

This file is part of the following work:

Sim, Nigel Graham Donald (2012) *Distributed processing for statistical data mining*. PhD Thesis, James Cook University.

Access to this file is available from:

<https://doi.org/10.25903/esyp%2Dgd93>

Copyright © 2012 Nigel Graham Donald Sim

The author has certified to JCU that they have made a reasonable effort to gain permission and acknowledge the owners of any third party copyright material included in this document. If you believe that this is not the case, please email

researchonline@jcu.edu.au

Distributed Processing for Statistical Data Mining

Thesis submitted by
Nigel Graham Donald Sim BE (Hons)/BSc

For the degree of
Doctor of Philosophy
In
Information Technology
at
James Cook University, Townsville

STATEMENT OF ACCESS

I, the undersigned author of this work, understand that James Cook University will make this thesis available for use within the University Library and, via the Australian Digital Theses network, for use elsewhere.

I understand that, as an unpublished work, a thesis has significant protection under the Copyright Act and;

I do not wish to place any further restriction on access to this work

15 March 2012

Signature

Date

STATEMENT ON SOURCES

DECLARATION

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

15 March 2012

Signature

Date

ELECTRONIC COPY

I the under-signed, the author of this work, declare that the electronic copy of this thesis provided by James Cook University Library is an accurate copy of the print thesis submitted, within the limits of the technology available.

15 March 2012

Signature

Date

STATEMENT ON CONTRIBUTIONS OF OTHERS

The research described and presented in this thesis was undertaken by the author under the supervision of Professor Ian Atkinson and Professor Danny Coomans, both of whom provided editorial and academic advice.

Financial support in the form of a scholarship was gratefully received from the School of Maths and Physics, and the School of Information Technology. Further support was received from the Graduate Research School in the form of research grants, and from the Australian Mathematical Sciences Institute, Australian Partnership for Advanced Computing, and the International Society for Business and Industrial Statistics in the form of conference travel grants.

I would also like to thank Dr Dmitry Konovalov, Dr Yvette Everingham and Dr Timothy Hancock for their discussions and expertise in the fields of data mining and statistics.

ACKNOWLEDGEMENTS

Firstly I would like to thank my parents, Helen and Robin, and brother Russell for their continued support and encouragement through the ups and downs encountered on the path to completing this project.

Additionally, I owe a great debt of gratitude to my supervisors Professor Ian Atkinson and Professor Danny Coomans for their guidance, inspirational advice, and ability to help me see the bigger picture when doubts abounded. I would like to also thank Gavin and Carol Blackman for their personal encouragement and advice in constructing scientific argument and prose.

Finally I would like to thank my friends and colleagues who have contributed to this thesis through their encouragement, discussion, friendship and feedback, with special thanks to: David Laing, David Browning, Adrian Knack and Angela Hughes.

ABSTRACT

The use of information technology (IT) in scientific investigations is now commonplace, due largely to the increased efficiency of IT procedures in managing and organising data sets now able to be generated through technologically aided data recording methods. While such data sets can be advantageous to investigative work, the size and complexity of these pose special challenges to exploring and revealing their information content.

Data mining procedures offer many general purpose tools that can be used to explore large volumes of data to find patterns and structures within the data sets that are able to relate response variables to observations. However data mining techniques need to be matched to the attributes of the input data sets. In general, data sets with larger numbers of input variables require robust and sophisticated techniques to reliably identify patterns and processes within the data structures. Additionally, more sophisticated data mining techniques take more computational time to execute than simple data mining techniques. However, some computational problems typical of data mining are amenable to being easily divided into discrete tasks able to be executed independently, in parallel across many computational resources.

The project reported in this thesis generated the components of an eResearch framework. A workflow language was developed to capture the critical aspects of a data mining process, allowing the parallel components to be exploited. Subsequent development of a distributed computing framework enabled leverage of existing data mining tools such as MATLAB and R to perform actual data processing. This distributed computing framework controls movement of data and execution of tasks based on the workflow submitted by the practitioner initiating the workflow.

The coordinating element within the distributed computing framework is a new task scheduling algorithm, termed “Neglected”. This algorithm is the major research contribution of this project. “Neglected” is a task matching algorithm that optimises total execution time of an experiment by minimising the unnecessary movement of data. This is achieved by matching resources to tasks, where a task's estimated completion time is within a margin of that task's best matching option.

The “Neglected” task scheduling algorithm was tested in simulation against a commonly used distributed computing scheduling algorithm, the “MinMin” greedy scheduler. The new algorithm significantly outperformed “MinMin” in terms of data transfer, and in most scenarios it also outperformed in terms of total compute time. This is attributed to the reduced transfer overhead required to satisfy the tasks assigned to each resource.

The “Neglected” scheduling algorithm offers improved efficiency in the use of resources and improved time to solution for workflow execution. This, together with the data mining workflow and execution framework, extend and improve overall efficiency, robustness and repeatability in the analysis of new and existing data sets by computationally intensive data mining techniques.

Table of Contents

1. Introduction.....	1
1.1. eScience and the Grid.....	3
1.2. Data mining.....	5
1.3. Scheduling Distributed Computing Execution.....	7
1.4. Project Summary.....	8
1.5. Contributions.....	8
1.6. Thesis Organisation.....	8
2. Background and Overview.....	10
2.1. Robust Data Mining of Large Data sets.....	11
2.1.1. <i>Cheminformatics – Data mining in Chemistry</i>	12
2.1.2. <i>Data Mining as eScience</i>	13
2.1.3. <i>Efficiently Speeding Up Execution</i>	16
2.1.4. <i>Review of Contemporary Data Mining Techniques</i>	19
2.1.5. <i>Typical Data Mining Experiments</i>	24
2.1.6. <i>Properties of an eScience Data Mining System</i>	27
2.2. Experiment Expression and Management Review.....	28
2.2.1. <i>Expression of a Data Mining Experiment</i>	29
2.2.2. <i>Experiment Management – Collection of Experiment Metadata</i>	32
2.3. Deployment and Integration Review.....	34
2.3.1. <i>Execution of Operations and Applications</i>	35
2.3.2. <i>Data Interchange and Conversion</i>	37
2.3.3. <i>Communications and Data Security</i>	41
2.3.4. <i>Recoverability</i>	42
2.4. Computational Platform Review.....	43
2.4.1. <i>Survey of Distributed Computing Topologies and Services</i>	44
2.4.2. <i>Master-Slave Execution Model</i>	50
2.4.3. <i>Scheduling Notation</i>	52
2.4.4. <i>Task Scheduling in Distributed Computing</i>	55
2.5. Previous Projects.....	58
2.5.1. <i>Data Mining Services Middleware</i>	59
2.5.2. <i>GridMiner</i>	60
2.5.3. <i>Nimrod</i>	60
2.5.4. <i>Kepler</i>	61
2.5.5. <i>Taverna</i>	62
2.5.6. <i>RapidMiner</i>	63
2.5.7. <i>WEKA</i>	63
2.5.8. <i>R Language</i>	64
2.5.9. <i>Summary of Previous Projects</i>	65
2.6. Summary.....	67
3. Distributed workflow, and eScience tool chain.....	68
3.1. Review of Data Mining Experiment Requirements.....	69
3.1.1. <i>General Data Mining Experiment Use Case</i>	70
3.2. Experiment Representation.....	71
3.2.1. <i>Workflow</i>	72
3.2.1.1. <i>Example workflow</i>	77
3.2.2. <i>Operators</i>	78
3.2.3. <i>Data</i>	80

3.3. Agent Based Execution System.....	82
3.3.1. <i>Operator Invocation</i>	83
3.3.2. <i>Distributed Execution</i>	85
3.3.2.1. Agent Communications.....	86
3.3.2.2. Agent Communications Protocol.....	87
3.4. Data Services to Support Distributed Execution.....	89
3.4.1. <i>Agent Data Access Component</i>	91
3.4.2. <i>Data Formats</i>	92
3.5. Experiment Metadata and Data Provenance.....	93
3.5.1. <i>Process provenance</i>	94
3.5.2. <i>Data Provenance</i>	97
3.5.3. <i>Provenance Storage</i>	98
3.6. Transforming Syntax Trees into Tasks.....	99
3.7. Conclusion.....	101
4. Development and Analysis of a New Task Scheduling Algorithm.....	103
4.1. Survey of Master-Slave Scheduling Heuristics.....	105
4.2. Scheduling of Heterogeneous Tasks in a Heterogeneous Environment.....	106
4.2.1. <i>Scheduling for Data Distribution</i>	113
4.3. Validation of systems via simulation frameworks.....	116
4.3.1. <i>Calibration of the simulation</i>	117
4.3.2. <i>Simulation of Allocation Algorithm</i>	119
4.4. Experimental Results.....	123
4.6. Utilisation of Resource Parallelism.....	133
4.7. Summary.....	134
5. Predictive modelling for human intestinal absorption of chemical compounds	135
5.1. Introduction.....	136
5.2. Materials and methods.....	137
5.2.1. <i>Experiment</i>	141
5.3. Results and discussions.....	142
5.4. Workflow and provenance.....	146
5.5. Conclusion.....	148
6. Conclusions and Future Work.....	149
6.1. eScience: Workflow and Provenance.....	149
6.2. Distributed Computing.....	150
6.3. Benefits to Data Mining Applications.....	151
6.4. Outcomes and Contributions.....	152
6.5. Future Work.....	153
6.5.1. <i>Resource Scheduling</i>	153
6.5.2. <i>Applications</i>	154
6.6. Concluding remarks.....	156
Nomenclature.....	157
References.....	159
Appendix A - Optimising Ensemble Predictions.....	175
A.1. Methods and materials.....	176
A.1.1. <i>Heterogeneous ensemble</i>	176
A.1.2. <i>Lasso post-processing</i>	177
A.1.3. <i>Evolutionary strategies post-processing</i>	177
A.1.4. <i>Blood Brain Barrier (BBB) data set</i>	179
A.1.5. <i>Friedman1000</i>	179
A.2. Results and discussion.....	179

A.3. Conclusion.....	181
Appendix B - Important Grid Agent Interfaces.....	182
B.1. IAgent.java.....	182
B.2. ITupleStore.java.....	183
B.3. IFileMovement.java.....	184
B.4. IPeerStatus.java.....	185
Appendix C - Workflow XML Schema.....	187

List of Tables

Table 2.1: Summary of cross-validation technique procedures, relating number of iterations required to n =observations and k =testsets.....	22
Table 2.2: Different classes of distributed resource, and some attributes of these.....	47
Table 2.3: Summary of notation used to describe task scheduling.....	53
Table 2.4: Summary of previous projects covering data mining.....	66
Table 3.1: Common provenance attributes about compute resources.....	95
Table 3.2: Common provenance attributes supplied by all operators.....	96
Table 3.3: Additional provenance information provided by the R operator.....	96
Table 4.1: GridSim network model comparison results.....	118
Table 4.2: Host configurations for simulation executions.....	121
Table 4.3: Normalised makespan for varying operator run times and repeat values.....	124
Table 4.4: Experiment data transfer saturation for the four scheduling algorithms, across the parameter space of operator size, task size and file count.....	126
Table 4.5: Final data placements for scheduling algorithms vs iterations (compute dominated).....	128
Table 4.6: Final data placements for scheduling algorithms vs iterations (data dominated).....	130
Table 5.1: Previous performance results for predictive models based in the HIA data set....	138
Table 5.2: Top 6 most frequently selected variables by data set using GA-PLS as reported by Hou.....	142
Table 5.3: Top 6 variables selected using random forests variable importance measure.....	143
Table 5.4: Top 6 variables selected using GA-PLS across the MCCV variable selection.....	144
Table 5.5: Mean predictive performance of HIA data set descriptors using PLS, SVM and RF predictive models using variables selected by Random Forest.....	145
Table 5.6: Mean predictive performance of HIA data set descriptors using PLS, SVM and RF predictive models using variables selected by GA-PLS.....	145
Table 5.7: Amount of time (serial) spent performing each step of the workflow.....	147
Table 1: R2 results of the ensemble post-processing techniques on the blood-brain barrier and Friedman1000 data sets.....	180

List of Figures

Figure 1.1: High level flow chart of the data mining process, and the outputs produced by it.	5
Figure 2.1: Process of distilling various sources of chemical observations into a table of data, from which a model can be built.....	13
Figure 2.2: An experiment is the application of a process on input data and algorithms, to produce output data and an experiment log.....	15
Figure 2.3: Typical data mining process considered in this thesis.....	16
Figure 2.4: Amdahl's law showing the speedup obtained by adding additional resources.....	18
Figure 2.5: Generic ensemble of independent weighted models.....	23
Figure 2.6: Bagging: Ensemble of independent weighted models built on subsets of the data.	23
Figure 2.7: Flow chart of cross-validation as applied in data mining.....	25
Figure 2.8: Abstract representation of the list of steps for the generalised process that a) data mining practitioners and b) algorithm developers, may use in their work.....	26
Figure 2.9: A typical data mining experiment template exhibiting iteration, nesting of iteration and sequential application of operators.....	27
Figure 2.10: A computational experiment is comprised of algorithms, data management and experiment management. Workflows provide the experiment management functionality, allowing the application to be more focused and general purpose.....	29
Figure 2.11: Adaption of disparate data sources to provide a consistent interface for the data mining workflow.....	38
Figure 2.12: Staging data via disk is a simple way of bridging legacy applications to the Grid.	40
Figure 2.13: The use of a shim can redirect file access in a way that is transparent to the Application.....	41
Figure 2.14: Compute resources potentially available to practitioners within an institution. Includes networks of workstations, Grids and clusters, and commercially provided Cloud resources.....	45
Figure 2.15: Grid Resource Broker provides a service to locate and acquire resources, based on market system.....	48
Figure 2.16: Example Data Grid scenario: Multiple resources located next to computational resources, and controlled by a replica catalogue. The User can request data be replicated between storage resources to provide local access to data by compute resources.....	49
Figure 2.17: Master-slave paradigm: Slave compute resources connected via a network perform tasks requested by a master resource.....	51
Figure 2.18: Abstract algorithm for the master node in a master-slave system.....	52
Figure 2.19: Abstract algorithm for a slave node in a master-slave system.....	52
Figure 2.20: Communications between master and slaves during task execution.....	52
Figure 2.21: Impact of decreasing bandwidth (or increasing data size) on makespan.....	56
Figure 3.1: Experiment document describe data elements, operators and a workflow, which all map to their counterparts in the eScience system.....	69
Figure 3.2: Top level elements within the experiment document.....	72
Figure 3.3: Parameter sweep blocks and subsetting example: Lines 1-4 generate data and write it to a store including variables which indicate the index of the iterators. Lines 5-6 bring those same iterators into scope again, and line 8 uses subsetting to retrieve the data stored in line 4. Method or sub-setting parameter names are shown in italics.....	73
Figure 3.4: Illustration of variable scoping. a) Variable x is in scope on line 3. b) Variable x is out of scope on line 3.....	75

Figure 3.5: Example workflow that only has a single level of iteration.....	77
Figure 3.6: An example workflow which illustrates how exposing inner iterations improves the degree to which a workflow can be executed in parallel.....	78
Figure 3.7: Example of an operator declaration for an R script.....	78
Figure 3.8: An abstract view of the invocation of an operator. Operators are invoked by the workflow, with operators looked up by name from an operator list and input and output data mapped to their ports.....	80
Figure 3.9: Data source declaration for the Iris flower data set.....	81
Figure 3.10: Hierarchy of parallelism from CPU level, to LAN level, up to WAN level.....	82
Figure 3.11: Components which exist within an agent: agent controller, workflow component and data management component.....	83
Figure 3.12: Flow chart of events when invoking an operator, including data preparation and retrieval.....	84
Figure 3.13: Sequence diagram of executing a remote task, including the staging of data.....	86
Figure 3.14: Data management component provides a service for operator adapters to access local and remote data.....	92
Figure 3.15: Provenance information collected by the operator is stored using the metadata component.....	98
Figure 3.16: Flow chart of components that are required to schedule a workflow. The workflow is segmented into discrete tasks, and these tasks are scheduled for execution.....	100
Figure 4.1: Attributes that contribute to scheduling: data, tasks, resources and network link.....	104
Figure 4.2: In a master-slave system there are a number of specialised queues and scheduling systems that may impact on the performance of an execution.....	108
Figure 4.3: Process of data migrating from its initial location. First it is copied from the Practitioners Workstation to Resource 1, then from Resource 1 to Resource 2.....	110
Figure 4.4: Compute-first matching heuristic. Matches tasks to the resource which will complete the computational component the quickest.....	112
Figure 4.5: Greedy (MinMin) task mapping heuristic.....	112
Figure 4.6: Sufferage mapping heuristic.....	113
Figure 4.7: Neglected matching heuristic maps tasks to resource when a resource is near, but not strictly, optimal for the task.....	114
Figure 4.8: Configuration of compute resources connected via a network used in scheduling simulation.....	120
Figure 4.9: Workflow used in simulation.....	121
Figure 4.10: Component diagram of simulation experiment software.....	122
Figure 4.11: CPU Usage Efficiency with increasing CPU dominance (x1).....	131
Figure 4.12: CPU Usage Efficiency with increasing CPU dominance (x5).....	132
Figure 4.13: CPU Usage Efficiency with increasing CPU dominance (x10).....	132
Figure 4.14: CPU Usage Efficiency with increasing CPU dominance (x100).....	132
Figure 4.15: Execution times for varying numbers of parallel arrangements of 20 computational elements, running a data-intensive workflow.....	133
Figure 5.1: Frequency of response variables in Hou data set.....	137
Figure 5.2: Workflow for HIA investigation.....	146

Chapter One

1. Introduction

Information Technology (IT) tools have become ubiquitous in modern day science, and are commonly applied as fundamental components of data acquisition and analysis across a broad spectrum of scientific fields. Not surprisingly, IT has also played a seminal role in extending horizons of many scientific endeavours – notably the life and chemical sciences – by providing tools and techniques for exploring data sets that were previously intractable. This has been due primarily to advanced IT techniques that a) enable massive computational power to be applied practically to managing, organising and exploring very large, complex data sets, and also b) vastly increases the capacity of data collection, through devices such as high speed, high resolution, robotic analysis instrumentation, sensor networks, and high resolution telemetry.

Similarly large data sets can also be produced through simulation, and calculating or deriving new attributes from previously collected data. For example, in the field of cheminformatics [1], chemical structures are analysed using computer applications that compute chemical, electrical and physical properties of a molecule. data sets generated by the above mechanisms typically result in enormous numbers of records containing many attributes. This phenomenon is often referred to as the data deluge [2] and is also illustrated by the increasing availability of data sets via the internet, and superscience instruments like the high energy physics experiment – the Large Hadron Collider (LHC) – that produce petabytes of data per year.

While the characteristic size and complexity of data sets has in part re-defined the potential scope of investigative work, it is evident that increased size and availability of data sets poses special challenges to exploring and revealing their information content. Analysis of large data sets requires the use of specialised techniques, including data mining [3], in order to expose the patterns and structures that relate the response variable to the many

candidate predictor variables. Data mining is particularly important when the system under investigation is not thoroughly understood, and new leads are required to focus future research. However, data mining does not discern between valid and spurious correlations, and may be misled by large numbers of variables. Robust techniques may be applied to begin to overcome these issues, but this requires a large amount of computing time, and still requires a domain expert to validate and interpret the results of the data mining investigation.

The project presented in this thesis explores the application of the *eScience* [4], *Grid* [5][6][7] and *distributed computing* [8] paradigms to the fields of computational statistics and data mining. eScience focuses on the use of IT tools within the scientific process while the Grid is a paradigm concerned with the interoperability of distributed computing systems. Data mining and computational statistics have a natural affinity with the concepts of eScience. Particular care must be taken to ensure rigorous record keeping is observed when performing experiments, as *in silico* experimentation can often be performed so readily that this attention to detail may be overlooked. This has consequences for the repeatability of experiments, and for the recording of novel work that may be further explored at a later date.

This study project has produced a framework that allows practitioners to express their data mining process as a workflow – a clear, repeatable plan or sequence of steps for an experiment – which enables the use of Grid computing [9] and Data Grid [10] resources to provide the required computational and data services. To use these resources efficiently and effectively requires an appropriate task scheduling algorithm. Many existing Grid computing frameworks use a task scheduling algorithm based on the greedy MinMin heuristic [11][12], that aims to minimise total computation time by accepting locally optimal solutions at each step. This thesis presents a new scheduling algorithm, Neglected, which is shown to outperform the MinMin heuristic in most situations across a range of typical experiment scenarios.

The sections following in this chapter give a brief introduction to eScience, the Grid and data mining, in sufficient detail to appreciate this work, with a more thorough analysis in

1.1. eScience and the Grid

The term *eScience* has been used to describes scientific endeavours that are largely enabled through the use of IT. IT can be employed to provide data management, simulation, instrumentation and/or experiment workflow orchestration. The actual scope and implementation details of these services will vary between implementations, but the essential purpose of eScience is to facilitate discovery which would not be possible without IT enhancements.

The Grid is the name of a modern IT paradigm that endeavours to seamlessly integrate researchers, instrumentation, and computing and analysis services on a large scale. The Grid paradigm proposes a small number of standards be used to interface between users, services and other widely distributed components. The purpose of keeping the number of required interfaces low is to improve interoperability and facilitate integration. Distributed identity management for users and services provides authentication and authorisation and is used to tie the Grid components together. This is intended to maximise compatibility and re-usability of the services and components, allowing them to be reconfigured and orchestrated for use in almost arbitrary workflows. Many Grid services build on existing components, such as Grid Computing services, which utilise transitional compute clusters and high performance computers via the Grid mechanism. Similarly, Data Grid services are typically used to expose traditional file systems and databases via standardised Grid interfaces.

When discussing the composition of a Grid system, it is important to highlight the roles that humans play. In Unified Modelling Language (UML) [13] terminology they are referred to as actors. In this project several human actors will be discussed. These are:

1. The *algorithm developer* who designs or implements data processing software;
2. The *workflow developer* or workflow integrator who develops the workflows which pull together the data sets and the algorithms; and

3. The *domain practitioner*, or *practitioner*, who will be applying the system provided to analyse their data sets, and also refining the workflows to improve predictive performance.

To develop this discussion, consider a scenario in which a workflow integrator actor adapts an existing algorithm for use in data mining in such a way that it can be used within the workflow framework being developed in this thesis. The domain practitioner actor has a number of data sets of interest to their research. The workflow integrator produces a workflow utilising an algorithm, which is run and analysed by the domain practitioner. Once the workflow exists, it is then possible for the domain practitioner to reuse the workflow on other data sets, and possibly tune parameters to optimise their investigations. It should be noted that although there are three distinct roles expressed here, it is not unreasonable to assume some or all of these would be performed by the same physical person, depending on their skill set.

The goals of eScience and the Grid are not novel within themselves. Since computers first became available, investigators, integrators and computer scientists have been attempting to use them to perform mundane, repetitive tasks of data collection and analysis. As the speed and capabilities of computers have increased they have been tasked with increasingly complex responsibilities. This also requires the integrators and practitioners to think more about the presentation of the problems for the computers. Simply providing a set of linear instructions to solve a problem does not lend itself to reuse, so more abstract forms of the instructions need to be derived which allow the solution to be used in many situations.

In general eScience experiments aim to be easy to repeat and modify, as compared to their physical analogues, so they can be easily copied, modified, repeated and even shared. This means that the process for developing an idea differs from an equivalent physical investigation. With software, it is possible to prototype ideas quickly, and without significant financial penalty, leading to the exploration of many potential lines of investigation. The consequence of this approach is that many pilot investigations can be run, possibly without

due documentation, before the primary or significant investigation is performed.

The use of Grid computing for data mining is intended to improve the time-to-solution/completion of a data mining workflow by increasing the number of available processing units available to perform work. By doing this efficiently it also enables larger data sets to be analysed – within the constraints of the data mining algorithm's ability.

1.2. Data mining

Data mining is “the extraction of implicit, previously unknown, and potentially useful information from data”[3]. This is achieved using computer programs – the data mining methods – that implement an algorithm for fitting the data set of interest to an understood structure such as an equation or a decision tree, resulting in a model that represents relationships in the original data set. This model can then be used to better understand the original data set and the system from which the observations were collected, and also used to predict the responses from new, unseen data sets, which has application to decision support systems and other business requirements (Figure 1.1).

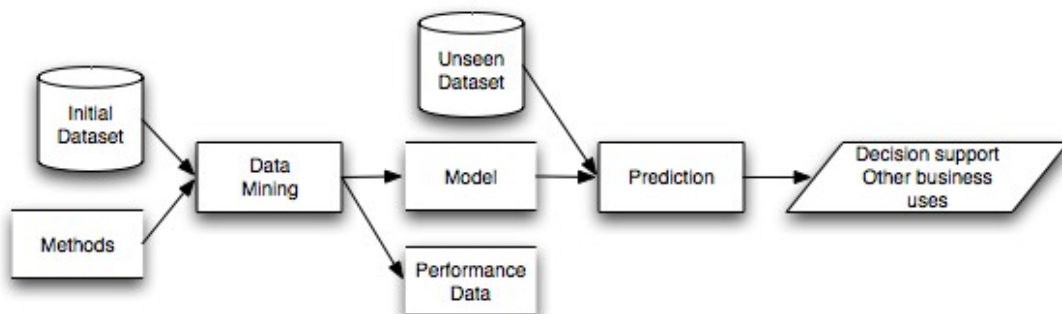


Figure 1.1: High level flow chart of the data mining process, and the outputs produced by it.

Data analysis in modern scientific investigations are moving towards the use of increasingly larger data sets, containing more observations and larger numbers of attributes. The increase in the number of observations is a consequence of improvements in automated data collection allowing more observations to be collected, and the increasing availability of public data sets which can be combined or integrated to produce a single larger data set. The

number of attributes collected for each observation is also increasing, once again driven by improved data collection at the instrument, and a growing number of synthetic or derived attributes showing promise or improving predictions. In the field of cheminformatics, where Quantitative Structure-Activity Relationship (QSAR) [14], Quantitative Structure-Property Relationship (QSPR) [15] and Quantitative Structure-Retention Relationships (QSRR) are studied, the data sets are composed of compounds (molecules), and an observed property of the molecule. The attributes of the molecules are calculated from the molecular structure, and can number in the thousands when topological, structural, electrical, solubilities, etc are considered.

