
	Sequential	The execution of a program can be done after the execution of the predecessor
	Parallel Split	Programs B and C have to be executed after program A
	Exclusive Choice	Program B or program C must be executed after program A
	Synchronization	Program C can only executes when programs A and B finished
	Simple Merge	Program C can execute if program A or program B finish

In the next subsection, we show some existing technologies that address the design of scientific workflows for Grid and web environments. There is no established consensus regarding which is the best language to specify a scientific workflow. In general each project uses a specific workflow design tool or workflow language. The Grid scientific workflows are being expressed in terms of jobs or web services. Since OGSA defines Grid services as extensions to web services, languages developed to compose business web services can also be employed to compose scientific web services in Grid environments.

## II.2.1 - DAGMAN APPLICATIONS

A directed graph [31] consists of a set of vertices and a set of arcs. A Directed Acyclic Graph, or DAG, is a directed graph with no cycles and can be used to represent a scientific workflow. The programs are the nodes (vertices) in the graph and the edges (arcs) represent the dependencies.

Condor [32] is a specialized batch system for managing compute-intensive jobs that uses the computer power of workstations that communicate over the network. Users submit their jobs to Condor, Condor puts the jobs in a queue, runs them, and then informs the user as to the result. Condor finds machines for the execution of programs, but it does not schedule programs (jobs) based on dependencies.



**Figure 5 - DAGMAN input file and respective Diamond DAG (adapted from [33])**

The Directed Acyclic Graph Manager (DAGMan) [33] is a meta-scheduler for Condor jobs. DAGMan submits jobs to Condor in an order represented by a DAG and processes the results. The DAG itself is defined by the contents of a DAGMan input file. The input file used by DAGMan specifies four items:

1. A list of the programs in the DAG,
2. Pre and Pos processing that takes place for each program submission,
3. The description of the dependencies in the DAG,
4. Number of retries in case of fails.

Figure 5 shows an example of a DAG and its DagMan input file. DAGMan allows Interworkflow parallelism to be achieved if the workflow input data is partitioned in advance. Intraworkflow parallelism can be achieved through the use of the descriptions of the dependencies in the DAG and by allocating concurrent programs to different nodes. DAGMan has been used to execute scientific workflows in Grid environments but it does not deal with the workflow for web services. However, the concept of using a DAG to represent a set of programs where the inputs, outputs and the execution are inter-dependent can be applied to describe the dependencies between web services.

### II.2.2 – BPEL4WS

Business Process Execution Language for Web Services (BPEL4WS) [34] represents a convergence of the ideas in XLANG and WSFL specifications, developed by Microsoft and IBM respectively. The BPEL Process (workflow) is an XML-based grammar that can be executed by an Orchestration engine like BPWS4J [35] from IBM. This engine reads the BPEL document and invokes the web services in the order required by the workflow. The workflow itself is a web service and is invoked as such.

```

<process name="process1"
  <partners>
    <partner name="client", .../>
    <partner name="provider", .../>
  </partners>

  <containers>
    <container name="request" .../>
    <container name="response" .../>
  </containers>

  <sequence>
    <receive name="receive1" partner="client" ..../>
    <invoke name="invokeservice"
      inputContainer="request"
      outputContainer="response"
      ...
    </invoke>
    <reply name="reply" partner="client" ..../>
  </sequence>
</process>

```

Figure 6 - BPEL Document Structure (adapted from [34])

The main structure of a BPEL document consists of the following elements: partners for interaction, containers for holding messages and a set of activities. The definition of a process begins with the *<process>* element. The next step is to declare the parties involved. These partners send and receive messages that are accessed by activities like receive and reply. In BPEL this data is stored in data containers. Once the partners and containers are defined, the activities that form the composition can be added to the document. Figure 6 illustrates the structure of a BPEL document that involves two parties: a client and a service provider.

BPEL4WS documents have two types of activities: primitive activities (table1) and structure activities (table2). The first type are low level activities representing the work in the process and the second type activities have the ability to define an ordered sequence of steps, to indicate that a collection of steps should be executed in parallel, to define a loop, to execute one of several paths, and to execute one of several alternative paths.

BPEL4WS process itself does not indicate how a partner is bound to a specific service. That is considered a runtime-binding step that must be supported by the BPEL4WS implementation.

**Table 2 – Primitive activities**

<i>&lt;invoke&gt;</i>	Invoke the operation of a web service
<i>&lt;receive&gt;</i>	Wait for a request
<i>&lt;reply&gt;</i>	Generate a response
<i>&lt;wait&gt;</i>	Wait for some time
<i>&lt;assign&gt;</i>	Copy data values
<i>&lt;throw&gt;</i>	Throw an exception
<i>&lt;catch&gt;</i>	Catch an exception
<i>&lt;terminate&gt;</i>	Finish the entire process instance
<i>&lt;empty&gt;</i>	Do nothing

**Table 3 – Structure activities**

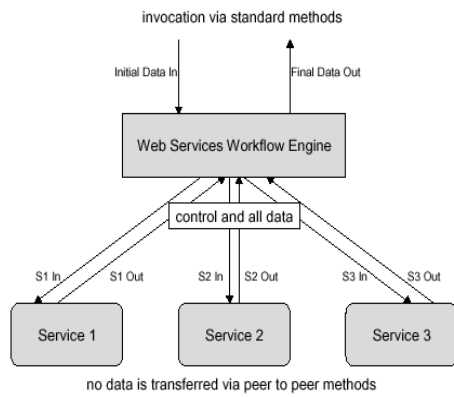
<i>&lt;sequence&gt;</i>	Sequential execution of primitive activities
<i>&lt;flow&gt;</i>	Parallel execution of primitive activities
<i>&lt;switch&gt;</i>	Case-statement approach
<i>&lt;while&gt;</i>	Defines a loop
<i>&lt;pick&gt;</i>	Executes one of several paths
<i>&lt;scope&gt;</i>	Groups a set of activities as a single transaction

### II.2.3 – GSFL

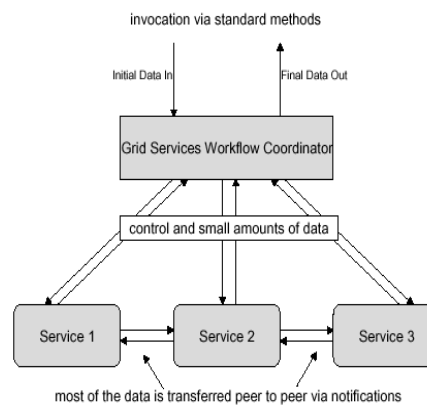
The Grid Services Flow Language (GSFL) [36] is an XML based language that allows the specification of workflow descriptions for Grid services in the OGSA framework. Differently from BPEL4WS, which defines a workflow in such a way that the workflow engine has to coordinate the execution of each web service, GSFL workflow specification allows the communication between the services. According to GSFL authors, Grid services usually exchange large amounts of data and thus, having a central workflow engine to distribute data between the services is a bad idea. Figure 8 depicts the both models.



Web Services Workflow Model



GSFL Workflow Model



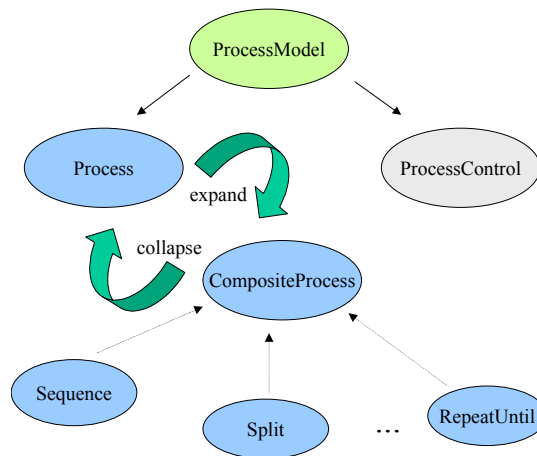
**Figure 7 – Web Services and GSFL Workflow Models [36]**

The main components of GSFL language are: Service providers, Activity model, Composition model and Lifecycle model. All services that are part of the workflow have to be specified in the list of service providers. They have a unique name and can be located using the locator element. The activity model lists all the operations belonging to the individual service providers. The composition model describes how the different Grid services are composed to form a new Grid service. It describes the control and data flow between various operations of the services, and also direct communication between them in a peer-to-peer way. The lifecycle model addresses the order in which the services are supposed to execute.

GSFL is a work in progress at Argonne National Laboratory in the context of Globus project. However there is not much documentation about it nor real experiments describing GSFL use. According to the language authors, ideas for future features include automatic integration with a graphical workflow editor and the development of constructs such as loops and switch statements.

## II.2.4 – DAML-S

DAML (DARPA Agent Markup Language) [37] goal is to develop a language and tools to facilitate the concept of the Semantic Web. HTML allows people to visualize the information on the web, but it does not provide much capability to describe the information in ways that facilitate the use of software programs to find or interpret it. XML allows information to be more accurately described using tags and DAML language is being developed as an extension to XML and the Resource Description Framework (RDF) which provide a lightweight ontology system to support the exchange of knowledge on the Web. The latest release of the language is named DAML+OIL.



**Figure 8 - Process modeling Ontology [37]**

DAML-S [38] is an ontology developed to describe web services and process (workflows) based on web services. The ontology to describe processes is illustrated in figure 8. A *Process* can have any number of inputs, outputs, participants and preconditions. It has a subclass *CompositeProcess*, which in turn has a variety of subclasses of control structures. Composite processes are processes that have additional properties called components to indicate the ordering and conditional execution of the subprocesses from which they are composed. There are two fundamental relations between processes and composite processes. The EXPAND relation associates a *Process* with the *CompositeProcess* describing its component subprocesses, while its inverse, the COLLAPSE relation represents the association of the *CompositeProcess* to its atomic *Process* form. Expanding is intended to provide a “glassbox” and collapsing a “blackbox” view of the process. The subclasses of control structures enable the execution of processes in sequence, parallel, with determined order, without order, satisfying conditions, etc.

DAML-S is an initiative of the Semantic Web community to facilitate automatic discovery, invocation, composition, interoperation and monitoring of web-services through their semantic description. However the language presents some drawbacks [39]. First, there are few and artificially examples about its use. Second, it is necessary to know DAML, WSDL and SOAP to start writing DAML-S descriptions, making its use difficult.

## II.2.5 – XScufl

XScufl [61] is the XML dialect of the Simple Conceptual Unified Flow Language. The language was developed to be used in myGrid project, in order to fulfill the

workflow requirements established by the project. A Xscufl file consists of the following tags: <processor>, <link>, <source>, <sink> and <coordination>. A processor tag defines a single processing step. A link tag defines a data link between two processors. A data link represents a flow of information of some processor output by an input of some other processor. A source tag defines a workflow source and is used to get input data to the workflow. A Sink tag is similar to the source tag and is used to send a processor output data to the workflow sink, making it visible outside the workflow. The coordination tag is used to restrict the execution of processors when there are no data dependencies between them.

