2008

Dynamic workflow management for large scale scientific applications

Emir Mahmut Bahsi
Louisiana State University and Agricultural and Mechanical College

DYNAMIC WORKFLOW MANAGEMENT FOR LARGE SCALE SCIENTIFIC APPLICATIONS

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
College of Basic Sciences
in partial fulfillment of the
requirements for the degree of
Master of Science in Systems Science

in

The Department of Computer Science

by
Emir Mahmut Bahsi
B.S., Fatih University, 2006
August, 2008

Acknowledgements

It is a pleasure for me to thank many people who made this thesis possible. It is impossible to exaggerate my indebtedness to my advisor Dr. Tevfik Kosar. With his support, his enthusiasm, his great efforts to canalize my work by providing invaluable advice, he is the person who should be congratulated before me for this thesis. I wish to thank my committee members for their support during the thesis. This thesis would not be possible without the contribution of Karan Vahi and Ewa Deelman in the implementation of Pegasus by giving useful, and timely information and instructions, Dr. Thomas Bishop for providing me background and giving explanatory information about his work in DNA folding application and also providing priceless feedback for the report, Prathyusha V. Akunuri and LONI team for their user support and prompt responses. I would also like to thank my colleagues and friends Mehmet Balman, and Emrah Ceyhan for their both technical and motivating supports. I acknowledge Center for Computation & Technology (CCT) for providing such a great working environment and financial support. I also thank NSF, DOE, and Louisiana BoR for funding my research. Lastly, and most importantly, I wish to thank my parents Mustafa Bahsi and Songul Bahsi. They bore me, raised me, loved me, taught me, supported me, and be the motivation factor of my life. To them I dedicate this thesis.

Table of Contents

List of Tables

List of Figures

Abstract

The increasing computational and data requirements of scientific applications have made the usage of large clustered systems as well as distributed resources inevitable. Although executing large applications in these environments brings increased performance, the automation of the process becomes more and more challenging. The use of complex workflow management systems has been a viable solution for this automation process.

In this thesis, we study a broad range of workflow management tools and compare their capabilities especially in terms of dynamic and conditional structures they support, which are crucial for the automation of complex applications. We then apply some of these tools to two real-life scientific applications: i) simulation of DNA folding, and ii) reservoir uncertainty analysis.

Our implementation is based on Pegasus workflow planning tool, DAGMan workflow execution system, Condor-G computational scheduler, and Stork data scheduler. The designed abstract workflows are converted to concrete workflows using Pegasus where jobs are matched to resources; DAGMan makes sure these jobs execute reliably and in the correct order on the remote resources; Condor-G performs the scheduling for the computational tasks and Stork optimizes the data movement between different components.

Integrated solution with these tools allows automation of large scale applications, as well as providing complete reliability and efficiency in executing complex workflows. We have also developed a new site selection mechanism on top of these systems, which can choose the most available computing resources for the submission of the tasks. The details of our design and implementation, as well as experimental results are presented.

Chapter 1

Introduction

Importance of distributed computing is increasing dramatically because of the high demand for computational and data resources. Large scale scientific applications are the main drivers for this demand since they involve large number of simulations and these simulations generate considerable amount of data. In order to enable the execution of these applications in distributed environments, many grid tools have been developed. Workflow management systems are one of such tools for end-to-end automation and composition of complex scientific applications. Several workflow management systems are introduced by the grid community and each of these systems have different functionalities and capabilities.

Large scale scientific applications are composed from several tasks which are connected each other via dependencies. These dependencies can be **data dependency** where one task may need output of another task as input or **control dependency** where execution of a task depends on success or failure of another task. On the other hand, some tasks are totally independent from each other and they can run in parallel. Therefore, these tasks should be organized in some order so that dependencies are satisfied and independent jobs are executed in parallel for efficiency.

One of the imperative problems of scientists who are using grid resources for large scale applications is managing every part of application manually, such as submission of tasks; waiting for completion of one task or group of tasks in order to submit the next; submitting hundreds of parallel simulations at the same time; and handling the dependencies between tasks. One solution to eliminate the human intervention and to simplify the management of such applications is via automation of the end-to-end application process using workflows. Besides, task failures are the critical points in the execution of those applications especially in automated systems and they should be handled cautiously. One solution could be detecting task failures prior to the submission and execution of subsequent tasks. Since those applications are running on grid resources, some steps of the applications need large amounts of data transfers. The time consumed in data transfers may form the large portion of the application completion time. Therefore, computational tasks and data transfer tasks should be managed separately and appropriate methods should be used for each of them. Resource selection can also be a factor that should be considered for performance. More simulations should

be run on the resources which provide more throughput in order to increase performance.

1.1 Contributions

Our work in this thesis has three main contributions:

i) Study, analysis and comparison of existing grid workflow management systems. First objective of our study was performing a survey of most widely used workflow management systems in order to analyze and compare their functionalities and capabilities. We were especially interested in dynamic behavior and conditional structures. After studying conditional elements in each system, we have focused on implementation and presented case studies by using some of these conditional structures. For the systems in which those conditional structures did not exist, we were be able to use other primitive constructs to build those structures.

ii) Implementation of end-to-end automated systems for real-life scientific applications. Our second intention was end-to-end automation of two large scale applications: DNA folding and reservoir uncertainty analysis. Our implementation is based on Pegasus workflow planning tool, DAGMan workflow execution system, Condor-G computational scheduler, and Stork data scheduler. The designed abstract workflows are converted to concrete workflows using Pegasus where jobs are matched to resources; DAGMan ensures that these jobs execute reliably and in the correct order on the remote resources; Condor-G performs the scheduling for the computational tasks and Stork optimizes the data movement between different components. Integrated solution with these tools allows automation of large scale applications, as well as providing complete reliability and efficiency in executing complex workflows.

iii) Development of a new site selection mechanism for workflow management systems. Our third goal was to implement a site selector that aims to achieve intelligent resource selection and load balancing among different grid resources. In order to achieve this goal we have implemented two site selectors for Pegasus. Based on the information retrieved from different resources, site selection algorithm maps tasks to sites in which tasks may have higher chance to be completed sooner. We have used our site selectors in UCoMS project and obtained better results compared to Random and Round-Robin site selection mechanisms, which are the default site selectors in Pegasus.

1.2 Outline

Rest of this report is organized as follows: Chapter 2 presents our study of different workflow management systems and their conditional behaviors. Chapter 3 explains our workflow enabling process for DNA folding and reservoir uncertainty analysis applications. Chapter 4 presents the two similar load balancing site selection mechanisms we have developed. In Chapter 5, we provide the related work in this area, and we conclude the paper in Chapter 6 along with the directions to improve the system as future work.

Chapter 2

Survey of Existing Dynamic Workflow Managers

As the complexity of the scientific application increases, the need for powerful grid tools such as workflow managers that handle those applications increases as well. While some workflow managers can only support basic constructs and leave the responsibility of creating dynamic behavior of the workflow inside the executables or user scripts to user, some workflow managers introduce conditional structures and let users benefit from them. The support for conditional structures and similar constructs in workflow management systems is essential for the execution of scientific applications since failure in a task may cause whole application to fail, and in some cases depending on the output or success of previous tasks, one of the tasks from a group of tasks is supposed to be chosen for execution. For instance a transfer failure task may cause whole system to fail especially if the file that is supposed to be transfer is input for a task. In those cases, such as failure of a task, choosing alternative task will prevent whole application to fail.

Several existing workflow managers have support for conditional structure in different levels. While some of them provide **if**, **switch**, and **while** structures that we are familiar from high level languages; some of the workflow managers provide comparatively simple logic constructs. In the latter case, the responsibility of creating conditional structures left to users by combining those logic constructs with other existing ones.

We have chosen some of the most widely used workflow systems to observe conditional behaviors and compare the ease of constructing workflows using them. The systems we have studied are; Apache Ant [1], Askalon [2], DAGMan [3], GrADS [4], Gridbus [5], ICENI [6], Karajan [7], Kepler [8], Pegasus [9], Taverna [10] [11], Triana [12], and UNICORE [13]. Four of these systems do not support any of the conditional structures. However, some structures in these systems can be used to build conditionals. For instance prescript mechanism in DAGMan can be used to imitate if statements. The remaining eight systems support at least one of the conditionals (see Table 2.1).

Table 2.1: Conditional Structure in Grid Workflow Managers

Name	IF	Switch	While
Apache Ant	Y	Y	N
ASKALON	Y	Y	Y
DAGMan	N	N	N
GrADS	N	N	N
Gridbus	N	N	N
ICENI	Y	X	Y
Karajan	Y	Y	Y
Kepler	Y	Y	N
Pegasus	N	N	N
Taverna	Y	N	N
Triana	Y	N	Y
UNICORE	Y	N	Y

Y: Supports.
N: Does not support.
X: Not much information found.

2.1 Support for Conditions in Workflow Management Systems

2.1.1 ASKALON

ASKALON [2], which aims to provide an invisible grid to application developers, is based on an XML-based workflow language called AGWL [14]. AGWL describes workflows in high level of abstraction. In AGWL tasks are connected by data and control flows.

AGWL supports two types of conditional activities: **if** and **switch** structures. Figure 2.1a and 2.1b show two data flows of **if** structure. The data flow is provided by connecting data-in and data-out ports to activities based on the control flow. However, control outcome of **if** or **switch** activity is not known at compile time. Therefore, which inner activity's data-out port should be connected to an activity outside of that conditional activity cannot be determined. As can be seen from Figure 2.1b, this issue is solved by connecting all inner activities' data-out ports to the data-out port of the conditional activity and also connecting the data-out port of the conditional activity to the next activity that comes after the condition structure.

In AGWL there are three types of loop activities: **while**, **for** and **forEach**. The vital part in loop structures in AGWL is handling data flows. There is a conditional structure in **while** structure which determines the loop execution. First task in the **while** loop is connected to the data-in port of the **while** structure or

data-out port of another task from the outside of **while** loop. Data-out port of the last task in the **while** loop is connected to the data-in port of the **while** loop in order to keep the data flow between iterations. **If** condition determines the **while** loop to be exited, data in the data-in port of **while** is mapped to the data-out port of **while** and the next activity after loop can take the data from there.

2.1.2 DAGMan

DAGMan (Directed Acyclic Graph Manager) has been developed as part of the Condor project [3], and acts as the meta-scheduler for Condor. DAGMan handles the dependencies between jobs in the workflow.

Since DAGMan is a simple workflow management system, it does not have advanced constructs such as conditionals. However, some users explored a way of imitating simple **if** structure. They are using pre-scripts to execute the current job based on the previous job result. Actually in every case the current job is executed but the inside of the job is replaced with the no_op task which does not have any effect in the execution of the workflow(Figure 2.1d).

2.1.3 Triana

Triana [12] is both a problem solving and a programming environment. Since it is written in Java, Triana can be installed and run almost on any system.

Triana has a simple user interface for composing workflows of scientific applications. Users do not have to worry about the XML representation of workflow.

Triana has two types of conditional processing element called **if** and **loop**. **If** structure has one input for data which needs to be forwarded and one input for condition. The input for condition is compared with the test value inside **if** structure. If it is smaller than the test value the input data forwarded to the first output otherwise it is forwarded to second output. Therefore, flow of control shaped based on the data flow.

loop structure in Triana has testing mechanism inside which takes an input and forwards input to outside of the loop if condition is met otherwise forwards input to the next task inside the loop. The output of the last task inside loop can be connected to the **loop** structure's second input thus **loop** can take the conditional input for the iterations after the first one.

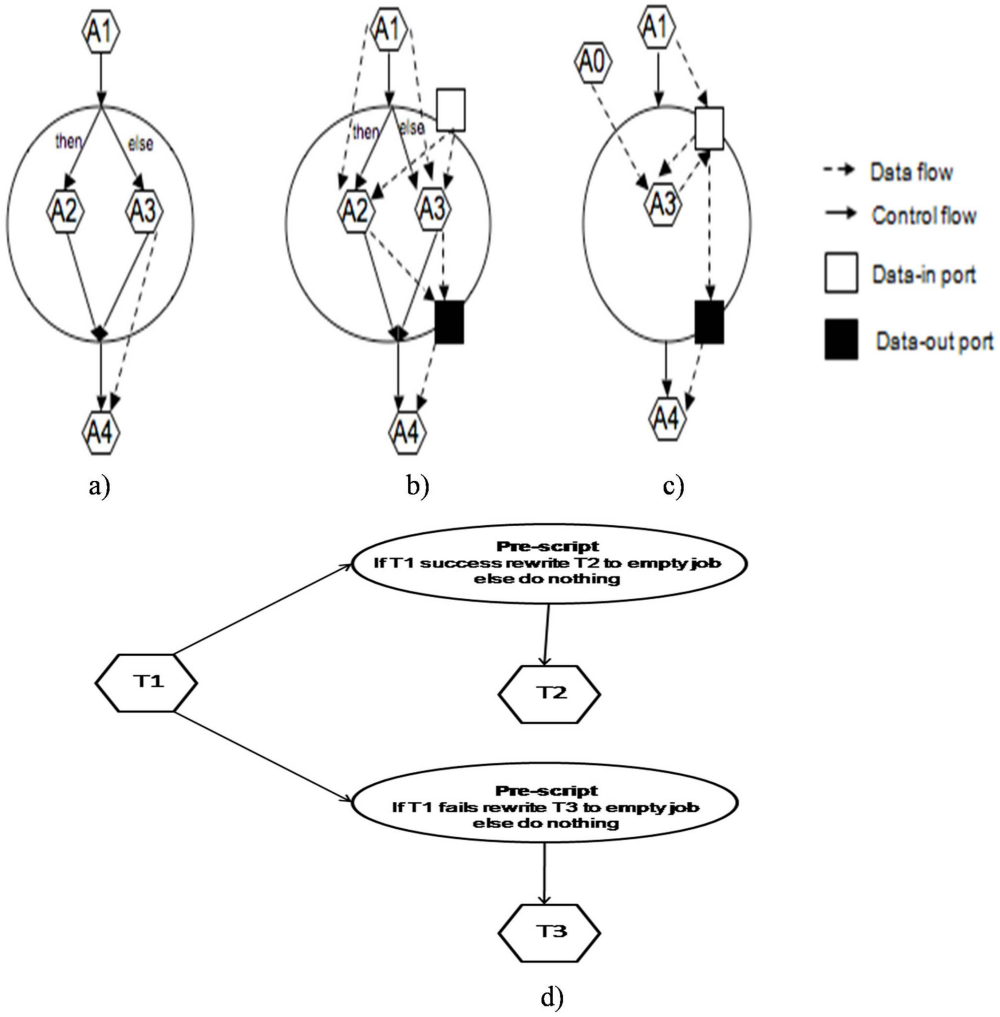


Figure 2.1: Conditional Structures in AGWL [14] - a) Data Flow in Illegal Form in **if** Activity b)Data Flow in Legal Form in **if** Activity c)**while** Loop d)Imitating Conditional DAG in DAGMan [3].