The analysis of data sets like those just described often requires new, or modified statistical and data mining methods. Traditional statistical methods may not perform well (on their own) when there are a very large number of attributes involved. As an example, a simple method like multiple linear regression (MLR) will not compute an accurate fit when there are large number of attributes as it lacks a robust mechanism for disregarding unimportant attributes. This can be addressed through the use of either a variable selection technique to reduce the attribute space, or by using a more sophisticated linear modelling technique such as partial least squares (PLS)[16]. Obviously PLS is not a novel technique, but this does demonstrate that techniques must be applied within their limits of applicability, which is likely to become a more serious issue when researchers are confronted with increasingly large data sets.

In developing new or modified techniques to work on large data sets another issue is the time it takes to execute or “fit” a model. Many robust models have a large time-to-solution. Consider the use case of model validation. 10-fold cross validation, which is the minimum accepted by many practitioners in the field, will escalate the execution time by approximately a factor of 10 due to the 10 repetitions of the process. On a small data set this increase in time-to-solution will not be significant, but the addition of new observations or attributes to the data set will increase the significance of this impact until it becomes an

important consideration. While in some situations it may be acceptable to wait for extended periods for results to be returned, it is often desirable to reduce the time-to-solution as much as possible. It is also desirable to have a strategy for scaling the techniques should future data sets become larger. In this thesis tools will be developed to reduce this time-to-solution.

1.3. Scheduling Distributed Computing Execution

Data mining workflows take one or more data sets as inputs, and apply transformations or operations to these inputs to produce outputs. Within the workflow data will pass from the primary sources of data, through the operators and finally to the data sinks. Depending on the type and size of the data, considerations about data location may affect the time-to-solution just as significantly as the number of computational elements, or their speeds. There are two components to the transfer time problem, 1) the network speed, and 2) the network latency. Network speed is significant for large data transfers, and ultimately limits the transfer time. Network latency - the amount of time it takes signals to travel between hosts - is significant for small data transfers as it begins to dominate the actual transfer time when the data occupies only a few packets. This penalty for transferring data is the typical motivation for performing coarse grained parallelism in distributed environments, and restricting fine grained parallelism to clusters and shared memory machines. In the general case the workflow scheduler should be able to place the tasks appropriately to minimise time to completion by taking into account processing speed and data transfer speed.

By modelling the workflow in finer detail it becomes possible to effectively have the workflow system adjust the coarseness of the parallelism by scheduling tightly coupled sections on low latency sections of the network. This has particular applicability in scenarios where there is a mixture of single CPU and multiple CPUs per machine, as the workflow can be decomposed to an extent where the CPUs are being used efficiently on a specific section of workflow which would have otherwise required large data transfers.

1.4. Project Summary

The project described in this thesis had the goal of designing and prototyping an eScience system to address the requirements of data mining practitioners. This involved the collection of requirements based on first hand data mining experience, a survey of existing tools used by data mining practitioners in their work, and then the design and implementation of a prototype system to demonstrate the utility of such a system. This involved developing a simple workflow language which could elegantly capture the experiments data mining practitioners typically execute, and developing a task scheduling algorithm to improve the time-to-solution for executing data mining workflow across distributed hardware.

1.5. Contributions

This thesis makes several contributions to the fields of eScience and data mining. These are summarised as follows:

1. This thesis provides analysis of some typical data mining workflows and develops a general template for these investigations. From this template it is possible to determine the limiting factors in parallelisation, and to enable the understanding of the computational and data requirements for supporting the execution of these investigations.
2. This thesis presents a workflow model which addresses the requirements developed from the general model of data mining, and the design and implementation of a workflow engine to execute this workflow model.
3. This thesis adapts contemporary master-slave scheduling algorithms to efficiently schedule tasks derived from the workflow. Simulation studies have been conducted using the GridSim [17][18] toolkit, investigating the performance of these scheduling algorithms on a number of different network topologies.

1.6. Thesis Organisation

The remainder of this thesis is organised as follows: Chapter 2 presents a detailed

background of the application areas, and the technologies involved. Chapter 3 develops a framework which can be applied to the application areas to achieve the benefits of eScience and the Grid. Chapter 4 formalises the scheduling problem, and presents a new heuristic to address the problem. Chapter 5 presents a study performed using the framework, utilising modern data mining techniques to model the absorption of pharmaceuticals via the human intestinal pathway. Chapter 6 presents conclusions and future work.

Chapter Two

2. Background and Overview

Increases in the volume and resolution of data available for scientific endeavours is a consequence of technological advancements which allow more, higher quality data to be collected more easily. In recent years many large scale projects have provided frameworks for the automatic integration and collection of data from analytical instruments [19][20], sensor networks and large scale scientific projects. These concepts are being enhanced and informed by initiatives such as the OpenData [21] movement, which aims to standardise data formats, and encourage free and uninhibited access to data collected by researchers, in particular involving publicly funded projects.

Increases in the production and availability of data sets create a situation termed the *data deluge* [2]. The ability of researchers to use this data will be dependent on the availability of tools to discover and robustly analyse it. eScience aims to provide these tools, and in this chapter the requirements around data mining will be evaluated in an eScience context.

This chapter continues the discussion of the large and complex scientific data sets from Chapter 1, and covers the use case of data mining and computational statistics discussing their adaptation into an eScience framework. Initially, the field of cheminformatics is introduced as an example of a field where the growing size of data sets offers opportunities to improve scientific outcomes. Then the requirement for the use of computationally intensive data mining techniques is illustrated with a discussion of cross-validation, meta-models and randomisation techniques. Next, requirements for performing data mining within an eScience context are explored, including the use of a workflow to express the data mining process. Following this, a summary of distributed computing practises is presented, which identifies candidate resources, and approaches to utilising them. And finally, this chapter presents an overview of projects which have previously contributed towards the requirements identified for this project.

2.1. Robust Data Mining of Large Data sets

As outlined in Chapter 1.2, data sets are increasing in size and complexity, requiring new methods and approaches for their analysis. There exist many contemporary data mining methods that can be applied to these large data sets, however, these methods often have long execution times. However, many methods, and other processes of data mining fall into a class of problem that is naturally suitable for distributed computing. In addition, concepts and processes within data mining experiments can be readily mapped to eScience concepts, allowing a natural adaptation of common data mining tasks to an eScience and distributed computing framework. This approach will be covered in detail in the following sections.

Data mining is performed through the application of computational techniques to produce predictive and explanatory models of an observed system. Data mining typically uses standard types of models, such as decision trees and linear equations, and fits parameters and structures of the model to the observations. There are many different data mining techniques which take different approaches to the fitting process. Some techniques produce models which are better for explanation than prediction and vice versa. These techniques were developed and used in a variety of disciplines including statistics, computer science, artificial intelligence and machine learning.

Data mining techniques have applications that reach far beyond the disciplines that contributed them. In general, data mining can be applied to most fields which collect quantitative data, and is typically applied when there is no existing model of the system under investigation. The application to a new problem area requires different models and fitting techniques to be evaluated to determine the properties of the data, and the suitability of the particular model for that data.

Robust data mining refers to data mining techniques that are able to handle either large data sets, or high dimensional data sets without negative impacts such as:

- Over fitting – when a model is too closely matched to the input data that it is unable to perform on new unseen data, and

- The influence of outliers – an observation “... that appears to deviate markedly from other members of the sample in which it occurs.”[22], which can skew the data.

The remainder of this section will introduce cheminformatics, being the application area considered in this thesis, introduce the data mining process in the context of eScience, discuss the options for improving the execution speed of an experiment, and will then analyse some contemporary data mining techniques and experiments. This will result in a set of requirements being developed that encompasses what is to be achieved by this project.

2.1.1. Cheminformatics – Data mining in Chemistry

In this thesis the application area of cheminformatics [1] will be discussed and used in examples. Cheminformatics is the application of data mining techniques to the prediction and understanding of chemical systems. The particular field of cheminformatics discussed here is Quantitative Structural Activity Relationships (QSAR) [14], which relates some measure of chemical activity (response) to the chemical properties (predictors). A typical pharmacological application of QSAR would be to relate the bio-availability or absorption of a drug to the chemical structure of that drug.

Predictors for cheminformatics include measured molecular properties such as the near infrared (NIR) and mass spectrometry (MS) spectra, measured physical properties such as melting point, and other calculated properties known as molecular descriptors which are derived from the 3D molecular model of the compound (Figure 2.1). There also exist specialised data mining techniques which can utilise the 3D models directly, but in this thesis the focus will be on the use of numerical data sets. All of the numerical data sets indicated here have the property of having a very large number of predictors. In the case of NIR or MS data sets there may be 10,000 or more ordered points and can be gigabytes in size, while descriptor data sets may have over 1,000 molecular descriptors. The significance of having so many predictors must be considered in the context of a pharmacological experiment which may have only a few hundred observations, meaning that spurious correlations

between the predictor and response become likely. In order to handle this, either a variable selection pre-processing step needs to be applied to reduce the number of predictors, or a robust technique which works well in high-dimensional space should be applied. Both of these options typically have significant execution times. Both of these approaches will be discussed in more detail in the next section.

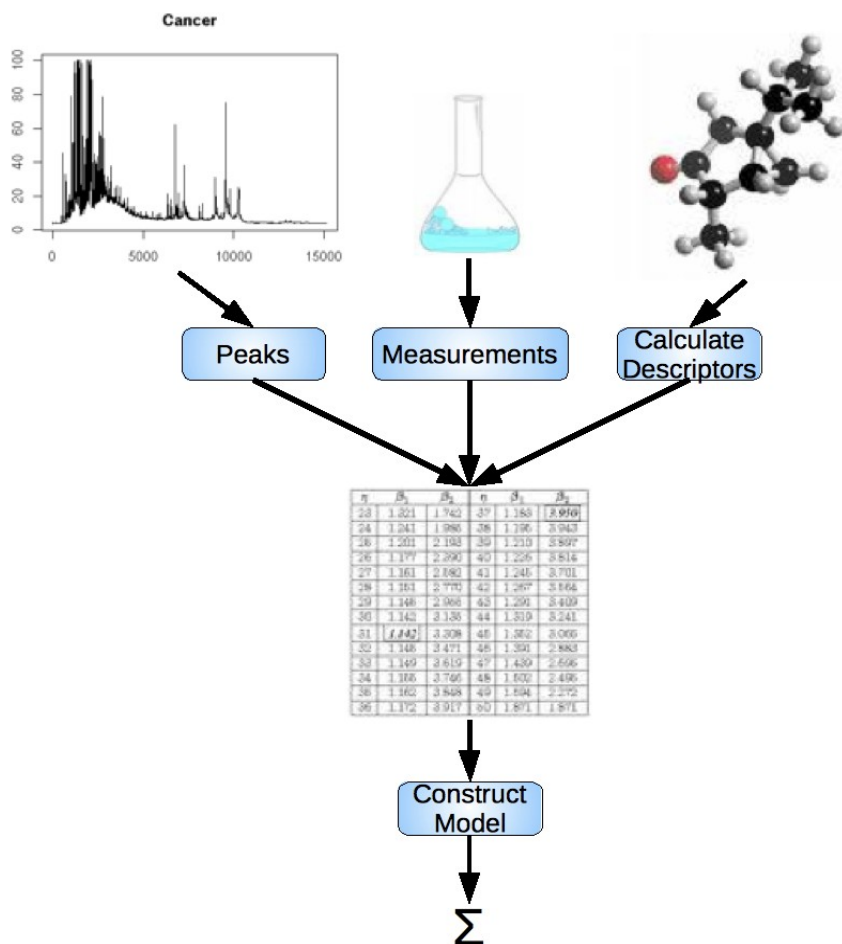


Figure 2.1: Process of distilling various sources of chemical observations into a table of data, from which a model can be built.

2.1.2. Data Mining as eScience

From the inception of many scientific investigations, information technology plays an integral role, beginning with experiment design and preliminary calculations, through to the laboratory reports and field trips where the data are collected. Direct data entry or data capture is being used to improve the quality and handle the quantity of measurements and observations, replacing situations where hand written tables would be used. Direct data

collection means that data can be curated or stored immediately along with experimental metadata associated with the data sets. Storing and organising the experimental data as it is generated gives the opportunity for data processing to begin while the remainder of the experiment is completed. This data can be used as quality control for the experiment, and to assist in tuning or optimising the experimental apparatus in the lead up to the main data collection stage of the experiment.

As described in Section 1.1, eScience is a term used to describe scientific endeavours that utilise information technology as an integral part of the process of scientific investigation. Exemplar eScience projects include Comb-e-chem [20] in chemistry, myGrid [23][24] in bioinformatics, and GEODISE [25] in engineering. Key themes in each of these projects include data and provenance management, experiment management via workflows, and distributed execution of tasks. Further, these projects, and others, strive to present this functionality as flexible services – middleware – that can be reused in other projects. As well, the required use of abstract experiment representations – workflows – ensures that the experiments being carried out are guaranteed to have at least a minimum level of standard documentation, aiding in the repeatability of the experiment and the integrity of the results.

It is proposed here that this holistic application of object oriented design, service oriented architectures, and the general desire for interoperability and reuse of components is critical in discriminating eScience from science simply done with computers. Often existing middleware providers are utilised, such as Grid computing, as the provider of computational resources.

A data mining experiment is a process that applies operations or transformations to input data to produce output data, models, and experiment reports (Figure 2.2). In eScience the experiment process is referred to as the workflow, and the experiment reports are the provenance and metadata of the experiment. eScience commonly relates to input and output data in the same way as data mining.

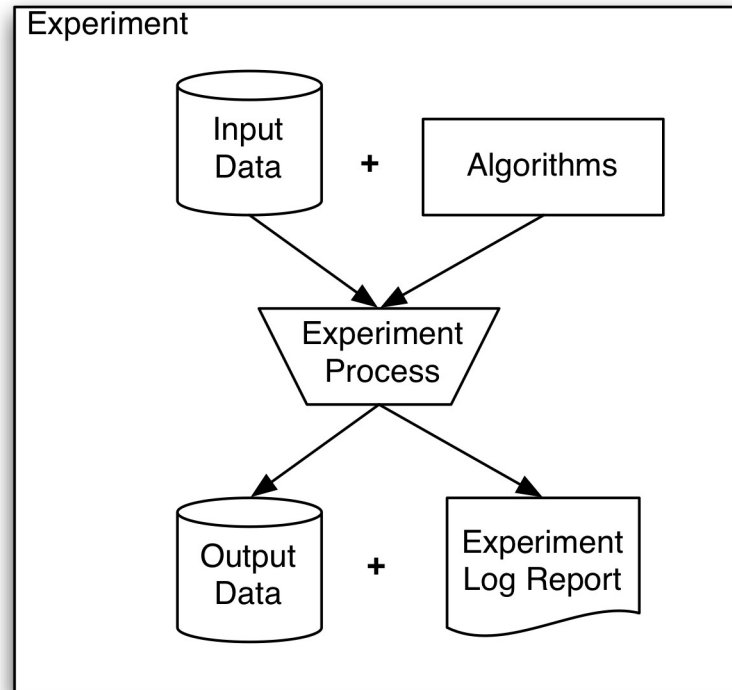


Figure 2.2: An experiment is the application of a process on input data and algorithms, to produce output data and an experiment log.

The model for a data mining process that will be considered in this thesis involves three distinct concepts: variable selection, cross-validation, and model fitting (Figure 2.3). Variable selection is a data set reduction technique that can improve runtime and predictive performance of data mining methods by removing clearly redundant variables from the data set; model fitting is the application of the data mining method; and cross-validation is an iterative technique for evaluating the real-world performance of a data mining method.

The process of eScience closely emulates the world of physical science. It involves data management, experiment management, workflow enabled data processing, as well as parallel and distributed computing. eScience sometimes enables physical science, while at other times eScience is itself an end, with experimentation occurring via simulation producing quantities of data which subsequently require analysis. In this thesis the entire data mining process is considered in the context of the eScience paradigm, with the intention to exploit eScience experiment and process management approaches, and to achieve an efficient speedup through parallel execution. In fact, it can be seen that the data mining process, when

presented in this way, is a very close match to eScience ideas.

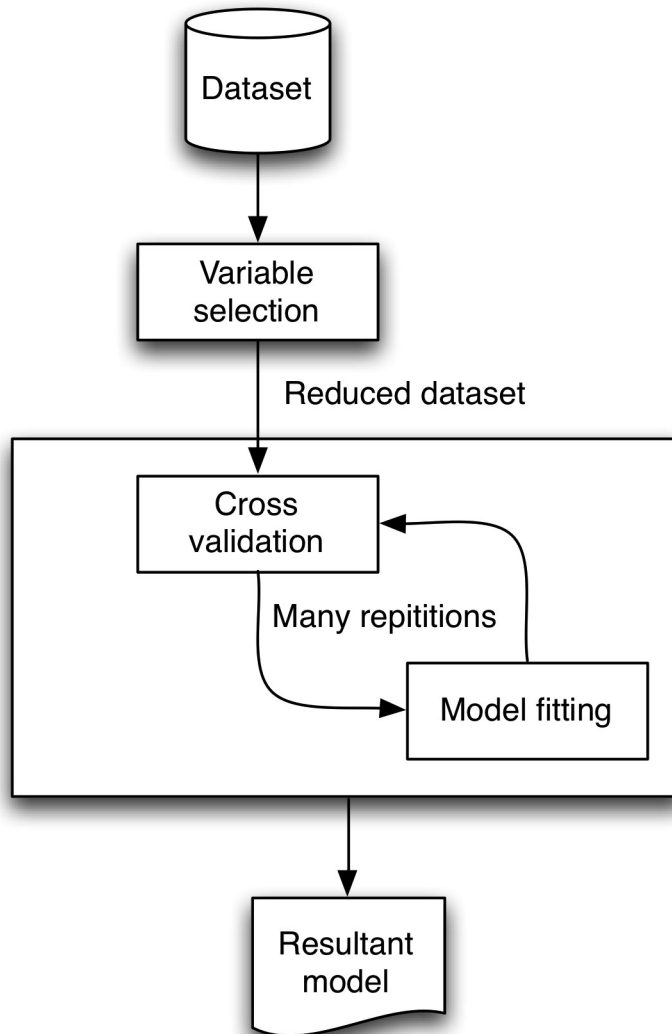


Figure 2.3: Typical data mining process considered in this thesis.

To achieve the outcomes of parallel data mining execution requires a data analysis system designed to support this functionality, but the benefits of faster execution time are significant. Whether the analysis is being performed for real-time applications, or the system is being used for the development of analysis techniques, reducing completion times from days to hours, or hours to minutes can be of significant value.

2.1.3. Efficiently Speeding Up Execution

The most effective method for improving the execution time of an application depends on many factors, such as: the language used to implement the application, the algorithms and

design of the application, and the way it uses data. Applications which are implemented using scripting languages can often achieve speedups by re-implementing (at least the slow method) in a compiled language, as this removes the interpreter and type checking overhead. Many scripting languages provide efficient ways to achieve common tasks, such as R[26], where you can achieve upwards of a 10 times speedup for loops if you use the *apply* method instead of a *for loop*.

Another approach to achieve speedup is to ensure the data are stored in an efficient way in memory, so that processing it does not incur unnecessary operations to transform it. For instance, the use of strongly typed, primitive arrays are much faster to access than object arrays. This is due to a couple of reasons: firstly the primitives are normally a smaller data type, so will fit better in memory and cache memory; and secondly because objects will normally be converted to primitives for the CPU to process, and this conversion incurs an overhead.

Once it is clear that the application is as efficient as is practical, it is possible to consider using multiple computing resources to perform the work. Parallel processing adds an extra level of complexity to the application, as it will now have to consider coordinating these resources, transferring data if required, and recombining the output data. This added complexity and overhead can make it impractical for smaller applications, but once the framework is established, it is possible to continue to scale to additional resources.

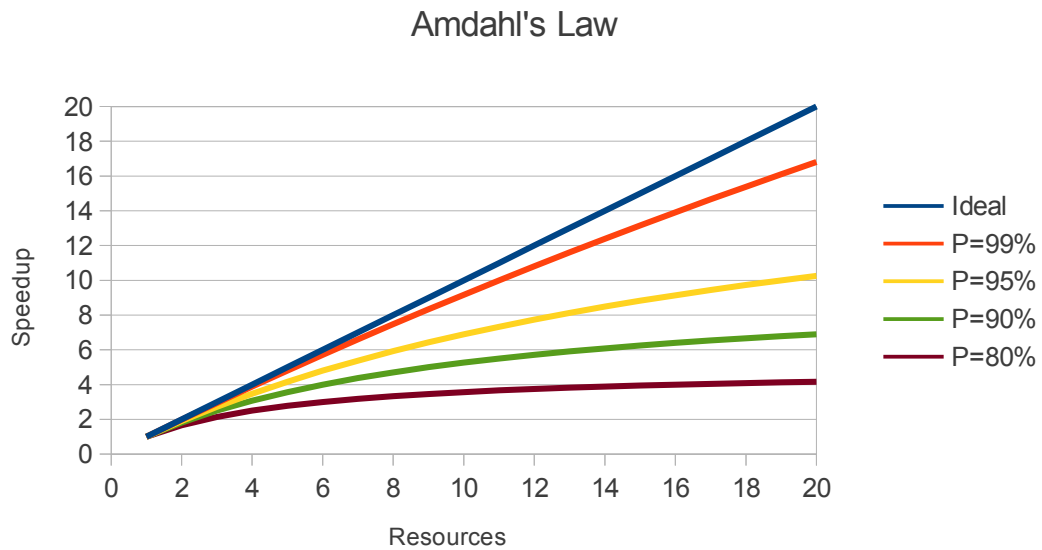


Figure 2.4: Amdahl's law showing the speedup obtained by adding additional resources.

The benefit gained by adding additional resources to an application is described by Amdahl's law, which expresses the diminishing returns of adding additional resources as a function of the percentage of the application that can be made parallel (Figure 2.4). This is expressed as:

$$Speedup = \frac{1}{((1-P) + \frac{P}{S})} \quad (2.1)$$

Where S = the number of resources, and P is the percentage of the application that can be made parallel. To be made parallel in an efficient way the application needs to be broken into tasks that have minimal data communications overhead, ideally only requiring communications at the start and end of the execution (loosely coupled), and where the execution time dominates the communication time (coarse grained).

Applications that can be broken into coarse grained, loosely coupled tasks are in the well-known class of parallel programs called embarrassingly parallel (EP) [27]. EP problems can be efficiently solved using up to n independent machines, where n is the number of models. Consider a single task that occurs in three distinct steps:

1. Load data into memory,
2. Compute model, and
3. Return result.

The time taken to perform steps 1 and 3 will be limited by the bandwidth and latency of the connection between the main memory of the machine and the storage resource being used to store the data. When the data are local to the machine it may be much faster than if the data are stored on a remote machine accessed via a network. For an EP problem the time taken to perform step 2 will be limited by the speed of the CPU in the machine. There also exists a spectrum of parallel applications which require data to be transferred between tasks during execution, and are described by the frequency with which these exchanges occur, from coarse grained to tightly coupled. In this spectrum of parallel applications the time taken to perform step 2 will also be affected by the latency and bandwidth between machines, in addition to the CPU speeds of the other machines and their own CPU speeds. This is because resources have to wait while other CPUs calculate and deliver the data they depend on, resulting in idle CPU time.

From this discussion it should be clear that efficient parallel processing is dependent on being able to break the data mining application into coarse grained, loosely coupled tasks. The discussion of parallel processing will be continued in Section 2.4. Next, contemporary data mining techniques will be introduced, and assessed as to how they can be efficiently parallel processed.

2.1.4. Review of Contemporary Data Mining Techniques

There are a large number of data mining algorithms and techniques employed in a field like cheminformatics. They include classification, regression and clustering techniques, and aim to produce both predictive and explanatory models to aid the practitioner in their work. The effectiveness of a technique will be influenced by the data, and the desired outcomes of the practitioner. Data dimensionality will be a primary driver in the selection of the

algorithm, with many methods failing as the number of variables increases, requiring variable selection as a pre-processor. Many methods can only utilise either continuous or categorical data, and may not be able to handle missing values. Depending on the form of the model, there will be a trade-off between a complex model which has a high degree of predictive power, but is hard to interpret, and a simpler model which has less predictive power but is more easily used as an explanatory model.

Once a model has been fitted to a data set it is important to know how well the model represents the data. There are a number of methods which compare the predicted and actual responses. Two very commonly reported figures are error and variance. These are commonly reported as mean absolute error, *MAE*, often calculated as a mean squared error, *MSE*, or its square root, *RMSE*, which all approach zero as the model improves. Variance can be calculated using Pearsons correlation coefficient, r , or it's square R^2 which approach 1 as the model improves. These methods are covered in more detail in [28] and [29].

However, an important concept to understand when fitting complex, or even some simple models, is that the model needs to be generalised (not over-fit) if it is to be of any use. That is, it needs to be able to make accurate predictions from data that was not in the original data set. Evaluation of this requires that an external test set be available to validate the performance of the model. This process is known as *cross validation* (CV). There are a number of specific approaches to cross validation summarised in Table 2.1. The selection of a CV technique will depend on the type of data, the size of the data set, and the requirements for time-to-completion and the accuracy of performance metrics.