Figure 9 illustrates the structure of an XScufl document that involves three processors in a sequential pattern. The coordination block states that processor “processor3” should only be allowed to transition from “scheduled” to “running” if the processor “processor2” has achieved status “completed”. The available states are: Scheduled, Running, Completed and Aborted.

```

<?xml version="1.0" encoding="UTF-8" ?>
  <s:scufl xmlns:S="http://org.embl.ebi.escience/xscufl/0.1alpha" version="0.1"
    log="3">
    <s:processor name="processor1" />
    <s:processor name="processor2" />
    <s:processor name="processor3" />
    <s:link>
      <s:input="processor1" .../>
      <s:output="processor2" .../>
    </link>
    <s:link>
      <s:input="processor2" .../>
      <s:output="processor3" .../>
    </link>
    <s:source>input_file</s:source>
    <s:sink>output_file</s:sink>
    <s:coordination name="testcoordination">
      <s:condition>
        <s:target>processor2</s:target>
        <s:state>completed</s:state>
      </s:condition>
      <s:action>
        <s:target>processor3</s:target>
        <s:stagechange>
          <s:from>scheduled</s:from>
          <s:to>running</s:to>
        </s:stagechange>
      </s:action>
    </s:coordination>
  </s:scufl>

```

Figure 9 – Xscufl example (adapted from [61])

According to Xscufl authors, the decision of myGrid to design the language was taken to address the support for a set of requirements, like cost, quality, semantic level specification and provenance that other languages do not offer. However, Xscufl is proprietary solution and can only be executed by the Freefluo [62] workflow orchestration tool.

### III – RELATED WORK

---

Cluster of PCs, Grids and the Web are being used as platforms for the execution of scientific programs and workflows. Many initiatives can be found in the literature exploiting these distributed environments to enhance the performance of scientific applications. This section describes some important works along their main characteristics related to the execution of scientific applications, particularly in the bioinformatics scenario.

In many scientific areas we can find works exploiting cluster of PCs to improve the response time of their applications. For example, in the bioinformatics field, there are several works that exploit parallelism in sequence comparison and alignment operations. Braun et al [12] explore in their work, the use of BLAST in batch mode processing of multiple query requests against a database replicated at all nodes. Meanwhile it is not showed details of the implementation and also practical results. Pappas [11] used a network of Dec Alpha workstations to set up a service for BLAST requests. The service was implemented using PVM for parallel interface. The sequence database was fragmented and accessed via NFS simulating a shared-disk configuration. Costa and Lifschitz [10] present a more accurate work where different approaches to distribute the query requests are examined. The services were implemented using MPI and were executed in a 32 PC cluster with different input databases and with replication and fragmentation polices. Their parallel algorithms show significant improvements for the sequential implementation of BLASTP, which compares protein queries to proteins databases. However, none of these works exploit parallelism to improve the performance of a combination of programs. They are focused on one single isolated program execution.

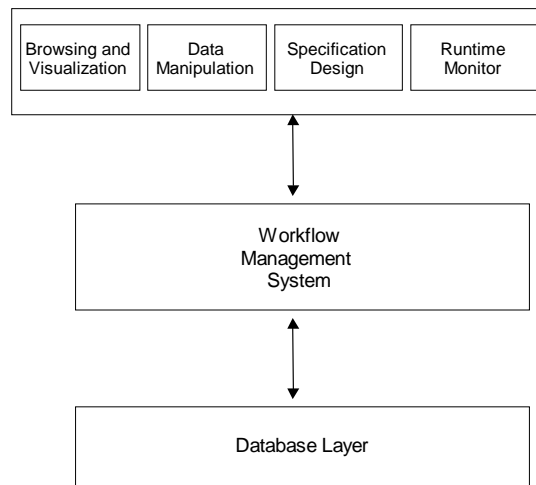
Already widely used in projects by physicists, astronomers and engineers, the grids are beginning to be used by also the bioinformatics community. The Grid Application Development Software (GrADS) project [40] is developing a framework to simplify the preparation and execution of programs on a computational Grid. Yarkhan and Dongarra [41] describe how to enable the biological sequence alignment application FastA to run on the GrADS framework. Their work adopts database replication strategies to distribute data with a master-slave approach to process the query sequences, and also improves the performance of a single application.

Since the main focus of our work is the execution of scientific workflows instead of the execution of isolated programs, we will next present in more detail, the main projects that are related to this subject. A set of characteristics was selected to be analyzed aiming to better understand the resources these works provide. We begin discussing the WASA [42] project. Although not developed for distributed environments, WASA was one of the first dealing with design and execution of scientific workflows. Next, we discuss nine projects that enable the specification and execution of scientific workflows in distributed environments: Meteor [43, 44], SDM [45], gRNA[46],

BioOpera [47, 48], SRMW[57, 58], Chimera [15, 18], Pegasus [16, 20, 49, 50], myGrid [17, 51, 52] and Gridflow [53].

## II.1 - WASA

The WASA project (A Workflow-Based Architecture to Support Scientific Database Applications) [42], proposes an architecture to integrate database and workflow technologies to support the management of scientific experiments in domains of geosciences and biocomputing. Figure 10 sketches the main components of WASA architecture.



**Figure 10 - WASA Architecture (adapted from [42])**

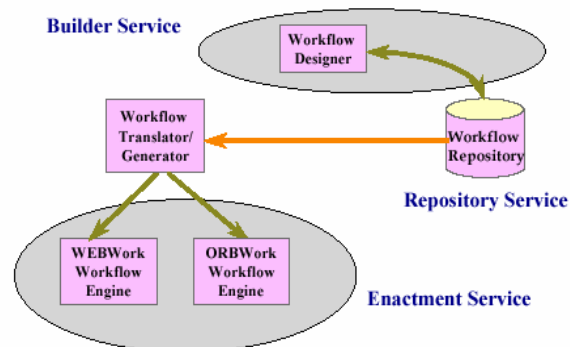
The user starts the design of a new workflow using the specification design facility. When the user finishes a specification, the runtime manager may be invoked to execute the experiment, and the results will be seen through the browsing and visualization facility. The Data Manipulation facility allows users to access and update data concerning the experiments. The database layer of WASA contains two categories of data: application-specific data and WASA-specific data. The first one is supposed to be stored in a number of databases and includes text data, references to bibliography data and other data resources. The second one is a repository used to store data needed to run a workflow.

WASA was not developed for execution in distributed environments. Therefore it is not clear if or how parallelism is achieved during the execution of a workflow. Another point that is not mentioned is whether WASA has the ability to deal with abstract workflows nor how workflow definitions are stored.

## II.2 - METEOR

Meteor is a workflow management system developed at LSDIS laboratory of the Computer Science Department at the University of Georgia [43, 44]. Its architecture

includes four services: Workflow Builder, Workflow Repository, Workflow Enactment and Workflow Manager (figure 11).



**Figure 11 - Meteor architecture [43]**

The Builder Service is used to graphically design and specify a workflow. The specification includes the dependencies between the tasks and data passed among them. It also includes details of task invocation. The workflow definition is independent of the runtime system and is stored in the workflow repository.

The Repository Service is responsible for maintaining information about workflow definitions and also interacts with the Enactment Service providing the necessary information about a workflow application to be invoked. Workflows are stored as XML documents. The user can access the workflow repository to perform dynamic changes.

Two enactment services for METEOR have been developed: ORBWork and WebWork. The first one based in the CORBA middleware and the second one, based on Web technology. Both services have code generators to be used to build workflow applications from the specifications stored in the repository. METEOR has been used by applications in medicine, engineering and biology.

METEOR was not developed to address workflow design and execution in cluster or grid environments. Therefore performance is not the main goal of the project. However, since the workflow components may be distributed to different hosts, the workflow administrator may distribute the processing of the tasks of a running workflow to a host that become available allowing for better load balance.

### **II.3 - SDM**

The Scientific Data Management Center Project (SDM) [45] is carried out by US Department of Energy and many Universities to address data management challenges in

science applications. Figure 12 shows the Scientific Workflow Management System architecture.

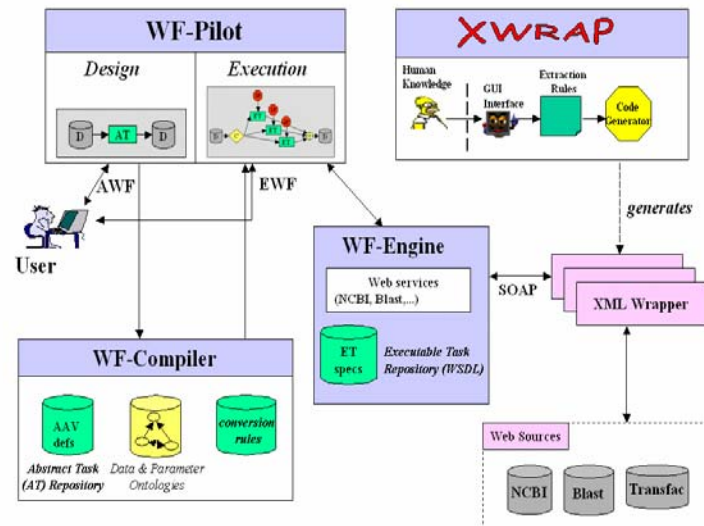


Figure 12 - Scientific Workflow Management System architecture [45]

The core idea of the project is to allow a scientist to design an abstract workflow from the repository of abstract tasks while the system generates an executable workflow from the abstract definition in terms of the available web services. In order to assist the user in the abstract workflow design, domain ontology is used to link abstract tasks with semantic types. Executable tasks are stored in a web service repository.

The workflow designer defines single abstract tasks in terms of executable tasks using the WF-pilot GUI. For example, after defining the abstract task of cluster-analysis, the designer has to create a concrete instance of cluster-analysis by associating it with a specific cluster analysis tool such as CLUSFAVOR. In general the designer may create several concrete instances for the same abstract task. In this case, the conditions that allow the system to select at runtime one executable task must be provided. If the system cannot determine a unique instantiation, the user is prompted for a decision at runtime.

In the execution mode, the user can add breakpoints in the workflow to inspect intermediate results and to decide which intermediate data should be made persistent. The Scientific Workflow Management System was used in a bioinformatics application, the Promoter Identification Workflow (PIW).

SDM aims to hide the low-level details of web services so the scientist can focus on the design of a scientific workflow at the conceptual level. Like METEOR, SDM was not designed to scientists to create workflows to be executed in cluster or grid environments and consequently performance is not the main concern of the system and



there is no performance results reported. Abstract tasks are stored in a abstract task repository and executable tasks in a web services repository. However details on how these repositories store the data are not discussed.

## II.4 – gRNA

gRNA [46] operates on a cluster of multiple computers that communicate over Ethernet and provides an environment for development of life sciences applications based in a set of APIs. Data from biological sources are gathered and stored in warehouses using XML structures. These data can be queried using the XomatiQ component, a visual XML-based query interface. Although the main goal of GRNA is data integration, the project has a workflow management system, named HyperThesis, that enable users to build, in a graphical way, ad-hoc workflows. HyperThesis stores the workflows definitions in a repository. However, it is not clear how GRNA addresses issues like workflow parallel execution, abstract workflow concept and the workflow definition storage.

## II.5 - BIOOPERA

BioOpera [47, 48] is being developed at the Information and Communication Systems Research Group of ETH Zürich, and provides resources for development and execution of scientific applications in cluster of PCs, UNIX workstations and Grid environments. Its architecture is divided into two layers: The User Interface Front-end and Back-end (figure 13).