2.1.4 Karajan

Karajan, which is part of Java COG Kit, is developed at the Argonne National Laboratory. Karajan is developed from GridAnt [15] and has additional features such as scalability, workflow structure and error handling [7]. Karajan has two different syntaxes: K-syntax which is very similar to high-level programming languages, and XML syntax which we selected to use in our studies.

Karajan has **if** and **choice** structures as conditionals. **if** structure can be shaped by using the following elements: **if**, **condition**, **then**, **else**, and **elseif**. **Choice** element is very similar to **switch** statement that we are used to in programming languages such as C and Java. Tasks inside the **choice** element are executed sequentially until a successful execution happens. If execution of a task ends successfully the next tasks inside the **choice** element are skipped and the task following the **choice** element is executed.

Karajan has two looping constructs: **while**, and **for**. **while** is used to execute group of tasks until a specific condition becomes false . **for** is used for iterating for a range of values.

In addition, Karajan has some other logical constructs that users can create conditions either using one or combining multiple of them.

2.1.5 UNICORE

UNICORE (Uniform Interface to Computing Resources), being a grid middleware, has an open, service oriented architecture. UNICORE aims to provide seamless, secure, and intuitive access to distributed resources [13]. Via a simple GUI in UNICORE, users can design and execute their workflows which are represented as Directed Acyclic Graphs (DAGs).

UNICORE has conditional execution (**if-then-else**), repeated execution (**do-n**), conditional repeated execution (**do-repeat**), and suspend (time conditional) action (**hold-job**) as advanced control structures and they use **ReturnCode**, **FileTest**, and **TimeTest** as testing conditions.

Control Structures:

- **if-then-else** structure chooses one of two branches for execution. If **ReturnCode** test is used as test condition, a dependency must exist between the previous task and **if-then-else**. It is client's responsibility to check dependency and not to submit non-deterministic jobs.

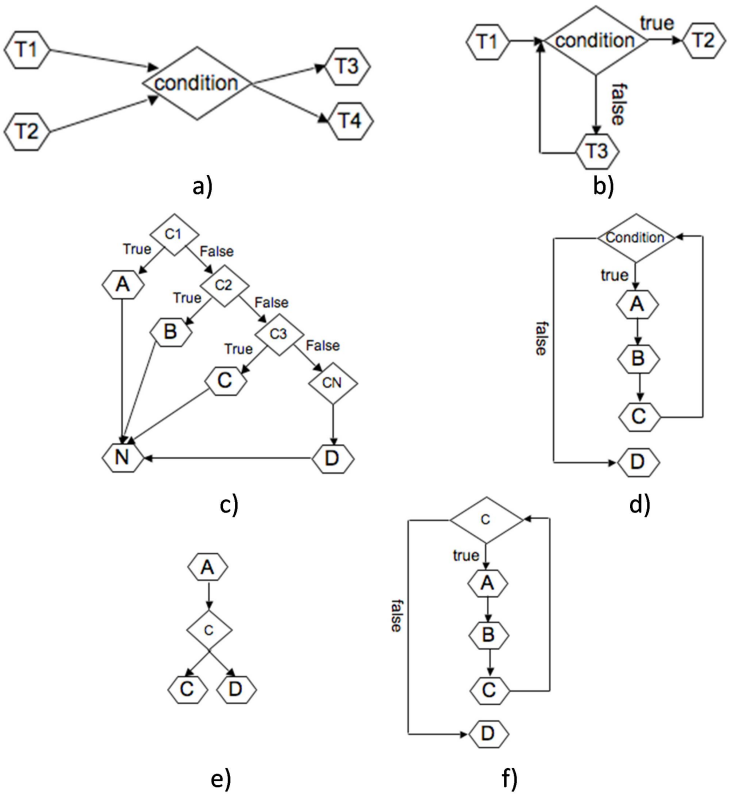


Figure 2.2: Conditional Structures in Triana, Karajan, and UNICORE a) **if** Structure in Triana b) **while** Structure in Triana c) **if** Structure in Karajan d) **while** Structure in Karajan e) **if** Structure in UNICORE f) **while** Structure in UNICORE.

- **DoRepeat** structure iterates group of tasks based on the result of a testing condition. The result of a task is used as return code if **ReturnCode** test is selected as condition.

- **HoldJob** construct, which uses **TimeTest** as the condition, waits for a specific amount of time before executing a task.

- **DoN** structure is similar to **DoRepeat** in the sense that both are iterating group of tasks. However, the number of iterations is specified while composing the workflow in **DoN** task. Therefore, it does not use any test conditions.

Test Conditions:

- **ReturnCode** offers three different choices to users to select from: a) comparing return value of the previous task and the value it has, b) successful execution of the previous task, and c) unsuccessful execution of previous task. Checking for success of executions in UNICORE increases the level of fault tolerance since an alternative task selection can be made in case of a task failure.

- **FileTest** forwards the control flow to a task based on the file status which can be file exists, file does not exist, readable, writable, and executable.

- **TimeTest** executes a task if specified time passed or has been reached.

2.1.6 ICENI

ICENI (Imperial College eScience Network Infrastructure), which is an integrated grid middleware to support e-science, provides and coordinates grid services for eScience applications. Via the GUI of ICENI users can easily build their workflows without caring about XML representation since YAWL (Yet Another Workflow Language) generates the XML format [16] [17] [18].

ICENI has two compositions: spatial and temporal. We are observing temporal composition which represents the workflow of the application. Each component in the workflow is composed by collection of nodes. The types of nodes are: **activity, send, receive, start, stop, andSplit, andJoin, orSplit,** and **orJoin** [6].

Although there is not a specific conditional structure in ICENI, a similar structure to conditions can be done using **orSplit** and **orJoin**. **orSplit** is the node where branching happens and **orJoin** is the node where branches converge. Successful execution of one branch is enough for **orJoin** to transfer control to next node. If one node between **orSplit** and **orJoin** is connected to a node coming before **orSplit**, then a loop structure occurs.

2.1.7 Kepler

Kepler, which is a popular workflow manager, aims to produce an open-source scientific workflow system for scientists to design scientific workflows and execute those workflows efficiently using emerging Grid-based approaches to distributed computation [8]. Kepler is derived from Ptolemy that has many conditional actors. For instance generic filters can use conditions to filter some tokens at the input ports to forward them to their output ports. However, instead of those conditional actors, we are interested in workflow control actors.

Comparator actor is one of the logic actors which has two input ports. It compares the inputs based on the following operators: <, <=, >=, == and returns a boolean output.

Repeat structure iterates the input tokens to the output by specified number of times.

BooleanSwitch actor has a data input, a control input and two output ports: **TrueOutput**, and **FalseOutput**. Based on the value of control input, input data is forwarded to one of the output ports. **BooleanSwitch** can be thought as the closest actor to **if** structure since Kepler does not have **if**. There is also **Switch** actor which is same as **BooleanSwitch** except it has many outputs. Data from the data input port is transferred to one of the output ports which is specified by the value of control input.

Select actor has one control input, one output, and a data input port which is divided into channels. **Select** transfers the data to output port from one of the channels of data input port that is specified by the control input.

BooleanMultiplexor has two data input ports, one control input and one output port. Based on the value of the control input value, one of the data input ports is selected to forward data to output port.

Equals actor has one data input port that has many channels. It compares all of the input port values and produces a true output if all of them are same, produces false otherwise.

IsPresent actor has one input and one output port. It produces true output if data exists in the input port for each firing [19].

2.1.8 Taverna

Mygrid [20] is a collection of comprehensive loosely-coupled suite of middleware such as workflow design and execution, data and metadata management which are designed to support silico experiments in biology. In bioinformatics experiments integrating resources is challenging because of the distribution and heterogeneity of data. Taverna [21] is the workflow manager of the myGrid project which connects distributed web services and other services which are generally provided by third parties.

In Taverna **if** and **switch** structures can be implemented by using **fail_if_false** and **fail_if_true** processors as can be seen in Figure 2.3c, and Figure 2.3d. In the implementation of **if** structure (Figure 2.3c) C and C' nodes represent **fail_if_false** and **fail_if_true** processors. Based on the value produced by T1 one of the C and C' processors fails and causes that branch to fail and the other one executes successfully and gives the control to the next task in the branch.

Similarly in the implementation of **switch** (Figure 2.3d) **fail_if_false**(represented as C) used to implement **switch** structure. The difference is there are java beanshell scripts (denoted by S), which produces a boolean value, comes before C processor in every branch. Based on these values C processors in each branch give the control to the next task or cause the failure of that branch.

2.1.9 Apache Ant

Apache Ant is a java-based software tool for automating build processes. Ant built files are written in XML and each build file should have one project which is a collection of **target**s. **Target** in Apache Ant represents set of tasks and has five attributes: **name**, **depends**, **if**, **unless**, and **description**. In order to compose a workflow, **target**s are connected via dependencies which should be specified in **depends** attributes. If execution of a **target** depends on a condition, **if** and **unless** attributes can be used [1].

Another way of building conditional behavior is using **condition** task. **property** attribute of **condition** task is set when a condition evaluates true. In order to create more specific conditions, conditional elements such as **and, not, or, xor, available, equals, isset,** and **contains** can be used inside **condition** task.

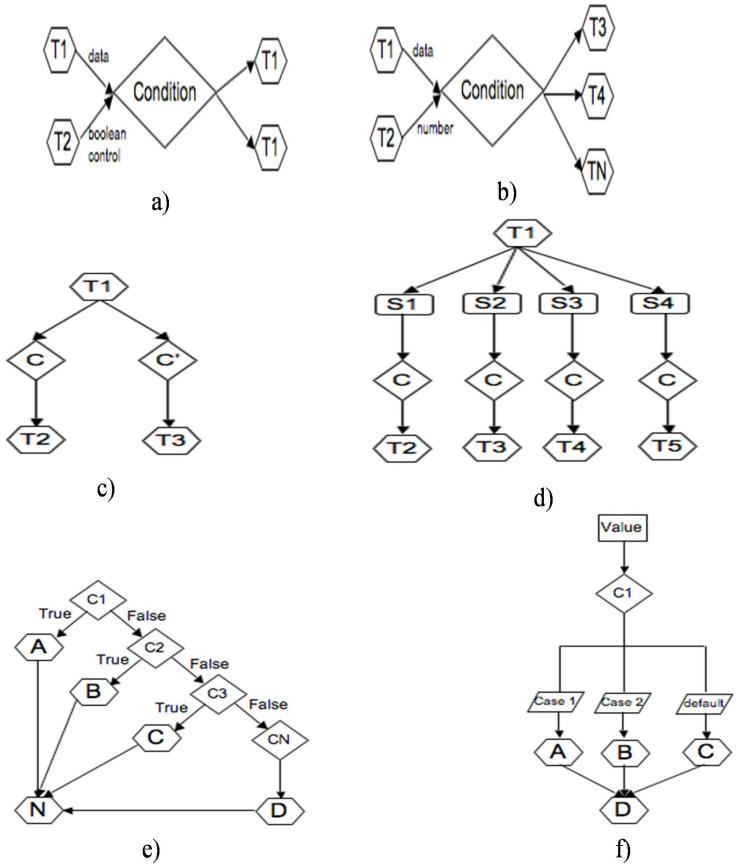


Figure 2.3: Conditional Structures in Kepler, Taverna, and Apache Ant a)**BooleanSwitch** Structure in Kepler b)**switch** Structure in Kepler c)**if** Structure in Taverna d)**switch** Structure in Taverna e)**if** Structure in Apache Ant f)**switch** Structure in Apache Ant

In addition to those core tasks some conditional and iterative tasks are implemented by Ant-contrib project [22]. Those tasks are not added to core tasks group to avoid increasing complexity but they can be used by including relevant source files. Those structures are:

- **If**: **If** structure executes some tasks based on the value of a condition which sets the value of the specified property to true if condition evaluates true. There are many conditional tasks that can be used inside **if** structure. Inside an **if** structure branching can be reached by using **elseif**, **then**, and **else** elements (Figure 2.3e).

- **Switch**: **Switch** structure has an attribute called **value** as the key to check the values that are presented in each **case** element inside **switch**. Based on that value tasks inside the **case** elements are chosen for execution (Figure 2.3f).

2.2 Case Studies

In this section we compare six of the studied workflow management systems in more detail using three different case studies. Those systems are: Kepler, Triana, Taverna, Apache Ant, Karajan, and UNICORE.

2.2.1 Case Study-I

In this case study, we have the following scenario: We have Task A which stages input data and Task C that process this data. The purpose of this study is to introduce an alternating task B that transfers input data from another resource when Task A fails. Figure 2.4 shows the implementation of this scenario in six workflow management system for which we give the details next:

Figure 2.4d represents the implementation of this scenario in Kepler in which we use **execute cmd remotely/locally** task. This task has two inputs: location of the machine where the command will be executed (called as **target** port), string representation of the command (called as **command** port). **exitcode**, which is one of the output ports of **execute cmd remote/locally** task, is connected to a **select** task's control input. When the first **execute cmd remote/locally** fails, based on the value of **exitcode select** task chooses the second alternative command to feed the second **execute cmd remote/locally** task. However, if the

first **execute cmd remote/locally** executes successfully, **select** forwards empty job since the file is already downloaded.

In order to perform our case study in Triana we have implemented our own staging task in Java which produces '4' for successful executions and '1' in case of failures. As can be seen in Figure 2.4e, **if** task is forwarding the flow of control to second **my_stage_in** task or skips it based on the value retrieved from first **my_stage_in** task. **If** task makes the decision by comparing the output of first **my_stage_in** task and test value which is set to '2'.

In Taverna since failure of one task causes all the following processors to fail we have modified our scenario slightly. An input from a user selects which source will be used for data stage in. In order to implement this scenario we have written a java beanshell task to convert user input data to a boolean value. Besides we used **fail_if_true**, and **fail_if_false** for branching, **get_web_page_from_URL** for staging data, **write_text_file** for saving data. As a result based on the user input (which is assumed a task output in real scenarios) one branch is selected for execution (Figure 2.4f).