Simply splitting the original data set into a test set and a training set is straight forward and computationally inexpensive, but it does not robustly evaluate the performance as artefacts in the test or training set can give misleading results. *Leave One Out* (LOO) validation iteratively selects each observation from the data set and removes it as the test set, performing the model building process on the remainder of the data, and averaging the resultant figures. Due to the small size of the test set this method is also prone to issues such

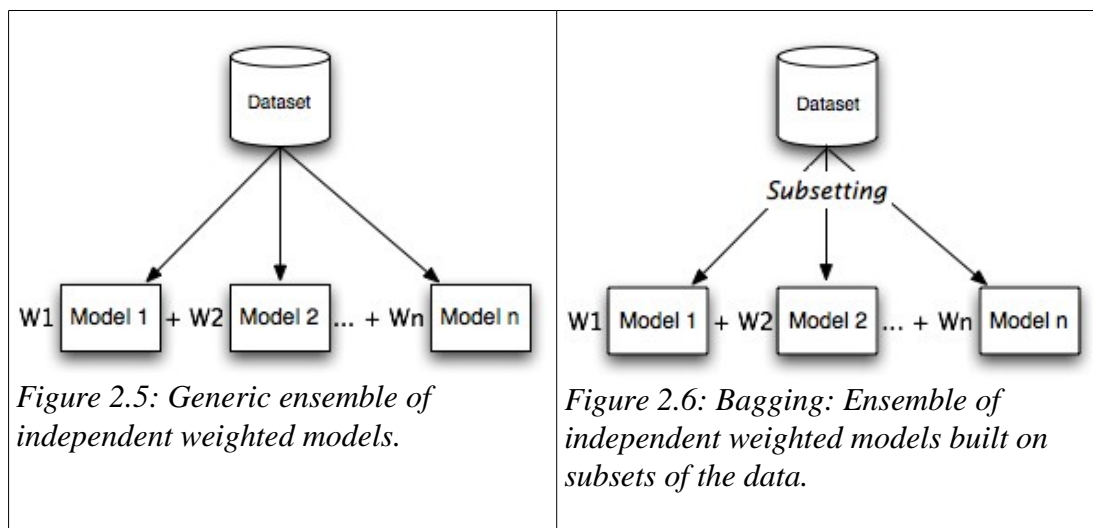
as high variance [30][31], and it comes at a higher computational cost.

A variation on the LOO is leave group out (or k-fold), where a fixed number of observations are removed into the test set. This number is typically N/k where N is the total number of observations and k is the number of times this will need to be repeated to use all observations in a test set. This offers better computational performance than LOO, and can also avoid the high variance drawbacks. Leave one out is a special case of leave group out, where $k=N$.

Monte-carlo cross validation (MCCV) [32] is a more robust and accurate performance estimation technique, which approaches the performance estimation problem as a monte-carlo simulation. At each iteration of the validation the data set is split, with training and testing performed as before. However, this process is repeated hundreds or thousands of times, and an aggregate of the measured performances is reported as the result, meaning it is computationally the most expensive. This process eliminates any bias which may accidentally occur due to the prior ordering of the data, but does so at great computational expense.

To appreciate the impact of the computational expense of each cross-validation technique, remember that each training iteration involves performing the model fitting, which itself may be a computationally expensive operation. For example, if a data mining technique takes 1 minute to fit the model on 100 observations, it will take ~ 1 minute to cross validate under a single split CV, 10 minutes under 10-fold CV, 100 minutes under LOO CV, and 1000 minutes under MCCV.

when combining their base learners. An example of this is bootstrap aggregation or bagging [33] (Figure 2.6), which trains many models on a sample (bootstrap) of the training set. The bootstrap sample is of the same size as the original training set, and is sampled with replacement, meaning that some observations will appear many times. When evaluating the bagged model all the individual models are evaluated, and the average of the predictions is used. The resultant model has a low variance and is more resilient to overfitting. Another example of an ensemble is boosting, which takes a base learning method and produces many individual models using subsets of the input data variables, and combine these individual models into a single larger model.



One popular ensemble technique which utilises bagging is Random Forests [34]. Random Forests uses decision trees as the weak learner. Each tree is trained on a bootstrap of the training data, and at each node a bootstrap of the predictors is used to calculate the split. This is repeated for a number of trees which are combined to make the Random Forest ensemble. Random Forests can perform well in the presence of many predictors, and produces accurate predictions in many situations.

Another popular data mining technique for cheminformatics is the Support Vector Machine (SVM) [35]. SVMs can be used for classification and regression and offer a robust approach to fitting models. Importantly, they can be used with various kernel transformations

on the input data which allows them to fit non-linear data sets.

2.1.5. Typical Data Mining Experiments

In this section the development cycles of two actors are discussed: data miners who are interested in analysing data sets; and algorithm developers who are interested in producing and evaluating new data mining techniques. Both these groups have the same core requirements, although they approach the problem from different angles. First the data miner's environment will be described, and then the algorithm developer's environment, highlighting the differences.

Data mining practitioners start their work by identifying a data set of interest. Whether they collect the data themselves or derive it from other data sets is not relevant, but in either case it is assumed they can identify the response and predictor variables so these can be considered as generic data sets. Assuming the data are in an unprocessed form it will now be analysed to test the quality of the data set, filtering out homogenous variables, and perhaps centring and normalising the variables.

Next a decision will be made by the practitioner about the validation technique that will be used. All validation techniques involve splitting the data set into two parts, training a model on one part, and evaluating the model's predictive capabilities on the remaining part (Figure 2.7), and each research field that utilises data mining tends to have its own accepted validation technique.

Following this, the actual modelling techniques to be used on the data must be selected. This decision will be guided by a combination of domain knowledge, the linearity or non-linearity of the data, whether the model is for prediction or understanding the data, and personal preference and experience. Nearly all but the simplest of modelling techniques will require a number of parameters to be set, requiring manual selection by rules, automatic tuning algorithms, or parameter sweep operations.

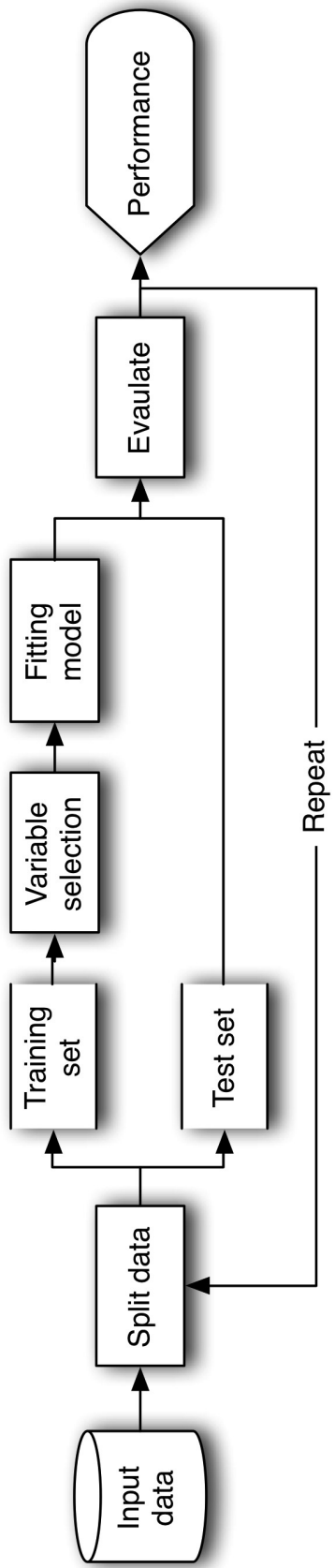


Figure 2.7: Flow chart of cross-validation as applied in data mining.

Finally the practitioner will execute the experiment with the selected validation, model, and parameter selection, and will analyse its performance. This may lead to changes in the set up, and re-running of the experiment in an attempt to improve the outcome. Ultimately the experiments will produce a model which the practitioner can use to predict new unseen data, or be used to describe the system from which the data was originally collected.

Data mining practitioners will hold the data set and probably the model constant, and attempt to tune the parameters to get the best performance from the models and the best understanding of the system.

Algorithm developers are typically interested in using many data sets, selected for their data type or origin (spectral, QSAR, etc) and using these to evaluate the algorithm or modelling technique they have developed. Therefore algorithm developers will hold the model and possibly the parameter selection constant, and evaluate across many data sets. An example of this kind of experiment is presented in Appendix A.

<p><i>a)</i></p> <ol style="list-style-type: none"> 1. clean data 2. cross validation 3. variable selection 4. model parameter selection 5. model building 6. summarise 	<p><i>b)</i></p> <ol style="list-style-type: none"> 1. data sets 2. clean data 3. cross validation 4. model building 5. summarise
---	--

Figure 2.8: Abstract representation of the list of steps for the generalised process that a) data mining practitioners and b) algorithm developers, may use in their work.

Figure 2.8 shows an abstract list of steps that a data mining practitioner and algorithm developer may use in their work. Indentation represents sub-steps that are performed in the context of the higher level indentation. In both cases, each iteration of the cross validation will require a model to be built. It is clear that there are common constructs between these two lists of steps. The common structures of these lists are extracted and presented in Figure 2.9 as a generic template that demonstrates these structures. It is not intended to be fully representative of the work practices of of all data mining practitioners, but rather it capture

the core elements of the process sufficiently for us to then map these practices to the paradigms of eScience and the Grid.

```
1. For each data set
2.   for each model type
3.     for each cross validation split point
4.       compute model
5.         record performance
6.       compute model cross validated performance
7.   for each model type
8.     compute cross validated performance across data sets
```

Figure 2.9: A typical data mining experiment template exhibiting iteration, nesting of iteration and sequential application of operators.

Figure 2.9 shows a typical data mining experiment, from which the core requirements for the experiment process language of the data mining eScience platform can be developed.

These are:

1. Nestable iterations or looping over data;
2. Invocation of actions with the data exposed by the iterations;
3. Storage of output from actions;
4. Chaining of multiple actions in sequence; and
5. Loops or actions executed in sequence.

The essential elements of this template include: the idea that data should to some degree be self-describing; the observation that the process of validation is an EP class problem; parameter tuning, whether through a sweep or an optimisation is also EP; capturing environment and performance information for each execution of the algorithm is a desirable outcome; and by reducing the time to completion it can enable more focused and hopefully productive efforts on the particular problem at hand. Further to this, the pre-processing, execution and post-processing of the data may also be composed of other elements which themselves could be transformed into EP problems.

2.1.6. Properties of an eScience Data Mining System

The previous sections presented a high level overview of the data mining process as it,

applies to cheminformatics, and some typical experiments that would be performed. The goal of the project reported in this thesis, was to develop a system to assist the data mining process by addressing the following requirements:

1. Repeatable experiments,
2. Capturing the process as completely as possible to minimise manual setup and data handling,
3. Well documented experiments, at least to the extent of capturing the inputs, outputs and experiment process,
4. Minimise execution time of experiments and allow the ability to scale up using arbitrary computing resources,
5. Make it possible to integrate existing data mining applications, and
6. Make experiments easily adaptable to different input data sets.

Clearly if a system such as this was to be made useful to the general data mining community it would also need to address other issues which are beyond the scope of this thesis, such as:

1. Recovery from failure,
2. Security, and
3. Integration into a wide variety of data, compute and service providers.

The requirements being explored in this thesis can be broken into three groups:

1. eScience and experiment management (#1, #2, and #3),
2. Integration and deployment (#5 and #6), and
3. Distributed execution (#4).

The following sections will each investigate the current state of research and practise in each of these three groups of requirements.

2.2. Experiment Expression and Management Review

A data mining experiment needs to be expressed in a way that makes it easy to

understand, adapt, reuse, and also execute. Scalability and fast execution is a requirement for this project, thus the expression of the experiment should make it possible, if not easy, to achieve this, by allowing the experiment to be decompose into individual tasks that can be executed in parallel. Then, once the execution is complete, the results of the experiment should be accompanied with the appropriate experiment metadata and data provenance information so the experiment can be archived, discovered, understood, and validated by a third party. These requirements will be investigated in the rest of this section.

2.2.1. Expression of a Data Mining Experiment

Computational experiments, such as data mining, may be performed using a high level language such as MATLAB, R or Java, and will include code which handles the data manipulation, algorithm execution and overall process flow. To better achieve the project requirements the experiment should be structured in such a way that the high level coordination concerns, such as input and output data sets and setup parameters, are separated from the lower level algorithm implementation concerns (Figure 2.10). Further, these lower levels should be parameterised such that changing the experiment setup does not involve changing multiple sections of the overall experiment.

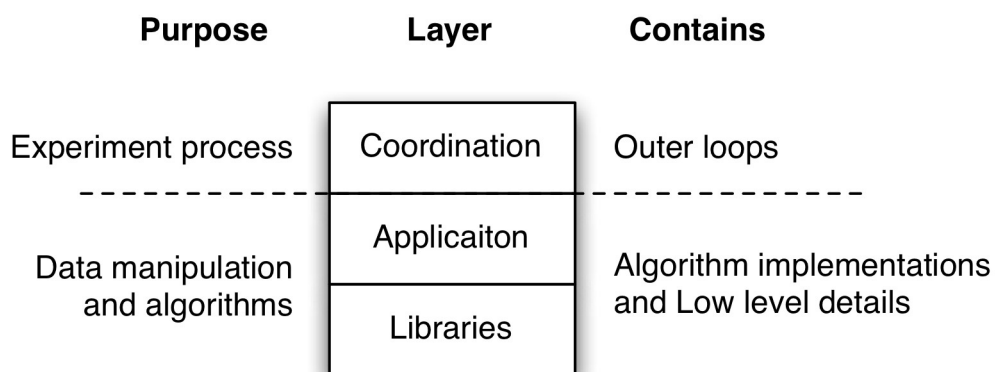


Figure 2.10: A computational experiment is comprised of algorithms, data management and experiment management. Workflows provide the experiment management functionality, allowing the application to be more focused and general purpose.

In general, the coordination layer should capture the main data operations and outer loops of the experiment, such as the implementation of the cross-validation, as these have the greatest potential to provide the coarse grained, loosely coupled tasks suitable for parallel execution. The further the boundary is pushed into the data mining application, the greater the potential for parallelisation, but the gains achieved will diminish as the overhead of coordination layer begins to dominate the speedup. As well, the coordination layer will apply constraints on the types of operations that can be captured, preventing the boundary being pushed further.

The task model used by the coordination layer will determine what types of, and degree to which, experiments can be captured. These models vary in terms of how compact, expressive or general purpose they are. These include:

1. *Process workflows* which chain together operations with dependencies,
2. *Dataflows* which explicitly track the data movement between operations in workflow like dependency graphs,
3. *Bag of tasks* which are independent tasks,
4. *Parameter sweeps* which iterate over all combinations of candidate parameters, invoking a single main task each time, and
5. *Syntax tree* which nests iterators and conditional propositions around operations which need to be performed.

The suitability of each of these coordination task models will be discussed based on the typical data mining experiment requirements presented previously. Process workflows represent the interdependencies between operations, where an operation needs to be completed before the next operation is signalled to begin. The operations in workflows can perform any actions, and will produce and consume signals or data. However, due to the way the signals propagate through the workflow graph it would be difficult to conceive how to fulfil the nested iteration requirement with this scheme, without providing high level operations that encapsulate the entire data iteration and execution process. This would

obscure the EP tasks that would provide the speedup opportunity.

Dataflows are quite similar to process workflows in that operators and data signals are represented by nodes and edges respectively. They differ however, in that dataflows do not operate from signals, but rather the availability of data. In this way the data flows from one end of the graph to the other, like a pipeline. Like a pipeline, data can be in-flight in multiple places along the pipeline, and simply block when waiting for the next operator if it is still processing the previous data package. However, like the process workflow, the dataflows provides too high level abstraction to expose the parallel aspects of an experiment without developing experiment specific actors.

Bag of tasks are a very coarse grained model for the coordination level. They represent each individual task that will need to be executed to fulfil the experiment. Although this is the form that the coordination layer will eventually decompose the experiment into, it is not suitable for modelling the experiment due to its verbosity.

Parameter sweeps, in their purest form, represent the execution of a single operation with each combination of parameters from a parameter space. Implementations of parameter sweep systems, such as Nimrod, list the parameter space and specifies the task to be executed for each item in the parameter space. This type of system does capture the intention of the nested iteration part of the requirements. However, in itself parameter sweeps provide only a subset of the requirements.

Syntax trees are an open ended model, with their suitability determined by what syntax is supported. As a reference to the variety and possibility of syntax trees, all textual programming languages are represented as a syntax tree at some point when being compiled or interpreted. By providing data iteration and execution the syntax tree can be used as an alternative representation of the parameter sweep, using compact syntax. By unrolling these data iteration loops it is possible to decompose the syntax tree into individual tasks, constrained by input data that are suitable for parallel processing. The chaining of loops and operations could be allowed by the syntax grammar, and implemented by the coordination

layer provided the data dependencies between the derived tasks are properly tracked. Finally, the order of execution of the operations within the iterations could be preserved by a constraint in the coordination layer.

From this discussion it is clear that the more general purpose syntax tree model supporting data iteration and operation execution is the most suitable fit to the data mining experiment requirements. It is a compact, abstract representation of the experiment that is contained enough to allow the coordination layer an opportunity to decompose the experiment into parallel tasks.

2.2.2. Experiment Management – Collection of Experiment Metadata

The execution of computational or *in silico* experiments like those addressed in this thesis follows the same patterns of scientific endeavour as any other scientific pursuit. Accordingly, clear, accurate and detailed record keeping is as important *in silico* as it would be in a physical laboratory. As all the information about experiment set up is already available in computerised form it should, in principle, be a much simpler task to capture this information in a standardised way, than it would be in a comparable physical laboratory. Various disciplines do specify standards and procedures for the collection and storage formats of metadata, but there exists no encompassing standard or procedure. For instance, a data set analysed in a cheminformatics study may provide chemical names or Simplified Molecular Input Line Entry Specification (SMILES) [36] descriptors under investigation, and will perhaps report the programs and techniques used to calculate the molecular descriptors used in their analysis. However, the parameters for these programs and optimisations which would have been performed along the way may not be thoroughly reported in the text. While there may be alternative avenues to discover this additional data in some circumstances, such as direct contact with the author, generally this lack of consistent detail makes the reported findings far less useful, and certainly less reproducible.

By definition, the Grid computing environment consists of heterogeneous collections of

computers, connected by a variety of interconnects or networks, running a variety of software environments. This variability introduces the potential that the experiment may not be reproducible or reliable. Extensive documentation of the hardware and software can begin to overcome this, allowing comparisons of the different execution environments which can explain unusual trends in the data. As part of this standardisation, it is suggested that mandatory test cases should be incorporated to validate the methods on the execution machines.

As discussed by Simmhan, et al. [37], provenance metadata can be collected about different parts of the data processing system, and can contain multiple levels of detail. Simmhan, et al. define two models for provenance: firstly data-oriented provenance which is concerned explicitly with the data, and secondly process-oriented provenance which models the derivation process and can be used to deduce the provenance of the data by inspection of the process inputs and outputs. Uses for provenance metadata in eScience as defined by Goble, et al. [38] includes data quality, audit trail, replication of experiments, attribution of the data owner and contextual information which would help interpret the data.

Greenwood, et al. [39] describe their experience using the myGrid [23][24] provenance system for capturing metadata related to bioinformatics experiments. The myGrid project is a framework for executing scientific workflows composed of Grid Services. Greenwood, et al. *ibid* describe the use of two types of metadata. Firstly, *derivation path* describes the process of transforming the input data via services, and includes database queries, applications and their parameters, and secondly *annotations* that can be attached to data to describe generic attributes such as ownership, as well as domain specific attributes such as the chemical names or protein sequences. They demonstrate the usages of these sets of metadata in making experiments findable, repeatable and quality assured.

Chimera [40] is another process oriented workflow system, which tracks the derivation path of the data. Workflows and provenance are stored using the Virtual Data Language [41]. Once run, workflows can be queried via a central catalogue, and re-run taking advantage of

previously derived results to hasten execution.

Provenance metadata provides critical information for validating the process and outputs of *in silico* experiments. Although there is no standardised storage format or collection mechanism, there are many examples of successful and useful provenance metadata collection systems, such as those just discussed, which can act as a template for future systems. The methods for the collection of provenance information will vary between application implementations, and do not need to occur within the main application. A pre and post process would be able to collect this information without the need for modification to the running main application. This could be written to a document store, and returned along with the results of the main application.

As an example of the application of this provenance information to the validation of an experiment consider the following. It is possible to collect generic apparatus information from *in silico* which includes static machine information such as hardware and software. Gathering this data creates a reference metric which can be used to group and loosely compare executions run on different machines. The same workflow run on a machine with the same memory and CPU should yield results that are computationally equivalent. If discrepancies are significant this would indicate there are issues relating to the hardware, software, or the application being executed. Then software dependencies can be investigated, ideally leading to the offending library, or methods being identified and addressed.

2.3. Deployment and Integration Review

This section discusses the requirements to leverage existing data mining, as this is where the expertise of the practitioners already lies, and to integrate with existing data sets. The separation of concerns between the experiment coordination and low level data mining introduced in Section 2.2.1 allows the data handling to be absorbed into the coordination layer, and enable the reuse of existing, “legacy” applications. As well, a discussion about some real world, desirable requirements is presented at the end of this section for

completeness.

2.3.1. Execution of Operations and Applications

Any application or data operation that is going to be executed in an experiment needs to be invoked through a standard interface that specifies the inputs and outputs of the execution. This interface is what separates the coordination layer from the implementation layer which may be a combination of new and legacy applications. Legacy applications, or legacy code, refers to any existing application which is in use, and which may not support any forward looking interoperability, and which cannot or will not be modified to adapt them. Legacy applications are typically written to expect a certain specific data format, with known data layouts and parameters which are exposed through an arbitrary interface. To enable legacy applications to be used in a Grid environment requires some analysis of the functionality that will need to be exposed, and on whether the code can be modified or if it will have to be wrapped to provide the required interface.

Integrating these independent systems is one of the challenges of eScience, and it is fundamentally addressed through the use of standardised systems, and rich, consistent system metadata. This is often referred to as a semantic-rich system, as this extra system metadata enables automatic system configuration, reuse, adaptation, and documentation. There are a number of established technologies, discussed throughout this section, that assist in achieving these goals.

The use of arbitrary computing resources for performing executions assumes that the resources provided will be suitably configured to run the desired applications, and that the applications are written in a portable way that enables them to be executed in the provided environment. Typical issues that will restrict the use of a resource include:

1. Requirements for the application to be installed on the resource,
2. Licensing preventing the desired application being used on the resource,
3. Incompatible versions of the application on the resource,

4. Missing or incompatible shared libraries available on the resource,
5. Missing runtime environment, or
6. Incorrect hardware architecture for the application.

Issue #1, #2 and #3 can only be addressed by selecting resources that provide the required software. The impact of remaining issues will vary depending on the type of application that is being used. This could be a native executable or library, scripting language or byte code language that runs on a virtual machine.

If the application is compiled to native machine code, one approach for ensuring portability across environments, addressing issue #4, is to statically link all the required elements into a single relocatable package, which reduces the assumptions about the libraries that are provided. However, unless the application is compiled for all of the possible machine architectures then issue #6 may prevent some resources from being utilised.

Issue #6 is overcome if the application utilises an abstract environments such as a scripting language, such as MATLAB and R, or virtual machine (VM), like Java and Python. Scripting languages and virtual machines differ in the degree to which they abstract the computing environment. Virtual machines often provide the best portability, so long as the application is purely executed in the VM, and does not invoke native code to perform part of the calculation. This is because the shared libraries that often accompany the application will most likely also be built to run in the VM, making them portable across hardware. Scripting languages on other hand, particularly MATLAB and R, are often high level abstractions on top of native libraries, meaning that portability is potentially hindered if the required libraries are not available or installable on the target resource.

Software agents run on the resources can be used for invoking the application on the resource provided. The role of the agent is to:

1. Inspect the execution environment for suitability,
2. Prepare the environment if required,
3. Invoke the application,

4. Monitor the application, and
5. Clean up the environment once execution is complete.

In the wider computing community software agents are used to perform many monitoring and maintenance tasks, and the exact definition of a software agent may vary. Within the parallel and distributed computing community, agents are utilised to perform many of the functions mentioned previously, with varying sophistication and scope. Cluster computing software such as PBS [42], Xgrid [43] and Condor [44], and Grid software such as Nimrod/G [45] use the agent model to stage data and launch applications on remote systems. They typically do not provide any form of interprocess communication (IPC) with this service, but can set up a third party system to provide this.

Once the application is able to be executed on the provided resource the next step is to ensure the input and output data are in an appropriate format for that application. This requirement is covered in the next section.

2.3.2. Data Interchange and Conversion

Larger catalogues of data are becoming available, assisted by increasing adoption of information technology and use of network services. These are provided by researchers making data available to collaborators, colleagues and sometimes the general public through data repositories. Provided the data are adequately described then it is possible for the data users to confidently perform their research without having to do primary data collection. For instance, a statistician developing or evaluating methods of classifying DNA microarrays will potentially be able to evaluate their work using one of many publicly available data sets.

To use a given data set within a data mining experiment requires that the data set be available in a format that can be understood by the data mining algorithms. Achieving this may require preprocessing before being used in the main data mining process. In order to completely capture the process it is important that these steps of format conversion and preprocessing be captured by the data mining system. Doing so helps to prevent unidentified

errors creeping into results due to ad hoc or undocumented steps in the experiment processes. Ideally, the data mining system would provide a way to encapsulate this step by conceptually providing a system for adapting or transforming the data, allowing the original data set to be used directly (Figure 2.11).

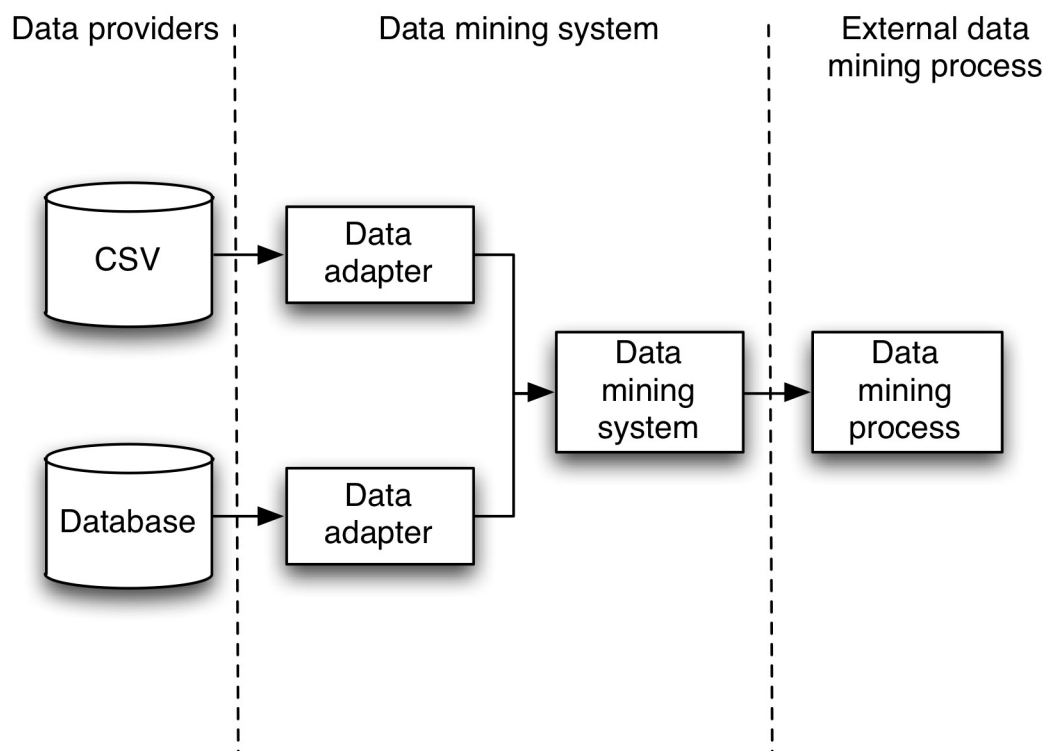


Figure 2.11: Adaption of disparate data sources to provide a consistent interface for the data mining workflow.