Workflows are specified using a graphical tool where the user defines its input, output, tasks and their dependencies. The tasks (programs) are selected from a library and are linked by drawing connections between them. The graphical representation is turned into a textual representation using an internal programming language to represent and manipulate workflows. When registering a new program, the user must specifies the input/output parameters, how to run it and the ranges of nodes where it can be invoked. This information is stored in a database and can be dynamically changed. The database also stores runtime information such as the states of the tasks, execution logs, load and availability of each node.

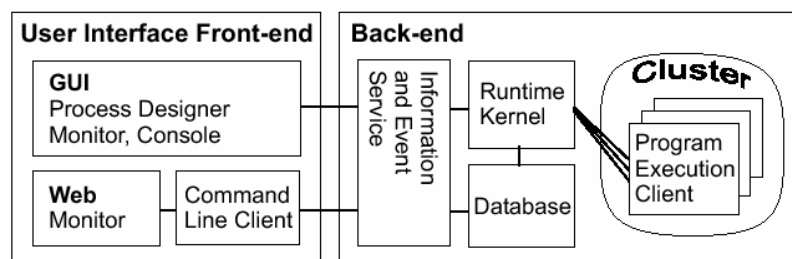


Figure 13 - BioOpera Architecture [47]

When executing a workflow, BioOpera analyses the control flow dependencies and concurrently schedules all tasks that are found to be independent. If enough computing resources are available, these tasks will be executed in parallel. The user can interact with BioOpera to restrict the set of nodes used for execution and to check intermediate results. It is also possible to kill, suspend, resume and restart a program or the entire workflow. The dynamic scheduling and load balance mechanisms allow BioOpera to work with a cluster that shrinks or grows in size dynamically. The system may start applications running on different operational systems (Linux, Solaris, Windows) and execution platforms (Condor, CORBA, RPC, Web Services).

Experimental results are shown for a bioinformatics application. In this case, data parallelism is achieved, but in the context of a single task and not in a workflow processing. This computation involved a cross comparison of the SwissProt protein database and was done by partitioning the database among several servers which performed the work. In BioOpera there is no use of abstract workflow concept. The workflow may be constructed through the combination of executable programs.

## II.6 – SRMW

The Scientific Resource Management project (SRM) [57, 58] proposes an architecture where scientific users can remotely access and share programs, data and scientific workflows definitions and experiments. SRM embeds the Scientific Publishing Metamodel (SPM) that relates models, programs and data through specific categories and semantic relationships. SRM is implemented as a web service architecture (SRMW), which provides interoperability among published data and programs. Web services classes may be used to classify data and programs, which can be used in service composition to become a workflow. The SRMW architecture is showed in figure 14.

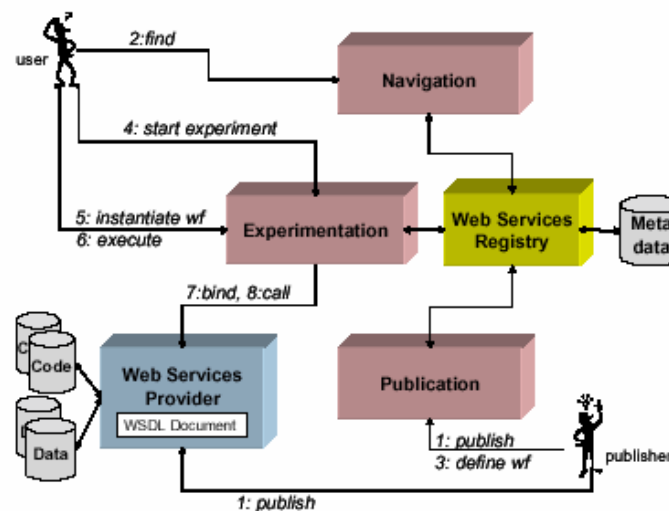


Figure 14 – SRMW Architecture [58]

The publication module publishes WSDL documents in a metadata repository. This metadata can be retrieved by the navigation module. The experimentation module monitors the execution by mediating communications between the user and the real service provider. Data and program publishers are responsible for building a Web Service Provider for their resources, so that they can become available to Web users. Publishers have to use the Web Service Registry module to register their code and data resources as services in the UDDI Service repository. Publishers should also publish the correspondent WSDL documents.

SRMW allows scientists to configure their own workflows by dynamically combining programs provided by different research teams and was used in biological experiments in a real structure genomic workflow. SRMW is a work in development. Parallel processing is not yet provided and only serial workflows can be defined through SRMW web interface.

## II.7 - CHIMERA

The Chimera [15, 18] prototype implements the Virtual Data Grid, an architecture to integrate data, and the computational procedures used to manipulate it. Chimera includes two primary components: a virtual data schema and a virtual data system. The first one is used to store the information about the procedures, invocations of those procedures and the datasets produced during their execution, while the second one has the goal of allowing users to construct and maintain this information in a distributed context. The virtual data schema is not tied to any specific technology like RDBMS or XML repository and defines five entities as illustrated in figure 15.

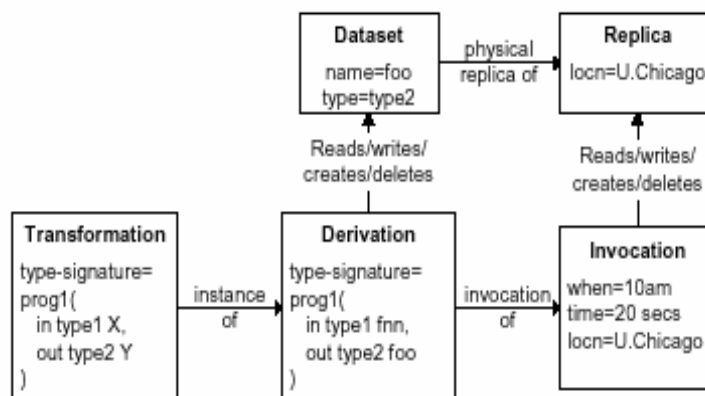


Figure 15 - VDC main components [15]

Dataset is the unit of data managed within the system and can have associated multiple physical copies with different properties such as location. Transformation represents computational procedures and a Derivation specifies the arguments to execute a transformation. Last, an Invocation store information about the environment (e.g., date, time, OS) in which its associated derivation was executed.

The Chimera Virtual Data Language allows the users to specify the transformations and derivations needed to generate the datasets. Figure 14 shows an example of the VDL statements used to define the transformations and derivations that correspond to a workflow, represented as a Directed Acyclic Graph (DAG), which is showed in the right side.

The VDL definitions are translated to XML format before being inserted in the Virtual Data Catalog (VDC). The abstract workflow (AWF) is then generated based on the information stored in the VDC. The AWF is represented as another XML document where the transformations and derivations are substituted by jobs and contain the logical names of the executables and files. Therefore a DAG is the result of the Chimera abstract planner and is expressed in terms of logical entities. These logical entities are mapped to physical instances by the concrete planner that selects an execution site for each node, and determines how to obtain and transport the data needed by each computation. The concrete DAG generated in this step is represented as a script file that is submitted into the Grid via Condor-G [32] and DAGMan [33] a meta scheduler for Condor jobs developed at Wisconsin University.

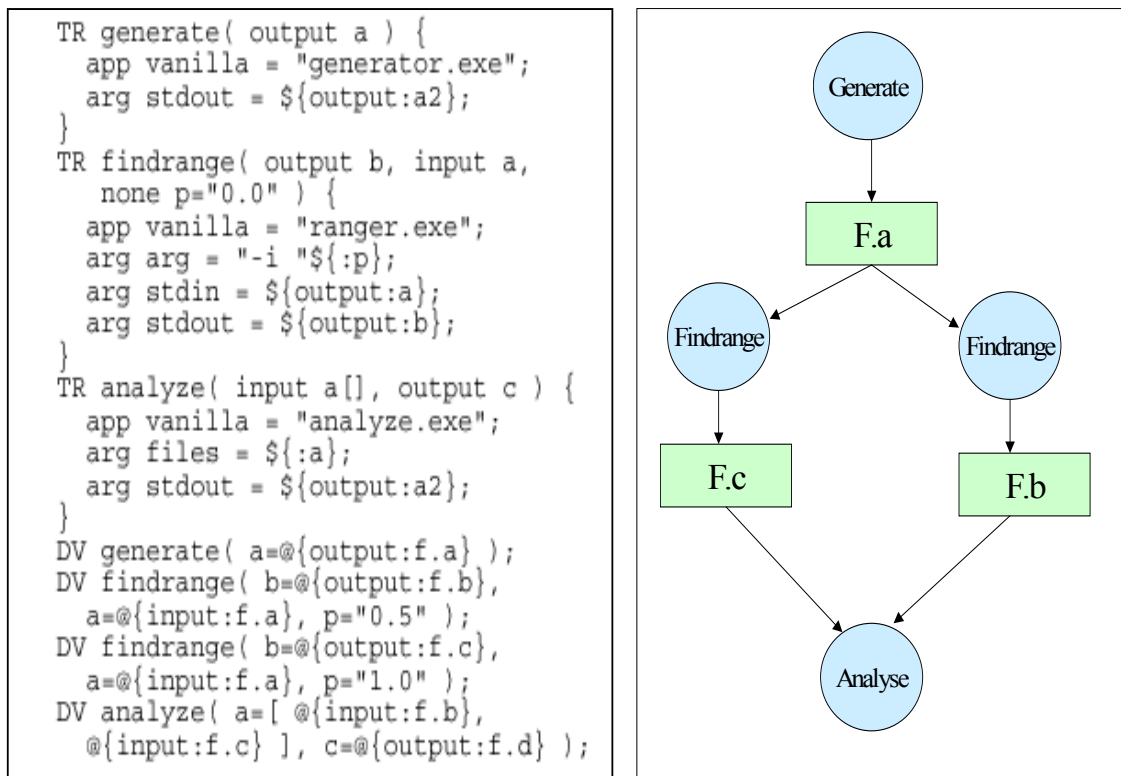


Figure 16 - VDL specification and the correspondent acyclic graph (adapted from [16])

Chimera has been used in the context of the GriphyN project in the data analysis problem of the identification of galaxy clusters within the Sloan Digital Sky Survey. In

this work, inter-workflow parallelism is exploited by the fragmentation of the input dataset and replicating the workflow programs among a large number of machines in four universities. Intra-workflow parallelism can be achieved in Chimera since Dagman can submit multiple Condor jobs at the same time. However, it seems that intra-program parallelism cannot be specified with VDL, that is, there is no way to define that a transformation in the abstract workflow can be executed in this way. Another limitation in Chimera is that workflow execution is restricted to Condor jobs.

## II.8 - PEGASUS

Pegasus [16, 20, 49, 50] is another project that addresses the problem of generating abstracts and concretes workflows for Grids and is being developed at ISI as part of GriPhyn and SCE/IT projects. Pegasus uses several Globus services and can execute workflows on the Grid. Pegasus can be integrated to Chimera. In this configuration (see figure 15), Pegasus receives an abstract workflow from Chimera, produces a concrete workflow and submits it to Condor-G for execution.