We have used **if** structure which is implemented by Ant-Contrib project in Apache Ant scenario. For condition of **if** task **http** element is chosen to check the existence of the source URL. Based on the result, one of the **wget** tasks that downloads the input is executed (Figure 2.4a).

Choice element is chosen in order to implement our scenario in Karajan. It includes two **execute** tasks which execute **wget** command to download input file from different sources and an **echo** task for printing error message if both **execute** tasks fail. Since **choice** element executes tasks sequentially until a successful execution is reached, second task is run if the first source is not able to provide the input file (Figure 2.4b).

Figure 2.4c represents our implementation of **if** scenario in UNICORE. We have written three scripts called A, B, and C and used **if** task which is already provided by UNICORE. Task A and Task B have **wget** commands inside which have different URL addresses for downloading the input file and Task C is a simple **echo** command. In the execution of the workflow **if** structure executes Task B when Task A fails to stage the input file otherwise execution of Task B is skipped.

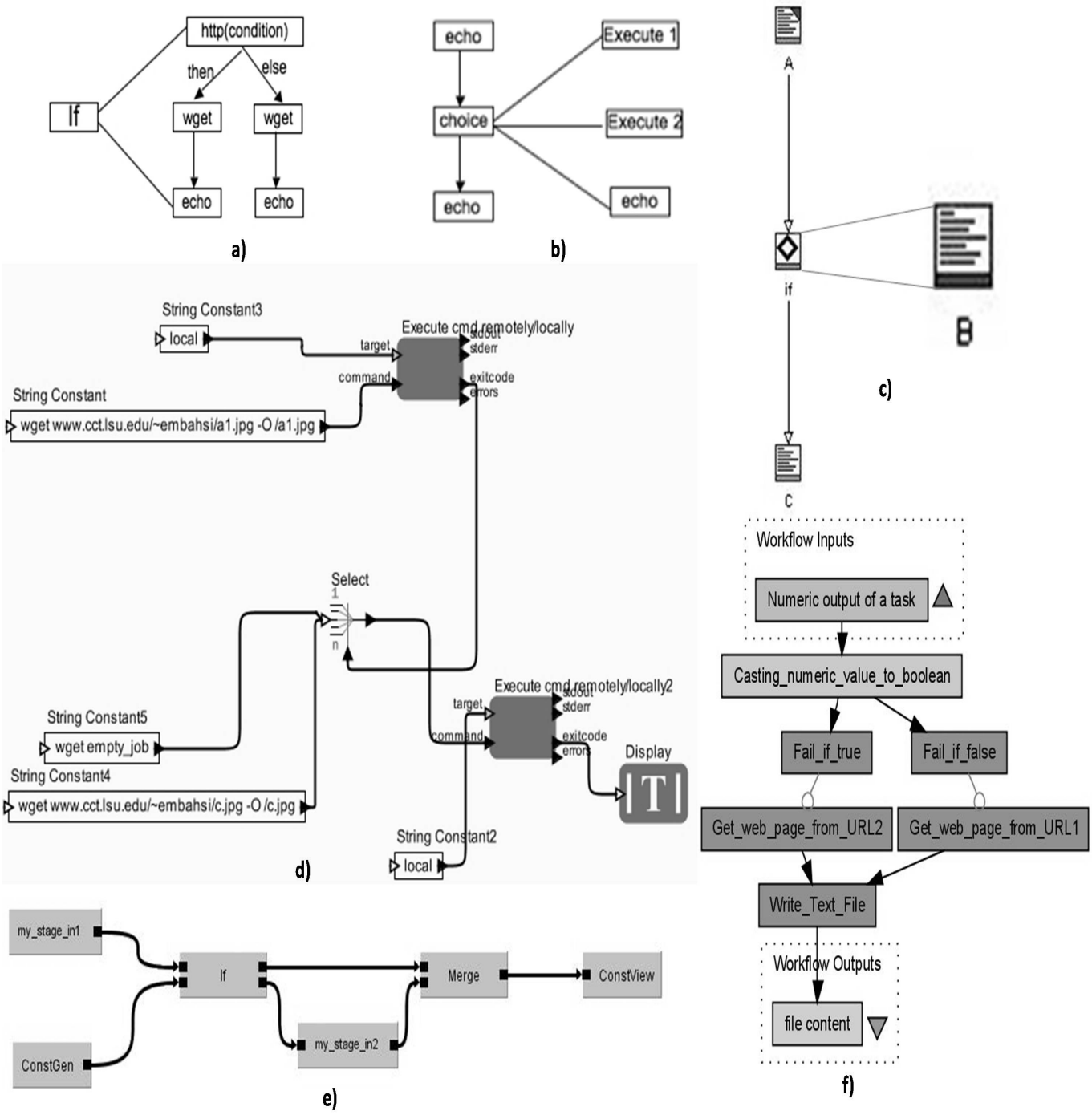


Figure 2.4: Implementation of **if** Structure in: a)Apache Ant b)Karajan c)UNICORE d)Kepler e)Triana f)Taverna

2.2.2 Case Study-II

In this case study, we are trying to imitate **switch** structure by trying to select an available resource for staging input file among more than two different choices.

As can be seen from Figure 2.5d, **switch** implementation in Kepler is very similar to **if** implementation in Kepler except some additional tasks. Since we need more than two alternative sources we are processing the **exitcode**s of the first two **execute cmd remotely/locally** tasks. If the first two sources could not provide the input file for stage in, second **select** task forwards the third alternative URL with **wget** command to the third **execute cmd remotely/locally** for staging.

For our **switch** implementation we choose **execute cmd remotely/locally** task since it produces **exitcode** to provide information about job situation. However, not every task in Kepler produces **exitcode** when a failure occurs; instead many of them throw exception. So in Kepler creating conditional behavior by using logic elements is highly dependent on which tasks are going to be used.

Similar to the implementation of **if** structure in Triana, we use our **my_stage_in** task for **switch** implementation (Figure 2.5e). However, in this case we use one additional **if** and **my_stage_in** tasks. Second **if** condition is used for giving control to the third alternative URL to be used for data stage-in if first two stage-in jobs fail to download the input data. New alternative sources can be added for downloading input file by adding more **if** and **my_stage_in** tasks.

In the implementation of **switch** structure in Taverna **get_web_page_from_URL, write_text_file** and **fail_if_false** tasks are used similar to the implementation of **if** structure (Figure 2.5f). Additionally, we have used three different java beanshell scripts for three branches and each script generates its own boolean value and passes to the **fail_if_false** task. Those branches, which receive the true input execute successfully and the others are not performed. **Switch** implementation can be extended by adding java beanshell scripts, **fail_if_false**, and **get_web_page_from_URL** tasks.

As can be seen from Figure 2.5a an additional **http** condition is used different than **if** scenario in Apache Ant. This **http** condition resides inside the **elseif** element of first **http** condition and makes the decision between running second or third source for downloading input data. **Switch** scenario can be broadened by applying additional **http** conditions, and **wget** tasks.

Figure 2.5b illustrates the **switch** implementation in Karajan. **Switch** implementation in Karajan is

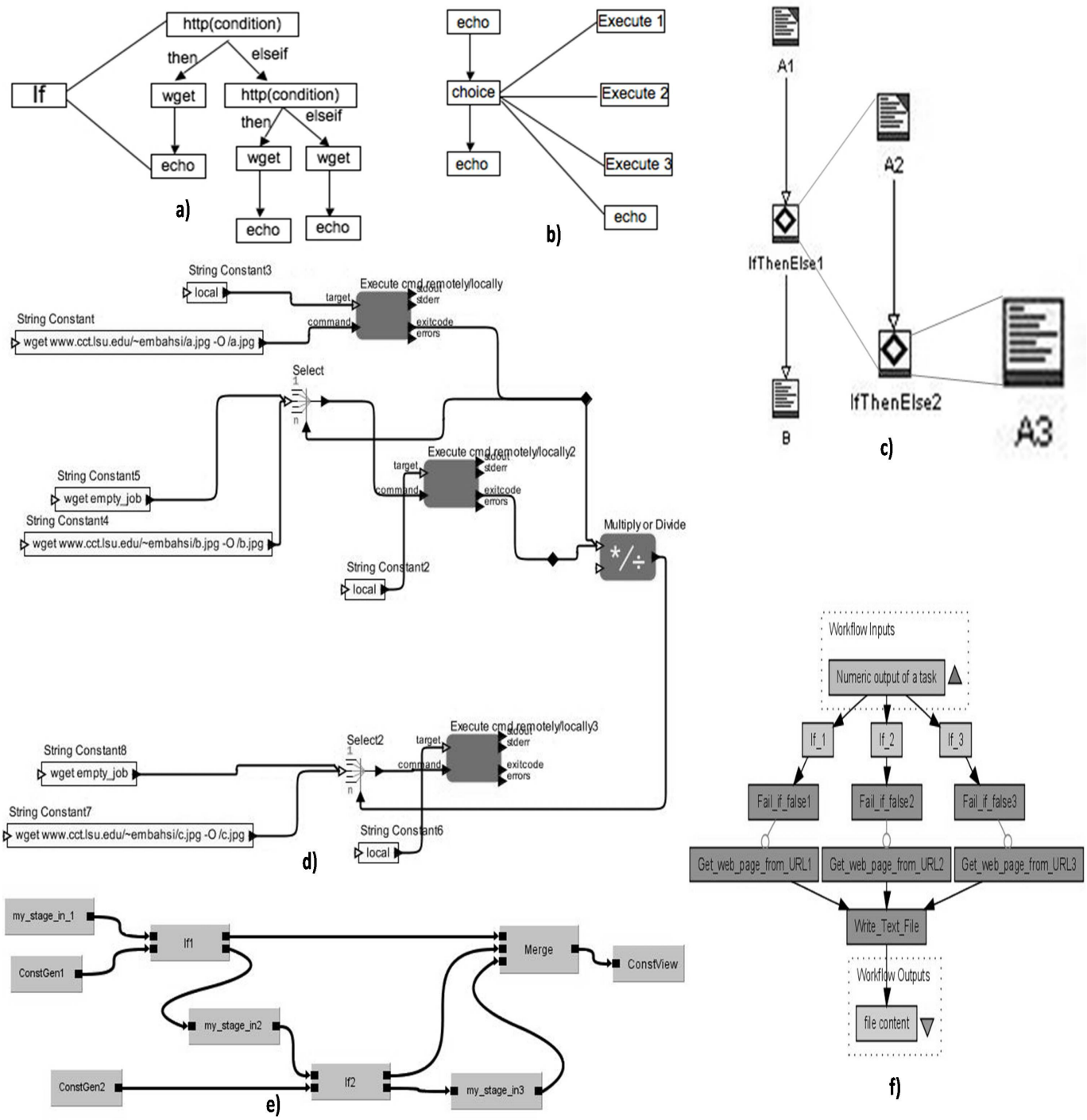


Figure 2.5: Implementation of **switch** Structure in: a)Apache Ant b)Karajan c)UNICORE d)Kepler e)Triana f)Taverna

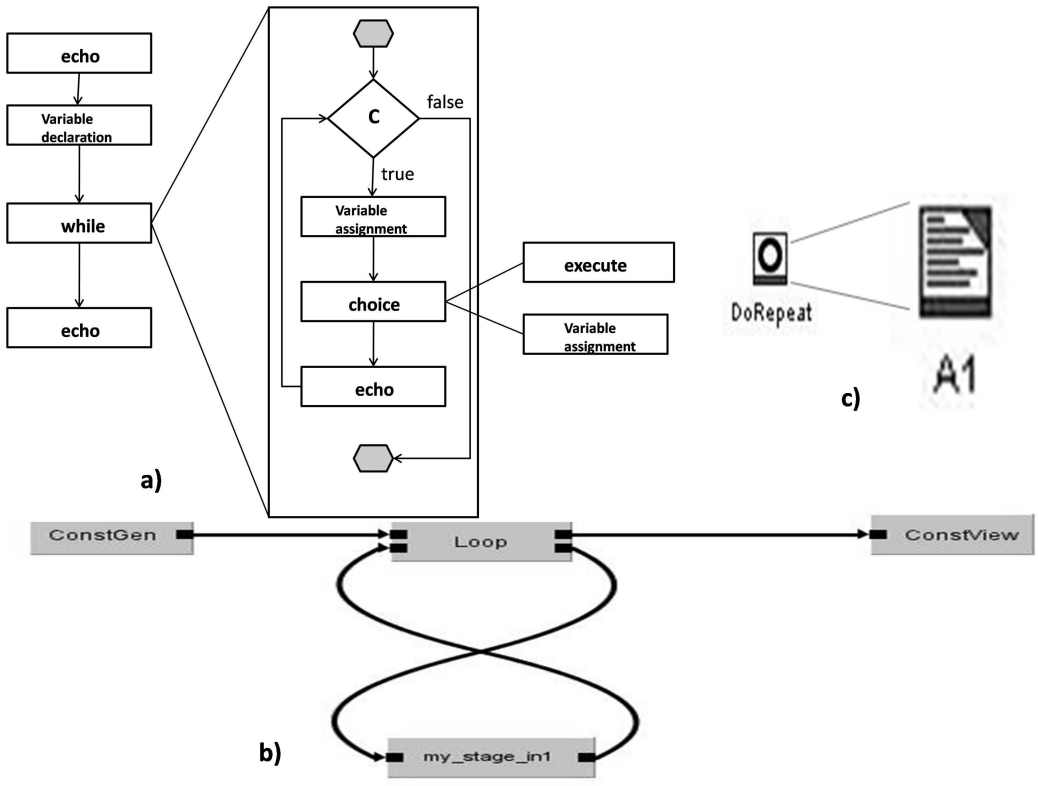


Figure 2.6: Implementation of **while** Structure in: a)Karajan b)Triana c)UNICORE

performed by adding additional **execute** tasks inside the **choice** element. If the first two **execute** tasks fail, third **execute** task is performed. Therefore, new alternative sources can be added to **switch** implementation by increasing the number of **execute** tasks inside the **choice** element.