Building systems which can interact in the way described here is a major focus of the Grid, as interfacing with existing data services is a major challenge for adoption of any new technology, including Grid, by existing practitioners. This problem is typically exhibited when legacy applications are to be used to analyse the data sets collected and curated using Grid technologies. While the description given is quite straight forward, it disguises many issues that arise including, interactions with proprietary, and legacy systems which may have been developed prior to standardised formats, and may not be modifiable.

There is almost no limit to possible data file formats, but in general there are normally only a handful of formats that will be prevalent in a given field. In cheminformatics data are often tabular – like a spreadsheet or database table – and stored in either text files, or binary

table formats like netCDF [46]. Conversion between these formats is relatively straight forward, but possible issues could include: loss of precision when converting to text formatted numbers; exact formatting rules such as delimiter characters; or column names. Other cheminformatics data formats such as microarray images or chemical structure files will encounter similar conversion issues.

Beyond just converting data between formats to allow interoperability between algorithms and applications, using data adapters to allow the data mining system to address and subset the data would enable the given data set to be iterated over as was discussed in section 2.2.1. However, not all data will require this functionality as a data mining process may not call for it. This is fortunate, as it becomes hard to manage having complex structured data represented within a generalised data processing system without the system becoming overly specialised and complex. For instance, a structured record representing a molecule may be easily handled within a cheminformatics application, but it is unlikely that the molecule will need to be subsetted or iterated over. Also, should it require analysis with a statistics application, it would have to be transformed into a format that can be represented with that application.

Once the data are of the correct format it must be provided for the application to use. The simplest method of providing data to legacy applications is to stage the data via the file system. This involves writing copying the file from its original location, potentially across a network, and writing it to the file system on the resource executing the application. This method provides a simple, portable method for allowing access to remotely sourced data (Figure 2.12), as it does not require any modifications to the end application and it does not require any special functionality from the operating system. The drawback is that the entire data file will need to be copied across the network, which is inefficient if the entire file is not used.

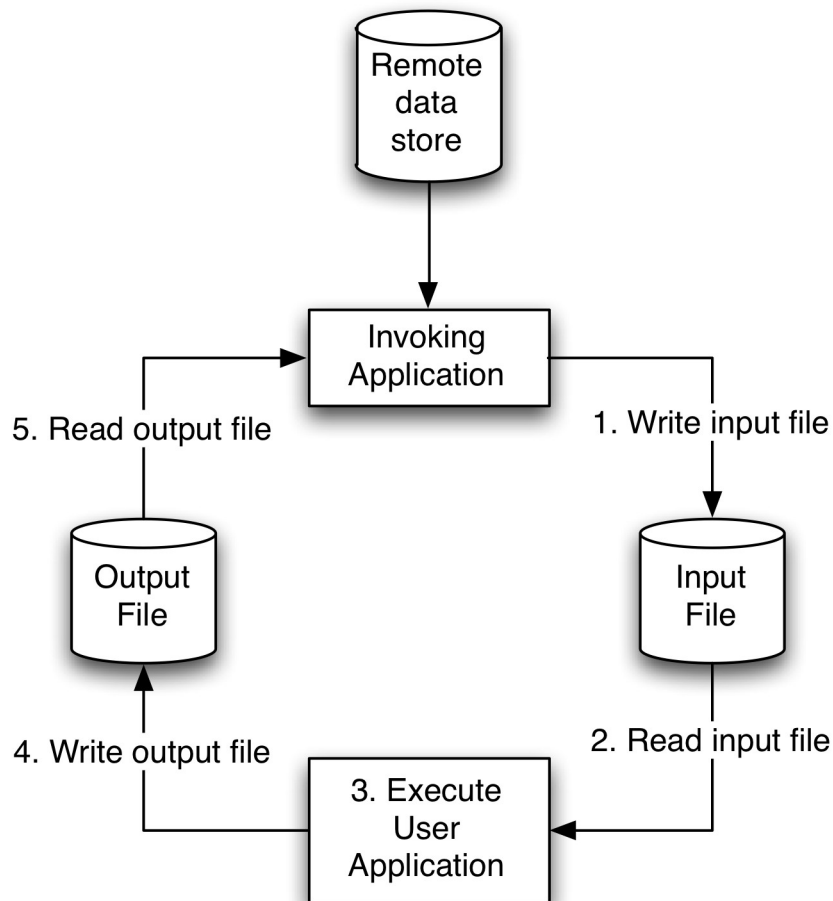


Figure 2.12: Staging data via disk is a simple way of bridging legacy applications to the Grid.

A potentially more efficient solution is to provide a transparent, remote access to the file. Remote access is potentially desirable if only a portion of the file is actually required, and provided that the file can be accessed without too much latency. However, this property may not be determinable at the commencement of execution. Further, providing remote access will either require modification of the executed operator so it can talk to the remote data store, or the deployment of a data access shim such as Parrot [47] to intercept file Input//Output (IO) calls made by the application and redirect them to the appropriate remote file store (Figure 2.13). This approach would also have deployment issues as not all operating systems will support such shims, and the shims themselves will be platform dependent.

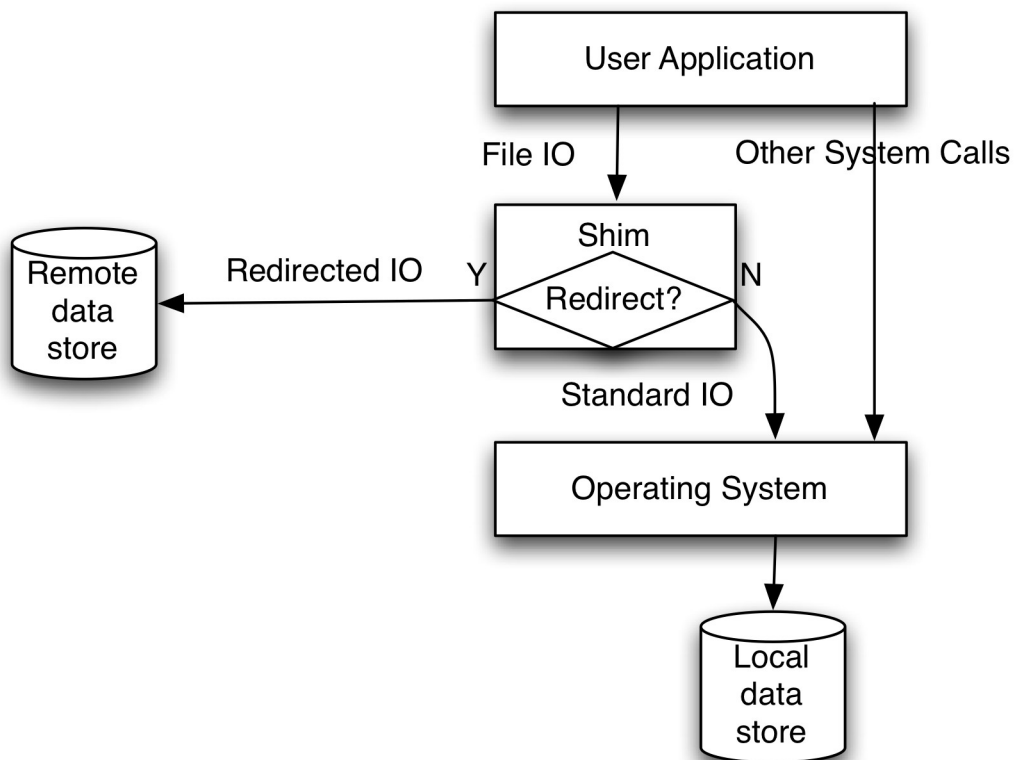


Figure 2.13: The use of a shim can redirect file access in a way that is transparent to the Application.

2.3.3. Communications and Data Security

Communications and data security is another important consideration for production eScience and Grid systems. Although, it is beyond the scope of this project a brief background and discussion is included here for completeness.

For a system to be considered secure then from the data are collected onwards the data needs to be stored in such a way that it is protected from unauthorised access, while still being available on demand for use by its owners. This must include all stages of data activities including data collection, transfer, storage and staging to computational resources. This can be achieved either by only using secure and trusted networks and computers, or through a combination of transport layer and on-disk encryption.

Once the data are available on the data processing systems there may be further need to

move the data around between elements within the computing network. Communications between processing elements needs to be secured as, in the case of the Grid, these computing elements may be agents or services executing at different institutions communicating over public networks.

Within the Grid world the X509 [48] public key encryption standard is used to provide both data security and actor authentication. X509 uses a hierarchy of trust, and digital signatures to establish the framework of authenticated communications, on top of the asymmetric encryption. Particular communications schemes employing X509 such as web services also time-stamp communications to further improve the trust and authentication aspects of the scheme.

By applying this technology to a computational network it is possible to secure communications and data from interception (theft), and man-in-the-middle style attacks. However, this does not address security on the resource where processing will occur. At this level there is reliance on the operating system to provide protection from other users and processes. This is identified as a particular issue when the resource being used for computations is not owned by the researcher or their institution, such as when rented commercial resources are used. This area requires further investigation in order to establish best practise when using third party resources.

2.3.4. Recoverability

Distributed computing systems by definition contain and rely on many components, such as computers, routers, switches, power and communications links. As the complexity of these systems increases so does the probability that a single component will be out of service at a given time. Hence, it is important to address this situation within any distributed processing system. This is especially so in the case of the Grid, which is not only distributed, but may fall under many administrative and technical domains of control. For instance, an execution which utilises the resources of two universities may be affected by a loss of

network connectivity between the two sites. During this time the execution system should respond appropriately to salvage the work it has already performed, and to try to recover the progress when the connection is re-established.

Similarly, the experiment execution should support being paused, unpaused or restarted with minimal loss of progress. This addresses the situation where non-dedicated computing resources are being used to perform the calculations, and these resources need to be reclaimed, or restarted during the period of the execution.

Finally, ungraceful machine crashes should be recoverable, again with minimal loss of progress. In traditional parallel and distributed programming this may be addressed by taking a snapshot of the running system as a security against machine crashes. In the case of a single, monolithic program a complete memory dump of the running application can be taken. This can be slow, and takes large amounts of storage as the physical memory, and scratch disk files all have to be duplicated. In the case of a distributed application, special measures have to be taken to get a checkpoint from all nodes simultaneously.

Task based experiment execution systems, such as Nimrod, approach the problem of recoverability through the use of transactional execution processing. Experiment execution systems contain many distinct tasks. Each task is considered to be atomic, either running to completion, or else does not affecting the global state, allowing recovery to occur by simply re-executing the failed tasks or those that were in progress when the system failed. The implementation must be made transactionally safe by ensuring all the data are located somewhere secure and not affected by system failure before marking the task as complete.

2.4. Computational Platform Review

Identifying the resources which are available to execute workflow helps refine the constraints on the system, and the attributes of the system will the suitability of various scheduling decisions. Network topology, resource configurations and availability of resources are all important considerations. This section discusses these variables and the

framework within which they will be utilised. For the purposes of distributed data mining the resources need to be able to provide a network enabled environment that can execute the required software. This will be determined by the hardware, software and network configuration of the system. From there, the performance, reliability, availability and cost of the resources will be the main selection criteria.

2.4.1. Survey of Distributed Computing Topologies and Services

A distributed computer typically refers to a virtual computer which has many CPUs that do not share the same main memory. This is commonly realised as a set of computing elements, connected by a network of high latency, as compared to computer bus speed. This is in contrast to multiprocessor machines such as multi-core CPUs and Symmetric Multiprocessor (SMP) machines which are comprised of many computational units within the same physical housing, sharing the same main memory (RAM) and hard drives. These units have low latency and high throughput for communications between computing elements.

A second property of distributed computing systems is that their communication latency and throughput can be variable. Unlike a multiprocessor configuration where the system bus is used exclusively for CPU, memory and peripheral communications, distributed computing networks may carry communications for other network devices. However, it should be noted that multiprocessor machines can be part of a distributed computing configuration. A possible set of distributed computing resources is depicted in Figure 2.14 where a practitioner uses commercial, Grid and workstation resources, connected via the internet, local area networks and firewalls.

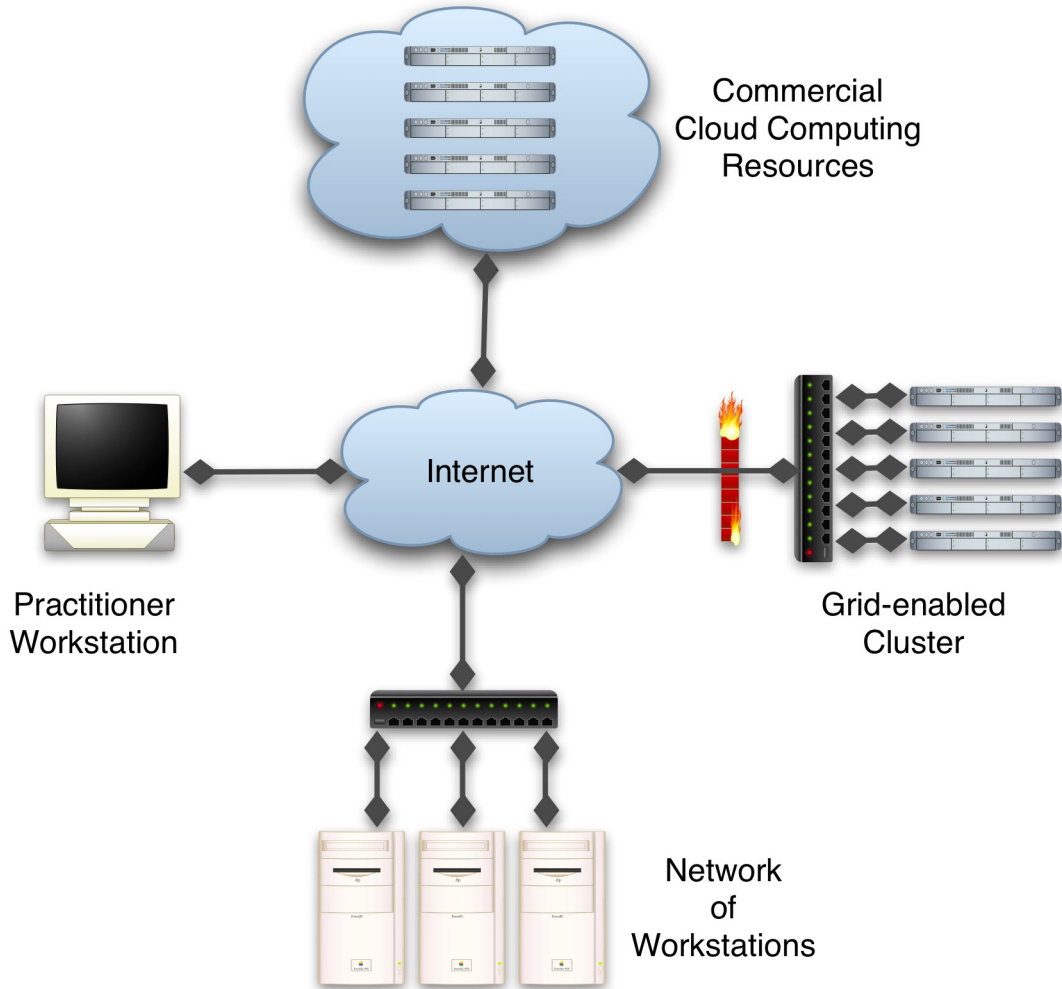


Figure 2.14: Compute resources potentially available to practitioners within an institution. Includes networks of workstations, Grids and clusters, and commercially provided Cloud resources.

Computing clusters are homogeneous collections of computing elements which share a single network for the exclusive use of the cluster. This allows the network to be tuned for the application of the cluster, which improves the throughput and latency fluctuation within this type of distributed computer.

A *widely distributed* compute resource is defined here as any collection of computing resources that are connected by multiple network hops, which may include segments which may need to be shared with other resources. This increases the network latency, increases network jitter, and potentially reduces the available network bandwidth and reliability.

Grid computing is a special case of widely distributed computing as the computing

elements are assumed to be heterogeneous, and are accessed in a utility style, on demand. This means there will be variability of both the computing power and network throughput and latency within the Grid. The computational resources which are exposed using Grid computing may include multiprocessor machines or entire clusters which are made available as single computing resources. These resources are typically owned by virtual or actual organisations, and the Grid is used as a mechanism to share these resources with members of the organisation and their collaborators in a seamless fashion.

Cloud computing is similar to Grid computing as it is exposed as a utility style computing service which is usually accessed via a multi-hop network. It has the additional property of offering reconfigurable computing services, allowing the client to provide the entire operating environment including operating system for execution on the resources. Cloud computing providers exist as commercial entities such as Amazon's EC2 [49] service, and generally offer generic resources on a pay-per-use model. The advantage of this model to researchers is the ability to acquire large amounts of computing power for relatively short periods of time, offering potential cost savings over owning physical hardware.

Table 2.2 shows some available types of distributed computing resources, as well as the accessibility, reliability, cost, and availability attributes of these resources. The trade-offs between the different classes of resource will determine which resources are appropriate for running a particular application.

Resource class	Accessibility	Reliability	Cost	Availability
Desktop workstation	Varies, mainly at night	Low – possibility of unscheduled restarts	Low	Ubiquitous
Grid/Cluster	High availability – Possibly long queues	Medium to high – depending on underlying resources	High	Specialised, owned by some large organisations
Cloud	High, depending on provider	Medium to high – depending on provider	Medium to High	Commercially available

Table 2.2: Different classes of distributed resource, and some attributes of these.

For Grid enabled resources to be utilised they must first be discovered. This can occur by the user providing a list of valid resource addresses, or using a discover service. A *Grid Broker* is one such service that allows resources to be discovered, accessed and orchestrated. Brokers can be provided with various criteria for selecting resources. One such architecture which has been proposed to formalise and leverage cost based computing is Grid Architecture for Computational Economy, GRACE [50]. GRACE proposed a market model, Grid economics, which allows resources to be advertised, and compute time sold in a market system. This requires a centralised Grid Resource Broker (GRB), which allows the user to search, purchase, and access Grid resources (Figure 2.15). The GRB model exemplifies one possible way to monetise existing idle resources, and allows commercial resource providers to help supplement the available pool of resources available for research computing.

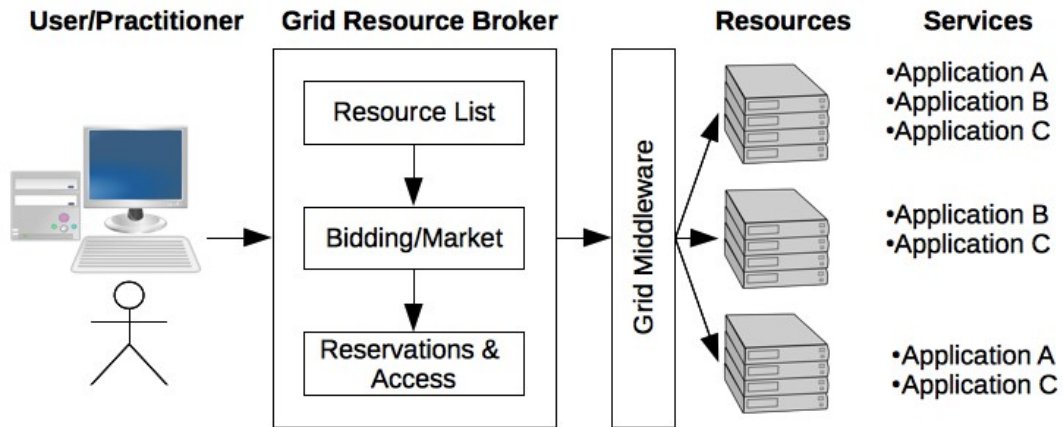


Figure 2.15: Grid Resource Broker provides a service to locate and acquire resources, based on market system.

Data transfer time within a distributed Grid system can be significant where large data sets are copied across wide area networks. This has implication for managing distributed executions, as it may not be possible to centrally store all data sets and access them on demand, rather data may be stored across distributed resources. In the context of Grid computing there is typically a complementary service used to manage data known as the Data Grid [10]. The Data Grid is concerned with all aspects of data management including access, security, metadata and replication. For the purpose of scheduling, the replication feature is of greatest interest. Two typical data grid implementations supporting replication are Globus Toolkit's GridFTP with RLS [51], and Storage Resource Broker [52]. These systems have the same general structure for storing, locating, replicating and retrieving files, but differ in the protocols, implementation and repository structure.

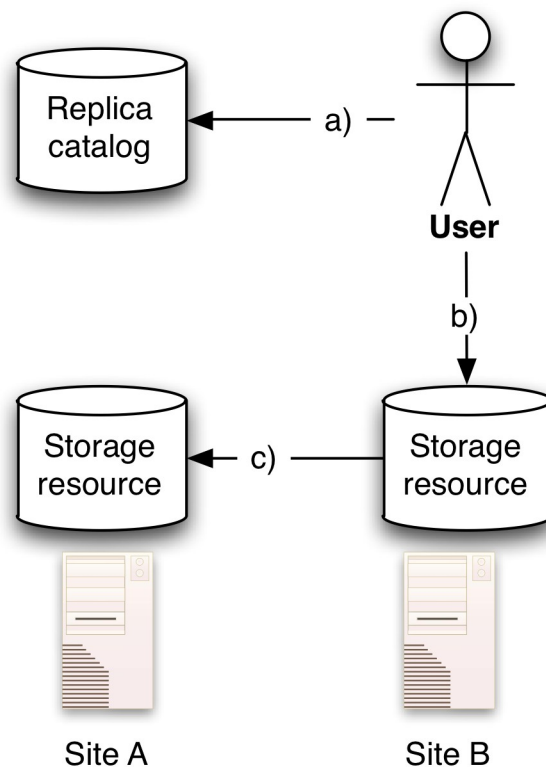


Figure 2.16: Example Data Grid scenario: Multiple resources located next to computational resources, and controlled by a replica catalogue. The User can request data be replicated between storage resources to provide local access to data by compute resources.

The general data grid system includes a replica catalogue and one or more storage resources (Figure 2.16). The replica catalogue stores a reference to all files under its control using a logical name. The logical name is mapped to all the physical instances of the file in the various resources. A client user or service can query the replica catalogue for all available instances of a file using its logical name (a), and then select the most appropriate replica (b). The client can also request new replicas of a file be created (c), which may be done ahead of time.

By utilising a data grid, or similar system with replication capabilities, it is possible to manage the data transfers to complement the task execution. Replication transfers can occur between sites, allowing bandwidth to be fully utilised. To optimally execute a task the speed of the compute resources, the locations of the replicas and the available bandwidth between

sites must all be considered. Often scheduling of data transfers is closely coupled with the scheduling of tasks, but it should be noted that some research has been conducted on decoupling data and task scheduling. Ranganathan, et al. [53] present results which demonstrate it is possible to schedule the replication of popular data sets and use a simple task scheduler to achieve acceptable results.

However, most task schedulers for the Grid are also responsible for data replication scheduling. As evaluated by Hamscher, et al. [54] many popular scheduling algorithms will optimise using a simple heuristic such as minimum execution time or minimum start time. Nimrod/G [45], which is discussed further in section 2.4.4, uses a greedy algorithm to create and dispatch a time and budget constrained scheduler.

Data grids may not always be suitable, as existing data may need to be ingested into the data grid before execution, or for adapters to be provided to allow legacy applications to access the data grid. This additional complexity might not be worthwhile in many instances, unless there is already significant, integrated, data grid infrastructure. However, many of the data grid concepts are worthwhile considering.

2.4.2. Master-Slave Execution Model

The execution model which is used herein is the master-slave paradigm [55]. It was selected due to the simplicity in implementation, and its common adoption amongst other distributed computing projects. In this paradigm there are a number of nodes or agents that are connected by a network (Figure 2.17). One of these nodes is designated as the master node, and is responsible for the overall execution of the experiment. All the other nodes are slave nodes, and they process commands from the master to perform work. This is a very common model for distributed computation as it centralises the complexities of task coordination, and allows the computational resources to be simple and light weight. This technique is used in scientific computing projects such as Nimrod, and volunteer computing projects such as BOINC [56], SETI@Home [57] and Folding@Home [58].

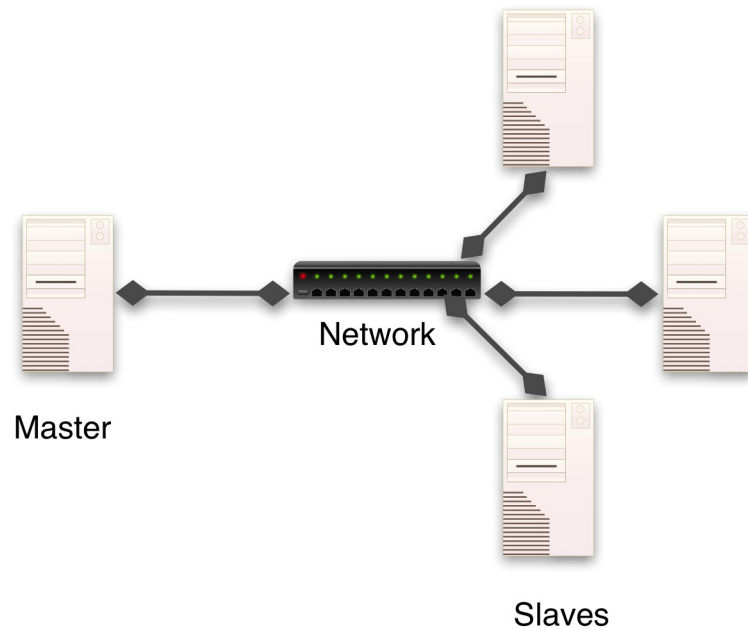


Figure 2.17: Master-slave paradigm: Slave compute resources connected via a network perform tasks requested by a master resource.