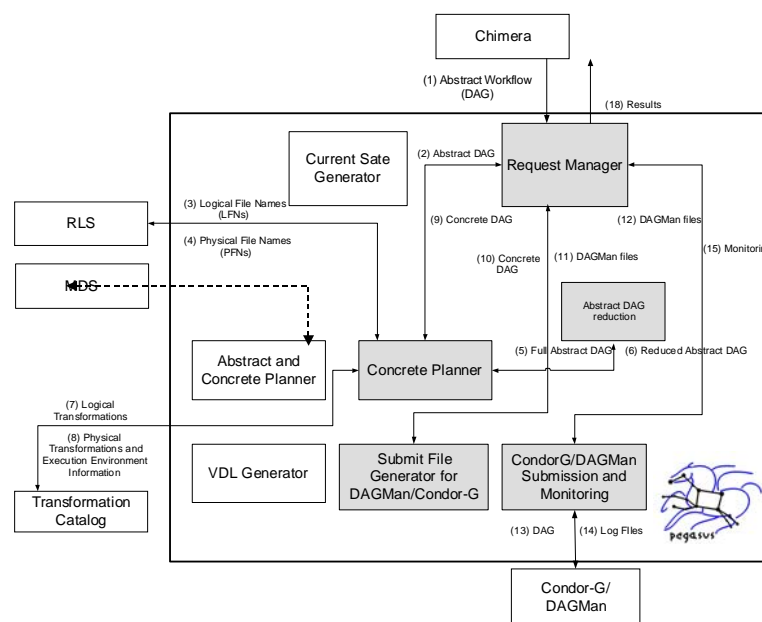


Figure 17 - Configuration of Pegasus when driven by Chimera [50]

The abstract workflow (AW) describes the transformations and data by their logical names. The concrete workflow (CW) specifies the location of data and the execution platforms. If there is any dataset specified in the AW that already exists, the Pegasus planner will reuse it and thus reduce the complexity of the CW. The choice of the location where the transformation will be executed is done querying the transformation catalog. Data files can also be replicated in various locations. The Metadata Catalog Service (MCS) links application-specific metadata with the logical

names of the data files. Given a logical file name, the Replica Location Service (RLS) can be used to find the physical locations for the file. The Monitoring and Discovery Service (MDS) is used to find the appropriate resources given the requirements of the application components. Pegasus contains a Virtual Data Language generator that can populate the Chimera virtual catalog.

Pegasus can also be configured to perform the generation of the abstract workflow based on application metadata. In this case, AI-based planning technologies are used to construct both AW and CW. Like Chimera, Pegasus uses Dagman and Condor to submit the workflow for execution and has been used in physics applications.

## **II.9 - MYGRID**

The myGrid [51, 52] project aims to develop middleware to support *in silico* experiments in biology and is building services for integration such as resource discovery, workflow enactment and distributed query processing. The architecture of the project is based on services, initially implemented by Web services but is intended to be delivered as Grid Services. Like Chimera, MyGrid is concerned with the provenance of derived data and aims to identify all input data, intermediate and final results together with the process used to create the results.

In order to fulfill the workflow requirements established by the project like specification of provenance data and workflow semantics, a workflow language (Scufl) and enactor (Freefluo) were developed. These tools allow interaction with the user during the enactment process. The user can then be asked to choose which of the services available at that time should be used. Although MyGrid has been developed for use by e-scientists in a grid environment, none of the works describe performances results regarding workflow executions. Also, issues like strategies to perform data and program distribution and replication are not mentioned.

## **II.10 - GRIDFLOW**

GridFlow [53] is a workflow management system being developed by several research institutes with the goal to enable grid users to construct, simulate, execute and monitor grid workflows. A workflow is represented as a flow of several different activities each activity represented by a sub-workflow. A sub-workflow is a flow of related tasks that is to be executed in a predefined sequence on resources within a local grid (one organization). Tasks are MPI and PVM jobs, data transfers or archiving of large datasets. Figure 16 shows GridFlow architecture. The User Portal enables users to construct a workflow in a graphically way. To construct a workflow, a user needs to define properties of each sub-workflow and task and their execution sequences. The abstract workflow is represented using a XML specification, which is send to the Workflow Management. If the user knows where a task or a sub-workflow will be executed, he can define this location within the portal. However if the user has no knowledge about the available grid services and resources, the workflow management service will provide the services automatically.

The workflow management service provides three functionalities: simulation, execution and monitoring. Simulation takes place before the execution of a workflow and provides the workflow schedule. Execution provides the workflow execution according to the simulation schedule. Monitoring provides interfaces that allow access to real-time status reports of tasks or sub-workflows execution. The Sub-workflow scheduling schedules tasks onto grid resources within a local grid.

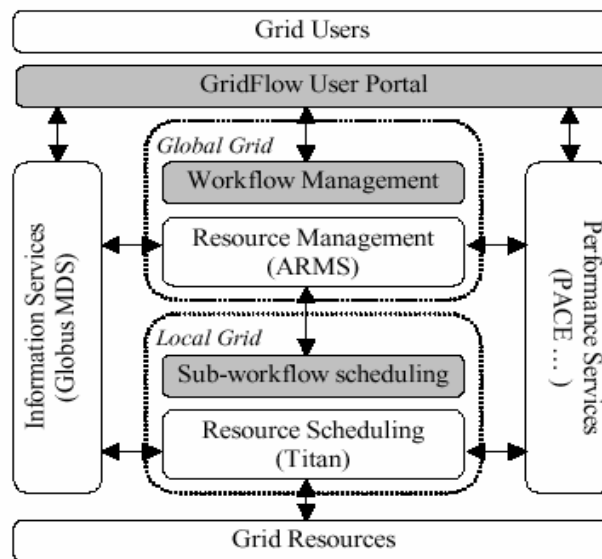


Figure 18 - GridFlow architecture [53]

It is not clear how users interact with the portal in order to specify the workflow and how the parallel environment of the grid is exploited to execute the workflow in parallel. Apparently the system does not provide a repository containing the tasks stored allowing users to choose the workflow components. The mapping between abstract and concrete workflow is not well explained either. Since tasks or sub-workflows are executed inside a local grid, it seems that parallelism can be achieved if a task is already a parallel application. The use of the GridFlow in a real application study was not reported.

## II.11 – DISCUSSION

Table 3 summarizes all the systems described in the last section according to a set of workflow features we found relevant to the context of this thesis: execution environment, workflow definition storage, run time, scientific area, parallelism and abstract workflow definition. The execution environment is related to the distributed environment, which is addressed by the system: web, cluster or grid. Storage informs the technology like RDBMS or XML files used to store definitions about the workflow. Run time specifies the kind of executables that are run by the system. Area is related to the

scientific area where the system has been employed and parallelism informs the level of workflow parallel processing that can be achieved by the system. Finally, AWF is characterized by its semantic level, language used to specify the abstract workflow and by the presence or not of a graphical user interface.

**Table 4 – Workflow Projects General Aspects**

System	Environment	Storage	Run Time	Area	Parallelism	AWF		
						Semantic	Language	GUI
WASA	Central	SGBD	?	Geoscience Biology	None	None	None	Y
Meteor	Web	XML Files	Corba	Medicine Engineering Biology	?	None	Internal	Y
SDM	Web	-	Web Services	Biology	Intra-Wf	Metadata Ontology	Internal	Y
GRNA	Cluster	SGBD	SQL Queries	Biology	?	Metadata	Internal	Y
BioOpera	Cluster	SGBD	Condor Corba RPC Web Services	Biology	Intra-WF	Programs	Internal	Y
SRMW	Web	XML Files	Web Services	Biology	None	Metadata	BPEL4WS	Y
Chimera	Grid	XML Files	Condor Jobs	Physics	Inter-Wf Intra-Wf	Logical Names	VDL	N
Pegasus	Grid	XML Files	Condor Jobs	Physics	Intra-Wf	Logical Names Metadata	VDL	N
MyGrid	Grid	?	Web Services	Biology	Intra-Wf	Metadata Ontology	XScufl	Y
GridFlow	Grid	XML Files	MPI PVM	?	Intra-Wf	None	XML	Y

As can be noted from the projects discussed before, several scientific workflow projects allow users to specify in a friendly way, the components that compose the workflow application. The use of tools like a graphical interface and an abstract workflow definition language can allow scientists to focus on the design of a scientific



workflow at the conceptual level, hiding the low-level details and intricacies of program interactions and invocations. Therefore, using such features, scientists would be free to concentrate on how to solve a problem rather than on how to map a solution onto available resources, or to acquire better performance. However there must be software components to provide an adequate workflow execution strategy.

The simultaneous execution of different programs within a workflow, i.e., intra-workflow parallelism, seems to be addressed in almost every system. However, the simultaneous execution of the same workflow, i.e., inter-workflow parallelism, is only showed in Chimera work. But in this case, the input data to be processed by the workflow was previously fragmented over the several machines. None of the other presented works perform the distribution of the workflow input data during the workflow execution. Therefore, in these works the input data is always processed by the first program of the workflow with no data parallelism.

Also, we did not find systems allowing the specification for data fragmentation and/or data replication for single components. Therefore, intra-program parallelism, which is very used to improve the performance of isolated scientific programs, appears to be not exploited during the execution of a workflow component in these systems. The graphical user interfaces and languages used to design the scientific workflows do not allow users to specify this kind of feature. Thus, in order to have intra-program parallelism exploited during the workflow processing, it is necessary to have this information provided by the workflow designer.

Regarding the workflow specification storage, XML is the data format most used to store the abstract workflow definitions. This is important since an XML provides a data structure for a document. Therefore, the use of XML for defining abstract workflows can certainly make the development of services that have to interpret an abstract workflow definition, like for example to generate a parallel execution strategy for the workflow, easier.

Most works address the design and execution of scientific workflows in Physics and Biology fields. This fact makes evident the need for scientists in these areas to compose and integrate different resources like programs and data to build their scientific applications.

## IV – WORKFLOW PARALLEL PROCESSING

---

Parallelism has already been used by scientific and database communities to improve the performance of programs and database operations respectively. SPMD (Single Program Multiple Data) and MPMD (Multiple Program Multiple Data) characterize strategies for parallel program execution. Also inter-query, pipelined and partitioned parallelisms characterize strategies for parallel query processing in DBMS. However, we did not find works characterizing strategies for parallel execution of scientific workflows. The goal of this section is to propose a set of parallel strategies and an architecture to execute scientific workflows in parallel. In subsection IV.1 we propose a characterization of such parallel strategies. Then in subsection IV.2 we show an architecture that provides the design and the execution of scientific workflows in parallel. Finally, in subsection IV.3 we propose a set of initial heuristics considering workflow patterns that can be used to generate the parallel execution design.