In UNICORE we use two **ifthenelse** tasks to imitate **switch** structure since **switch** structure does not exist in UNICORE. In addition, we make use of three different alternative sources for transferring input data by creating **wget** scripts: A1, A2, and A3. Executions of these scripts are controlled by **ifthenelse** and they are performed sequentially until one of them successes. **Switch** implementation can be expanded by adding an **ifthenelse** task and a **wget** command for each additional alternative resource (Figure 2.5c).

2.2.3 Case Study - III

In this section we aim to imitate **while** structure by implementing a loop structure which iterates some part of workflow under a condition is met. We used the same scenario as downloading input data in the loop structure. Even though keep on trying the same source until the input file is downloaded is not an efficient

way for fault tolerance, we wanted to protect the integrity of our case study scenarios.

As can be seen Figure 2.6b, **while** implementation in Triana is performed by using a **loop** task and **my_stage_in** tasks. Based on the failure code received from **my_stage_in** task **loop** task repeats the execution of **my_stage_in** until success code is retrieved.

Implementation of **while** scenario in Karajan we use **while** element, a **choice** element inside the **while**, and a **condition** element inside the **while** structure that checks a variable's value. This variable is set to '0' inside the **choice** element when the execution of input download fails. At the end of each iteration of **while**, control comes back to the **condition** element and it decides whether loop should be executed again or not by checking the value of the variable which is set inside **choice** (Figure 2.6a).

As can be seen from Figure 2.6c **while** scenario in UNICORE is implemented by using a **DoRepeat** task which includes a **wget** script called A1. **DoRepeat** task iterates the A1 task until a successful completion occurs.

2.2.4 Discussion

Our study shows that level of conditional support in each workflow manager is quite different. While some systems such as UNICORE and Karajan support almost all the conditional structures which will satisfy users needs in most scenarios, some other workflow managers have very limited support for conditional structures in terms of functionality and usage. In addition, some workflows have conditional tasks that are very specific for a use case and cannot be used in other situations. For instance, in **if** and **switch** implementations, we use **http** task as the condition to check the existence of a file for stage-in. Therefore, **http** task can only be used in such cases and cannot be used as a combination with some other tasks.

In some of the workflow systems, failure of a node may cause whole workflow to fail. Since this may degrade the support for conditionals, users may overcome this issue in some cases by selecting proper tasks. For instance, in Kepler although many of the tasks cause whole workflow to fail in a task failure **execute cmd remotely/locally** can be used in some scenarios since it produces an exit code in case of an unsuccessful execution. While failures of processes in Taverna cause whole workflow to fail, there are some mechanisms such as retry, delay, and backoff in order to increase the level of fault tolerance. In Triana control flow is achieved by passing data to the appropriate branch. In Triana we have written our own task

my_stage_in to implement three case studies. Likewise, users may need to write their own tasks in Triana in order to retrieve an exit code or an output value from that task. Although it seems like extra work, in many situations this can be handled by doing some modifications in the codes of existing tasks or by adding an extra output port.

One important factor for choosing a workflow manager can be ease of use. Based on our experiences, UNICORE is the system which we spent least amount of time on both installation the system and implementation of the workflow. The most effective mechanism in UNICORE in terms of conditional support is the **ReturnCode** test condition that exists almost every conditional task. This condition can be very useful in two very common cases: when an alternative task needed to be run in case of a task failure and a task execution depends on the return value of one task.

For some users, the length of code for implementing the workflow can be necessary. Based on our implementations we have written the shortest codes in Karajan and Apache Ant while the most code generation is needed in Triana and Taverna. However, the systems that generate longer codes have graphical user interfaces to compose the workflow by drag-and-drop mechanism in which users do not have to worry about coding. Still, having a graphical user interface and the length of code can be a considerable issue for some users.

if-type conditional structures can be studied in two parts: **exclusive choice** is the point where one of the branches is chosen for execution and **simple merge** where those branches are merged without synchronization.

There are some points in the workflow called as **multi-choice** where a number of branches are chosen for execution. However, in these situations it can be very hard to decide which branches should have been synchronized in the point of merge.

Improper usage of **split** and **join** constructs may result a deadlock situation. For instance, using an **OR-Split** and **AND-Join** may result deadlock since **OR-Split** may prevent execution of some branches while **AND-Join** will wait all branches to finish their executions.

In every workflow some tasks may fail in the execution. In order to prevent whole workflow to fail in those circumstances, workflow manager should have some mechanism that enables the execution of alternative tasks. In addition, since making decision of executing alternative tasks in compile time is impossible

in case of a task failure, this decision should be made in run-time. Therefore, like the control flow, data flow should also be provided by workflow managers.

Chapter 3

Workflow Enabling Scientific Applications

In this chapter, we present how we have automated a real-life scientific application via use of advanced workflow management tools. This application is DNA folding.

Thomas Bishop and his research group at Tulane University study DNA and chromatin structures and their dynamics which can be represented by the help of mathematical modeling procedures and molecular dynamics.

3.1 Science Background

Identification of how DNA sequence proteins rotate the global structure and dynamics of chromatin is one of the hot topics in biomolecular sciences and still needs more research to be performed. In order to find some useful information that may give them some hints to resolve the effects of the protein on the conformation and dynamics of the DNA, they try different molecular dynamic simulations which model DNA and protein interactions [32].

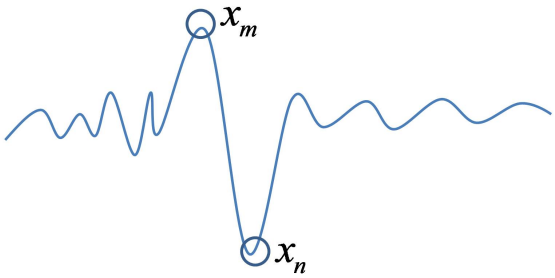
There can be three different approaches designed for modeling DNA folding. The simplest and the fastest one is coarse grain model which can process 10M base pairs in the order of 10 min to process themin single CPU. However, it ignores many variables that effect the simulation but beneficial in the sense of giving a general understanding. Typical formula used is a coarse grain model is shown in Figure 3.2.

A more informative approach than the coarse grain model which can be called as cheap atomic model. This method ignores water in which DNA structure exists originally. This method is very slow compared to coarse grain model. It can process 3000 base pairs in a day with a small cluster that has 3 nodes each with 4 processors.

Solvated atomic model can be counted as one of the most informative approaches. However, it is the slowest model among those three models. It will take about 2 weeks to process 146 base pairs in the same cluster that has 12 processors.

Figure 3.1: Folded DNA Structure [33]

Sequence: ATTGAGT…..



$$\sum_{i=1}^{146} \frac{1}{2} k_i (x_i - x_{oi})^2$$

where;

x_{oi} : equilibrium of ith basepair step

x_i : conformation of ith basepair step of nucleosome

k_i : stiffness of ith step

Figure 3.2: Coarse Grain Model Formula

3.2 Biological Tools Used for Simulations

3.2.1 Amber

Amber(Assisted Model Building and Energy Refinement) is a suite of programs for biomolecular simulation and analysis which is developed in University of California, San Fransisco. It is written in Fortran 90, and C with the support for most major Unix-like systems and compilers. Programs that are included in Amber are:

- Leap: is used for preparing input files for the simulation programs,
- Antechamber: automates the process of parameterizing small organic molecules using GAFF,
- sander: is the central simulation program and provides facilities for energy minimization and molecular dynamics with a wide variety of options,
- pmemd: is somewhat more feature-limited reimplementation of sander by Bob Duke. It was designed with parallel processing in mind and has significantly better performance than sander when running on more than 8-16 processors,
- nmode: calculates normal modes,
- ptraj: provides facilities for numerical analysis of simulation results,
- MM-PBSA: allows for implicit solvent calculations on snap shots from molecular dynamics simulations [24].

3.2.2 3DNA

3DNA is a sophisticated software package for the area of biomolecular simulations which is based on a standard reference frame [26]. It is used for analysis, rebuilding and visualization of three-dimensional nucleic acid using a standard reference frame. More specifically, 3DNA can process complex patterns in DNA and RNA structures such as antiparallel and parallel double helices, single-stranded structures, triplexes, quadruplexes and many more. 3DNA accepts DNA and RNA structures in the format of Protein Data Bank (PDB). Recognition and classification of all base interactions and the double helical character of proper base

pair steps are performed by the analysis utilities. Full atomic models with the sugar-phosphate backbone and publication quality 'standardized' base stacking diagrams and rectangular block demonstration of nucleic acids are generated by the rebuilding tools of 3DNA. In order to have a complete visualization, 3DNA can also place the base pairs and helical regions in a structure and arrange structures. [25]

3.2.3 NAMD

NAMD(Nanoscale Moleculer Dynamics) [27], which is a molecular dynamics simulations program designed for high-performance simulation of large biomolecular systems, can run in indivual workstations as well as small cluster and large-scale supercomputers. NAMD can also work with AMBER, and CHARMM potential functions, parameters, and file formats. Unlike other molecular dynamics tools, NAMD is developed especially for distributed memory parallel systems. In order to provide maximum scalibility NAMD uses a spatial decomposition scheme and a multithreaded, message-driven design. [28]

In addition, as permitting to perform standard simulation to beginners, providing advance functionalities for experts, and being introduced to the science committee free of charge, NAMD is appreciated by many users.

3.2.4 VMD

VMD [29] is designed for visualizing and analyzing molecular objects especially for biopolymers such as proteins and nucleic acids. By applying many rendering techniques and coloring styles VMD can demonstrate any number of structures at the same time. A selected subset of atoms can be displayed with a chosen rendering and coloring method. Each of these displays are called representations. Atoms viewed in each representation are selected with the help of logic operators and regular expressions.

VMD can also be used for visualizing MD simulations. It can perform animation from a running MD simulation as well as from an input file.

3.2.5 GLUE Languages

- Tcsh(Tenex C shell) [31] is a Unix shell based on and compatible with C shell(csh). It has features such as command line completion, commmand line editing and many more.

- AWK [30], which is created at Bell Labs in the 1970s is a programming language for processing text-based data. It is a standard feature of most of the Unix-like operating system nowadays.

3.3 Grid Technologies Used for Applications

3.3.1 Condor/Condor-G

Many scientific applications need large amount of computational power over a long period of time, instead of delivering huge amount of power over a short period of time. In other words, those applications need High-Throughput Computing environments which concentrate on completing as many jobs as possible over a long period of time instead of High-Performance Computing which is highly-tuned systems for small number of jobs executed for short periods of time.

Condor Project, which began in 1988 in the University of Wisconsin-Madison, aims to provide high-throughput computing environment for scientific applications.It is a workload management system designed especially for compute-intesive jobs. Like other full-featured batch systems, Condor provides a job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs either one by one or all of them composed as a workflow. Condor passes those jobs into a queue and selects the resources where the jobs will be execute and the time when jobs will start execution based on a policy. In addition, Condor monitors the progress of the jobs, and informs the users about the jobs' statuses.

As large clusters are especially designed for needs of large scale scientific applications, total of idle cycles of workstations in institutes can compete with clusters' computational power. That was one of the motivitations of the Condor team in the development of Condor. The main goal is to get benefit from unused computing resources that would be wasted otherwise, and to schedule jobs to those resources.

Condor-G stands for Condor-Globus. Using Condor-G, users can also submit their jobs to remote clusters which have job schedulers other than Condor. In order to achieve this Condor creates globus RSL file that is specific to each job manager and submits jobs with globus-authentication. Without the help of Condor-G users would have to connect to remote host, create their job submit file specific to that job manager and submit every job manually or create globus RSL file for every job, and submit them to remote machine without actually logging in.

3.3.2 DAGMan

As we have mentioned in section 2.1.2, DAGMan is the workflow manager of Condor which allows users to submit large workflows to Condor. As can be understood from the name it manages DAGs (directed acyclic graphs) and dependencies between jobs. In a DAG, nodes represent the jobs and edges respresent dependencies between those jobs. DAGMan is the meta-scheduler of Condor. Based on the job orders represented by dependencies in a DAG, DAGMan submit those jobs to Condor. Each DAG of an application should be defined by DAGMan input file. Each job inside the DAG should also be defined seperately as a Condor submit file.

3.3.3 Stork

Stork [23] which is a batch scheduler specialized in data placement and data movement, aims to make data transfer a first class entity in a distributed computing environment. It introduces a high level of fault tolerance for data transfer. It chooses one of the underlying transfer mechanisms for data transfer jobs. In case of a failure in the selected transfer protocol Stork tries alternative protocols that are specified between two sources with a retry mechanism until a specified number of times.

Stork is bundled with Condor release packages. In DAGMan, data transfers can be specified as either a Condor job and the responsibility of transfer is left to the users' transfer program or a Stork Data job with the keyword "DATA" in the DAG submit file.

3.4 Implementation

We have the scripts written by Dr. Thomas Bishop, which aim to perform number of simulations for DNA binding. Those scripts are designed using the second approach which we call as minimal MD model as we have discussed in Science Plan Section. The order of how scripts are executed illustrated in the Figure 3.3.

While the Figure 3.3 gives a general idea about the execution flow of scripts, description of each script is explained following:

- Connect: Connecting to system either sitting actually in front of the machine or connecting to it remotely via ssh.