A significant amount of work has been previously undertaken looking at various aspects of the master-slave paradigm, and a common technical model is used to describe the particular aspects of a master-slave system. This model covers network access, pre-emption of tasks and execution speed, and is useful for analysing algorithms and comparing performance.

The following discussion of the master-slave paradigm assumes a single port per node, which means that communication is limited to one full bandwidth transfer at any given time. Tasks cannot be preempted, meaning they cannot be stopped and started, or migrated between computational hosts. Other parameters such as the speed of the nodes, and the number of simultaneous tasks which can be executed will vary during the discussion.

The abstract algorithm for the master node in a master-slave system is shown in Figure 2.18. The algorithm involves computing an objective function for each task to determine the best node to execute the task, and then dispatching the scheduled tasks to their assigned resources.

1. For each task:
2. Select most appropriate node from idle pool (Schedule)
3. Assign tasks to nodes (Dispatch)

Figure 2.18: Abstract algorithm for the master node in a master-slave system.

The slave nodes run the counterpart algorithm to the master algorithm, seen in Figure 2.19. In the slave there is a queue of tasks, and the slave will simply execute the tasks as they arrive.

1. While running:
2. Dequeue task
3. Execute task

Figure 2.19: Abstract algorithm for a slave node in a master-slave system.

The assignment of work occurs with the master not waiting for the completion of the individual slaves, rather tracking their state via a pool for idle slave nodes. When the slave receives an assignment it is removed from the idle pool, and when it completes its work it is re-added to the idle pool (Figure 2.20). Commonly, following the initial allocation of tasks, slaves will receive a new allocation immediately after reporting the completion of their task, skipping over the idle pool. Alternatively the scheduling events may occur on specific time intervals.

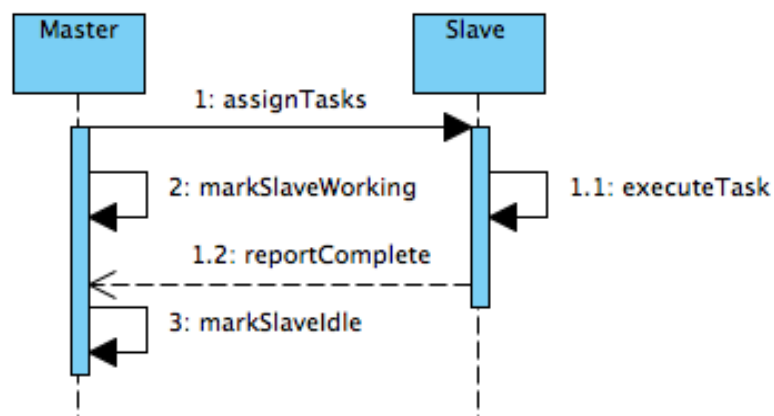


Figure 2.20: Communications between master and slaves during task execution.

2.4.3. Scheduling Notation

This thesis will describe a master-slave system with the following notation, which has

been influenced by Leung, et al. [59] and Pineau, et al. [60] and which is summarised in Table 2.3.

Symbol	Meaning
t_j^R	Release time of task j
t_{ij}^E	Execution time of task j on resource i
t_j^I	Execution length in instructions of task j .
s_i	Speed factor of resource i
t_j^C	Completion time of task j
t_{ki}^T	Transfer time to get data set k to resource i
t_i^W	Wasted time of a resource
b_k	Size of data set k
E_i	The efficiency of using resource i
T_{max}	The makespan, or completion time of the workflow
l_{il}	Bandwidth between resource i and l .
E_i	Efficiency of the use of resource i .

Table 2.3: Summary of notation used to describe task scheduling.

Consider a master-slave system with one master node and I slave resources, $R = \{R_1, \dots, R_I\}$. Each slave resource, R_i , has a CPU speed factor s_i which represents how quickly the resource can perform work. There are K data sets, $D = \{d_1, \dots, d_K\}$, where data set d_k has size b_k and requires t_{ik}^T to be transferred to resource R_i being a function of its size, and available bandwidth, l_{ij} . There are also J task processes, $P = \{P_1, \dots, P_J\}$. Each task process P_j has a set of input data $D_j \subseteq D$, a release time after which it can be scheduled t_j^R , a length, t_j^I , based on the number of instructions required, and an execution time, $t_{ij}^E \propto t_j^I s_i$, on resource R_i . Once executed, a task process has a completion time of t_j^C . The completion time is a combination of the release time, the execution time and whatever data transfer is required.

In line with Pineau *ibid* this thesis adopts the definition of the one-port model as being

“where the master can communicate with a single slave at any time”.

Efficiency of a resource is the amount of time it spends processing, compared to the makespan, or the time it is being held for exclusive use. In the situation where data transfer and computation do not overlap T_{max} can be expressed as its components, where t_i^W is the time a resource R_i wastes being idle, and is not transferring data or executing task processors.

$$E_i = \frac{\sum_j (t_{ij}^E)}{T_{max}} = \frac{\sum_j (t_{ij}^E)}{\sum_j (t_{ij}^E) + \sum_k (t_{ik}^T) + t_i^W} \quad (2.2)$$

The final completion time, time-to-solution or *makespan* of the workflow is the completion time of the final task in the workflow,

$$T_{max} = \max_j (t_j^C) \quad (2.3)$$

Within Grid and commodity computing multi-objective optimisations are also common [61][45][62], and can include both makespan/deadline targets in addition to a budget constraint that is computed based on a cost being applied to computational, storage and network resources.

The scheduling problem is described by which information in t_j^R , t_{ij}^E , s_i and t_j^C is known in advance. In one group of literature [63][64] the scheduling problem has two classes, on-line and off-line. Maheswaran, et al. [64] describes these classes as:

“In the on-line mode, a task is mapped onto a machine as soon as it arrives at the mapper. In batch mode tasks are not mapped onto the machines as soon as they arrive; instead they are collected into a set that is examined for mapping at prescheduled times called mapping events.”

The consequence of this is that Venugopal, et al. [63] describes the scheduling problem they addresses as being in the class of off-line scheduling problems. However, Pineau, et al. [60] and Pinedo [65] have a different framework to handle on-line and off-line classes of problems. They maintain the definition based on the information available at the start of the

workflow, but do not prescribe that on-line scheduling needs to immediately be mapped to a resource, or that rescheduling cannot occur.

Alternatively, Pinedo [65] defines off-line and on-line on a continuum of information, and include a third class called stochastic scheduling. That is, *off-line* has all information for the jobs up-front, *stochastic* where the number of jobs is known but release times, deadlines and runtimes are all unknown and drawn from a random distribution. Then *on-line* scheduling is where no information is known up-front, including the total number of jobs.

Based on the definitions in [65], which appear to be the most authoritative, the problems addressed both in [63] and in this thesis are in fact of the third class of problem, termed *stochastic*. That is, all the jobs are available at the start of the execution, with a release time of 0, but unknown runtimes which are drawn from an unknown distribution.

2.4.4. Task Scheduling in Distributed Computing

To utilise the available compute resources an algorithm is needed which can assign tasks to the resources such that all tasks are executed and any prerequisites of the tasks are fulfilled. This algorithm is known as a scheduling algorithm. It will typically be more sophisticated, and involve other constraints or objective functions, such as minimising the makespan [59]. The scheduling problem is NP-complete [66], meaning that there is no optimal algorithm which will run in polynomial time. For this reason polynomial time heuristics are used to compute schedules.

The scheduling problem has a number of variables which need to be considered, covering the tasks, compute resources and the network. These include task run times, task dependencies, compute resource speed, number of CPUs, and network speeds. These variables may be unknown at the start of the execution, and may vary during the execution.

Data mining workflows may contain many compute intensive tasks and include data dependencies between particular tasks, or groups of tasks. These data dependencies need to be resolved, and the data transported, before dependent tasks can be executed. The size of the

data transfers may vary from gigabytes for some input data, to just kilobytes for models and summary vectors. Scheduling the execution of these tasks on a distributed set of computing resources in an efficient way motivates this investigation into the scheduling of data dependent workflows on distributed and Grid resources.

The scheduling problem in a distributed computing environment has significantly different factors than the scheduling problem within a single machine. Communications latency and bandwidth limitations between computational elements becomes more significant, so does the frequency with which these communication channels are used. This will be a limiting factor for the efficiency and minimum makespan of an execution. To illustrate this, consider a single processor which is executing tasks. In Figure 2.21 the scenario in *a* has a higher bandwidth than the scenario in *b*. The tasks, T, require data, D, which needs to be transferred across a network. As the network bandwidth decreases, the transfer time increases, and this increases the makespan. This means that for a task to be efficiently executed the task run time must dominate the data transfer time, which places a constraint on the types of tasks which can be efficiently executed using distributed computing.

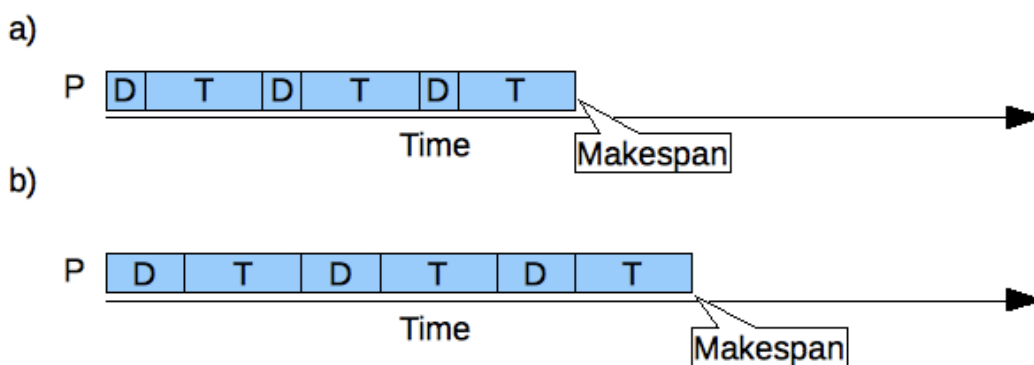


Figure 2.21: Impact of decreasing bandwidth (or increasing data size) on makespan.

In addition to makespan and efficiency, cost is another function which can be optimised by the scheduler. This approach assigns a cost against a resource based on the amount of time the resource is occupied, and would typically reflect the real world ownership costs of the resource. Different resources will have different attributes such as speed and cost, and the

trade-offs between the different classes of resource will determine which resources are appropriate for running a particular application. Research into scheduling systems which utilise budget constraints has been previously conducted by Buyya, et al. [45][50], and others. These systems can utilise market based Grid resource brokers such as GRACE to discover and acquire resources based on performance and cost attributes, as well as other sources of cost information such as user supplied values. For instance, Nimrod allows the practitioner to specify a parameter sweep workflow, supply a list of resources, and then run the experiment with both time and budget constraints. Nimrod will attempt to satisfy both constraints by choosing resources based on cost and performance using a greedy algorithm.

Different distributed and Grid resource environments exist, and are generally characterised by heterogeneity of processor speeds, memory, storage, network speed and network latency. This results in a less than ideal computing platform. Two particular configurations of interest are Grids comprising workstation networks, and hybrid workstation networks supplemented with Grid enabled clusters. Workstation networks are typically highly heterogeneous due to differing requirements of the end user, and different purchase dates of the systems. However they are a low cost computing option. In contrast specialist clusters are more homogeneous as they are typically purchased at the same time with the same specifications, and deliberately matched to maximise parallel performance, making them a higher cost option.

Section 2.1 identified heterogeneous configurations of computational resources and has shown them to be appropriate for experimental cheminformatics work, making better use of existing resources and utilising centralised or Grid resources where required. Individual machines in the network may have one or more CPUs or computational elements, and are assumed to have sufficient local storage (hard drive) space, to hold the data that they are processing. Consideration needs to be given to the real world requirements for network communications between these resources, as security firewalls can and do interfere with direct communications. In particular communications to clusters, where the individual

cluster nodes are typically inaccessible from the wider network can be problematic. In this scenario the cluster “head node” needs to be used to relay messages to the cluster nodes, either using a proxy as is done by Nimrod [45], or by using an agent as presented here.

Master-slave systems requires a resource, known as the master, to host the experiment and coordinate the execution, while one or more slave resources perform the work. An anticipated scenario for workflow execution would be for the practitioner to be using their workstation or high performance computing environment as the master resource, meaning the network connections between the master and the slaves will impact on the workflow execution performance. If required the workflow could also be executed from a dedicated workflow service, removing the workstation from being part of the execution environment. Initially all the data are located on the master resource. Once the workflow begins to be executed the data are copied to the remote slave resources where it is required for processing. This assumption differs from some other data intensive schedulers, such as Venugopal [61], who assume a random distribution of data across participating slave nodes.

Data dependency between computational tasks is a constraint on scheduling options, as the list of candidate tasks for execution will vary depending on the available data. In contrast to other Grid applications like bag-of-tasks, where independent tasks which all relate to one application execute under common constraints, data dependencies brings the possibility of starvation of slaves due to the unavailability of data to execute more tasks. This reduces the throughput of the system, and the efficiency of the execution as machines sit idle.

2.5. Previous Projects

Both Grid and distributed computing, as well as the eScience paradigm, have been maturing over many years and consequentially there are many existing software solutions in the problem space as described in the previous sections. Many are research implements which focus on single issues within the problem space or focus on specific applications, and they do not intend to provide the solution to the specific problems described here. For

instance, the Blast application which is used to compute sequence alignment has been “Grid enabled” many times. For example, Squid [67] is a Perl based project to execute a Blast search in parallel, and SOAP-HT-Blast [68] exposes the same application as a Web service, making it available to be run on the hosting resource from other applications on other machines.

The following section covers several projects which make significant contributions to various aspects of executing data mining workflows on distributed resources. These are Nimrod, Kepler and Taverna, which will be discussed in the context of the problems set out previously. Aside from Grid computing there are also a number of data mining suites that are important to discuss here, such as Rapid Miner, the R language and WEKA. These projects do not attempt to provide a parallel or Grid execution solution, or embrace the eScience paradigm directly, but they do mark an important point in the evolution of thinking and technology along this road.

At this point it is important to distinguish data mining performed on distributed resources, from the field of distributed data mining. The distinction is that distributed data mining is the development of algorithms that can perform data mining on data which is distributed and not moved. That is, the data mining application is sent to the data, typically because the data is too large to transfer. Data mining on distributed resources, on the other hand, is the use of distributed computing resources to perform data mining, typically where the data mining application and the data can be transferred.

2.5.1. Data Mining Services Middleware

Many Grid data mining projects, such as GridMiner (below), the work presented by Perez, et al. [69], and WEKA4WS [70] approach the use of the Grid for data mining by developing service oriented architectures. That is, presenting data mining applications as services that can be invoked remotely by practitioners. The focus of these projects is to resolve architectural and interoperability issues with the services frameworks, and they do

not concern themselves with the low level requirements addressed in this thesis. A service oriented implementation may well present a data mining service that is, in fact, executed in parallel. And, it may be desirable to one day present this thesis project as a service.

2.5.2. GridMiner

GridMiner [71][72] takes a pure Grid approach to providing a data mining environment. The project is built using Globus Grid services, providing a high level BPEL workflow enactment engine to execute experiments. Data access is also provided by Data Grid enabled OGSA-DAI databases. GridMiner relies on parallelism within the Grid service implementations, and does not attempt to expose and exploit parallelism within an experiment. It is a good demonstration of a services, and standards focused project, but it leaves behind many of the legacy data mining application that are a key focus of the requirement for this thesis.

2.5.3. Nimrod

One of the identified required features is the ability to perform parameter sweep operations in parallel. This functionality is the primary focus of the Nimrod software [45]. Nimrod introduced a novel concept of using market based placement of jobs. In setting up the Nimrod workflow the investigator would specify a deadline and a budget for the completion of their application. Then the Nimrod software would communicate with a broker to find appropriate computing resources to use for the execution. Each resource has a price per unit time associated with it, faster computers typically being more expensive per unit time than slower machines. This allows the investigator who needs faster time-to-completion to acquire additional resources to achieve this.

The workflow used by Nimrod is a simple parameter sweep model. The investigator constructs a series of nested loops to iterate over the parameters of interest. Within these loops a job execution operation is specified which substitutes the parameters as command line arguments. Alternatively the parameters can be substituted into a file template.

Nimrod will stage in the data and application, and stage out the output as specified by the investigator. The philosophy of this approach is to enable legacy applications to be executed on Grid resources without modifying the application in question.

Each job in a Nimrod workflow is entirely standalone, and thus there is no onus on determining an order of execution. Each unique parameter combination is stored in a database, and marked as done when the application has successfully completed. This allows Nimrod to be restarted without loss of previously completed work.

Nimrod uses an agent based architecture, with the agents written in C/C++ and the master being a Python application. They communicate over a HTTP communications protocol.

Nimrod does fulfil some of the requirements set out at the beginning of this chapter. However, the workflow language does not allow the nesting or chaining of iterations, so steps such as pre-processing cannot be expressed.

Since the completion of this project Nimrod/K [73] has been developed and reported on. Nimrod/K makes significant progress toward the goals of the project presented in this thesis. Specifically, Nimrod/K utilises the Kepler workflow engine (discussed in the following section) to enable the execution of parameter sweeps across subsections of a workflow, with the tasks executed on distributed resources using the Nimrod system. This allows some of the parameter sweep style components of a data mining workflow to be expressed, and executed in parallel.

2.5.4. Kepler

The Kepler [74][75] scientific workflow system is centred around the desktop, and uses a process workflow paradigm. This means that instead of considering the path that data will take from the source, through transformations, to its sink, Kepler considers each action to be triggered by the completion of the previous. This has many similarities to a data flow system, if signals are considered to be data flows.

Kepler supplies a suite of interesting and useful workflow actors, including logical and

arithmetic operations, reading and writing data and graphics, as well as invoking external programs and services.

Kepler does not fulfil the parallel processing aspect of the requirements because it does not provide an understanding of properties of the data being passed between actors, such as its ability to be parallelised. Also, it is desktop centric, and does not allow for unattended execution. This supports the observation that Kepler is not appropriately oriented toward Grid processing. Kepler does provide Grid service invocation, which lends itself to being used as an overall process co-ordinator, but this alone does not enable parallel Grid processing.

2.5.5. Taverna

Taverna [76] is a group of applications and standards which have arisen from the UK eScience community to facilitate desktop bioinformatics workflows. Its parts include a workflow engine, desktop interface, task enactment services, and tools to wrap existing applications for use within Taverna. Their goal was to create an open source bioinformatics environment which could create, execute, record and share experiment workflows.

Their workflow system is described in an XML language called ScufI (Simple Conceptual Unified Flow language). It consists of three element types: processors which transform a set of inputs into a set of outputs, data links which dictate how data moves between processors, and coordination constraints which control the order of processing when no direct data dependencies exist between processors. The processors which are provided with Taverna allow for the invocation of remote web services, applications wrapped for webservices using Soaplab, nested ScufI workflows, local applications, constant outputs and Talisman [77] which is used for rapidly developing Grid services.

Provenance information is recorded for each task in the workflow. This occurs when the tasks start and finish, and captures the parameter information about the task.

Taverna is strongly oriented towards web service enactment, and thus provides only

coarse grained parallelism for the execution of the workflow. Of course, the webservices may themselves be executed using clusters or other parallel systems, and this complexity is hidden from the user and Taverna.

2.5.6. RapidMiner

RapidMiner 4.3 [78], formally known as Yet Another Machine Learner (YAML) is a Java machine learning framework and workbench. Following the tree based workflow model, RapidMiner allows the practitioner to assemble a workflow comprised of sequential or nested operators, which perform work on a data set. It uses a very strict structure for dataflows between elements, allowing only a single data set to be accessed by each operator, and relying on the operator themselves to perform any filtering required on the data. This design consideration is acceptable in the context of a single computational element, as the data will always be located where the processing is occurring.

For instance, in the case of a cross validation operator will receive the data set from the previous operator and split the data into a test and training set. The cross validation operator is allowed to have two nested operators, the first being a learner, the second being an evaluator. The cross validation operator then passes one subset of the data to the learner, and then passes the resultant model plus the remaining data to the evaluator, which returns a PerformanceVector, containing the performance evaluation.

2.5.7. WEKA

WEKA [3] is a university developed Java data mining package, which provides a suite of classification and regression methods, inside a rich environment. The WEKA workbench provides both a graphical workflow tool, and a more technical interface which allows the investigator to visualise and data mine their data. It includes a comprehensive suite of tree, linear regression and meta-models.

The classes within WEKA can, and have been used in other projects to provide data mining and analysis features. To achieve this, the client application needs to provide an

adapter within the application from the data representation to the WEKA data interface.

There have previously been projects aimed at parallelising complements of WEKA, specifically the cross-validation step. WEKA4WS [70] is a project that produced a version of WEKA that was exposed as a web service, however it is limited to executing a single experiment on each resource. It is possible to start multiple separate experiments on different resources, and have them execute in parallel, but this does not improve the time to solution of any individual experiment.

2.5.8. R Language

The R language [26] has a long history in informatics research. It is a complete programming environment designed for statistical analysis, and based on the ideas and structures of the S+ statistical environment. Like any language it has some fundamental data structures, in this case the matrix, array, list and data frame (a very useful R structure that allows heterogeneous data sets to be addressed, filtered and subsetted). There are other derivations of these structures within R, but these four are representative of the main structures. The first two are quite self-explanatory. Lists are similar to arrays, except they can use strings as their keys. They are equivalent to maps or dictionaries from Java or Python respectively. Lists are very useful for storing complex objects, such as models, which might have multiple sets of parameters, or other homogenous data which needs to be stored in relation to each other.

The data frame is based on a the list data type, but with some restrictions, as each key must contain an array equal to the length of the arrays in the other keys. Another way of looking at this is: a data frame in a matrix where each column can be of different data types. Data frames are very useful in numerical analysis situations, as different attributes in the data set can be of different data types.

The R language lends itself to extension in other languages. Natively it can use Fortran functions, and many core packages are in fact implemented in Fortran, with a set of R

functions available to assemble the data in a form that Fortran can use. This is normally done to improve the speed of the methods, as R is an interpreted language and suffers a performance penalty for this.

Parts of the bioinformatics community have adopted R as their preferred working environment. This has spawned the Bioconductor [79] project, which provides a number of bioinformatics specific libraries.

2.5.9. Summary of Previous Projects

A comparison of the projects discussed above is presented in Table 2.4. The comparisons are made against the three requirement groups presented in Section 2.1.6. It can be seen that the requirements are not fulfilled by any of the systems, but all systems make valuable contributions towards them. In particular WEKA and RapidMiner provide insightful workflow management for data mining, while Taverna and Kepler provide excellent provenance management, and Nimrod provides excellent contributions to the execution of tasks.

The significant gap remains the lack of a mechanism for breaking a workflow into coarse grained tasks. Then the task execution will need to be optimised to accommodate the potentially large data set that will be moved around during the execution. Finally, the provenance collection systems will need to be extended to better cater for the heterogeneous environments that executions will occur in.

Project	eScience and experiment management	Integration and deployment	Distributed execution
GridMiner	<ul style="list-style-type: none"> • Experiment management provided with high level workflows 	<ul style="list-style-type: none"> • Only integrates to web/Grid services • No legacy application support 	<ul style="list-style-type: none"> • Provides support for existing Grid services
Nimrod	<ul style="list-style-type: none"> • Support for simple experiments only, • Experiment is confined to a single directory meaning good repeatability. 	<ul style="list-style-type: none"> • Only supports executables, no specific language integration 	<ul style="list-style-type: none"> • Excellent parameter sweep support
Kepler	<ul style="list-style-type: none"> • Excellent high level experiment representation, • Workflow representation not conducive to parallel execution. 	<ul style="list-style-type: none"> • Support local execution of R and MATLAB • No distributed support for R or MATLAB 	<ul style="list-style-type: none"> • Provides support for existing Grid services, • No support for parallel workflow execution.
Taverna	<ul style="list-style-type: none"> • Excellent high level experiment representation, • Workflow representation not conducive to parallel execution, • Excellent capture of provenance. 	<ul style="list-style-type: none"> • Individual applications can be wrapped as services, which can be consumed. 	<ul style="list-style-type: none"> • Provides support for existing Grid services, • No support for parallel workflow execution.
RapidMiner	<ul style="list-style-type: none"> • Data mining centric workflow language 	<ul style="list-style-type: none"> • Support R and WEKA through extensions 	<ul style="list-style-type: none"> • No support
WEKA	<ul style="list-style-type: none"> • Workflow management is excellent 	<ul style="list-style-type: none"> • No direct support, but can be extended by implementing new plugins 	<ul style="list-style-type: none"> • Limited support for some steps of the execution through 3rd party extensions
R language	<ul style="list-style-type: none"> • Excellent flexibility for experiments, but experiment management is up to practitioner 	<ul style="list-style-type: none"> • Many integration points to WEKA, C, Fortran and many other systems 	<ul style="list-style-type: none"> • Supports simple distributed computing functionality through SNOW module, but the distribution and coordination of the execution is up to the practitioner

Table 2.4: Summary of previous projects covering data mining.

2.6. Summary

Data mining is an important tool in many fields, including cheminformatics, as it is used to extract useful information from large, complex or poorly understood data sets. However, data mining needs to be able to scale up to meet the computational throughput required to analyse larger and more complex data sets. Grid computing offers some remedies for this through scalable computing resources, and distributed compute and data paradigms. The problem for eScience is to bring together the existing software and data systems, and build them into a greater framework which adds value for the practitioner. The requirements to be met in order to achieve this outcome include reliability and recoverability, interoperability, and use of distributed resources.