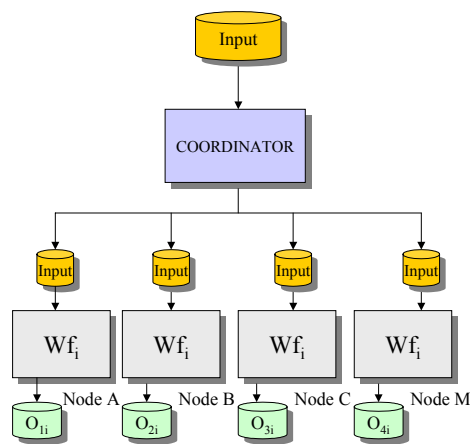
### IV.1 - WORKFLOW PARALLEL STRATEGIES

Scientific programs often generate and process large datasets. Therefore, the execution of such programs can be a time-consuming task and data parallelism strategy has already been used to improve the performance of isolated programs. When combining these programs to compose scientific workflows, the simultaneously execution of different programs can also be exploited. We propose here a characterization of three kinds of strategies to process a scientific workflow in parallel: **Inter-workflow** parallelism, **Intra-workflow** parallelism and **Intra-program** parallelism. These strategies can be employed in order to provide parallel solutions to execute scientific workflows with better response times.

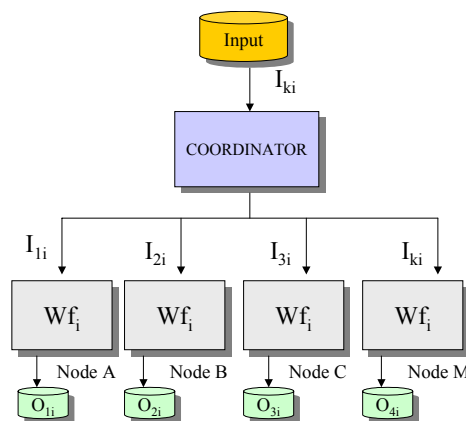
One typical scenario in scientific applications is having a set of input data to be processed by a workflow. We define a workflow  $Wf$  as a set of programs to be executed according to modeling patterns. A modeling pattern defines the order and the conditions to execute each program. Let  $WF$  be the set of workflows  $Wf_i$ , where  $1 \leq i \leq N$ . For each  $Wf_i$  there is a set of programs  $P_{ji}$  ( $1 \leq j \leq P$ ) that compose the workflow  $Wf_i$ , a set of input data  $I_{ki}$  ( $1 \leq k \leq M$ ) that has to be processed by  $Wf_i$  and a set of output data  $O_{ji}$  ( $1 \leq j \leq P$ ) generated by each program  $P_{ji}$ . **Inter-workflow** parallelism can be characterized by the simultaneous execution of the same workflow  $Wf_i$ , each one processing a subset of the input data  $I_{ki}$ . This parallelism can be reached by allocating all the programs  $P_{ji}$  of a workflow  $Wf_i$  at each node of the system but can only be exploited if the input data elements  $I_{kj}$  can be processed independently. The set of output data  $O_{ji}$  generated in each node must be joined in order to produce the final result.

The distribution of the input data to the nodes can be done basically in two ways: by groups of  $I_{ki}$  elements or individually. In the first case, as illustrated in figure 19, dividing the number of input data elements by the number of nodes can generate subsets of the input data. In the second approach, as illustrated in figure 20, the input data elements can be distributed in a round-robin way, as soon as one node finishes the

processing of a previous input data element. The distribution of the input data to the nodes plays an important role, due to the number of messages exchanged during the workflow parallel processing. In the first case, the number of messages exchanged by the coordinator and the nodes will be equal to the number of the nodes in the system, while in the second case, this number will be equal to the number of the input data elements to be processed. If the number of nodes is less than the number of input data elements to be processed, the first alternative apparently seems to be better. However, there is no guarantee that the time needed to process different input data elements will be the same. Therefore, one node can finish its processing and stay idle while another node still has a lot of input data to process. The second approach, the individual distribution of the input data to the nodes, minimizes this problem since the coordinator can distribute an input data element as soon as a node becomes able to process it.



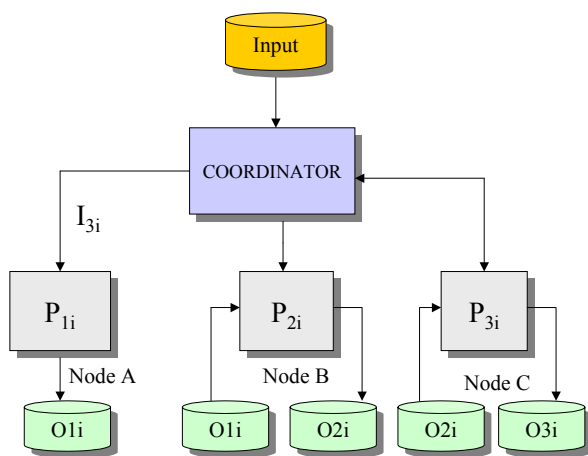
**Figure 19 – Inter-workflow parallelism with grouped input data distribution**



**Figure 20 - Inter-workflow parallelism with individual input data distribution**

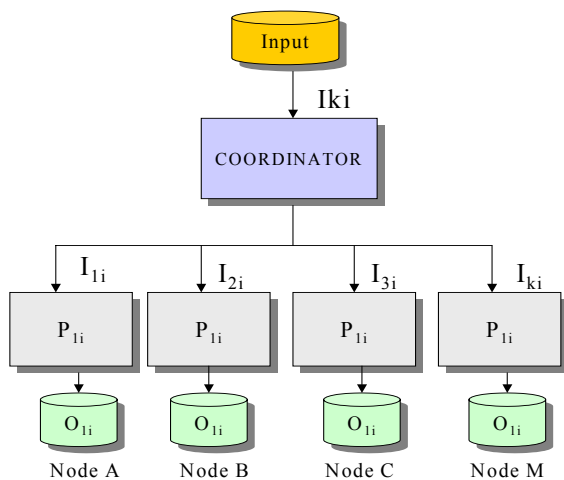
**Intra-workflow** parallelism can be characterized by simultaneous execution of more than one program  $P_{ji}$  of the same workflow  $Wf_i$ . This parallelism can be reached by allocating the programs  $P_{ji}$  of a  $Wf_i$  workflow at different nodes of the system. Intra-query parallelism is achieved in DBMS when an operator continually streams its result to

another operator allowing both operators to work in parallel. This pipeline parallelism can also be achieved in a workflow execution. For example, supposing a workflow with three programs in a sequential pattern as shown in Figure 21. **Intra-workflow** parallelism could be achieved by having program  $P_{3i}$  in node C processing an output  $O_{2i}$  previously generated by program  $P_{2i}$ , program  $P_{2i}$  processing the output  $O_{1i}$  previously generated by program  $P_{1i}$ , and program  $P_{1i}$  processing an input data  $I_{3i}$  provided by the coordinator. There are others workflow patterns where intra-program parallelism can naturally be exploited. Whenever a parallel-split, exclusive-choice, synchronization, or simple-merge occur **intra-workflow** parallelism can be applied. However, the coordinator must provide the programs with the input data needed by them to perform their processing and control the synchronism between the executions of the programs.



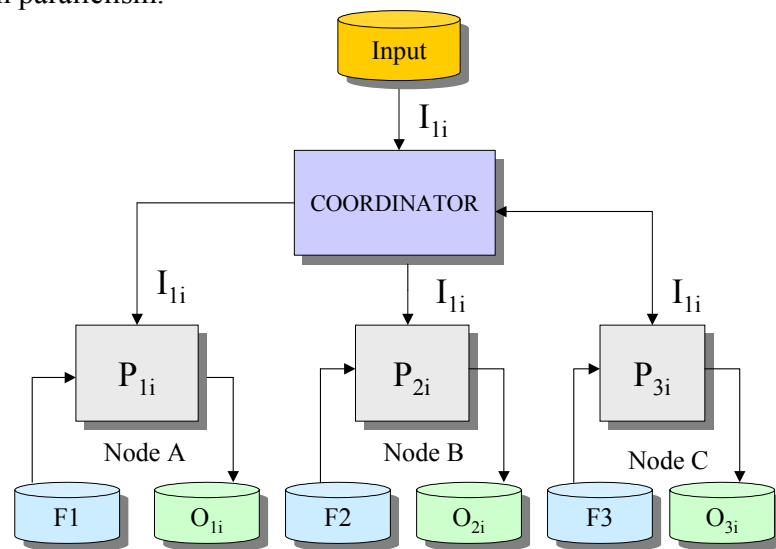
**Figure 21 - Intra-Workflow Parallelism for a pipeline of programs (sequential pattern)**

**Intra-Program** parallelism can be characterized by simultaneous execution of the same program  $P_{ji}$  of a workflow  $Wf_i$  in different nodes. If the input data elements  $I_{kj}$  for a program can be processed independently then intra-program parallelism can be achieved by allocating the same program  $P_{ji}$  at different nodes and by distributing the set of input data between the nodes, like in the inter-workflow strategy. Figure 22 illustrates this situation.



**Figure 22 – Intra-program parallelism for input data processing independently**

However, if the set of input data cannot be processed independently, **intra-program** parallelism can also be achieved if a program  $P_{ji}$  processes a dataset that can be partitioned. In this case, the program can process in each node the same input data element over a fragmented dataset. This kind of data parallelism has been widely used by the database community and offers great opportunities to increase performance in shared-nothing DBMS [19]. In a parallel DBMS architecture, the database is fragmented according to some criteria and the generated data fragments are allocated to the different nodes. Consequently, slow operations like a full table scan in a large table, can be performed in much less time since each node have to scan a subset of the original table. In workflows, such strategy could be adapted for programs that process large datasets if this data partitioning does not interfere with the final result. Figure 23 illustrates this kind of intra-program parallelism.

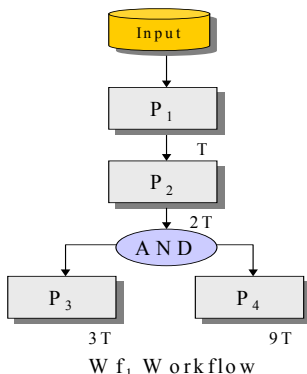


**Figure 23 - Intra-Program Parallelism with dataset partition**

It must be noted that whenever **intra-program** parallelism strategy is used, the set of output data  $O_{ji}$  generated in each node must be joined in order to allow the execution of the next program of the workflow or to produce a final result, unless the next workflow program can process the antecessor fragmented results.

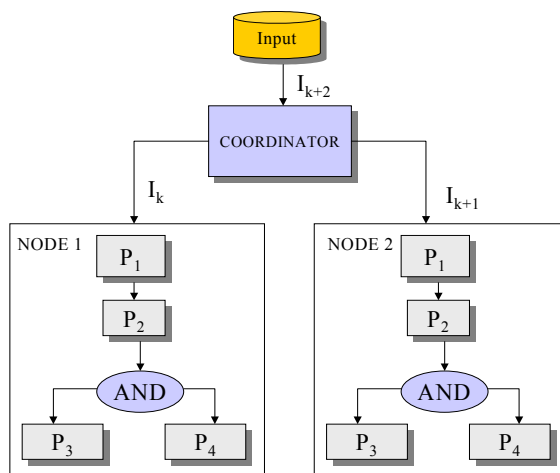
**Inter-workflow, intra-workflow** and **intra-program** strategies can be combined or used independently to execute a scientific workflow in parallel. However, to choose the best strategy to parallel process a set of input data is not straightforward. Most of the works discussed in section III exploit **intra-workflow** parallelism if two or more programs can execute at same time, like in a parallel-split pattern. But there is no guarantee that this strategy will always be the best. To illustrate this fact, we will make a theoretical analysis using a workflow  $Wf_j$  that presents a sequential and a parallel-split pattern and is composed by four programs  $P_1, P_2, P_3$  and  $P_4$ . Figure 24 illustrates the  $Wf_j$

workflow. The time to  $Wf_1$  process sequentially one input data is  $15T$  and the time to execute each program individually is  $T$ ,  $2T$ ,  $3T$  and  $9T$  respectively, where  $T$  is assumed to be a time unit.