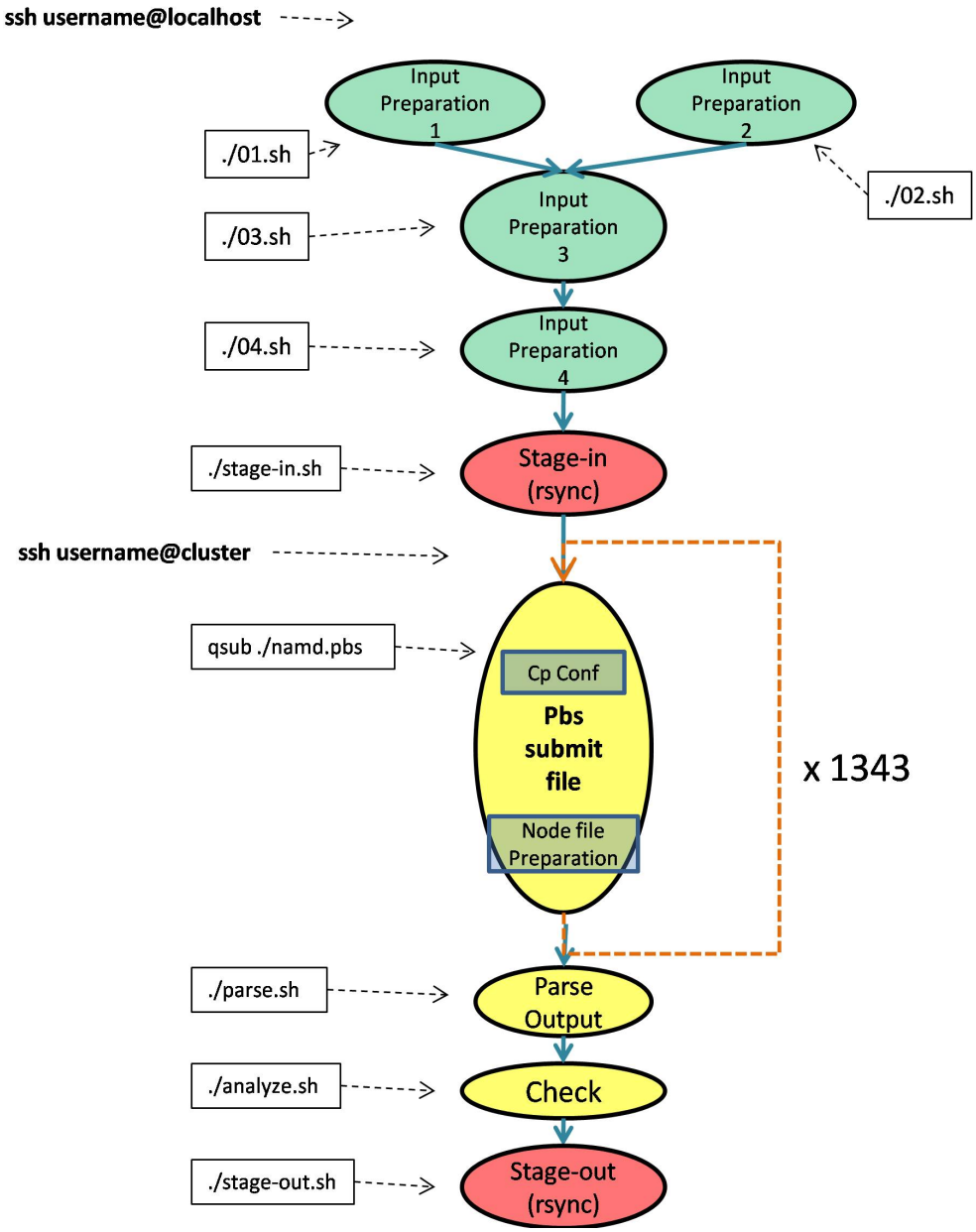


Figure 3.3: Execution Flow of MD Simulation Scripts

- Input Preparation 1: Downloading DNA structure. Processing it with a script to produce pdb([A-H].pdb) files. Downloading nucleotide sequence (V01175.fasta).

- Input Preparation 2: Creating structure parameter files (dna.[0-1343].par) for rebuild by using sequence file (V01175.seq) and another parameter file (ki-all-11.par). Using rebuild command, which is provided by 3DNA, to generate nucleic acid structure as pdb files (dna.[0-1343].pdb) based on structure parameters from parameter files (dna.[0-1343].par). Creating seq.[0-1343].txt files.

- Input Preparation 3: Deleting unnecessary files. Creating the other necessary inputs ([0-1343].crd, [0-1343].parm7, [0-1343].pdb, [0-1343].vmd.pdb) that are not created in first 2 steps. To create these input files some of the outputs (dna.[0-1343].pdb) of first two scripts are used.

- Input Preparation 4: Creating proper directories (sys, sys/analy, sys/sims) under main directory (1eqz). Moving input files ([0-1343].crd, [0-1343].parm7, [0-1343].pdb, [0-1343].vmd.pdb, dna.[0-1343].pdb, dna.[0-1343].par, seq.[0-1343].txt, dna.[0-1343].log), which have been prepared in previous steps, from different folders to a directory (sys/[0000-1343]). The aim is to collect the input files for each simulation in separate directories ([0000-1343]) under one directory (sys) to make it neat and easy for data transfer.

- Stage in: Transfering data from localhost to cluster. The directory that has all simulation files (sys) is transferred from localhost to cluster via rsync shell-command.

- Connect: Connecting to cluster via ssh.

- PBS Submission: Copying configuration file (min1.conf) for **namd** from a specific folder (utils) to each simulation directory (sys/[0000-1343]). Same configuration file is copied since every instance of simulation uses the same configuration file. Each simulation, which is basically a **namd** job, is submitted to PBS queue sequentially. Submition of each **namd** job is done after the completion of previous one. Prior to each **namd** simulation a nodefile is prepared which is needed while running **namd** in MPI. Each namd job is executed in 4 processors via MPI. namd executable takes configuration file(min1.conf) as command line argument. The output is stored in min1.out file in each simulation directory. The output has the energy values.

- Parse Output: The line that has the energy value 2000 is parsed for each output of namd (min1.out) and written to console.

- Check: The number of lines that has the energy value 2000 in each **namd** output file (min1.out) is found. Based on this number simulation is determined as fail or pass.

- Stage-Out: After the simulation the directory that has the simulation results (1eqz/sys) in cluster is transferred back to local machine.

Although scripts are able to perform simulations successfully there are still some deficiencies that should to be considered. First of all, since those scripts should be run in a sequence, they are forming an execution flow. Currently, user is responsible for submitting a script for execution and wait for the completion of that script, check whether any error occurred prior to submitting the next script in the execution flow. This causes an extra work for users since collection of scripts are considered as one scientific application. Besides, every simulation is executed in a sequence although they are independent and could be executed in parallel in order to get results quickly. Data transfer in the current execution flow is handled by rsync command of Linux for which there can be more efficient and fault tolerant mechanisms. Last but not least, there is not much fault tolerance in the whole execution flow. If any of the scripts fails, user should check that script manually and should rerun it.

In order to cope with those deficiencies we have implemented a workflow in Condor. As can be seen in the Figure 3.4, each script is defined as a job in the workflow and they are connected via dependencies. When the DAGMan submit file and Condor submit files for each job are defined, users can submit the workflow to Condor with one command line and Condor executes each job in the way that they are defined and connected. **namd** simulations are defined as parallel jobs in the workflow in order to increase the throughput by using different nodes and even different clusters. **namd** simulation jobs in the workflow (namd_0 mpi - namd_1343 mpi) are condor submit files that are not calling any scripts. They are directly executing namd executable in the machine they are submitted. If the job will be executed in the remote site which has any job manager (PBS, Loadleveler or LFS), Condor generates globus-rsl file for that job and runs it remotely. In addition, we have preferred using Stork against rsync command which is more efficient and fault tolerant. It is more fault tolerant in the sense that we can specify alternative transfer protocols

and it has retry mechanism which is useful for network problems. In order to increase the level of fault tolerance in the system, we can add retry mechanism for the jobs where necessary. For instance, in our experiments, some simulation jobs were failing although nothing seemed wrong in the executable, or input file. However, when we reran the same simulation we were getting the correct results. Therefore, retry mechanism is tremendously practical for the simulation jobs in our workflow.

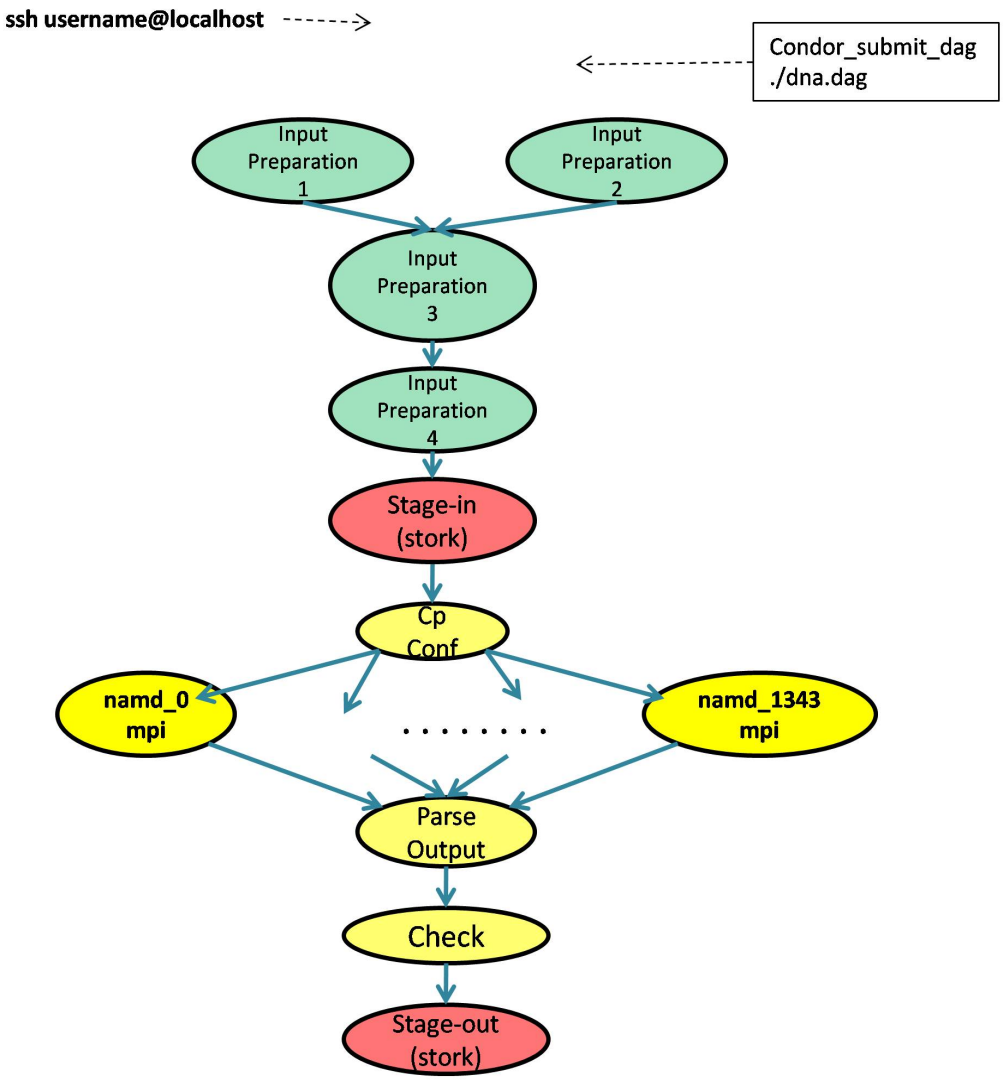


Figure 3.4: Condor WorkFlow of MD Simulation Scripts

Chapter 4

New Site Selection Mechanisms for Pegasus System

Some of the grid workflow management systems require their users to prepare scientific applications in concrete workflows. In concrete workflows, every task in the workflow must be defined in detailed. Some types of information that should be provided for each task are site URL, executable path, arguments for the executable, input files, error files, log files, and some Globus arguments such as project allocation, the queue that is going to be submitted.

On the other hand some of the grid workflow managers need abstract workflow from their users. In those systems static information for each task is stored in catalogs such as site catalog, transformation catalog, and replication catalog. For instance many types of the information about sites are stored in site catalog such as; URL, environment variables, jobmanager type, working directory. Likewise executable details such as path of executable, which site the executable resides are stored in transformation catalog and locations of all the input files and output files that are needed or produced stored in replication catalog. In those systems, once catalogs are prepared well, it becomes easier for users to compose their workflows without worrying about file locations, site locations and executables. In addition, users do not have to create submit files for each jobmanager (PBS, Loadleveler or Condor) for each site since those systems generate the submit files automatically based on the sites that are provided in the planning of the workflow. SWIFT [34] and Pegasus [9] are the example of this type of workflow managers. Both of them are using Virtual Data System (VDS) [35] that provides a set of tools for expressing, executing, and tracking the results of workflows.

One goal of such systems is to make workflows independent from locations of sites. This is a very essential issue for scientific computation since sites in grid can be down, and become running frequently. In those situations for the systems, which require users to submit concrete workflows, users need to create new submit files for every task in the workflow and they have to decide which tasks will be executed in which sites. However, for the systems that work with abstract workflows, only catalogs should be modified and users need to specify the sites (in which workflow will be executed) on the planning phase. One uncertain

point in these systems is site selection mechanism where there are more than one site that is eligible for a task to be executed. In these situations there should be a site selection policy to choose one of the sites for execution of the task. There are simple site selectors such as constant, weighted, round robin, and random. In addition, opportunistic site selectors keep the statistics of the previous job execution times and select the site that takes the shortest time to finish tasks. There are also policy site selectors that schedule tasks to the sites that they can receive favorable policy. Most of the site selection mechanisms aim to schedule the tasks in sites that execution of the whole workflow will be completed in the shortest time.

Based on our research and needs of scientific applications, we have developed a site selector mechanism for Pegasus workflow manager which performs load balancing among different resources.

4.1 Pegasus

Pegasus, which is developed at ISI (Information Science Institute), is one of the grid workflow management systems, that expects abstract workflow from its users. It processes abstract workflows in order to map jobs into resources and manages the execution. Figure 4.1 represents usage of Pegasus in converting abstract workflows to concrete workflows and the tools Pegasus uses. At the top level Pegasus accepts abstract workflows and these workflows can be produced in different ways such as; portals, GUIs, and workflow languages. Abstract workflow is converted to concrete workflow by Pegasus in which computation jobs are mapped to specific resources and data transfers are made clear. In order to accomplish this Pegasus generates the DAG file and proper condor-submit files for each job. Based on the resources specified in condor submit files, jobs are submitted to resources. If some jobs will be submitted to remote resources, proper globus-rsl files are created by Condor and job submission is done by Globus.