There have been a number of projects to date which have addressed some of these requirements, either for specific disciplines or in a generic way. In analysing these projects with a view to their application to data mining several inadequacies in the existing approaches have been identified and have informed the development of the workflow language presented here. However, many of the previous projects make substantial contributions towards the desired outcome, and may provide partial solutions that can be adopted or adapted to provide a complete data mining eScience toolset. This will be discussed in the next chapter.

A data mining eScience platform needs to provide a workflow language which allows the practitioner to integrate existing applications in a single workflow, using constructs or syntax that allows the abstract data mining process to be expressed. The platform should then be able to break the workflow into inter-dependant tasks, and schedule these tasks for execution using distributed compute resources.

Chapter Three

3. Distributed workflow, and eScience tool chain

The project described in this thesis, in part, involved the designing of a distributed experiment workflow execution system – here in referred to as the workflow system – to fulfil the requirements for a distributed data mining workflow system as outlined in Chapter 2. This chapter discusses the architecture and design decisions of the workflow system. It covers the workflow language and its artefacts, and their interactions with external systems, as well as the platform for utilising resources and managing data. The details of the algorithms used in the scheduling component will be discussed in Chapter 4. And, a real world implementation of workflow and execution system is demonstrated in Chapter 5.

The system is an implementation of the master-slave distributed computing paradigm, and integrates a data distribution system based on tuple space and data grid principles. The system executes experiments, which encapsulate data sources, executable code and the workflow which describes what is to be done. Each slave resource runs a piece of software called an *agent*, which is remotely controlled by the master in order to orchestrate the workflow execution.

The computational experiment is expressed in an *experiment document* (Figure 3.1) using Extensible Markup Language (XML)[80]. XML is suited to this role as it is an expressive, extensible structured document format that supports validation and is well supported by software libraries in many languages. The experiment document describes *data elements* that describe data sources and sinks. There are various data elements that allow for data files to be processed at various granularities, from sets of files to individual records within files. data are made accessible to be processed by the system via adapters, that make various data sources available through a consistent interface, separating data input requirements from data format requirements. When data are not available on a particular resource it can be replicated (copied) from other resources. Workflow operators are provided by external applications, in

particular R and MATLAB as they are common powerful data mining languages.

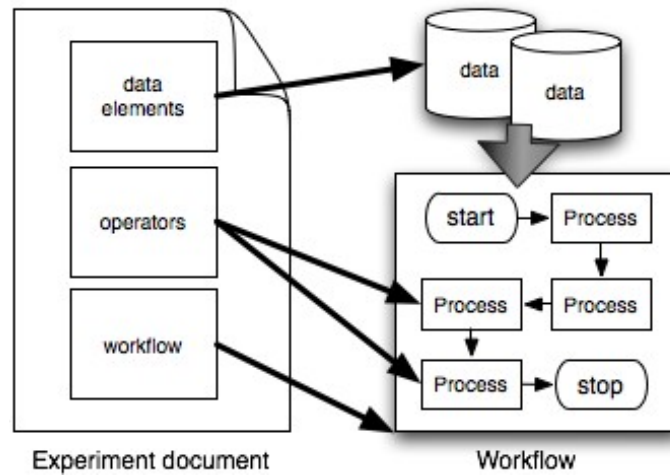


Figure 3.1: Experiment document describe data elements, operators and a workflow, which all map to their counterparts in the eScience system.

The workflow language implements parameter sweep functionality such as is found in Nimrod [45] for nested and sequential iterations, and also allows for data sub-setting to enable more complex processes to be captured. This allows complex experiments to be expressed in the workflow at an abstract level, leaving the details of the algorithm implementation to more mature data mining applications.

Experiment and data provenance information is captured by the agents executing the individual tasks of the workflow. Tasks are annotated with information about execution time, input data, output data, and execution environment. This information is useful in verifying the integrity of an experiment by auditing the execution environment, and detecting anomalies and errors in the intermediate data. In a heterogeneous environment software version mismatches may unexpectedly cause program errors or erroneous data to be produced.

The design of the distributed data mining system is described in detail through this chapter.

3.1. Review of Data Mining Experiment Requirements

As introduced in Chapter 2.1.2, the data mining process involves three key classes of

artefacts to describe an experiment, as they provide sufficient context to reproduce the experiment:

1. The data sets, both producers and consumers;
2. The operations being performed on the data; and
3. The workflow which describes the order of operations.

Requirements for these artefacts in this project are:

1. data sets of various sources and formats must be made available through a consistent interface;
2. Operators must make use of legacy software, including scripts in specialised data mining environments such as R and MATLAB; and
3. Workflow must support:
 1. Nestable iterations;
 2. Invocation of actions with the data exposed by the iterations;
 3. Storage of output from actions;
 4. Chaining of multiple actions in sequence; and
 5. Loops or actions executed in sequence.

In fulfilling these requirements many technical, technology and architectural decisions will need to be made. The most pragmatic decision will typically be made where the choices do not impact on the central research questions and investigation.

3.1.1. General Data Mining Experiment Use Case

The practitioner begins by selecting the data object or objects that are of interest, then the operations which will be used on the data, and subsequently uses these to construct the workflow to fulfil their investigation. For the data mining scenario the construction of the workflow will essentially require setting up appropriate validation patterns, such as 10-fold cross validation, and invoking the operations which are required to be execute the modelling technique. This results in the workflow description replacing the outer loops of the data

processing, with the actual processing being provided by the operators. This mimics the form of a normal procedural programming language, which is beneficial because its form will be familiar to the practitioner, and also because it separates the orchestration details from the algorithmic details.

Collectively the data, operators and workflow will be referred to as the *experiment*, with the series of experiments from which conclusions are drawn and which will be refined over time being referred to as the *investigation*. The practitioner can now execute the experiment locally or on a set of Grid resources, and collect the output data set using it to make decisions about how to proceed in their investigation. This will probably mean manually refining the parameters of operations in the experiment, adding in additional sweeps to determine parameter responses, or changing the actual operations which are being executed. Essentially the process of discovery is made up of number of linear and non-linear components. By analogy if the processes is considered as a control system, the investigator is a non-linear component in the feedback loop.

When the practitioner launches the workflow from their workstation, or other networked machine it is the only node with all the data and workflow information. As remote nodes become available they must be configured, and have data and workflow requests sent to them so they can perform their tasks. There is assumed to be no pre-staging of data or programs in this environment as nodes will have been recently acquired for this task and are assumed to be in a clean state. Nodes can differ in what hardware architecture they use, and what operating system, memory, network, and pre-installed applications they have.

3.2. Experiment Representation

The experiment is represented as an XML document (Figure 3.2) that consists of three top level elements: *data*, *operators* and *workflow*, and two descriptive attributes of the experiment, *name* and *version*. The data elements describe the input and output data; the operators describe the transforms available for use in the workflow; and the workflow

section describes the actual work to be done. While the document is self-contained, it can also be assembled using components from other sources, such as common repositories or libraries of operators and data elements. An example of this would be a data set repository, cataloguing data sets and describing them in XML fragments which can be imported into new experiments.

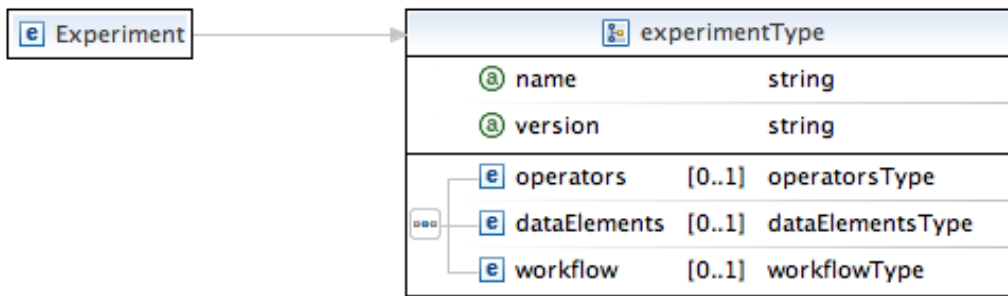


Figure 3.2: Top level elements within the experiment document.

3.2.1. Workflow

A workflow is the process that is to be followed to perform a data mining experiment. As discussed in Chapter 2.2.1 there are a number of forms that workflows can take, including parameter sweep, process workflow, data flow and task lists. It is clear from the data mining template in Chapter 2.1.5 that the data mining process is well suited to the parameter sweep style workflow, as it uses deeply nested loops to perform executions on different combinations of input data. However, many pure parameter sweep workflows have limitations that need to be addressed in a new parameter sweep style language.

Data mining requires that the workflow has the capacity to chain multiple parameter sweep groups together in order to express the different steps of the data mining process. A process workflow would express these requirements using a flow chart with boxes for process steps and arcs for transitions. However, many process workflow implementations do not support iteration in a way that would allow the parallel execution of those loops. For instance, the process workflow language Business Process Execution Language (BPEL) [81] was identified by Slominski [82] as a candidate for scientific workflows, however he notes

that it is limited by not supporting parallel loops. Data flows may be suitable for some data mining applications, however, like process workflows, data flows lack an elegant way to express iteration. Pure parameter sweep languages such as Nimrod and AppLeS[83] only allow a single task template to be applied per parameter sweep, so would require an extension to allow multiple parameter sweep experiments to be chained together. Finally, task lists are unsuitable as they require the practitioner to define each task individually. Although, it should be noted that parameter sweeps can be broken up into task lists, which are then useful for execution.

```

1. For each data set:
2.   for each method:
3.     {method}(in=data set, out=model)
4.     write to models (data set=data
      set,method=method,model=model)
5. for each data set:
6.   for each method:
7.     for each second_method:
8.       for each models[data set=data set,method=method]
9.         {second_method}(in=data
      set,first_model=model,out=second_model)
10.    write to output (data set=data
      set,method=method,second_method=second_method,model=second_model)

```

Figure 3.3: Parameter sweep blocks and subsetting example: Lines 1-4 generate data and write it to a store including variables which indicate the index of the iterators. Lines 5-6 bring those same iterators into scope again, and line 8 uses subsetting to retrieve the data stored in line 4. Method or sub-setting parameter names are shown in italics.

The parameter sweep style was adopted as the basis for the data mining workflow language as it naturally supports the parallel processing of tasks, nesting of loops, and execution of the operators in sequence, additionally it has simple syntax that is straightforward to parse and easy for the practitioner to understand. Two additional pieces of functionality are required to fully accommodate the requirements: multiple parameter sweep blocks and data set sub-setting. Support for multiple parameter sweep blocks there may be more than one top level loop, such as when there is preprocessing in addition to the main process. The constraints on the order of their execution is implied by their data set reading and writing dependencies. The workflow in Figure 3.3 contains two parameter sweep blocks,

the first in lines 1-4, and the second in lines 5-10. Data input and output of the elements within each block will determine which block is executed first.

Data sub-setting allows a parameter sweep to only consider a subset of the tuples in a data set, based on the value of tuple attributes. As an example of the use of sub-setting consider the workflow in Figure 3.3. Its purpose is to test all combinations of data sets, two stages of methods. In order to express this without nesting iterators after operators two groups of nested iterators are used. The first group, lines 1-4, generate the first stage of method models. The second group revisit the same iterators (lines 5-6), retrieve the data stored in the first group using subsetting (line 8), then perform the application of the second method and store the results (lines 9-10).

The workflow representation is designed to be as brief as possible to assist in keeping the separation between the workflow and the data mining operators, and to keep the workflow language general purpose for data mining as practical. The general elements of the workflow are containers which contain *actions* and other *containers*. A container groups other workflow elements while defining the variables that are in scope for them, and an action manipulates the data within its scope by mapping data to the input and output ports of the action. From Figure 3.3, lines 1 and 2 will bring variables from *data set* and *method* into the scope of the action on line 3. The action on line 3 will be invoked with its *in* port mapped to *data set* and its *out* port mapped to *model*. This same data, including the value in *model* will then be available to be written out by the *write* operation on line 4.

The workflow uses a data-centric model which allows the system to maintain a consistent state, including in widely distributed executions. The system requires that data sources be addressable by row and by variable, similar to a database. The individual rows, or tuples in database terminology, can consist of any number of variables and combination of data types. Each variable is a vector of a primitive type, integer, double or string, or it can be a file. Files are a special case of a string and are made available to the operator as a string reference to the file.

The workflow language provides a means of moving through the data sources, processing and transforming the data, and depositing the outputs into the data sinks. These will be referred to as *keywords*, in reference to their similarity to the built in functions in other programming languages. The first keyword is the *data iterator* which is responsible for moving sequentially through the records of a data source, bringing the row into scope. This keyword is the principal point of parallelism as it offers a container which may contain a discrete set of operations. Another similar keyword which defines a set of operations is a *group*, which simply combines operations in a container which cannot be further segmented.

Next the data manipulation operation, *execute* that presents data, which has been brought into scope by the data iterator keyword, to an external operator which manipulates the data and returns an output. The results returned by the operator are then in scope and available to subsequent operators, including *write* keyword which commits data to a data sink.

The data iterator and write keywords in this workflow language are similar to the *in* and *out* keywords in the traditional tuple space distributed computing model Linda [84]. However, they differ in a number of subtle ways, specifically the restricted way in which the data iterator is used to sequentially move through the input space. There will be more discussion of this in section 3.4.1.

```
a)
1. for cv in 1 to 10
2.     x = operation()
3.     write(x=x)

b)
1. for cv in 1 to 10
2.     x = operation()
3. write(x=x)
```

Figure 3.4: Illustration of variable scoping. a) Variable *x* is in scope on line 3. b) Variable *x* is out of scope on line 3.

Scoping of a variable is initialised either by a data iterator, or by an *execute* returning a

new named variable. The variable in the workflow is only inherited by the nested elements in the workflow. For instance, see Figure 3.4a, this write operator will be able to access variable x , while in Figure 3.4b the write operator will not be able to access variable x . Furthermore, all variables are immutable, meaning that once they are set they cannot be changed. The data iterator does not break this rule, as at the end of the data iteration all of its variables become out of scope, and are recreated with new values.

The syntax available in this workflow is slightly more sophisticated than in some other workflow systems, as it provides both data assignment and iteration. Parameter sweep workflows which use textual syntax, such as Nimrod [45] only allow simple data to be used within the workflow. Many graphical languages like Kepler [74] allow you to specify the data flows between operators, but expressing constructs such as iteration can be difficult if possible at all.

By providing the syntax similar to that of a procedural programming language it becomes possible to more easily move the division between the workflow system and the target applications. As long as the invocation of the operator and data handling occur efficiently then there will be minimal or no penalty for doing this, and it will expose more places where an experiment can be parallelised. The process of parallel execution involves breaking the workflow into discrete tasks, with data dependencies; then executing these on the remote machines. The exact process will be described later, but for now the simple model is to increase the number of times which the inner operator block is iterated over.

3.2.1.1. Example workflow

Consider the following data mining process. (Figure 3.5) n is the number of cases in the data set; $random_sample(n, m)$ selects m unique values from the interval 1 to n ; the notation $data\ set[-testset]$ indicates the data set excluding test set samples; $train(data\ set)$ produces a model on the data set; and $predict(model, data\ set)$ predicts the response of the data set inputs using the model.

```
1. for cv in 1 to 1000
2.     testset = random_sample(n,n/2)
3.     model = train(data set[-testset])
4.     y_ = predict(model, data set[testset])
```

Figure 3.5: Example workflow that only has a single level of iteration.

It is clear that there is no data dependency between iterations of the loop defined at 1, however, there are also no viable points for dividing up tasks between multiple processors except by unrolling the outer loop. However, if more was known about the train operation then perhaps there would be opportunities to increase the level of parallelism. From experience it is known that train will typically take the longest time to run as it may need to explore the data set, while predict may not require so much time as it only requires the application of the model developed previously.

If train was a simple, fast method like multiple linear regression then there would be little gained by further parallelisation, as overheads of distributed computing such as network latency would be significant. However, there are many instances where the runtime of the learning method is longer and investment in parallelism is warranted, as is the case with sophisticated methods such as genetic algorithms, random forests and support vector machines, or meta methods such as bagging or boosting.

Consider the boosting method applied to a tree method. Tree methods can take several seconds to minutes to construct a model, depending on the size of the training data. Boosting will require that the training method be applied to many bootstrap sets of the data. All these models are used in the final prediction of the outcome and are averaged to determine the final result. The workflow used previously could be used in this instance to represent the data mining operation, as a boosting method could be considered to be one single operation. However, it is of more use to drill down into this meta method to expose its parallelism (Figure 3.6). By doing this the number of unique tasks is increased from 1000, to $1000 + 1000 * 100 = 1,001,000$. This increased granularity is useful if there are enough available resources to execute with, as it can optimise the total run time.

```

1. for cv in 1 to 1000
2.     testset = random_sample(n, n/2)
3.     for boost in 1 to 100
4.         bootstrap_set = bootstrap(-testset)
5.         model[boost] = train(data set[bootstrap_set])
6.         y_ = predict(model, data set[testset])

```

Figure 3.6: An example workflow which illustrates how exposing inner iterations improves the degree to which a workflow can be executed in parallel.

3.2.2. Operators

Workflow operators define the external processes which produce or transform data. Operator implementations need to be interchangeable to facilitate a plethora of data processing options, with implementations provided for all the legacy applications that need to be supported within the workflow, either using concrete implementations for individual applications, or configurable operators that support a range of similar applications. Implementation details are discussed in Chapter 3.3.1.

```

1.     <callR name="lm">
2.         <script>
3.             <![CDATA[
4.                 data(iris)
5.                 s = dim(iris)[1]
6.                 testset = seq(s)[(itr*s/10):((itr+1)*s/10)]
7.                 m<-lm(Sepal.Length~Sepal.Width, iris[-testset,])
8.                 save(m, file=modelfile)
9.             ]]>
10.        </script>
11.        <parameter name="modelfile" type="file" direction="out"/>
12.        <parameter name="itr" direction="in"/>
13.    </callR>

```

Figure 3.7: Example of an operator declaration for an R script.

Each operator needs to be declared within the operators element of the experiment document. This declaration will give a symbolic name to the operator, and define what execution method will be required, and what input and output data are required. Figure 3.7 shows an example of an operator declaration for an R script. The declaration is comprised of a named element which is specific to the operator type. Enclosed in that element is information about the operator, in this case an inline script, along with declarations of the input and output parameters (sometimes called ports). An element like this will exist for each

operator that is being used in the experiment.

Operators are named objects within an operator list in the experiment that are invoked by references in the workflow using the execute operator. Figure 3.8 shows the abstract invocation process: the workflow engine looks up the operator in the list; the required data are retrieved from the data store; the operator is instantiated, the input data are set, the operator invoked, the output data extracted; then the output data are written to the data store. The data input occurs by passing the variables named in the workflow into the names parameters, and similarly retrieving the output data from the output parameters and write it to the named variables. The input data parameters can be used for configuration, or for passing data.

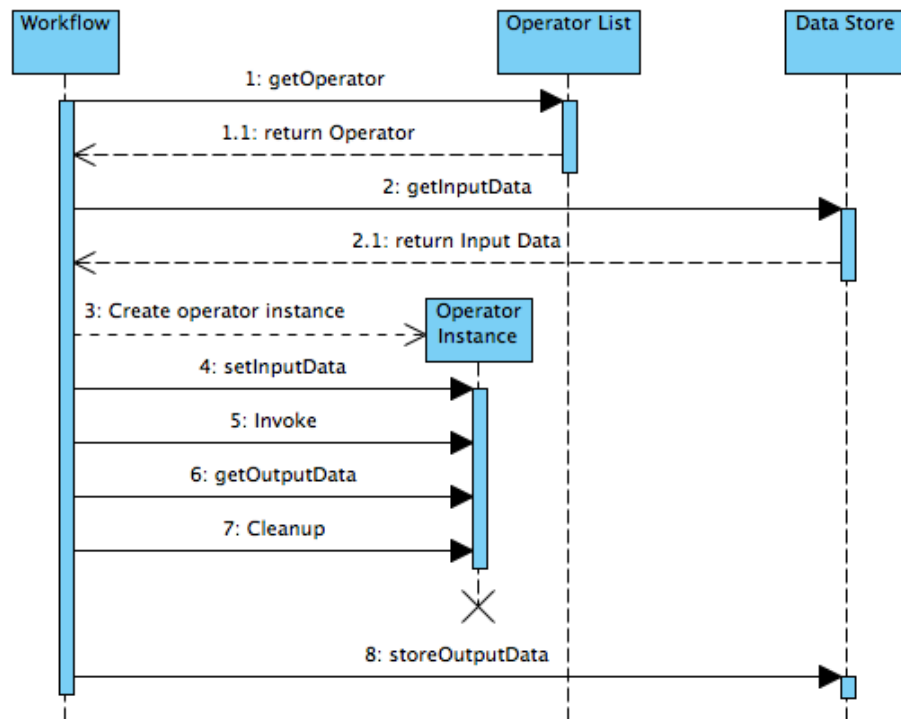


Figure 3.8: An abstract view of the invocation of an operator. Operators are invoked by the workflow, with operators looked up by name from an operator list and input and output data mapped to their ports.

3.2.3. Data

Data access is provided using sources and sinks which adapt common data stores and storage formats. Sources are indexed by row, and sinks are append-only. The intention is to expose as much internal detail about the sources as possible, as this allows the workflow to

take advantage of parallelism by unrolling the iterations over the data. The declaration of the data sources and sinks are similar to operator declaration, and include a unique name, data type, role, and then mappings for each attribute within the data source or sink. Attribute types supported by this implementation is limited to strings, integers, doubles and files. Details about the attribute and adapter implementations will be covered later in this chapter.

```
1. <data name="iris" type="text/csv" role="source"
   uri="file://./data/iris.csv" >
2.   <format type="col" index="0">
3.     <variable name="SepalLength" type="double" value=""
   length="-1"/>
4.   </format>
5.   <format type="col" index="1">
6.     <variable name="SepalWidth" type="double" value=""
   length="-1"/>
7.   </format>
8.   <format type="col" index="2">
9.     <variable name="PetalLength" type="double" value=""
   length="-1"/>
10.  </format>
11.  <format type="col" index="3">
12.    <variable name="PetalWidth" type="double" value=""
   length="-1"/>
13.  </format>
14.  <format type="col" index="4">
15.    <variable name="Species" type="integer" value="" length="-
   1"/>
16.  </format>
17. </data>
```

Figure 3.9: Data source declaration for the Iris flower data set.

Figure 3.9 shows a data source declaration for a data set containing data about the iris plant. It is a CSV file that has 5 column attributes. Each column is named and defined by a column offset and data type. An iterator operating on this source would proceed row by row through the file, presenting a tuple containing SepalLength, SepalWidth, PetalLength, PetalWidth and Species.

3.3. Agent Based Execution System

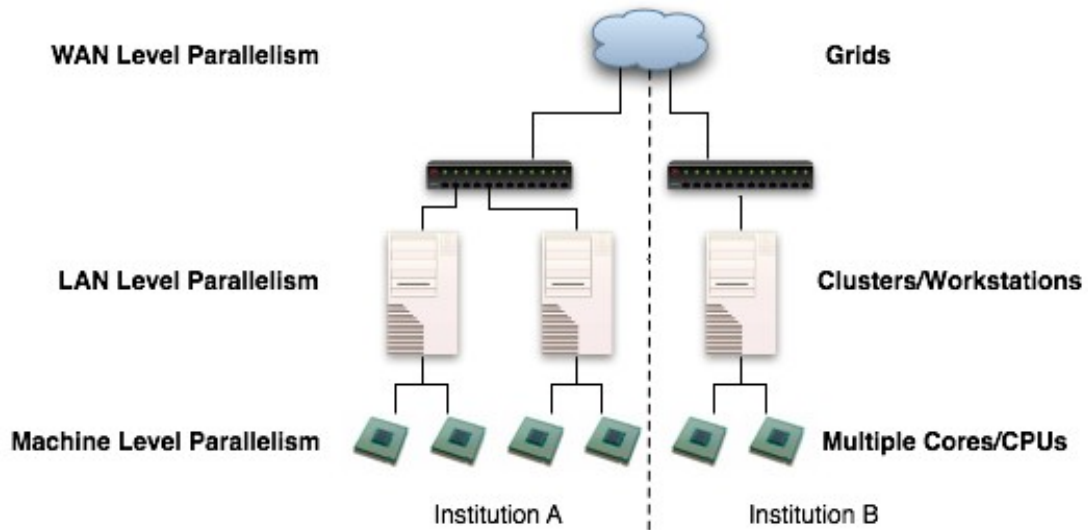


Figure 3.10: Hierarchy of parallelism from CPU level, to LAN level, up to WAN level.

A master-slave system as introduced in Chapter 2.4.2 is used to harness the computational resources available across the heterogeneous environment. It is implemented using an agent model to exploit the system level parallel capabilities of each machine, in addition to the Grid level parallelism across the entire system (Figure 3.10). The central “master” agent holds the workflow being executed, and is responsible for assigning parts of the workflow to the “slave” agents so that the entire workflow is executed. The master agent holds location information about individual tuples and this information is used by its scheduler component to make scheduling decisions. When a slave has more than one processing element it provides this information to the master scheduler, which will assign additional tasks to the slave, which will locally schedule the tasks onto the processors.

The agent model is comprised of a number of components which perform specific tasks. These are the core agent control module, the data management component, and the workflow management component (Figure 3.11). The central component is the agent control component which is responsible for coordinating the construction and control of other components. It is a network accessible module, and exposes the command and control interface discussed in 3.3.2.1. Many commands it provides are related to starting, stopping

and monitoring the Workflow Manager, and the agent as a whole.

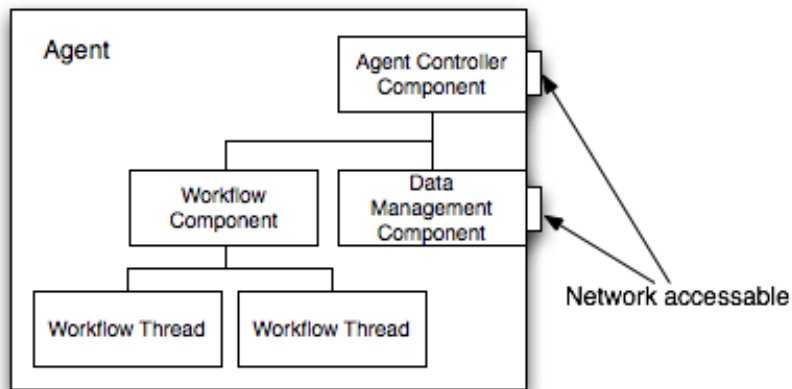


Figure 3.11: Components which exist within an agent: agent controller, workflow component and data management component.