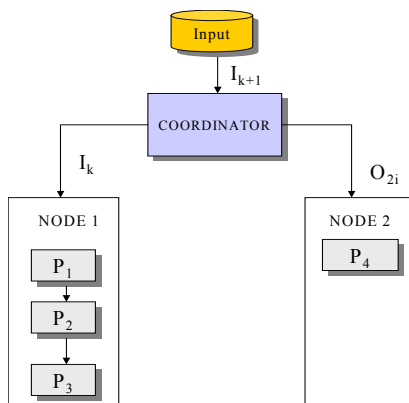


**Figure 24 – The theoretical  $Wf_1$  workflow**

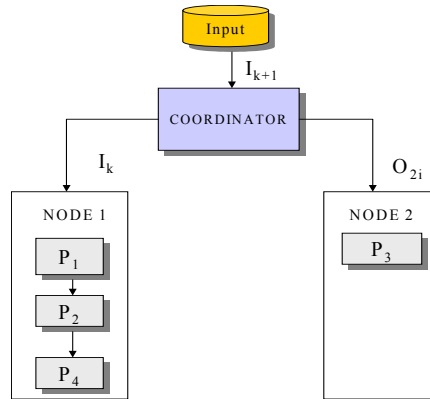
Supposing that  $Wf_1$  can be processed in a system with two nodes, we illustrate two parallel strategies for executing  $Wf_1$  and compare the estimate time to process  $Wf_1$  according to these strategies. An **inter-workflow** strategy is shown in figure 25 while two **intra-workflow** strategies are shown in figures 26 and 27. In these cases, the parallel-split pattern among the programs determined the distribution of the programs.



**Figure 25 – Inter-workflow strategy to execute  $Wf_1$**



**Figure 26 – First intra-workflow strategy to execute  $wf_1$**



**Figure 27 - Second intra-workflow strategy to execute  $wf_1$**

Table 4 gives the estimated time to process this  $Wf_1$  according to the inter-workflow and intra-workflow strategies showed in the figures 25, 26 and 17 supposing a homogeneous environment for the execution and ignoring the time transfer program outputs between the nodes. The first column of the table indicates the number of input data elements processed. The other three columns indicate the estimated time for executing  $WF_1$  according to the different parallel strategies.

**Table 5 – Estimated time to process  $Wf_1$  in a homogeneous environment**

# Input data	Inter-workflow Strategy	First Intra-workflow Strategy	Second Intra-workflow Strategy
1	15T	12T	12T
2	15T	24T	21T
3	30T	36T	30T
4	30T	48T	39T
5	45T	60T	48T
6	45T	72T	57T

As can be seen, except for the processing of one input data, executing the workflow according to the **Inter-workflow** strategy provides, at least in theory, better performance than the **Intra-workflow** strategies. However, since programs P3 and P4 can execute at same time, there are two possible **Intra-workflow** strategies to be done: executing program P3 in node 1 with programs P1 and P2 and executing program P4 alone in node 2; or in an inverse way, executing program P4 along with programs P1 and P2 in node 1 and program P3 in node 2. The second intra-workflow strategy seems to be better since program P4 is slower than program P3. Therefore it is important to gather information about the estimated time for execution of the individual programs to subsidize workflow components allocation and execution.

The estimated results presented in table 4 took into account a homogeneous environment for executing  $Wf_i$ . However, if  $Wf_i$  executes in a heterogeneous environment like a grid, the performance can be totally different. Table 5 shows the estimated times for  $Wf_i$  execution, supposing an heterogeneous environment where the first node has three times more CPU power than the second node. As can be noted, in this situation the first intra-workflow strategy seems to be better.

**Table 6 - Estimated time to process  $Wf_i$  in a heterogeneous environment**

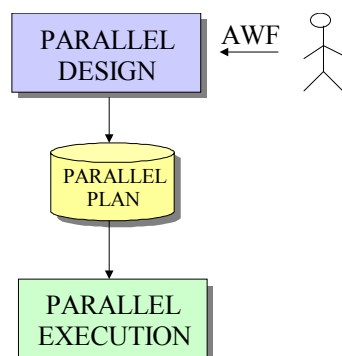
# Input data	Inter-workflow Strategy	First Intra-workflow Strategy	Second Intra-workflow Strategy
1	5T	4T	10T
2	15T	8T	19T
3	15T	12T	28T
4	15T	16T	37T
5	30T	20T	46T
6	30T	24T	55T

Another important information that must be gathered is the possible use of **intra-program** parallelism with some workflow component. However, the decision to employ **intra-program** parallelism strategy cannot be deduced automatically. Many scientific programs cannot be executed according this strategy due to program requirements. Therefore the scientist responsible for the workflow design must provide this information.

The idea of this thesis is to provide non-specialists in parallel processing a parallel solution to execute their scientific workflow. However, the execution of a scientific workflow in parallel can be done according several strategies. We intend to do experiments with different forms of parallel execution and in different environments, in order to propose heuristics to provide scientific workflows parallel processing.

## IV.2 – WORKFLOW PARALLEL DESIGN & EXECUTION

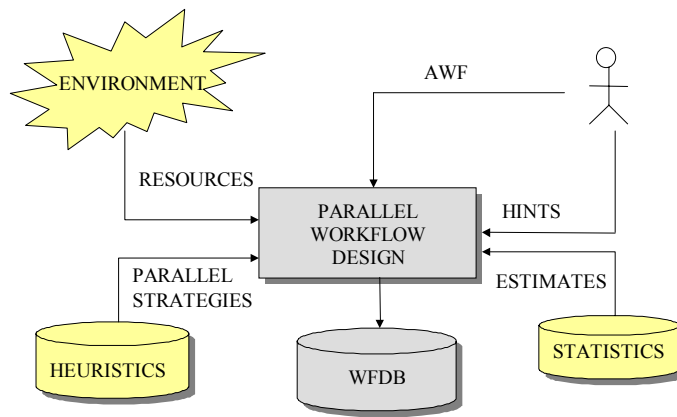
The execution of scientific workflows in parallel can be decomposed in two modules: Parallel Design and Parallel Execution. The goal of the parallel design module is to propose a parallel plan to execute a workflow given an abstract workflow definition of a scientific workflow. The goal of the parallel execution module is to perform the execution of the workflow regarding a parallel plan. Figure 28 illustrates the architecture for a workflow parallel execution.





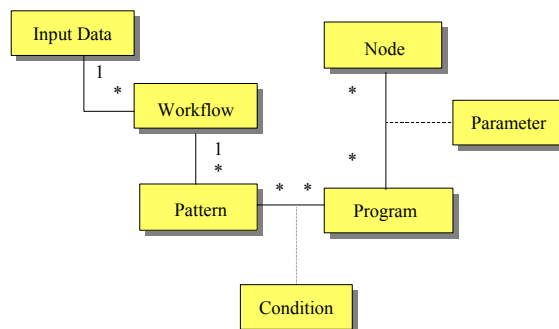
**Figure 28 – Workflow parallel execution architecture**

In order to provide a parallel plan exploiting the parallel strategies described in section IV.1, the parallel workflow design must consider other kinds of information besides that embedded in the abstract workflow. Information about the environment, like the computational resources where the workflow will be processed, statistics like the estimate execution time for the workflow programs and datasets sizes, hints regarding the possible use of intra-program parallelism strategy and the set of heuristics regarding the parallel strategies to be employed must be analyzed to propose a plan for the parallel execution of the workflow. Figure 29 illustrates the parallel workflow design.



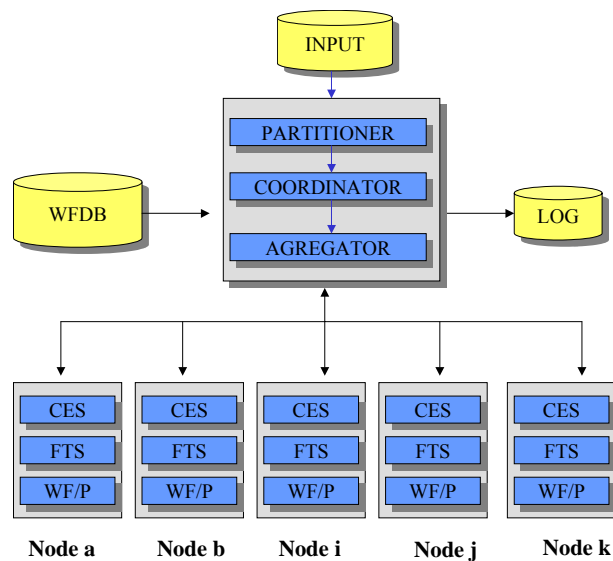
**Figure 29 - The parallel workflow design architecture**

The goal of the WFDB is to store the parallel plan to execute a workflow in parallel. Therefore, WFDB keeps information about the workflow components and their dependencies relationship. It must also inform how the input data can be fragmented in order to allow data partition during the workflow execution. For each program, the WFDB indicates if intra-program parallelism strategy can be employed or not, and provides the nodes where the component can be executed with the parameters needed for its execution. The dependencies between the workflow programs are defined by the relationship between programs and patterns. The order of the execution of a program is given by the condition associated with this relationship. Figure 26 illustrates the WFDB class diagram.



**Figure 30 - WFDB Class Diagram**

The parallel execution module is responsible to process a scientific workflow according the parallel plan. To have a set of input data processed in parallel by a workflow, this data and the workflow components must be distributed between the nodes. Figure 31 shows the architecture for the parallel execution module. The Partitioner is responsible for the fragmentation of the workflow input data. The Coordinator is responsible for the distribution of the input data to the nodes and to coordinate the execution of the workflow. To accomplish this task, the coordinator looks for the program that must be invoked in the WFDB, and sends a message to the appropriate node. If intra-program parallelism strategy is employed, then it is necessary to join the partial results for each program execution. The Aggregator component is responsible for joining any partial results into a single file. On each node, besides the workflow components, there are also two services: The Component Executor Service (CES) and the File transfer Service (FTS). The first one executes the component specified by the coordinator and the second service transfers results of executed programs between nodes.

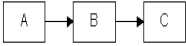
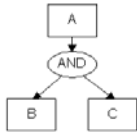
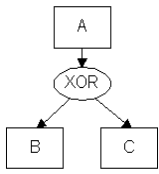
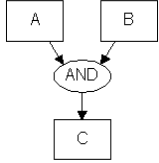
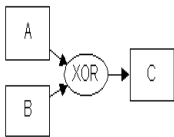


**Figure 31 - Parallel Workflow Execution Architecture**

### IV.3 – INITIAL HEURISTICS

Based on the basic control patterns listed in section II, we propose a set of initial heuristics to execute the workflow programs in parallel. Table 4 illustrates the workflow patterns and respective heuristics. Depending on the kind of the heuristic, additional actions must be performed in order to have the parallel execution achieved.