4.2 Load-Aware Site Selectors for Pegasus

We have developed two site selectors for Pegasus in order to provide simple load balancing between sites as a result to have the shorter completion time for a workflow. Jobs are mapped to resources sequentially in Pegasus. Both site selectors query every site and get the number of free nodes, total number of nodes, and number of jobs in the queue for each site only when the first job needs to be mapped. They store the information and use that information for all other jobs that are needed to be mapped. In the first site

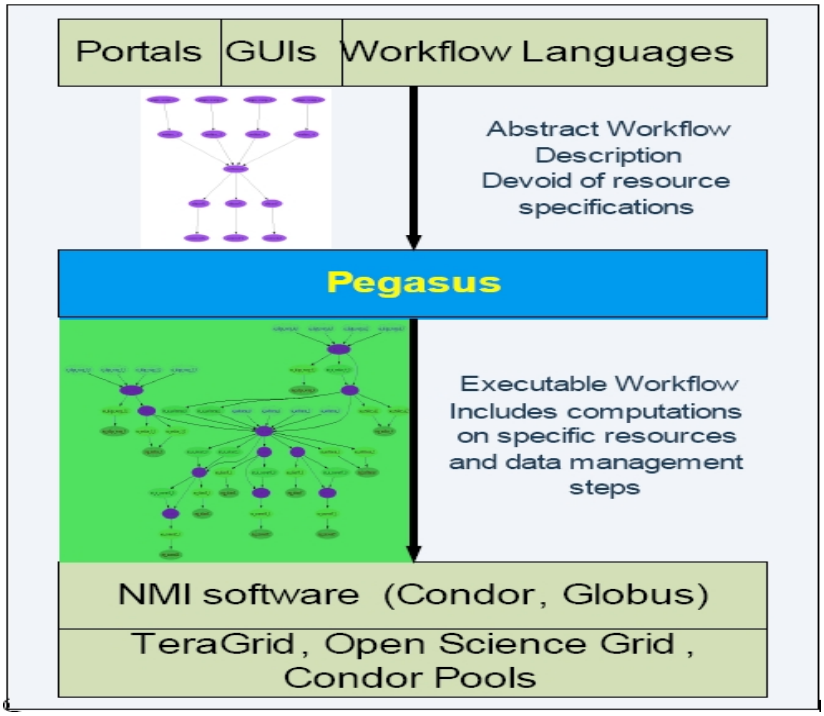


Figure 4.1: Pegasus in Practice [36]

selector(SS1); for each job the sites, which have free nodes, are mapped in round-robin fashion. While mapping the jobs number of free nodes for each site is decremented by one and number of jobs in the queue is incremented by one if job limit is exceeded. Job limit is the number of jobs that can be assigned to a site at a time by each user. Since job limit can be different for each site, users can specify job limit for each site in a file. This file is read by load balancing site selector of Pegasus and if user does not provide such a file job limit is estimated by using the total number of nodes in a site. This mapping strategy runs until there are no free nodes left in each queue. When there are no free nodes left in each queue, fullness of queues are evaluated by considering number of jobs in the queue and total number of nodes in each site. The site whose queue has the smallest fullness value is mapped for the next job and the number of jobs in the queue is incremented by one for that site.

Figure 4.2 summarizes every step in the planning and execution part of the workflow in Pegasus using one of the newly-implemented site selectors. When a user submits an abstract workflow, Pegasus-planner collects information from replication, site, and transformation catalogs. Based on the information collected, eligible sites are evaluated and sent to the site selector for each job. Site selector queries all the sites and stores the received information only once when the first job is needed to be mapped. By using this

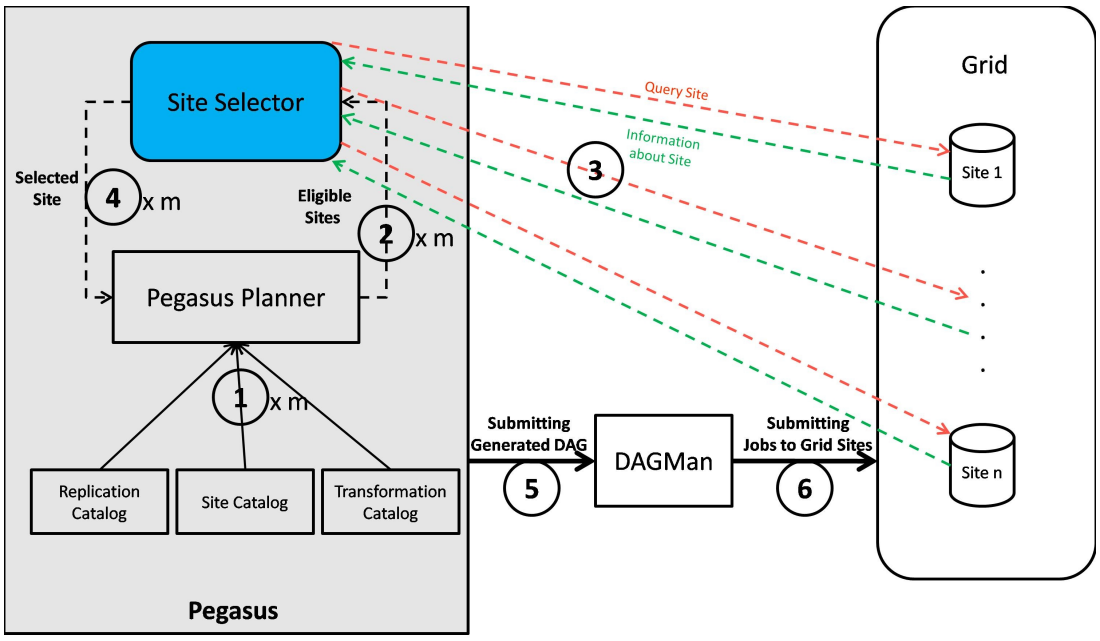


Figure 4.2: Using Newly-Implemented Site Selectors in Pegasus

information in the load balance algorithm one site is selected for execution of each job. After each job in the workflow is processed in the same way, generated dag file and job submission files are submitted to DAGMan. DAGMan submits the jobs to the sites in the sequence which is defined in the DAG file.

Figure 4.3a and 4.3b are examples of how site selection can be done in Pegasus using SS1 site selector. In Figure 4.3a Site1, Site2, and Site3 has 8, 6, and 4 empty nodes, respectively. We have set of jobs that are independent from sites, in other words, they can be executed in any of the sites provided. These jobs are mapped to the sites sequentially considering the free nodes left in each site. Until the 12th job they are mapped to three sites in round-robin fashion. After the mapping of 12th job, there will be no free nodes left in Site3. Therefore, 13, 14, 15, and 16 are distributed among Site1 and Site2. At that point, since there will be no free nodes left in Site2, last two jobs (17, and 18) are mapped to Site1.

Similar to Figure 4.3a, Figure 4.3b has a set of queued jobs to be scheduled and we have three sites available. However, none of the sites has free nodes. Site1, Site2, and Site3 have 128, 64, and 32 total number of nodes, and 10, 6, and 5 number of jobs in their queue, respectively. Since none of the sites has available nodes, fullnesses of sites are compared to map jobs in order to achieve load balancing. As mentioned before fullness is the ratio of number of jobs in the queue to total number of nodes in the site. Fullness of each site is calculated each time a job needs to be mapped and the site which has the lowest

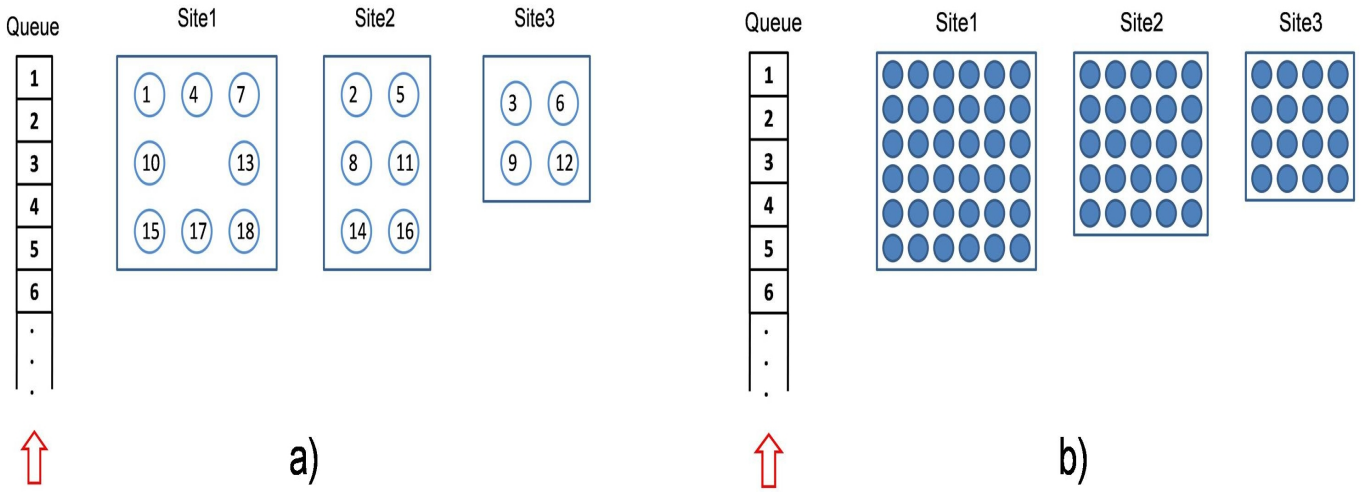


Figure 4.3: Example of Using Our First Site Selector (SS1) on Mapping Jobs among Three Different Sites a)Having Free Nodes, b)not Having any Free Node

fullness value is selected for job submission. In the Figure 4.3b 1,2,3,5,6,8,9,11, and 12 jobs are mapped to Site1, 4, 7, 10, and 13 jobs are mapped to Site2, and job 14 is assigned to Site3.

Similarly, in our second site selector (SS2) we use the same information retrieved from sites. Comparing to SS1, we give importance to the queued jobs even though there are available nodes in a site. So, instead of scheduling jobs based on round robin when there are free nodes, SS2 considers fullnesses of sites for each job scheduling. Fullness of a site in SS2 is also calculated slightly different than SS1 based on the formula : (# of nodes requested by queued Jobs - # of Free Nodes) / Total # of nodes. SS2 is simpler than SS1 and it works better than SS1 in the circumstances where there are heavy jobs in the queue and the number of free nodes is not sufficient for the execution of that job. We have also experienced that situation in our experiments in the following case study.

4.3 Case Study: UCoMS Workflow

4.3.1 UCoMS

UCoMS (Ubiquitous Computing and Monitoring System), which is an ongoing project with the collaboration of Louisiana universities, aims to discover and manage energy resources. UCoMS research can be divided as research in wireless and sensor network systems, usage of grid computing in large scale scientific applications, and monitoring systems. Results of the research in those areas should provide solutions

to effectively facilitate drilling and operational data logging and processing, on-platform information distribution and displaying, infrastructure monitoring/intrusion detection, and management of complex surface facilities and pipelines [37].

4.3.2 Implementation

We have developed an automated end-to-end system for UCoMS project by using different Grid tools. Our design goals and achievements can be categorized as:

- Automation
- Reliability
- Separation of computing and data tasks
- Running jobs on heterogeneous batch systems
- Increased performance
- Resource Independency
- Utilization of Resources to gain extra performance

Since the system is composed of different tasks, our first objective is to automate the tasks. This decreases the work done by scientists via getting rid of need to execute each job manually. Instead, everything is done by Pegasus system. In order to add reliability to our system, we designed our system in a way that failure of a task does not cause whole workflow to fail if the problem can be solved by retrying failed task. Data and computational tasks are separated since they have different needs and the level of reliability and performance can increase if they are handled by special tools. Therefore, Pegasus uses Condor, and remote job scheduler (e.g. Condor, PBS, Loadleveler, ...) for computational tasks and Stork for data transfers. Pegasus also manages to run jobs in heterogeneous batch systems via using Condor-G. Pegasus generates condor submission file with defining the universe as globus for each remote task and rest is handled by Condor. By using Pegasus, application has gained site independency. With this mechanism there is no need to specify resources for each simulation that is supposed to run on remote sites. Pegasus handles the site selection and

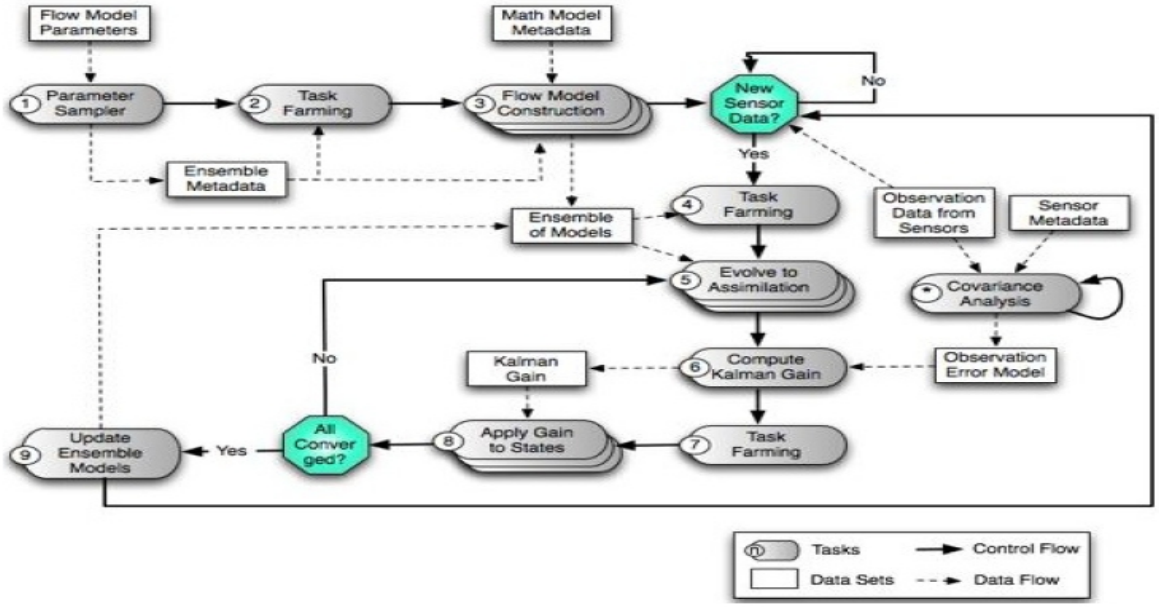


Figure 4.4: UCoMS Execution Flow [38]

generates proper condor submit files. In order to increase the performance by utilizing resources we have used our own site selectors which we have implemented for Pegasus. Via these site selection mechanisms, we aim to get high throughput by mapping large number of jobs to least loaded site and small number of jobs to the most loaded site.

Figure 4.4 shows the general UCoMS tasks' execution flow which is independent from Grid resources. Figure 4.5 shows the generated abstract workflow for Pegasus system. We used Pegasus in order to automate the workflow and make the application independent from sites.

The abstract workflow is generated by a java class for the ease of use. Users can create their abstract workflows by providing a name for the file as command line argument while executing this java program.