The *Data manager*, like the agent control component, is network accessible. It provides data location, registration and transfer services for the agent, which is typically set up by the control agent, but used by the workflow component. These operations are performed by direct communications between the data modules of different agents, not via the agent controller. This provides a good separation of concerns between the command and control, and the data management aspects of the agent, and also allows for different protocols to be used for the different components.

The *Workflow manager* component is responsible for the local workflow. It maintains a pool of worker threads which execute tasks from the workflow, and is responsible for notifying the master of the current execution state, via the agent control component.

3.3.1. Operator Invocation

Workflow operators are external applications and libraries which the domain practitioner will use to transform or analyse their data. The operators are invoked by the workflow execution engine, as specified by the experiment workflow. To invoke the operator the execution engine requires the operator type, the input and output parameters of the operator, file dependencies, and operator specific parameters such as scripts or invocation command lines. On completion of the operator the execution engine can return the output data and continue the execution of the workflow.

Task execution is atomic, as the operator is not operating on the main copy of the data, and the data are immutable so failure of the operator will not corrupt the global state of system. It is not until the output data are returned to the tuple space that the operator is able to affect the global system state. Also, operators must be purely functional, that is, the operator must not hold any state information inside, or outside itself, other than that passed through its ports.

The operators implemented in this project are Java, R, MATLAB and native library operators. Each operator provides the necessary utilities to transfer data between the workflow stores and the operators, and to invoke and monitor the process. In the case of the two scripting environments, R and MATLAB, this is done by deserialising the data to disk, and then starting the application with a bootstrapping script which reads the data into the application, invokes the users code, and then serialises the output data back to disk to be picked up by the workflow engine.

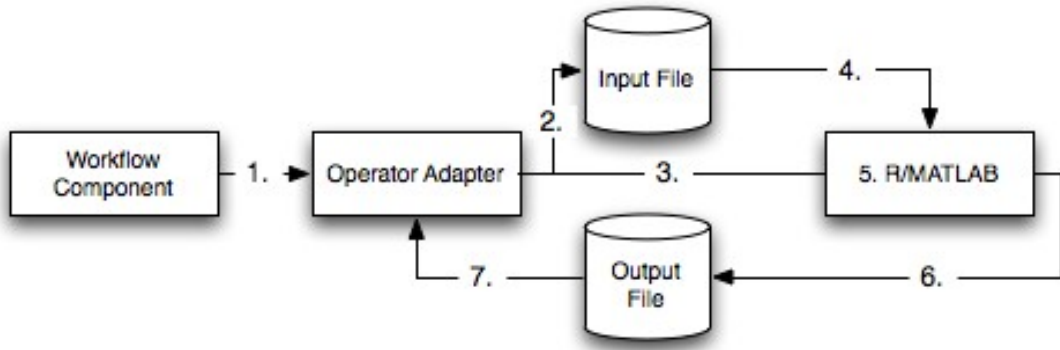


Figure 3.12: Flow chart of events when invoking an operator, including data preparation and retrieval.

Consider the following example in Figure 3.12:

1. Workflow engine invokes the operator adapter, passing the operator, and references to the data in scope;
2. The operator writes the data to a temporary file on disk;
3. The operator invokes the application, R or MATLAB, with a bootstrap script;
4. The application reads the data from the file;
5. The application runs;
6. The application writes the output data to another temporary file on disk, then exits;
7. The operator reads the output file, and places the data in the current workflow scope.

The Java operators directly pass data, without it being staged via files on disk. This has efficiency advantages, as the data can be held in memory once by the agent, which is then accessed by the operator, as opposed to potentially having the data in memory in the agent and in the target application. There may also be speed advantages as data does not have to travel via disk, and there is no additional process startup time. However, using a Java operator does require the data processing functionality required by the practitioner to be implemented in Java, which may not be the case.

Operators support arrays and matrices of primitive data types, including byte arrays and files for accommodating arbitrary structures of data. This is discussed in greater detail in Chapter 3.4. The operator execution itself will be performed by the operator adapter

appropriate to the operators' requirements.

3.3.2. Distributed Execution

Distributed execution is an important feature of the agent execution model, and in this project the distributed execution functionality is implemented using the master-slave paradigm as introduced in Chapter 2.4.2. In this section the services provided by the agents to allow this functionality is discussed. The specifics of how the workflow is split up or segmented into tasks is discussed in Chapter 3.6 and the scheduling algorithm used to place the segments on executing nodes is discussed in Chapter 4.

In a distributed environment task failure may be more probably than in a single machine environment due to the increased complexity of the environment. This design calls for failure recovery is the responsibility of the master agent. Should any slave agent fail, all its incomplete tasks should be returned to the waiting pool for assignment, and any individual task that fails should also be rerun. Repeated failures may indicate an actual issue with the task, so at this point the user will need to be notified. This feature is not part of this implementation.

3.3.2.1. Agent Communications

Basic tasks that are required of the agents are:

1. To have work assigned;
2. Report completion or error conditions; and
3. Resolve data references and place data into the global context.

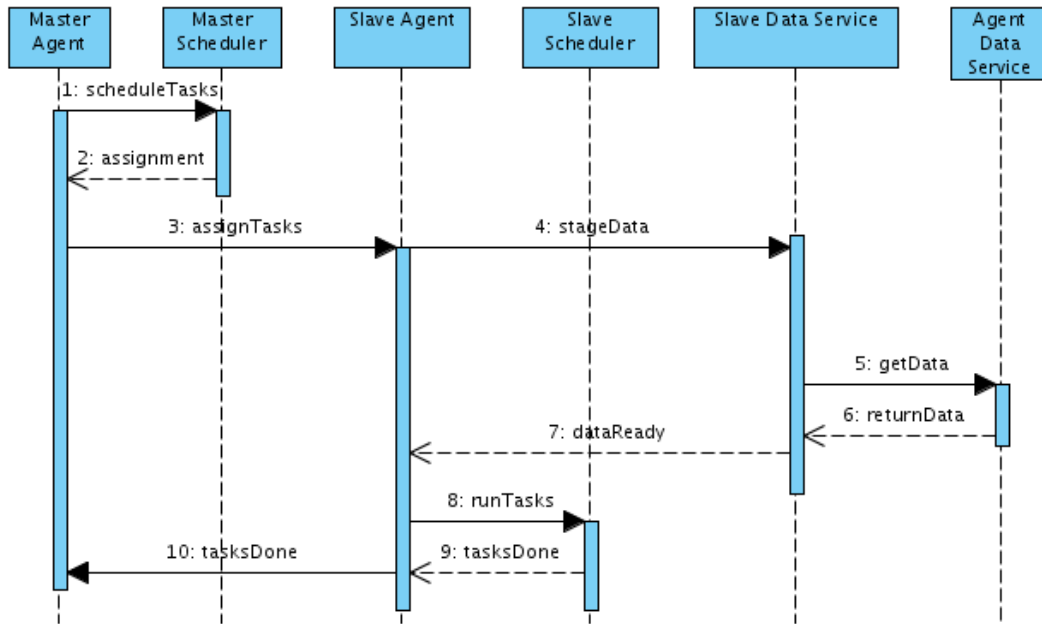


Figure 3.13: Sequence diagram of executing a remote task, including the staging of data.

The most important sequence of interactions is the execution of tasks (Figure 3.13). It involved interactions between the components of the master agent and the slave agent that is to execute task. Communication between agents are achieved through direct network remote procedure calls (RPC) between agents, and can be initiated by either master or slave agents depending on the nature of the request.

3.3.2.2. Agent Communications Protocol

The wire protocol used for agent communications has the following requirements:

1. It needs to be efficient in transporting binary data;
2. It needs to be able to recover from packet loss; and
3. It needs to support encryption.

There are a number of candidate technologies which can fulfil this requirement, and a brief discussion around these is presented in the remainder of this section.

Efficient transport of binary data, requires that the protocol be able to support 8-bit communications, and not just 7-bit ASCII text. This effectively excludes an XML only

approach, as 8-bit data would need to be encoded in base 64 or similar to be transported, which has less than 100% efficiency in encoding. Many XML based communications schemes support a multi-part attachment which transports the binary portion of the message in separate section of the same stream, allowing the versatility of XML for the command and control protocol, and the efficiency of a pure binary protocol for data transport.

Robust communication features like recovery from packet loss are usually provided by using the Transmission Control Protocol (TCP) [85]. This is utilised by a majority of the Internet applications, like HTTP [86] (web browsing), SMTP [87] (email) and XMPP [88] (instant messaging). TCP provides a data stream, a continuous series of bytes, and guarantees that the packets making up the stream will arrive in order and completely. This is achieved using a sequence number in the packets, with the stream receiving endpoint sending acknowledgements (ACKs) for all the packets it receives. If one does not arrive then the sending endpoint will resend the missing packet.

If each packet had to be acknowledged before the next one were sent then there would be at most one packet in flight at any time, putting an upper limit on the data rate of packet size/RTT. To improve this situation TCP has the concept of a window, which is the number of unacknowledged packets which can be outstanding at any time. This allows TCP to fill the network pipeline. The window size uses a slow start algorithm to increase the window size until packet loss begins to occur.

TCP provides a good general purpose approach which is good for most applications, but it does incur an overhead, and the throughput of the stream can be adversely affected by a single lost packet.

The alternative to TCP is the User Defined Protocol (UDP), which does not make any guarantees about delivery, and rather than being connection oriented and providing a stream, UDP delivers individual packets. It is up to the application to determine if there are missing packets, and to assemble them into the appropriate order. This is useful in situations where latency is critical, and where out of order packets can be dealt with in other ways, such as in

voice communications where a single missing packet represents 20 ms of speech. In this case it is better to drop the packet and continue, rather than wait for it and have a noticeable latency in the conversation.

In data communications UDP can be more appropriate than TCP, as it does not matter what order the data arrives in, so long as it is reassembled on disk or in memory in the correct locations. This means that throughput does not need to be jeopardised by unnecessary retransmission of data, or delays in processing data due to out-of-order packets.

The final requirement, support for encryption, is almost universally provided by using a secure sockets layer (SSL), which, once again, is the same process used to secure web transactions (HTTPS) and secure remote shells (SSH). SSL is a mechanism which replaces the standard TCP connection with an encrypted tunnel to the remote host, secured by a suite of encryption algorithms. It can also allow the endpoints to identify themselves using a certificate signed by a trusted third party, typically a certificate authority (CA). SSL can be applied to practically all wire protocols, including those discussed here, and as it is a flexible and well understood approach, it will be used here for this purpose.

The drawback to encryption is that each packet must be encrypted as it is being sent, which leads to an increase in CPU load. If the network connection is sufficiently fast, this overhead can become a bottleneck, preventing the network connection from being fully utilised as data will be held up in the encryption process.

GridFTP [89] is a good option for data transfer, and it fulfils the above requirements for robustness and security. GridFTP provides file transfer capabilities over an X509 secured, authenticated connection. It also provides the ability to stream the data over multiple sockets, which may better utilise network bandwidth. However, in this implementation GridFTP is not supported due to complexity and time constraints, but it would be an ideal candidate in a future implementation.

For this project, Java RMI has been selected as it offers native binary transport of the primitive array data types. RMI also supports encryption and authenticated channels using

custom socket factories, and due to its use of TCP supports low level packet loss recovery. More complex communications failures are the responsibility of the agent to negotiate, with exceptions used to report these failures.

Additional, real-world requirements would include a robust mechanism to recover from network failure, or network slowdown. These are not addressed in this implementation of the system due to time constraints, as they are not the focus of this particular project.

3.4. Data Services to Support Distributed Execution

As proposed previously, the key requirement for a data system to supporting distributed execution using the Grid is the ability for the system to prepare and present this data to individual computing elements in a timely and efficient way. In practise this means reducing the unnecessary transfer of data by only transporting the subset of the data which is required. For instance, if a model being computed only requires a subset of the variables available in the data set, then only the required variables should be sent to the computing element. While this proposition is quite simple, it makes the assumption that the data service understands the source data format, and is capable of performing the actions of segmenting the data set. To achieve this the required operations are defined, and implemented for each of the supported data types. The operations are:

1. Get full row as tuple;
2. Get full column/variable as vector/matrix;
3. Get subset of variables as a data frame; and
4. Get subset of variables and rows as a data frame.

In general data sets are made up of a number of discrete observations, comprised of one or more variables. These variables may be of a primitive type (integer, double, string), and are either single dimensional (vectors) or multidimensional (matrices). Scalar values are represented as vectors of length 1, with categorical data being encoded by either unique string or integer values depending on the source type. This description covers many

interesting data sources including simple observational data, waveforms, and images. Binary data are also supported via files, which are a special case of the string type.

As mentioned in Chapter 3.2.3 data set tuples are immutable, hence no data set update operations are supported. The dataflow is intended to be from a data source to a data sink. A special class of data set exists which can first operate as a data sink, and then become a data source, but no further updates are allowed to it after it becomes a data source. This is to ensure data integrity in a highly parallel environment. This locking is detected at the workflow level, but also enforced at the store level, and can either involve locking the entire store to additions, or, if the retrieval only uses a subset, blocking any further insertions into that subset.

This accessing method is not unlike a database or tuple store, but as it only provides a subset of the functionality of either of these other data models, hence no claim is made that the data accessing method is either a database or tuple space. A database would allow, at a minimum, for updates and deletes to occur to the data set. While a tuple store provides a distributed memory space which can be locked, updated and deleted from. Locking, updating and deleting is not required in the parallel execution system discussed here, due to the restriction on the order and type of data transformations.

Each data set described for use in this system can either be used for input, output or both. Input sources may only be read from, and are thus safe for any order of access, although only whole of set or iterations are supported. Similarly, output only supports the appending of data. Data sets which support both input and output need to be used in a particular way. These types typically start out empty, and are populated by writing to them. Once the writing is concluded then they can be read from either iteratively, or in totality as a matrix or data frame. When the data begins to be read back it means that the data set can no longer be written to, as this would invalidate the operations using the data as an input. Thus, the workflow must complete all the operations which write to a data set before it can begin other operations which reads from that data set. This constraint ensures data consistency, and

simplifies the concurrency requirements.

3.4.1. Agent Data Access Component

The data access is provided as a component or service of the agent platform. At any point where data access is required by the workflow engine or the operator adapters it will occur using this service. The service makes all available data appear as if it is local, which may require it to be fetched from a remote site. The master agent, which is run in the practitioners environment has direct access to all the data sources, so is the primary source of all data. For slave agents to access the data, they need to fetch copies of it from the master agent and hold it in a local cache.

Movement of data into an agent's cache can happen in both a push and pull operations. That is, the master agent can push records into a slave agent's data cache, or slave agents can pull data from any other agents cache. This functionality is akin to the replication capabilities of many data grid packages such as SRB [52], except in this case it is handling a special type of data and addressing scheme, and handling short term data storage with a particular access pattern.

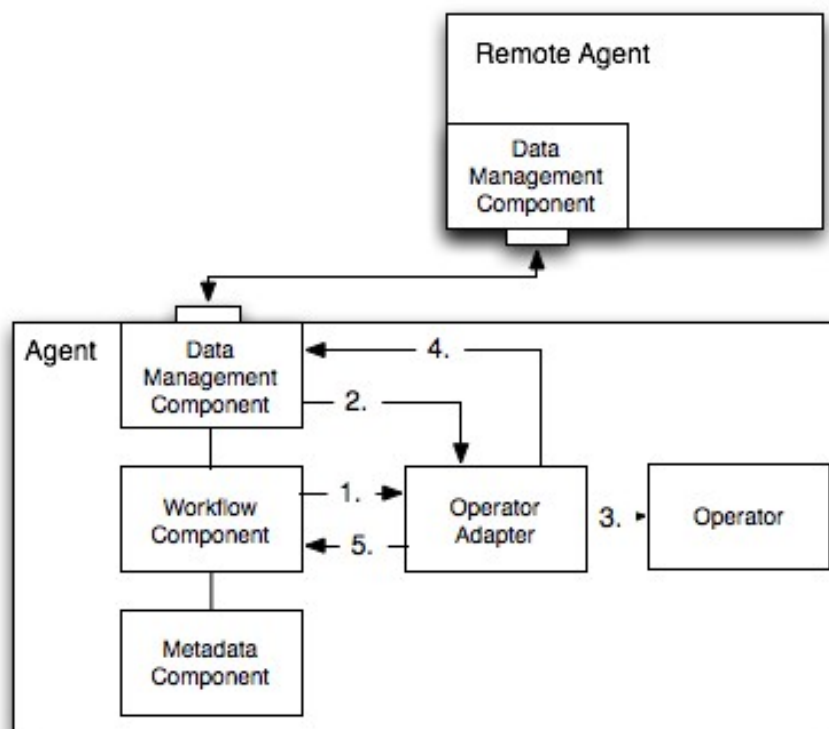


Figure 3.14: Data management component provides a service for operator adapters to access local and remote data.

Scheduling decisions around data movements is the responsibility of the workflow scheduler, so if the data are not locally available it is due to the directions given by this global scheduler. The scheduler can orchestrate the movement of data across slow or laggy links once, and have the agent at the remote end of the link act as a source for other agents in that area of the network. This helps to optimise the start times of the task processing by reducing the congestion around the root agent. This has been looked at by Venugopal [63] where he shows that optimising for the transfer time and data location is complementary to optimising for time-to-solution.

3.4.2. Data Formats

The responsibility of reading and saving the data from its file storage format falls on the data management system, and therefore some understanding of the required data storage formats is required. Typically in cheminformatics data are stored in tabular formats such as comma separated variable, tab separated variable and Excel files. All of these methods of

storing data are trivially read, or converted to a format which can be read. Data types covered by these formats include QSAR descriptors, near-infrared spectra and mass spectrometry, and most observational data. However, this may not be sufficient to cover all DNA-microarrays, and other image based formats. Cases such as these require that specialised readers be written for the data management system, but semantically the formats pose no particular issue as the data are simply a large numerical data set with either many cases and attributes, or with a large value matrix.

Data which is normally expressed as a network or a structure, such as a 3D molecule representation or a gene activation graph may not be so easily adapted into the full functionality of this management system. For instance, there may not be a standard or meaningful way to address a subset of a graph or 3D structure. This does not make these types of data incompatible however. All these formats can have numerical or string representation, graphs being represented as a matrix of connected nodes, or molecules being a series of atoms, bonds and angles. As these subsets cannot be generated within the data management framework it may be more desirable for these types of data to be stored and replicated as normal data files.

Data interchange between the system and the operators is done such that platform specifics such as whether the system is big-endian or little-endian does not matter. Endian format has no implications for text interchange formats. When binary data are read it is transmitted in big-endian format – the native format for the Java Virtual Machine – and when passed to the operator it is read in big-endian format, with any internal conversion taken care of by the receiving operator. Other binary interchange formats such as network common data format (netCDF) [46] enforce platform independence in a similar way.

3.5. Experiment Metadata and Data Provenance

Provenance information, previously introduced in Chapter 2, describes the context and lineage of a process or data. In this discussion the taxonomy proposed by Simmhan, et al.

[37] will be used to describe the type, use and collection method of provenance metadata. Within the context of this distributed workflow system the subjects of the provenance metadata will include the data, the process which manipulates it, and the environment in which the process runs. These different subjects require different attributes to describe them and will be collected by different components in the distributed system. Their data are stored in different documents, with references between them. Metadata about the data uses attributes that describes the data in a generic way, not specific to this system, while the process provenance is specifically refined to the experiment and workflow representations used here.

The workflow itself, which describes all the steps required to produce the output, is an important piece of provenance information as ideally it can be rerun with the verified input data to ideally produce the same, or similar output data. As many data mining techniques utilise stochastic monte carlo methods, the exact same output cannot be produced twice in a row, unless there was a single, central, pseudo-random number generator used to generate all the randomness within the entire network of executing agents. In this case the same starting seed would generate a deterministic sequence of random numbers. However, if the techniques do produce any meaningful information then they will be repeatable, and parameters for accepting two runs as equivalent can be established through domain understanding. For instance, a variation of 0.1% between outputs may be considered acceptable for a given problem.

3.5.1. Process provenance

When executions occur in a heterogeneous environment it becomes critically important that contextual information about the execution of each task is recorded. The information of highest importance will be dependent on the specific operator being used, but some generalisations can be made. Describing the environment in which the workflow is executed is important for verification, as software versions and other hardware and software variations

may introduce unexplained errors into the system. Common information includes CPU and operating system information (Table 3.1).

Attribute name	Use of attribute
Node ID	Indicate which node this data relates to
Host name	Host name of the resource.
IP addresses	List of all interface IP addresses for the resource.
Operating System	The operating system identification string.
CPU info	The make model and speed of the CPU.
CPU count	The number of physical processors (or cores) available to execute processes.
RAM	The amount physical memory available on the resource.

Table 3.1: Common provenance attributes about compute resources.

Each task is also described using provenance metadata with critical information such as input and output file information, execution node and execution environment (Table 3.2) allowing the task environment to be accounted for. Execution information such as system time and peak memory are also collected as diagnostic attributes to help identify abnormal task executions. Each operator type is required to provide its own provenance document type, which, in the cases of R and MATLAB, include software and library versions (Table 3.3). This allows variations between executions to be investigated, and improves repeatability of experiments.

Attribute name	Use of attribute
Node ID	Indicate which node in the network this work was performed on.
System time	The amount of system time used to perform this calculation.
Peak memory	The peak amount of RAM required to perform this operation.
Environment	Map of environment variables.
Input files (Parameter name, size, checksum, input tuple)	Information to verify that data was not changed or corrupted in transport.
Output files (Parameter name, size, checksum, output tuple)	Information to verify that data was not changed or corrupted in transport.

Table 3.2: Common provenance attributes supplied by all operators.

Attribute name	Use of attribute
R version	The version of R in use.
Packages	Name and version of all available packages

Table 3.3: Additional provenance information provided by the R operator.

It may not always be desirable to collect such fine grained provenance information, as it may result in the size of the provenance information exceeding the size of the actual data. In the case where all operators are recording provenance information, then a nested iteration of 10 and 100 items would produce 1000 provenance documents for the innermost operator. This may be excessive, and not useful. Instead it may be more useful to simply collect general provenance information about the set of computing resources which performed the work. Essentially this is collecting workflow level provenance information instead of operator level provenance information.

This provenance collection is implemented in much the same way as is described above. The workflow system operating on the executing agent will prepare a provenance report document which describes the attributes outlined in Table 3.2. Once the operator has executed it is queried by the agent to return the relevant information specific to the operation which has just occurred. The executing agent then merges the pre-execution and post-

execution provenance documents and queues the combined document to be filed against the workflow execution step. This is transferred back to the master agent for collation and permanent storage.

At the conclusion of an experiment there will be a large number of provenance documents associated with the experiment. The provenance documents can be analysed individually and used to develop a better, system level understanding of where the time is spent in the execution. Code execution times for different parts of the parameter space can be utilised to focus optimisation efforts on particularly slow pieces of code. At a higher level the workflow documents collectively provide a detailed description of the process entered into to transform the input data to output data.

3.5.2. Data Provenance

Data provenance involves an input data set document which describes where all the input data sets originate from, and the data elements of the process provenance. For instance, if the original data originates from a public repository, or research publication, then the input data set document will contain the URL to the repository or publication, along with other identifying information such as file size and checksums. This allows the original data set to be located and verified, and also acts as attribution for publication purposes.

The process provenance metadata covered in the previous section describes the operations which occur to create any intermediate and output data. The input and output data elements within the process provenance contain similar attributes to the input data set document, for the purpose of identifying and verifying the data at the beginning and end of each task in the process.

Unless the entire workflow and data are handed to an independent and trusted third party to execute then it is not possible to entirely guarantee that the data outputs presented for check-summing and time-stamping are in fact from that particular run of a workflow, or are not adjusted to present more favourable results. However, the chain of metadata provided

here would also need to be manipulated to ensure that the output was consistent with the process the provenance information described.

Much like the provenance information required by other experimental disciplines, the information here is expected, in good faith, to be accurate. This is not a fraud detection system, its intended use is for documentation and verification, rather than validating the claims of a researcher. The mechanism for verification of claims remains to be for other groups to repeat the experiments performed, and to obtain the same results and conclusions.

3.5.3. Provenance Storage

During workflow execution provenance documents or records are transported to the master agent where they are stored in a flat file structure on disk. At this stage of the execution there is no requirement to be able to query the provenance store, the store at the master is simply appended to.

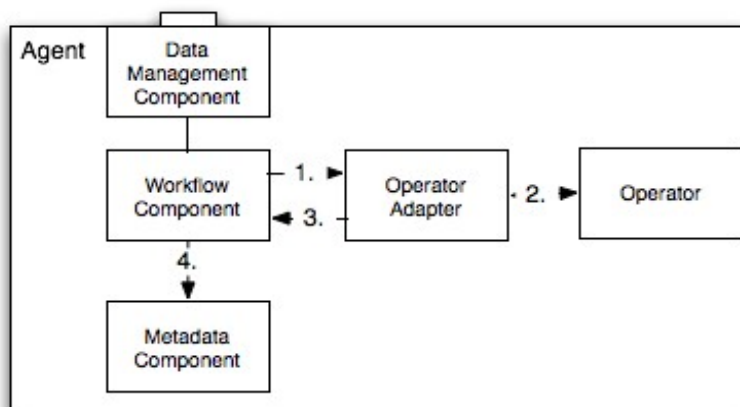


Figure 3.15: Provenance information collected by the operator is stored using the metadata component.

Once the workflow execution is complete the experiment can be registered with a central experiment repository. The central repository used in this case is the ICAT. The ICAT is an experiment metadata system, derived from the CCLRC metadata standard, and implemented by the Australian ResearCH Enabling enviRonment (ARCHER) project [90]. The service provides structured experiment management elements, suitable for experiment based research. Through its web services framework it was extended to store auxiliary XML

documents against the experiment elements. Using this mechanism, the composite workflow provenance XML document is stored in the ICAT, making it available for inspection by people looking to understand this experiment. As well as this, the experiment XML document is optionally stored, as it provides further information relevant to this task. Finally, the data set capacity of the ICAT is used to describe the input and output data sets from the workflow, and link them with other experiments.