**Table 7 – Workflow patterns and respective heuristics**

PATTERN	DESCRIPTION	HEURISTICS	ADDITIONAL ACTION
 <p>Sequential</p>	The execution of a program can be done after the execution of the predecessor	Execution of the programs in the same site	None
 <p>Parallel Split</p>	Programs B and C have to be executed after program A	Programs B and C executes in different sites. Program A executes along with either B or C	The output of program A must be sent to the sites of programs B and C
 <p>Exclusive Choice</p>	Program B or program C must be executed after program A	Programs A, B and C execute in the same site	None
 <p>Synchronization</p>	Program C can only executes when programs A and B finished	Program C execute in the same site of program A or program B	The output of program A or the output of program B must be sent to the site of program C
 <p>Simple Merge</p>	Program C can execute if program A or program B finish	Program C executes in the same site of program A or program B, preferably along the faster one	The output of program A or the output of program B must be sent to the site of program C

## V – PRELIMINARY RESULTS

---

The goal of this section is to describe some preliminary results obtained with the execution of MholLine workflow. Two different strategies for parallel execution were implemented: inter-workflow parallelism and intra-workflow parallelism. Both strategies showed speedup results.

### V.1 – MHOLLINE WORKFLOW

Genome sequencing projects are producing a vast amount of protein sequences, emerging the need to use high throughput methods to predict structures and assign functions of these proteins. Analysis of several genome sequences indicates that the function cannot be inferred to a significant fraction of the gene products. In fact, isolate sequence homology searches do not always provide all of the answers, since some proteins may not keep sequence homology throughout evolution. On the contrary, the molecular (biochemical and biophysical) function of a protein is tightly coupled to its three-dimensional structure.

One of the methods that contributes to the prediction of protein three-dimensional structures is the comparative modeling, a computational procedure that predict the most reliable structure for a sequence using related protein structure as template. This approach consists of four steps: finding known structures (templates) related to the sequence to be modeled; alignment of the sequence with the templates, building a model, and the validation of the structure. Actually, other type of structure information can also be generated by threading approach, detecting structural similarities that are not accompanied by any detectable sequence similarity, becoming possible its use for fold recognition, pointing for a possible protein function. There are several programs addressing each of these steps. MHOLline is a biological workflow that combines a specific set of programs for the comparative modeling approach. For template structure identification it uses the BLAST [8] algorithm for the search against the Protein Data Bank.

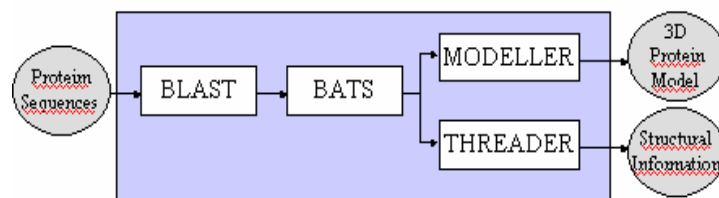


Figure 32 - The MholLine Workflow components

A refinement in the template search step was implemented with the development of a program called BATS (Blast Automatic Targeting for Structures). BATS identifies the sequences that comparative modeling technique can be applied, choose the templates

sequences from the BLAST output file depending on the given scores for expectation values, identity and sequence coverage, and also construct the input files for the automated alignment and the model building carried out by MODELLER [9], of the selected sequences. When a 3D model cannot be built using the comparative modeling approach, BATS identifies these sequences and construct the input file for the execution of THREADER3 [55] that will aggregate structural information through threading, for sequences that didn't generate any 3D model. The MHOLline workflow is illustrated in figure 32.

## V.2 PARALLEL MHOLLINE DESIGN

The distribution of the programs and the data among the nodes can be done in different ways. We adopted in this work two scenarios. In the first one, all the workflow programs and respectively files were placed in every cluster node. This strategy allowed inter-workflow parallelism to be exploited. On a second approach, the Modeller and the Threader programs, which are the slowest ones, were replicated, while Blast and Bats run on only one node. This strategy allowed intra-workflow parallelism to be exploited. Figure 33 illustrates these approaches.

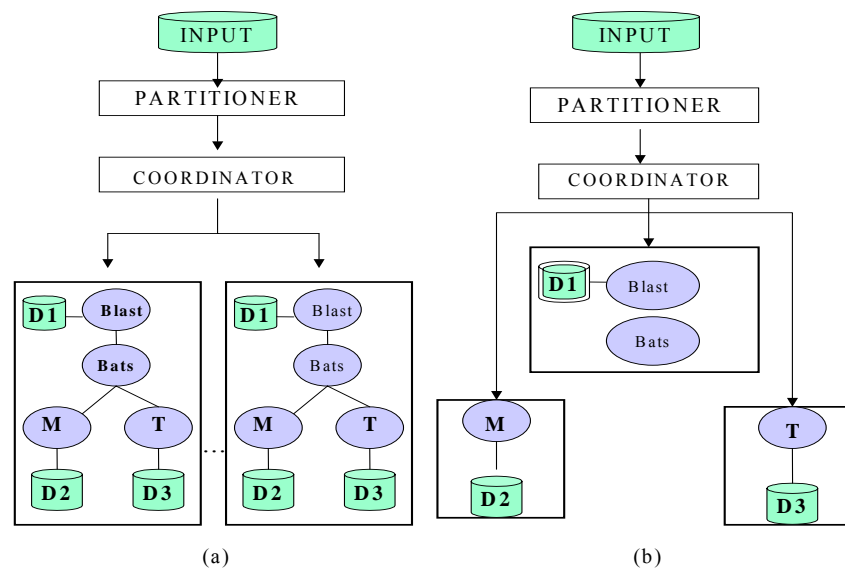


Figure 33 - Implemented Architectures

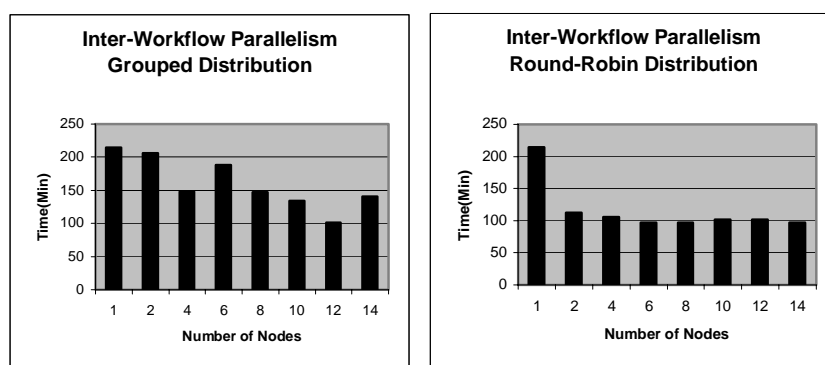
## V.3 EXPERIMENTAL RESULTS

This subsection presents the results obtained during the execution of MholLine according to the architectures described in last section. The experiments took place in an

Itautec Cluster Mercury running Linux Red Hat 7.3, with 16 nodes. Each node has 512Mb of RAM memory, 18 Gb of disk storage and two Pentium III processors with 1 Ghz. They are linked with a Fast-Ethernet (100 MB/s) network. Perl programs wrapped the workflow components and we used MPI to implement the coordinator. The input file had 66 sequences and the data sources used in our experiments were: pdbaa [56] with nearly 30 MB used by BLASTP, a set of pdb files extracted from Protein Data Bank with approximately 5 MB, used by Modeller, and the tdb files with almost 300 MB used by Threader. BATS selected seven input sequences to Modeller (sequences 4, 6, 7, 8, 11, 64 and 65) and three entry input sequences to Threader (sequences 12, 14, and 22). The figure 34 shows the results for the executions using the inter-workflow parallelism strategy with a grouped and a circular distribution of the input sequences.

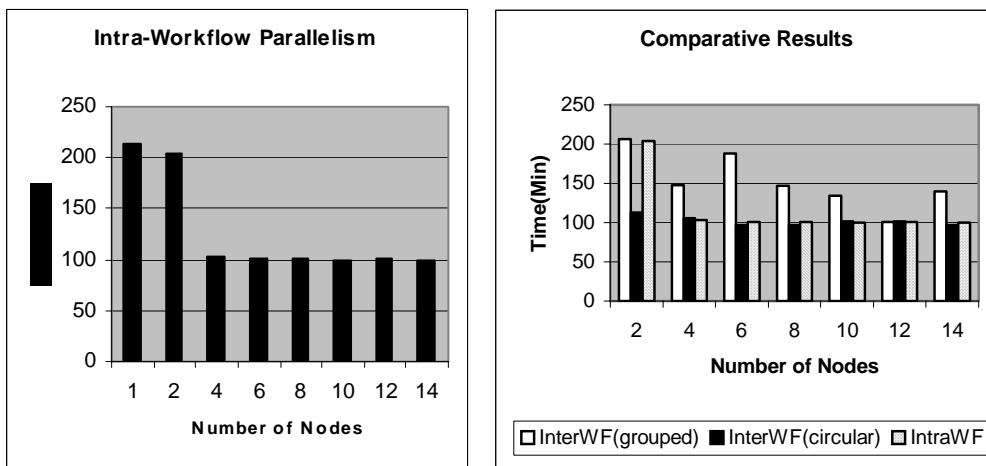
As can be observed, the circular distribution acquired better performance than the grouped distribution. Since Threader is the slowest program of the workflow, the best results were obtained when it could be executed in parallel. The fully parallel execution of Threader only happened in the grouped distribution, when running with 12 nodes. Running with 4, 8, 10 and 14 nodes implied having two sequences submitted for threading in the same node. When running with 2 and 6 nodes, the three sequences were submitted for threading in the same node. The second distribution strategy provided better load balance among the nodes and consequently best results.

Figure 35 shows on the left chart the results for the intra-workflow strategy. Since BLAST and BATS run much faster than Modeller and Threader, we allocated the first two programs in only one node, and replicated the last two programs among the rest of the nodes in such way that in one node or Modeller or Threader could execute. The results for this approach were similar to the results observed in the inter-workflow parallelism with circular distribution. In fact, in both situations, the number of the sequences processed by Threader in parallel was the same, except when running with two nodes.



**Figure 34 – Inter workflow parallelism strategy with grouped and circular distribution**

Aiming to provide a better comparative analysis, Figure 35 also shows on the right side, the results for the three approaches together.



**Figure 35 – Intra workflow parallelism strategy and for the three executions together**

As can be observed, using more than 6 nodes did not interfere in the speedup. This happened due to the few number of sequences selected by BATS to be submitted to Threader and Modeller in our experiments. However, the results show linear speedup when executing with two nodes in the interworkflow parallelism strategy and when executing with four nodes in the intraworkflow parallelism strategy.

These initial experiments allowed us to evaluate two different parallel strategies to execute a scientific workflow. The results show that parallelism can be used to increase the performance of a bioinformatic workflow. We believe that the techniques showed here can also be applied to others scientific workflows. However, it is mandatory to perform more experiments exploiting the combination of the all parallel strategies discussed previously in order to identify heuristics to automate the choice of the best parallel strategy. In the next section, we list the next steps we intend to do to accomplish this goal.