4.3.3 Results

We have executed UCoMS workflow on up to two LONI machines : Eric, and Poseidon; and one machine in LSU Distributed System Laboratory: dsl-condor. Eric and Poseidon are identical and each one has an interactive node and 128 compute nodes. Since dsl-condor is not a cluster and it is a single machine, it has one node for both interaction and computation.

We used Random and Round Robin site selectors in order to compare the results of our two site selectors.

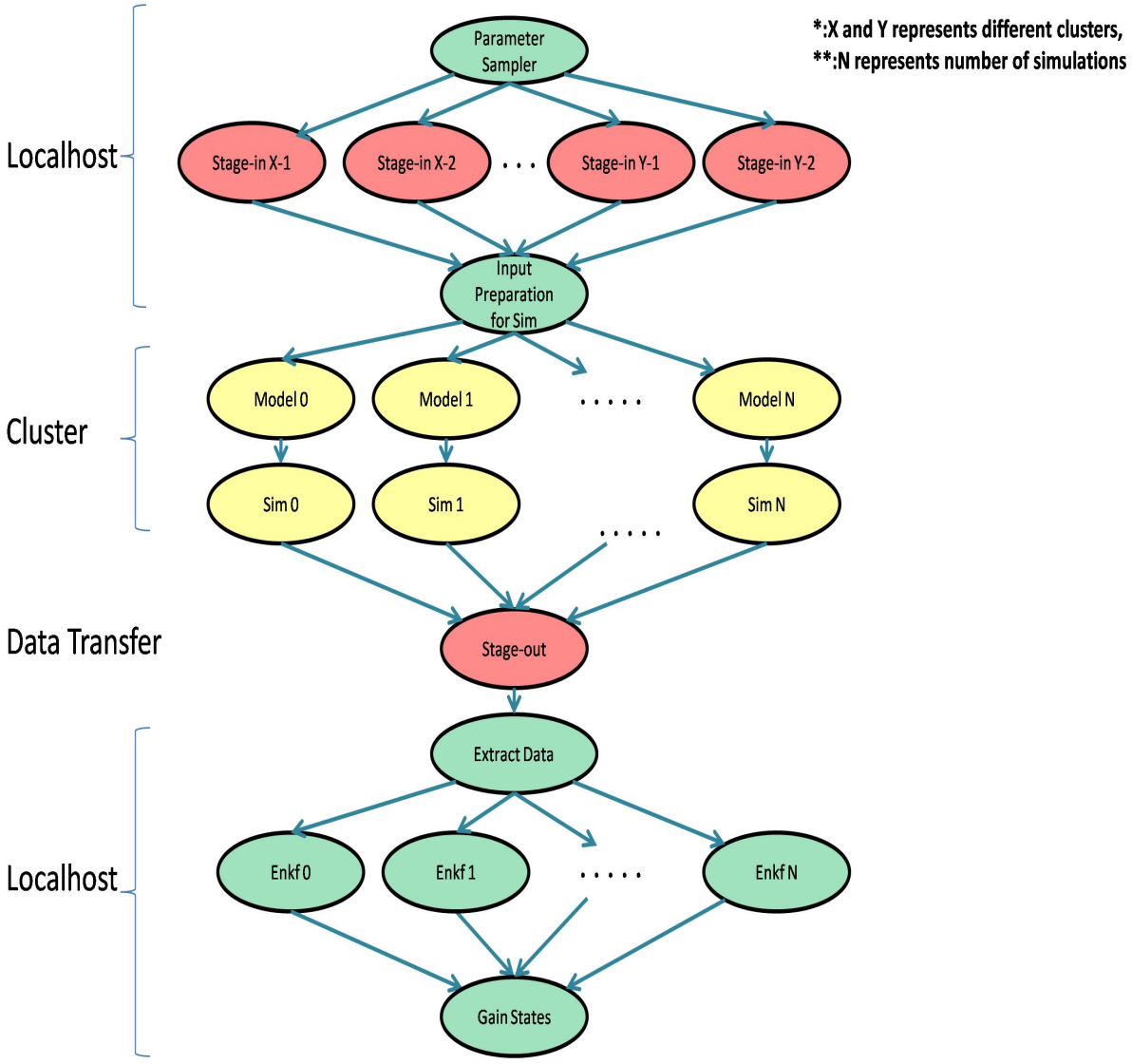


Figure 4.5: UCoMS Abstract Workflow for Pegasus System

Since in UCoMS workflow two dependent jobs in the modeling and simulation levels should be scheduled to same grid resource we have to use group site selector in Pegasus which uses random algorithm inside. In order to compare our results with round robin scheduling policy, we have implemented a round robin site selector that performs the grouping also.

Our experimental results show the time consumed with the different loads of sites for the different number of simulations performed by four different site selectors. As we have expected, overall round robin scheduling does a slightly better job than random site selector. We believe this difference arises since jobs are scheduled equally to sites in round robin which is one kind of load balancing without considering the loads among sites. However, as the simulation number increases the difference becomes negligible. The reason explained as: for the high number of simulations, the difference between the numbers of jobs assigned to each site become very small comparing to total number of simulations. Therefore, random site selector behaves like round robin site selector.

While SS1 beats the random and round robin site selectors, SS2 also gives better performance comparing to SS1 in overall. Performance difference between SS2 and SS1 comes from the implementation difference. They give very similar results in most of the cases except for the situations where there are free nodes in a site and there are queued jobs that request more nodes than the available nodes. In those situations SS1 may give even worse results than both random and round robin site selection mechanisms since SS1 do not consider queued jobs when there are free nodes available. Therefore, SS1 assigns many jobs to the site in which jobs may get very less number of resources. Table 4.1 demonstrates such a case where the performance difference is obvious between SS1 and SS2. There are free nodes available in site Poseidon but they are not enough to run the job that is waiting on the queue. While SS1 assigns more simulation to Poseidon, SS2 chooses Eric for the execution of most simulations. However, for the sites that use backfilling mechanisms, we believe SS1 will give the best results among others.

Besides, based on our experiments we have found that the value of joblimit, which is defined by site scheduling policy, can be the critical factor in site selection algorithms if jobs require small number of resources such that they cannot fill the all available nodes before joblimit is reached. In those situations, our both site selectors give worse results than random and round robin site selectors. The reason is joblimit becomes the key factor in the scheduling since SSs give much importance to the number of free nodes and

Table 4.1: There Exist Jobs in the Queue of Poseidon and Available Nodes at the Same Time

Site Selector	Simulation Count	Site Fullness				Time
		Eric		Poseidon		
		F	Q	F	Q	
Round Robin	80	16	0	47	48	40:21
SS1	80	16	0	47	48	39:42
SS2	80	16	0	47	48	36:50
Random	80	16	0	47	48	40:49

the difference between free nodes and queued jobs more than job limit. This situation occurred in most of our results since our simulations require one node and we have high number of simulations. Therefore, joblimit is reached before all of our simulations are scheduled and before sites have run out of free nodes. Table 4.2 illustrates such a scenario. Although loads are different in sites, joblimit becomes more important than the number of free nodes since simulations require very less amount of resources. As can be seen, random and round robin algorithms give better results than SS1 and SS2 algorithms.

Table 4.2: Different Loads among Sites where Joblimit Becomes Critical Factor

Site Selector	Simulation Count	Site Fullness				Time
		Eric		Poseidon		
		F	Q	F	Q	
Round Robin	80	21	0	32	0	35:18
SS1	80	21	0	45	0	38:43
SS2	80	21	0	45	0	40:00
Random	80	21	0	45	0	35:58

On the other hand, for the situations where joblimit is not critical factor such as some sites have less number of available nodes that can be filled with the jobs whose number is smaller than joblimit. In this situation and similar ones, SSs give the best results especially if load in sites differ considerably. Table 4.3 illustrates such a case where there is not waiting job in the queue but the load differs because of the difference between number of free nodes in each queue. Joblimit does not become the limiting factor since one of the sites have less nodes than the value of joblimit.

In addition, we have seen that the results collected from small number of simulations are not very dependable. Table 4.4 shows such a scenario. SS2 is expected to perform a better job than SS1. However, results are very close and SS1 gives slightly better performance than SS2. Therefore, simulation numbers

Table 4.3: Different Loads in Sites where Joblimit does not Become Bottleneck

Site Selector	Simulation Count	Site Fullness				Time
		Eric		Poseidon		
		F	Q	F	Q	
Round Robin	40	12	0	3	0	40:52
SS1	40	12	0	3	0	34:24
SS2	40	12	0	3	0	33:41
Random	40	12	0	3	0	38:44

should be increased to get more reliable results.

Table 4.4: Results with Small Number of Simulations

Site Selector	Simulation Count	Site Fullness				Time
		Eric		Poseidon		
		F	Q	F	Q	
SS1	20	60	0	43	48	20:33
SS2	20	60	0	43	48	21:07
Random	20	60	0	43	48	21:20

As a result, we state that the performance difference between both SS1 and SS2 algorithms purely related on the scheduling policies of sites.

Even though SS2 and SS1 do not give results as we have expected, they perform better than both random and round robin site selectors in overall. Results are expected to better if simulations require more grid resources. In those cases joblimit will not be the key factor and our site selectors are supposed to perform better.

Chapter 5

Related Work

5.1 Surveys in Workflow Management Systems

One of the most popular publications on the subject of grid workflow engines is brought out by Jia Yu et al [39]. This research consists of classification of workflow management systems based on five major criteria such as: workflow design, scheduling, fault tolerance, information retrieval, and data movement. There are four key factors in workflow design: structure of the workflow (DAG or non-DAG), workflow modeling (abstract or concrete), workflow composition and quality of service. Scheduling of workflows is examined from the view of scheduling architecture, decision making, planning scheme, scheduling strategy, and performance estimation. Fault management in workflow management systems is divided into two categories: task-level fault recovery and workflow-level fault management. Finally, transfer of data can be performed by user directly, or can be automated.

In addition to taxonomy of workflow management systems, most widely used workflow management systems are observed according to taxonomy criteria and classified into different categories.

Geoffrey Fox and Dennis Gannon summarize the discussion at the Global Grid Forum GGF10 Workflow workshop in [40]. They have characterized the applications based on the applications described in the workshop. In addition, different techniques in the design of workflows are discussed. Several workflow management systems such as Triana, Kepler, Taverna, Grid-Flow, GRMS, SkyFlow, Pegasus, and DAGMan are examined in terms of their structure, design, success of handling the complexity of workflows, and their ease of use. High importance is given to the issues in workflow enactment which includes efficiency, robustness and monitoring of the workflow. Some other interesting issues that are also studied are: Security, using workflow document as part of the scientific provenance of a computational experiment, the way of binding data sources to workflow patterns and templates.

SHI Meilin et al [41] have performed a survey in the workflow management area explaining the current WFMS research, pointing some workflow-related concepts and its typologies. Basic concepts and WFMS related concepts such as: workflow, activities of workflows, process in the workflow and different models

are clarified. This research does not only studies the grid WFMs but also discusses WFMs in business and in many different areas.

Based on the survey WFMS can be categorized into four different typologies: a) structured or ad-hoc, b) document-centric or process-centric, c) email-based or database-based, d) task-pushed or goal-pushed.

WFMS should have main components such as: a) process definition tool, b) workflow enactment service, c) client application, d) invoked applications, e) administration and monitoring tools.

In addition existing workflow managements are compared based on the following criteria: a) flexibility, b) object-oriented structure, c) intelligence, d) support for synchronous cooperation, e)support for mobile users, and they are categorized as: a) web based WFMs, b) distributed WFMs, c) transactional WFMs, d) interconnecting heterogeneous WFMs.

5.2 Similar End-to-End Processing Systems

Emrah Ceyhan et al [38] has designed a grid-enabled workflow system for reservoir uncertainty analysis. Reservoir analysis is part of the UCoMS project. The main aim of their project was to automate all steps in the analysis such as staging input data, distribution of simulations to grid resources, staging data out, post-processing the received data, and monitoring the whole application visually. For this project Condor-G is used as the batch scheduler, DAGMan as workflow manager, and Stork as the data transfer tool. Round robin algorithm is used as the scheduling policy.

Tevfik Kosar et al [42] has designed and implemented a system for transferring data reliably, automatically and processing of the data in large scale astronomy applications. Via this system data can be transferred to grid resources where processing is performed, and transferred back. System is responsible for recovering any failure that can be encountered in any step of the application. Besides, all steps are accomplished without human interaction. Data staging part of the workflow is given high importance. Therefore, system is designed such that data transfer and computation are designed and scheduled differently. As a result, data-movement and processing failures are separated and data-movements are optimized.

5.3 Other Site Selection Mechanisms

One project [43] developed in Harbin Institute of Technology in China, for multisite resource selection and scheduling. Their main concern in this project is parallelization of synchronous iterative applications in order to reduce the completion time of the job instead of maximizing system utilization. Their site selection algorithm (CGRS) is based on a density-based grid resource-clustering algorithm.

CGRS algorithm groups the grid resources based on the network delays in order to lower the network delays within each cluster comparing to network delays between sub-clusters. Next, resources in each cluster are arranged in the order of computational capacity. Then, possible schedules are produced and evaluated for all resources of the cluster until the best final schedule is selected. A single resource selection algorithm is also used in case of failure of multisite resource selection algorithm as a rescuer.

Byoung-Dai Lee et al [44] implemented an adaptive resource selection system for Grid-Enabled Network Services. In the system proposed, there is a network service front-end and it includes global scheduler. Global scheduler is responsible for scheduling jobs to different sites based on the information retrieved from those sites periodically. While scheduling, each grid resource is expected to have a local scheduler and every local scheduler assumed to be running shortest-remaining-time resource harvesting method.

In this research two adaptive site selection policies are introduced: Weighted Queue Length Based Heuristic (WQL), and Multi-level Queue Based Heuristic. WQL is designed based on the assumption of site with a shorter queue will typically finish a service request earlier. WQL evaluates the load of sites by considering total number of jobs in the queue, total number of jobs assigned to that site, and a weight value which is inverse proportional to the speed with each site can complete sample requests. Every job is scheduled to the site that has the smallest load value. Since grid environments have very dynamic behaviors, even though shortest-remaining-time scheduling algorithm is used in every site, long-running jobs can get higher priorities than smaller jobs after a threshold waiting time exceeded. Considering these circumstances MLQ is introduced. MLQ aims to let higher priority jobs to be suffered less from the dynamic changes of priorities by grouping requests with similar characteristics together and forwarding them to the same site. In addition, faster service time for low priority requests is expected by assigning lower priority jobs to faster sites. Global scheduler keeps predicted lower bound and upper bound values of run-times for each site.