## BIBLIOGRAPHY

---

1. Protein Data Bank, [www.rcsb.org/pdb/](http://www.rcsb.org/pdb/)
2. BLAST, [www.ncbi.nlm.nih.gov/BLAST](http://www.ncbi.nlm.nih.gov/BLAST)
3. GenBank, [www.ncbi.nlm.nih.gov/Genbank](http://www.ncbi.nlm.nih.gov/Genbank)
4. ModBase, <http://salilab.org/modbase/>
5. SwissProt, <http://www.ebi.ac.uk/swissprot>
6. Bio Grid, <http://www.eurogrid.org/wp1.html>
7. Deelman E., Blythe J., Gil Y., Kesselman C., Mehta G., Vahi K., Blackburn K., Lazzarini A., Arbree A., Cavanaugh R., Koranda S., "Mapping Abstract Workflows onto Grid Environments", *Journal of Grid Computing*, V.1, No. 1, pp25-39,2003.
8. Altschul S.F., Gish W., Miller W., Myers E.W. and Lipman D.J. (1990) "Basic local alignment search tool." *J. Mol. Biol.*, 215: 403
9. Sali A. and Blundell T.L.(1993) "Comparative protein modeling by satisfaction of spatial restraints." *J. Mol. Biol.*, 234: 779
10. Costa R., Lifschitz S., "Database Allocation Strategies for Parallel BLAST Evaluation on Clusters", *Distributed and Parallel Databases*, vol. 13, num. 1, Jan 2003, pp 99-127.
11. Pappas, A., "Parallelizing the Blast applications on a network of Dec Alpha Workstations", available at <http://www.cslab.ece.ntua.gr/~pappas>.
12. Braun R., Pedretti K., Casavant T., Scheetz T., Birkett C., Roberts C., "Three Complementary Approaches to Parallelization of Local BLAST Service on Workstation Cluster, 5<sup>th</sup> International Conference on Parallel Computing Technologies (PaCT), LNCS. Vol. 1662, 1999, pp. 271-282.
13. Waugh A., Williams G., Wei L., Altman R., "Using Metacomputing Tools To Facilitate Large-Scale Analyses of Biological Databases", in *Proceedings of Pacific Symposium of Biocomputing*, 2001, pp. 360-371.
14. Foster,I., Kesselman, C.. Chapter 2 of "The Grid: Blueprint for a New Computing Infrastructure", Morgan-Kaufman, 1999.
15. Foster,I., Voeckler, J., Wilde, M., Zhao, Y., "Chimera: A Virtual Data System for Representing, Querying and Automating Data Derivation", *Proceedings of the 14th Conference on Scientific and Statistical Database Management*, Edinburgh, Scotland, July 2002.
16. Blythe, J., Deelman, E., Gil, Y., Kesselman, C., Agarwal, A., Mehta, G., Vahi, K., "The Role of Planning in Grid Computing", ICAPS 2003
17. Greenwood, M., Wroe, C., Stevens, R., Goble, C., Addis, M., "Are bioinformaticians doing e-Business?", *Proceedings Euroweb 2002: The Web and the GRID - from e-science to e-business - December 2002*.
18. Foster,I., Voeckler, J., Wilde, M., Zhao, Y., "The Virtual Data Grid: A New Model and Architecture for Data-Intensive Collaboration", *Proceedings of the 14th Conference on Innovative Data System Research – CIDR-2003*, January 2003.



19. ÖZSU, T., VALDURIEZ, P., 1999, Principles of Distributed Database Systems. Prentice Hall, 2<sup>nd</sup> Edition.
20. Blythe, J., Deelman E., Gil,Y., “Planning for Workflow Construction and Maintenance on the Grid”, ICAPS 2003, Workshop on Planning for Web Services, Trento, Italy, June 2003.
21. Rössle S., Carvalho, P., Dardenne, L., Bisch, P., “Development of a Computational Environment for Protein Structure Prediction and Functional Analysis”, Second Brazilian Workshop on Informatics, Brazil, 2003.
22. Jacob, B.,”Taking advantage of Grid computing for application enablement”, <http://www-106.ibm.com/developerworks/library/gr-overview/>
23. Global Grid Forum, <http://www.ggf.org>
24. Foster, I., Kesselman, K., Nick, J., Tuecke, S., “The Physiology of the Grid: An Open Services Architecture for Distributed Systems Integration”, Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.
25. Foster, I, Kesselman, C, Tuecke, S., Nick, J., Tuecke, S., “Grid Services for Distributed System Integration, Computer 35(6), June 2002
26. Globus Alliance, <http://www.globus.org>
27. OGSA-DAI, <http://www.ogsadai.org>
28. Rajasekar, A., Wan, M., Moore, R., “MySRB & SRB – Components of a Data Grid”, The 11<sup>th</sup> International Symposium of High Performance Distributed Computing, Edinburgh, Scotland, June 2002.
29. Hoschech, W., McCance, W., “Grid Enabled Relational Database Middleware, Global Grid Forum, Frascati, Italy, October 2001.
30. Workflow Management Coalition, <http://www.wfmc.org/>
31. Aho, A., Hopcroft, J., Ullman, J., “Data Structures and Algorithms”, Addison-Wesley Publishing Company, 1987.
32. Condor-G, <http://www.cs.wisc.edu/condor/condorg/>.
33. DagMan, <http://www.cs.wisc.edu/condor/dagman/>.
34. BPEL4WS, <http://www.ibm.com/developerworks/library/ws-bpel/>
35. BPWS4J, <http://www.alphaworks.ibm.com/tech/bpws4j>
36. Krishnan, S., Wagstrom, P., Laszewski, G., “GSFL: A Workflow Framework for Grid Services”, <http://www-unix.globus.org/cog/projects/workflow/gsfl-paper.pdf>
37. DAML, <http://www.daml.org>
38. Ankolekar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D.L., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T.R., Sycara, K., and Zeng, H., "DAML-S: Semantic Markup for Web Services". International Semantic Web Workshop (SWWS), 2001
39. Sabou, M., Richards, D., Splunter, S. van, “An experience report on using DAML-S”. Proceedings of WWW 2003 Workshop on E-Services and the Semantic Web (ESSW'03), 2003
40. Kennedy et al, “Toward a Framework for preparing and Executing Adaptive Grid Programs”, Proceedings of Next Generation Systems Program Workshop, International Parallel and Distributed Processing Symposium, 2002, Fort Lauderdale, Florida.

41. Yarkhan, A., Dongarra, J., "Biological Sequence Alignment On The Computational Grid Using The Grads Framework", submitted to Journal on Grid Computing, July 2003.
42. Medeiros, C., Vossen, G., Weske, G., "WASA: A Workflow-Based Architecture to Support Scientific Database Applications", DEXA 1995, pp 574-583.
43. Kochut, K., Arnold, J., Seth, A., Miller, J., Kraemer, E., Arpinar, B., Cardoso, J., "IntelliGEN: A Distributed Workflow System for Discovering Protein-Protein Interactions", Distributed and Parallel Databases, vol. 13, num. 1, Jan 2003, pp 43-72.
44. Hall, D., Miller, J., Arnold, J., Kochut, K., Sheth, A., Weise, M., "[Using Workflow to Build an Information Management System for a Geographically Distributed Genome Sequence Initiative.](#)" *Genomics of Plants and Fungi*, R.A. Prade and H.J. Bohnert, Eds, Marcel Dekker, Inc., New York, NY, 2003, pp. 359-371.
45. Altintas, I. et al, "A Modelling and Execution Environment for Distributed Scientific Workflows", Proceedings of the 15<sup>th</sup> International Conference on Scientific and Statistical Database Management, SSDBM 2003, pp. 247-250.
46. Bhowmick, S., Singh, D., Laud, A., "Data Management in Metaboloinformatics: Issues and Challenges", DEXA 2003, pp392-402, 2003.
47. Bausch, W., Pautasso, C., Schaeppi, R., Alonso, G., "BioOpera: Cluster-aware Computing", Proceedings of the 4th IEEE International Conference on Cluster Computing, 2002.
48. Bausch, W., Pautasso, C., Schaeppi, R., Alonso, G., "Programming for Dependability in a Service-Based Grid", 3<sup>rd</sup> International Symposium on Cluster Computing and the Grid, Tokyo, Japan, May 2003.
49. Blythe, J., Deelman, E., Gil, Y., Kesselman, C., "Transparent Grid Computing: a Knowledge-Based Approach", Proceedings of the 15<sup>th</sup> Conference on Innovative Applications of Artificial Intelligence, Acapulco, August 2003.
50. Deelman, E., et al, "Mapping Abstract Complex Workflows onto Grid Environments", Journal of Grid Computing, vol. 1, number 1, pp. 25-39, 2003.
51. Stevens, R., Robinson, A., Goble, C., "myGrid: Personalised Bioinformatics on the Information Grid", Bioinformatics vol 19, suppl 1, 2003, Eleventh International Conference on Intelligent Systems for Molecular Biology.
52. Addis, M. et al, "Experiences with e-Science Workflow Specification and Enactment in Bioinformatics", Proceedings of UK e-Science All Hands Meeting, September 2003.
53. Cao, J., Jarvis, S., Saini, S., Nudd, G., "GridFlow: Workflow Management for Grid Computing", 3<sup>rd</sup> International Symposium on Cluster Computing and the Grid, Tokyo, Japan, May 2003.
54. Aalst, V., Hofstede, A., Kiepuszewski, B., Barros, A., "Workflow patterns", *Distributed and Parallel Databases*, 14(3), pages 5-51, July 2003.
55. Miller R.T., Jones D.T. and Thornton J.M. (1996) "Protein fold recognition by sequence threading tools and assessment techniques". *FASEB J.*, 10: 171
56. PDBAA, <ftp://www.ncbi.nih.gov/blast/db>
57. Cavalcanti, M., Baião, F., Rössle, S., Bisch, P., Targino, R., Pires, P., Campos, M., Mattoso, M., "Structural Genomic Workflows Supported by Web Services", In: 14th International Conference on Database and Expert Systems Applications

- (DEXA 2003), International Workshop on Biological Data Management (BIDM'03), Prague, Czech Republic, sep 2003, IEEE CS Press, ISBN 0-7695-1993-8, pp. 45-50.
58. Cavalcanti, M., Baião, F., Rössle, S., Bisch, P., Targino, R., Pires, P., Campos, M, Mattoso, M., "Managing Structural Genomic Workflows Using Web Services", submitted to Data and Knowledge Engineering Journal special issue on Bioinformatics.
  59. Scientific Data Management framework Workshop, Argonne National Lab, USA, August2003, <http://sdm.lbl.gov/~arie/sdm/SDM.Framework.wshp.html>
  60. e-Science Workflow Services, Edinburgh, December 2003, <http://www.nesc.ac.uk/esi/events/303/index.html>
  61. Xscufl, Taverna Users Guide, <http://www.cs.man.ac.uk/~markg/taverna/TavernaUserGuide.pdf>
  62. Freefluo, <http://freefluo.sourceforge.net/>