These values form the range of run-time for each site. Based on these ranges jobs are assigned to the sites whose range includes the predicted run-time requests of the jobs.

An opportunistic algorithm for site selection in grid environments is developed by Luiz Meyer et al [45]. The system varies comparing to many other systems in the sense that it does not need to query grid resources to collect information to provide load balancing. The system uses the VDS [35] architecture with additional components and small extensions in VDS. Planner of VDS uses newly implemented site selector to map jobs to grid resources. The selected site that is returned by the site selection algorithm is passed to DAGMan for scheduling the job into the grid.

One implemented module additional to the VDS architecture is control database which is responsible for logging the records of jobs that are scheduled. Each record is formed by the job identification, status of the job, and selected site identification. This database is used to predict the performance of each site by checking the number of jobs completed for each site. Site selector uses the ratio of (number of ended jobs / number of submitted jobs) for each site to assign the subsequent jobs.

One other additional component is for monitoring the submitted jobs which is called MonitQueue. This component aims to remove jobs which are not presenting a desired performance from the Condor queue. The removed jobs are re-planned and rescheduled.

Chapter 6

Conclusion & Future Work

As the demand for grid environment increases because of the complexity of scientific applications, many grid tools are introduced to grid community. Workflow management system is one such tool which is the vital part for execution of large-scale scientific application in distributed computational resources. We have compared most widely used workflow management systems in terms of their conditional behaviors. The answer for the question of: ”Which system has the highest support for conditional structures?” is not clear since selection of a WFMS depends on application needs. However, based on our survey and implementations, we have reached the following results from which scientist may take advantage while they are choosing workflow management system for their applications:

- Each WFMS has different level of support for conditional structures. While some systems have primitive logic elements that can be used to build advance conditional structures, others already have those high level conditional structures. In addition, although some systems do not have conditional elements, some other mechanisms can be used for implementing conditional behavior.

- Ease of installation and usage of each workflow management system varies.

- Instead of returning an error code in case of failures, some structures fail and cause whole workflow to fail in some systems. This complicates building dynamic workflows.

- Some systems let users to implement their own elements. Via using this capability application specific conditional structures can be implemented.

- We have also observed that the systems, which have graphical user interfaces, may generate longer codes comparing to systems where users are required to implement their workflows manually.

Based on our observations and application needs we have chosen Condor for automation of DNA folding application since it is one of the easiest and reliable tools. We have composed the workflow application using separate scripts and parallelized simulations in order to gain performance. In addition, we have increased the

fault tolerance by using Stork for data transfer and retry mechanism of Condor. For UCoMS project we have used Pegasus for automation of workflow. By using Pegasus we have benefited from Condor capabilities and some unique features of Pegasus such as ease of creating abstract workflow, and site independency mechanism.

Our main contribution is implementing new site selectors for Pegasus system. By using these site selectors less number of jobs are assigned to heavy-loaded resources and more jobs are assigned to the grid sites that have more available resources. We have used this new site selection mechanism in UCoMS project. Our results show that there is considerable performance gain by using our site selectors comparing to random and round robin site selectors which are already provided via Pegasus.

Our newly implemented site selectors have one deficiency that they map all jobs in the planning time before the execution. Although first set of jobs in the workflow scheduled correctly, following jobs may suffer from the change of the load in sites. Scheduling jobs to sites very close to the running time is expected to eliminate this problem. For this purpose we will map the jobs level by level in the dag for the next step. Therefore, mapping one level of a dag will follow the execution of the previous level.

In addition, to increase the intelligence level of load balancing we need to dig little bit more into the site selection policies that are used by grid organizations.

We are also planning to use Pegasus for the DNA folding applications as well.

Bibliography

[1] Apache Ant accessed December 2006 [Online]. Available: http://ant.apache.org/

[2] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, Podlipnig, J. Qin, M. Siddiqui, H. Truong, A. Villazon, and M. Wieczorek, ASKALON: A Grid Application Development and Computing Environment, *6th International Workshop on Grid Computing*, Seattle, USA, IEEE Computer Society Press, November 2005

[3] P. Couvares, T. Kosar, A. Roy, J. Weber, and K. Wenger, Workflow Management in Condor, In Workflows for e-Science, Editors: I.Taylor, E.Deelman, D.Gannon, M.Shields, Springer Press, January 2007 (ISBN: 1-84628-519-4)

[4] K. Cooper, A. Dasgupata, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, F.Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Olugbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z. Shi, A. YarKhan, J. Dongarra. New Grid Scheduling and Rescheduling Methods in the GrADS Project, *NSF Next Generation Software Workshop,* International Parallel and Distributed Processing Symposium, Santa Fe, IEEE CS Press, Los Alamitos, CA, USA, April 2004.

[5] R. Buyya and S. Venugopal. The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report. In *1st IEEE International Workshop on Grid Economics and Business Models*, GECON 2004, Seoul, Korea, IEEE CS Press, Los Alamitos, CA, USA, April 23, 2004; 19-36.

[6] A. Mayer, S. McGough, N. Furmento, W. Lee, S. Newhouse, and J. Darlington. ICENI Dataflow and Workflow: Composition and Scheduling in Space and Time. In *UK e-Science All Hands Meeting*, Nottingham, UK, IOP Publishing Ltd, Bristol, UK, September 2003; 627-634.

[7] J. Yu, and R. Buyya, A Taxonomy of Workflow Management Systems for Grid Computing. Technical Report GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, 2005. http://www.gridbus.org/reports/GridWorkflowTaxonomy.pdf.

[8] Kepler Project accessed December 2006 [Online]. Available: http://www.kepler-project.org/

[9] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. H. Su, K. Vahi, M. Livny. Pegasus: Mapping Scientific Workflow onto the Grid. *Across Grids Conference 2004*, Nicosia, Cyprus, 2004.

[10] About myGrid accessed December 2006 [Online]. Available: http://www.mygrid.org.uk/?&MMN_position=1:1

[11] T. Oinn, M. Greenwood, M. Addis, M.N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M.R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe, Taverna: Lessons in Creating a Workflow Environment for the Life Sciences, Concurrency and Computation: Practice & Experience, Volume 18, Issue 10 (August 2006) Workflow in Grid Systems Pages: 1067 - 1100, 2006, ISSN:1532-0626, John Wiley and Sons Ltd., Chichester, UK

[12] I. Taylor, S. Majithia, M. Shields, and I. Wang, Triana WorkFlow Specification, GridLab Specification available at : www.gridlab.org/WorkPackages/wp-3/D3.3.pdf

[13] D. Erwin, et al., UNICORE Plus Final Report - Uniform Interface to Computing Resources, The UNICORE Forum e.V., ISBN 3-00-011592-7, 2003.
Online:http://www.unicore.org/documents/UNICOREPlus-Final-Report.pdf.

[14] T. Fahringer, J. Qin, and S. Hainzer, Specification of Grid Workflow Applications with AGWL: An Abstract Grid Workflow Language. *IEEE International Symposium on Cluster Computing and the Grid 2005 (CCGrid 2005)*, Cardiff , UK, May 9-12 2005. IEEE Computer Society Press.

[15] Cog Kit GridAnt Project Page accessed June 2008 [Online]. Available:
http://www.globus.org/cog/projects/gridant/

[16] Grid Workflow: ICENI, accessed June 2008 [Online]. Available:
http://www.gridworkflow.org/snips/gridworkflow/space/ICENI

[17] LESC - London e-Science Centre ICENI, accessed December 2006 [Online]. Available:
http://www.lesc.imperial.ac.uk/iceni/

[18] YAWL: Yet Another Workflow Language, accessed August 2007 [Online]. Available:
http://www.yawl-system.com/

[19] S. S. Bhattacharyya, C. Brooks, E. Cheong, J. Davis, II, M. Goel, B. Kienhuis, E. A. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuendorffer, J. Reekie, N. Smyth, J. Tsay, B. Vogel, W. Williams, Y. Xiong, Y. Zhao, H. Zheng, Ptolemy II Heterogeneous Concurrent Modeling and Design In Java-Volume 1: Introduction to Ptolemy II. Memorandum UCB/ERL M05/21 EECS UC Berkeley, CA 94720, July 15, 2005.

[20] About myGrid, accessed December 2006 [Online]. Available:
http://www.mygrid.org.uk/?&MMN_position=1:1

[21] Taverna 1.5.2 Manual, accessed August 2007 [Online]. Available:
http://www.mygrid.org.uk/usermanual1.5/index.html

[22] Ant-contrib Tasks, accessed June 2008 [Online]. Available:
http://ant-contrib.sourceforge.net/tasks/tasks/index.html

[23] Stork Project accessed June 2008 [Online]. Available: http://www.storkproject.org/

[24] AMBER - Wikipedia, the free encyclopedia, accessed June 2008 [Online]. Available:
http://en.wikipedia.org/wiki/AMBER

[25] Xiang-Jun Lu and Wilma K. Olson, 3DNA: a software package for the analysis, rebuilding and visualization of three-dimensional nucleic acid structures, *Nucleic Acids Research, 2003*, Vol. 31, No. 17 5108-5121

[26] Olson et al., J. Mol. Biol. 313(1), 229-237, 2001

[27] James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kale, Klaus Schulten, Scalable Molecular Dynamics with NAMD, *Journal of Computational Chemistry*, Volume 26, Issue 16, Date: December 2005, Pages: 1781-1802

[28] Mark Nelson, William Humphrey, Attila Gursoy, Andrew Dalke, Laxmikant Kale, Robert D. Skeel, Klaus Schulten, NAMD: a Parallel, Object-Oriented Molecular Dynamics Program, *International Journal of High Performance Computing Applications*, Vol. 10, No. 4, 251-268 (1996), DOI: 10.1177/109434209601000401

[29] William Humphrey, Andrew Dalke, and Klaus Schulten, VMD: Visual Molecular Dynamics, *Journal of Molecular Graphics*, Volume 14, Issue 1, February 1996, Pages 33-38, doi:10.1016/0263-7855(96)00018-5.

[30] Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger,Addison-Wesley, 1988. ISBN 0-201-07981-X. The AWK Programming Language

[31] tcsh - Wikipedia, the free encyclopedia, accessed June 2008 [Online]. Available: http://en.wikipedia.org/wiki/Tcsh

[32] Thomas C. Bishop, Molecular Dynamics Simulations of a Nucleosome and Free DNA,*Journal of Biomolecular Structure and Dynamics*, Volume 22 No. 6 (p 615-878) June 2005 ISSN 0739-110

[33] Karolin Luger, Armin W. Mader, Robin K. Richmond, David F. Sargent and Timothy J. Richmond, Nature 389, 251-260 (18 September 1997), doi:10.1038/38444

[34] Yong Zhao, Mihael Hategan, Ben Clifford, Ian Foster, Gregor von Laszewski, Ioan Raicu, Ti beriu Stef-Praun, Mike Wilde Swift: Fast, Reliable, Loosely Coupled Parallel Computation, services, pp. 199-206, *2007 IEEE Congress on Services (Services 2007)*, 2007

[35] VDS - The GriPhyN Virtual Data System, accessed June 2008 [Online]. Available: http://www.ci.uchicago.edu/wiki/bin/view/VDS/VDSWeb/WebMain

[36] Ewa Deelman, Time and Space Optimizations for Executing Scientific Workflows in Distributed Environments, Invited talk at NeSC Workflow Optimization in Distributed Environments, (October 2006) [Online]. Available: http://pegasus.isi.edu/presentations/edinburgh.pdf

[37] The UCoMS Project Home Page. June, 2008, http://www.ucoms.org/

[38] Emrah Ceyhan, Gabrielle Allen, Christopher White, and Tevfik Kosar, A Grid-enabled Workflow System for Reservoir Uncertainty Analysis, *In Proceedings of CLADE'08 (in conjunction with HPDC'08)*, Boston, MA, June 2008

[39] Jia Yu and Rajkumar Buyya, A Taxonomy of Workflow Management Systems for Grid Computing, *Journal of Grid Computing*, Volume 3, Numbers 3-4 / September, 2005

[40] Geoffrey Fox and Dennis Gannon, Workflow in Grid Systems Editorial of special issue of Concurrency&Computation: Practice&Experience based on GGF10 Berlin meeting ; *Concurrency and Computation: Practice & Experience* Volume 18 , Issue 10 (August 2006) Pages: 1009 - 1019, 2006

[41] Shi Meilin, Yang Guangxin, Xiang Yong, Wu Shangguang, Workflow Management Systems: A Survey, Communication Technology Proceedings, 1998. ICCT '98. *1998 International Conference*, vol.2, On page(s): 6 pp., 22-24 Oct 1998

[42] Tevfik Kosar, George Kola, Robert J. Brunner, Miron Livny, and Michael Remijan Reliable, Automatic Transfer and Processing of Large Scale Astronomy Datasets. *In Proceedings of 14th Astronomical Data Analysis Software & Systems Conference (ADASS 2004)*, Pasadena, CA.

[43] Weizhe Zhang, Binxing Fang, Hui He, Hongli Zhang, Mingzeng Hu Multisite Resource Selection and Scheduling Algorithm on Computational Grid, *Parallel and Distributed Processing Symposium, 2004*. Proceedings. 18th International.

[44] Byoung-Dai Lee, Jon B. Weissman, Adaptive resource selection for grid-enabled network services, Network Computing and Applications, 2003. NCA 2003. *Second IEEE International Symposium*, On page(s): 75- 82, ISBN: 0-7695-1938-5.

[45] Luiz Meyer, Doug Scheftner, Jens Vockler, Marta Mattoso, Mike Wilde, and Ian Foster, An Opportunistic Algorithm for Scheduling Workflows on Grids, *VECPAR'06*, Rio De Janiero, 2006.

Vita

Emir Mahmut Bahsi was born in October 1984, in Bursa, Turkey. He has received his bachelor degree in computer science at Fatih University in Istanbul, Turkey, in June 2006. In his undergraduate studies he has implemented a web-based apartment management system and this work became his thesis and entitled as ”Apartment Building Management Web Application.” He joined Louisiana State University to pursue master’s degree in August 2006. At Louisiana State University he has studied and performed research in the area of high performance computing and grid computing. He has focused on grid workflow management systems and collected his work in his thesis as: ”Dynamic Workflow Management for Large Scale Scientific Applications.”