3.6. Transforming Syntax Trees into Tasks

The scheduling algorithms presented here schedule individual tasks, or blocks of work onto computational resources. However, experiments constructed by data mining practitioners are in the form of workflows. These workflows, represented as syntax trees, need to be transformed into individual tasks which can be scheduled and executed.

A syntax tree is an abstract representation of a process, which uses syntactical constructs to express actions based on data and operators. For the purpose of expressing data mining workflows a simple syntax has been developed which can capture the iterative data operations which represent many data mining problems. The syntax includes four elements: data iteration, element grouping, operator invocation, and data output. Using these elements the syntax tree encodes data dependency and task grouping into its model through the nesting of operators and outputs within data iterators. This is convenient when constructing the workflow, but needs to be decoded by the scheduler for execution, ensuring that data dependencies are met.

The process of breaking down syntax trees into tasks is referred to as segmenting, and this is a service provided by a *segmenter*. The segmenter is responsible for producing individual tasks, which can then be scheduled for execution by the scheduler (Figure 3.16). To do this the segmenter uses information about the availability of data from the data store, and then generates tasks based on this information.

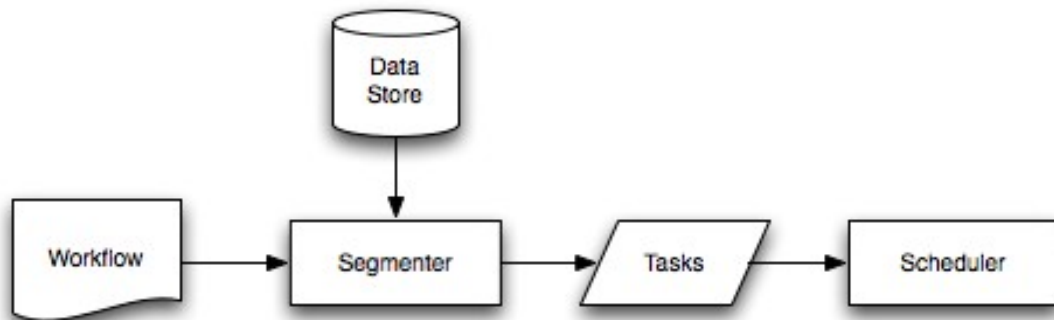


Figure 3.16: Flow chart of components that are required to schedule a workflow. The workflow is segmented into discrete tasks, and these tasks are scheduled for execution.

The syntax tree is decoded following a set of rules which govern data dependency. A workflow contains many individual blocks of iterators. Each block is treated independently by the segmenter. Each block must only contain non-iterator elements in its innermost iterator. If operators do occur in places other than the innermost loop, then it is possible to normalise the workflow by breaking the block into two pieces, and storing the output of the operator in an intermediate store. An exception to this rule occurs if an operator group occurs at a higher level in the syntax tree. An operator group indicates that all the elements within it must be processed as a sequential block, and not decomposed into individual tasks.

Data scoping occurs by data being brought into scope by iterators, or the output of operators, and is passed into nested iterators. All data are immutable, so cannot be updated by operators.

Each workflow block will contain elements that produce and consume data. All of the data sets involved in each case are calculated on the normalised workflow, and used to determine when to release new tasks from the segmenter. To be eligible for decomposing a block must have all its input data sets available, meaning all blocks (and the tasks produced from them) which output to these data sets must be complete. Once this occurs tasks can be produced. All tasks produced with this method are independent of each other and can be scheduled in any order.

When transforming the syntax tree into discrete tasks which are sent to the slaves, it is

possible and in some cases desirable to control the degree to which the nested loops are decomposed into discrete tasks, perhaps by inserting operator groups and changing the order of the iterators. Previous work has been done by Muthuvelu, et al. [91] looking at grouping fine-grained tasks from within a bag of tasks to improve execution efficiency through lower overheads. In the case of a syntax tree it would be possible to achieve a similar outcome by allowing one or more levels of iterator to be fully or partially assigned to a slave. This may also provide benefits to multi-processor systems, giving that slave an opportunity to efficiently utilise all of its computational elements, while minimising the data transfer requirements as common dependencies will already be available on that machine.

The effectiveness of this strategy will vary depending on the amount of time required to execute the computational tasks, the quantity of data required to be transferred, and the number of computational resources available. If tasks are short then it will improve efficiency by grouping tasks like this, as there will be a queue of tasks available to the slave without the need for the slave to consult with the master for each individual task assignment. However, if there are many available slaves and the tasks are long then assigning an entire iterator may cause starvation of some of these slaves.

3.7. Conclusion

This chapter discussed the design of the workflow execution system based on the requirements from the general model of data mining developed previously. The system has three distinct components, workflow, data handing and algorithm execution. These components are specifically designed to leverage distributed computing systems. The data management system is based on the distributed memory paradigm of tuple space; the workflow language allows for discrete tasks to be generated for parallel execution; and the algorithm execution occurs in a purely functional way.

Language integration was demonstrated using R, chosen due to its popularity within the cheminformatics community, with this approach easily extended to Java and MATLAB as

well. This serves to show the flexibility of leveraging of existing languages to provide the domain functionality, leaving the workflow language to be simple and domain agnostic.

The experiment metadata and provenance captured by the system was discussed and linked to the desired usages of this metadata for provenance tracking, experiment integrity, and performance monitoring. Ultimately the experiment workflow needs to be executed on distributed resources, requiring that the workflow be broken into discrete tasks for execution. The workflow language enforces immutable variables, thus it is possible to unroll the loops and create discrete tasks, once all input data sources are finalised.

The execution is performed using an agent based implementation of the master-slave distributed computing paradigm. Slave agents are run on all compute resources, and are controlled by the master to transfer data and execute tasks. The scheduling algorithm utilised by the master is discussed in the next chapter, but is also inter-changeable within this framework. This implementation satisfies the requirements for data mining workflows outlined previously in Chapter 2, and allows expression and execution of many data mining processes within a consistent, high-level framework.

Chapter Four

4. Development and Analysis of a New Task Scheduling Algorithm

The execution of a computational workflow in a distributed environment requires the coordinated assignment of tasks to each of the computational resources, and the movement of input and output data between these resources. Where there are data dependencies between tasks in the workflow the prerequisite data must be available before the dependent task can be executed. The movement of data between resources incurs a time cost as the transfer time is a function of the size of the data and the network conditions between hosts. Ideally, the scheduling of workflow tasks needs to be adaptive to accommodate unequal run times for the individual tasks, resulting from the influence of stochastic algorithms or input data variations, and unequal computational performance of the executing machines.

This chapter defines and addresses the scheduling problem for distributed workflow systems which was introduced in Chapter 2.4.4, and discusses a formal resource and application model for the analysis of this problem. The master-slave paradigm, introduced in Chapter 2.4.2, is adopted as the basis of a solution to the scheduling problem. This resource and application model and the master-slave paradigm are applied to the workflow model from Chapter 3, to produce an algorithm for scheduling data-dependent workflows on distributed computing resources. The objective of the algorithm is to minimise overall execution time, makespan, of the workflow. This new algorithm is then evaluated through simulation in a variety of network scenarios, and compared against other common scheduling algorithms used for scheduling tasks in Grid computing environments.

Many variations of the scheduling problem are known to be NP-Complete[66] or NP-Hard[59], meaning there is no optimal solution which can be computed in polynomial time. On top of this, the scenario proposed for this project has tasks, resources and networks as non-deterministic, making any computed scheduling solution an approximation. When

framed in the master-slave paradigm there are a number of useful heuristics which have been previously proposed with the objective of reducing the total execution time. These include resource sorting algorithms which assign to the fastest processor or communications link first, and greedy heuristics which assign based on shortest total execution time.

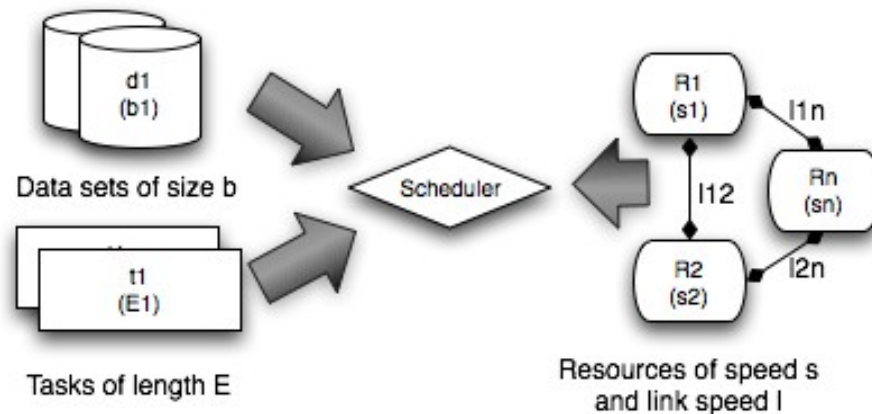


Figure 4.1: Attributes that contribute to scheduling: data, tasks, resources and network link.

Many scheduling algorithms reported in the literature focus on scenarios where one or more of the attributes of job length (E), bandwidth (l), resource speed (s) and data size (b) are homogeneous (Figure 4.1). However, the real world experiments which will be scheduled require all these attributes to be heterogeneous, making the scheduling problem harder. This required the development of a new scheduling heuristic which was robust to these attributes. Through simulation experiments two greedy schedulers were observed to determine their performance. It was observed that they had a tendency to require all the data to end up on all the resources. Further analysis of the greedy algorithm indicated that replication of data, d_i , into the network reduced the cost of further replication d_i , resulting in tasks requiring d_i to be favoured. This led to the development of a new algorithm which uses a relative measure of total estimated run time which aims to increase efficiency by not assigning tasks to resources which require a large amount of data transfer if other resources are already equipped to execute the task. The new scheduling heuristic was again tested in simulation, and a dramatic reduction in total data transfer was observed, as was a significant reduction in total run time in many data intensive scenarios.

4.1. Survey of Master-Slave Scheduling Heuristics

As introduced in Chapter 2.4.3 the scheduling problem has a number of versions; off-line where all attributes of the tasks and resources are known at the start of the execution; and on-line where attributes of the tasks are not known at the start of the execution. The off-line version of the scheduling problem falls into the class of NP-complete [66], and requires an approximation or a heuristic to achieve reasonable results. So it follows that the online version is at least NP-complete. Variables within the system which affect the scheduling problem include task size, resource speed and network speed. When all of these variables are homogeneous the scheduling problem, optimising for makespan, is solved by round robin allocation. Few researchers address the case where all variables are heterogeneous, and often make the assumption that all tasks are equal although may be of unknown size, and then apply a statistical model to estimate task lengths based on previous executions [60][92]

In the scenario where there are homogeneous jobs and a heterogeneous pool of machines there exists a number of quite effective scheduling algorithms [93][94][60]. In this situation it would be possible for the completion time of each task to be estimated, either using a precomputed resource execution rate, or a running average of the execution rate. The problem is then a matter of finding an assignment of tasks to machines so that the end time of the last task on each resource is equal to the result being a minimisation of the makespan. Due to the set-up overhead involved in staging data to each host, tasks are started on machines one at a time, which is another way of stating the one-port model.

The MJF [93] algorithm assigns jobs to the fastest processors first, and achieves good performance. The shortest communications time first (SCTF) [95] algorithm tends to outperform MJF, especially in the turnaround time.

In [60] Pineau, et al. consider homogeneous tasks within the context of a heterogeneous pool of machines assuming a one-port network model, calculating theoretical bounds for makespan, max-flow and sum-flow under on-line and off-line scheduling. They analysed Shortest Remaining Processing Time (SRPT), List Scheduling (LS), Round Robin (RR),

Round Robin c_j (RRC) which orders for shortest communications time, Round Robin w_j (RRP) which orders by shortest execution time, Schedule Last Job First (SLJF), and Schedule Last Job First With Communications (SLJFWC). SLJF and SLJFWC were off-line algorithms, and were modified for on-line scheduling by having them assign tasks to the nodes which would finish them the quickest. The result of their experiment suggested that:

1. In a homogeneous platform all scheduling algorithms had similar performance;
2. Most algorithms have similar performance with homogeneous communications and heterogeneous processors.
3. Communications heterogeneity impacted greatly on the performance of many of the algorithms, with LS and SLJFWC performing better than the other algorithms.

This suggests that communications time is more important than processing speed when scheduling tasks, which is supported by the results of the SCTF algorithm in [94]. One explanation for this outcome is that task starvation occurs while data are delivered to other resources because under the one-port network model only one communications task can occur at a time. This means that resources will sit idle for longer periods if slow communications links are utilised before faster ones. As executions on different resources occur concurrently the selection of slower resources does not have the same limiting affect on other resources being utilised.

4.2. Scheduling of Heterogeneous Tasks in a Heterogeneous Environment

As stated previously the data mining workflows being executed contain heterogeneous tasks, and the target execution environment will be made up of heterogeneous networks and compute resources. Heterogeneity can be dealt with in different ways. For instance, benchmark metrics or execution rates can be measured for compute resources, and these used to compare candidate resources. Similarly, network links can be modelled using instantaneous or aggregate bandwidth measurements that can be utilised for estimating

transfer times. Heterogeneous tasks are harder to optimally schedule as, unlike heterogeneous compute resource or network links, the size of each task is not known in advance and, as each task is only executed once, any information about individual tasks can only contribute to knowledge about that class of task. Information built up about classes of tasks contain assumptions regarding the variation in run time between executions due to normal program execution variability, while information built up about machines and network links may remain more predictable and deterministic over time as these elements can be monitored and modelled in a more robust way.

Despite master-slave computing being a conceptually simple model, there are many variations and assumptions that may impact on the performance of the system in the real world. As covered in Section 2.4.2 the basic master-slave process involves the master scheduler assigning tasks from the workflow to the slaves, and the slaves retrieving the required data, running the task, and returning the result. Underlying this is a number of smaller, specialised schedulers and queues that allow each of these steps to be performed (Figure 4.2). The master will have the task scheduler which is the focus of this chapter, a data scheduler for outbound transfers. While the slaves may have a number of configurations involving a local task scheduler, and incoming data scheduler. This becomes more complicated when there are multiple cores, CPUs or threads running on the slave, as then there are design decisions around whether the computing elements are independent with their own data scheduler (Resource A), or whether they share a common data scheduler (Resource B), keeping in mind that they will be sharing the same network link.

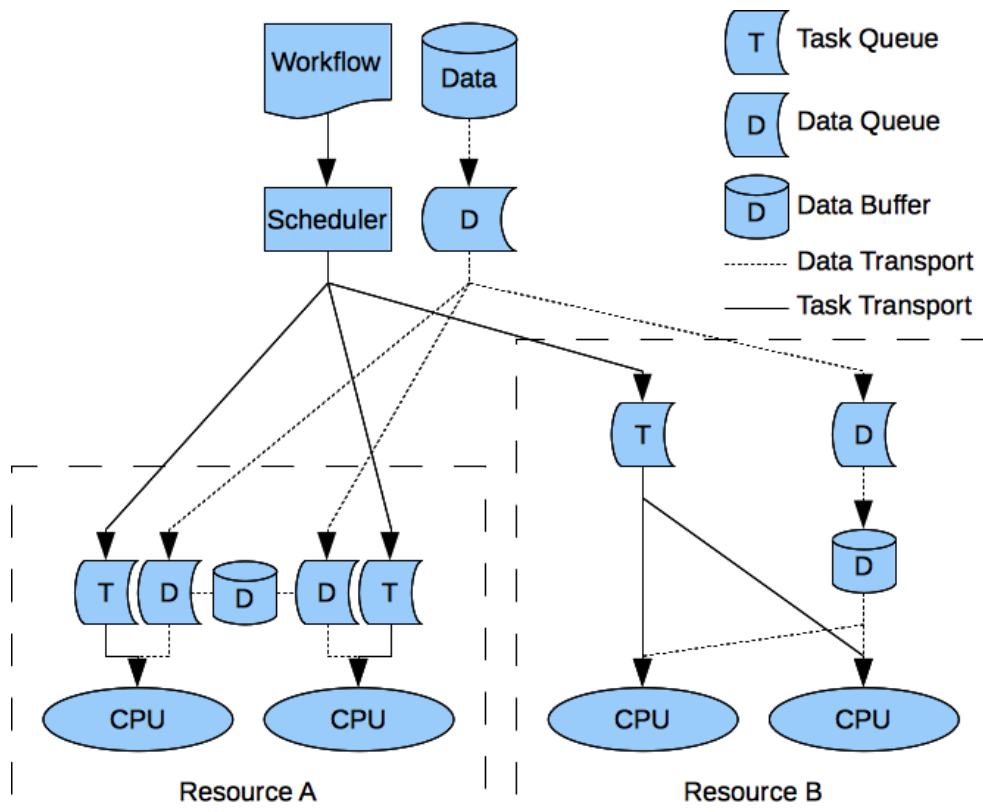


Figure 4.2: In a master-slave system there are a number of specialised queues and scheduling systems that may impact on the performance of an execution.

To illustrate the impact of the different components consider the following scenarios:

1. If two resources have received tasks that require data then they will each request the data from the master. The master's data scheduler must decide how to best transmit the data to the competing resources. Under the one-port model this is achieved using a first in first out (FIFO) queue, meaning that only one resource's request will be processed at a time, and this will occur in the order in which the requests arrived. Alternatively, given that it is possible that the full bandwidth may not be utilised, it may be possible to begin sending data to the second resource at the same time, perhaps applying some priority or bandwidth sharing to these transfers.
2. A resource executing a task may have multiple data files that it requires. These files may be available from multiple locations and have different connecting bandwidths. Again, under the one-port model only one transfer can occur at a

time, so file ordering may be important. But otherwise bandwidth sharing could be applied.

3. A resource with multiple computing elements may have a shared data scheduler, and queued tasks may have overlapping data requirements (Resource B). The order of retrieval of these files will impact on which tasks become available for execution.
4. A resource with multiple computing elements that do not share a data scheduler will have to compete for bandwidth, much like would occur when using other shared resources (Resource A).
5. When a resource has multiple computing elements, they can either share a task queue, and the master can assign enough tasks to that queue to occupy the elements (Resource B), or each element can have its own queue, so the master scheduler must explicitly assign tasks to each computing element (Resource A).

To simplify this investigation and focus on the scheduling of tasks from the master, the one-port model is applied to the master's data scheduler. In resources with multiple computing elements tasks are executed when they become available, ie their data requirements are met, and are assigned to the first available computing element (shared task queue). When tasks are assigned their data requirements are queued in the local FIFO data queue.

At the beginning of an experiment data are located in a single centralised location, ie at the master or practitioners workstation. data are copied to the slave nodes as it is required, which has consequences for the startup time of future tasks. As illustrated in Figure 4.3 data initially located only on the practitioners workstation will be copied to the executing resource 1 after it is assigned a task. Then, when resource 2 is assigned a task it may also get the data from the practitioners workstation, but it may get the data from resource 1, depending on the schedulers decision. This means that any scheduling algorithm will need to track the data locations, and machine interconnects in order to properly evaluate the startup

times of tasks on a given machine. This further adds to the heterogeneity of the system, as the startup time, and the compute time of a task will vary between machines, and will be dependent on at what stage during the execution the task is scheduled.

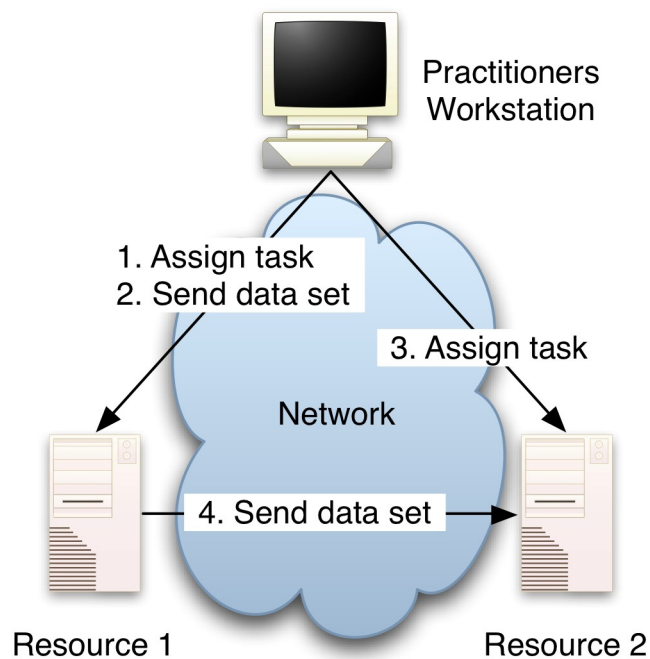


Figure 4.3: Process of data migrating from its initial location. First it is copied from the Practitioners Workstation to Resource 1, then from Resource 1 to Resource 2.

As discussed in Chapter 2.4.3 the efficiency of an execution is a function of the execution time, data transfer time and machine idle time. In the traditional master-slave model the data are unique to the task, and thus the strategy for mapping tasks can become a network link scheduling problem, as well as an execution resource scheduling problem. Data transfer bottlenecks put an upper limit on efficiency, as t^T is non-zero. But as data are shared between tasks, once all machines have copies of all the data t^T goes to zero (0). Therefore, if there is a finite set of data, and an infinite set of tasks the efficiency will tend towards 1. This property is important to appreciate, as it says that an algorithm which optimises the data location problem may outperform one which neglects data transfer for a given number of tasks, but as the number of tasks increases, the relative makespan will converge.

The execution times of the tasks are not known in advance, so a task scheduling strategy

which can adapt to the situation needs to be used. As stated previously it is not possible to preempt or migrate tasks, so allocation decisions either have to be honoured completely, or the task cancelled and rescheduled elsewhere. To accommodate these constraints a number of scheduling heuristics are proposed which accommodate these unknown quantities. Some common functionality is used by all of these algorithms, and includes a task length estimation and transfer time estimation which are used by the heuristics to maintain a constant flow of work to the slaves keeping efficiency high.

Tasks have data input requirements and an execution component. Each task has a class, i.e. an application, and the assumption is that the execution time of each class has a random runtime, drawn from an unknown distribution. This distribution is reconstructed by recoding the task runtime and CPU speed of the executing host when the task completes. The scheduler will be allocated initially based on the assumption that all tasks are equal, and will begin to use the mean task length for the given class of task, based on the timing information returned from the completed tasks.

The primary objective of the allocation algorithm is to minimise the makespan of the execution and to maximise the efficiency of the system usage. The efficiency of a machine's usage will be the ratio of time spent executing tasks, compared to time spent sitting idle or waiting for data. These two metrics may be correlated, as increasing the efficiency of machine usage will decrease the idle and transfer times, thus reducing the makespan. However, it is possible that the transfer time will dominate the execution time to the extent that using a single computational resource would be the optimum solution, which would mean that system efficiency would be minimum. Analysing each computational element in the system will reveal how efficiently each scheduling algorithm utilises the available resources.

In this section three existing scheduling algorithms are presented that address the scheduling problem. One is an on-line scheduler that schedules tasks to resources when resources become available, the second is an offline scheduler which computes an entire

schedule at the start of the execution, updating it as new information becomes available, and the third is another offline scheduler.

```

1. for each tasks  $P_i \in P$ :
2.   for each resource  $R_j \in R$ :
3.      $tC_{ij} = tE_{ij} + tR_j$ 
4. mark all  $P_i \in P$  as unmapped
5. while there are unmapped tasks in  $P$ :
6.    $P_i, R_j =$  find earliest completing task-machine mapping  $\in tC$ 
7.   find the fast data hosts for  $P_i$  on  $R_j$ 
8.   assign  $P_i$  to  $R_j$ 
9.   mark  $P_i$  as mapped
10.  update  $C$ 

```

Figure 4.4: Compute-first matching heuristic. Matches tasks to the resource which will complete the computational component the quickest.

The first scheduler (Figure 4.4) is based on the Compute-first heuristics presented in [61]. Compute-first ignores the data component of the execution and simply maps tasks to resources based on minimising the computational component of the makespan objective function using MinMin greedy selection. Once the task is mapped the fastest available data hosts are used to provide the required data for the execution of the task.

```

1. for each tasks  $P_i \in P$ :
2.   for each resource  $R_j \in R$ :
3.      $tC_{ij} = tE_{ij} + tT_d + tR_j$  (Based on fastest available data
4.     sources)
5. mark all  $p_i \in P$  as unmapped
6. while there are unmapped tasks in  $P$ :
7.    $P_i, R_j =$  find earliest completing task-machine mapping  $\in tC$ 
8.   assign  $P_i$  to  $R_j$ 
9.   mark  $P_i$  as mapped
10.  update  $tC$ 

```

Figure 4.5: Greedy (MinMin) task mapping heuristic.

The second scheduler (Figure 4.5) is another MinMin greedy scheduler, based on those presented in [96] and [61]. This algorithm considers both the estimated execution time and the estimated data transfer time when computing the objective function. Tasks are then mapped to resources based on minimising the objective function.

The third scheduling heuristic (Figure 4.6), Sufferage, maps tasks to machines based on which task will “suffer” the most from not being mapped. Maheswaran, et al. Present this

heuristic in a comparison against min-min and other scheduling heuristics and it is shown to outperform these other heuristics in a simulation of a heterogeneous resource pool and heterogeneous task list. Sufferage was not designed to explicitly handle the data transfer so for the purposes of this experiment the transfer time estimation along with the execution time estimation is used in place of execution time estimation alone. The fundamental principle of Sufferage remains unaffected.

```

1. for each tasks  $P_i \in P$ :
2.   for each resource  $R_j \in R$ :
3.      $t_{C_{ij}} = t_{E_{ij}} + t_{T_d} + t_{R_j}$  (Based on fastest available data
       sources)
4. while there are unmapped tasks in  $P$ :
5.   mark all  $R_i \in R$  as unassigned
6.   for each task  $P_i \in P$ :
7.
8.      $P_i, R_j =$  find earliest completing task-machine mapping  $\in$ 
        $t_C$ 
9.     sufferage = second_earliest  $t_C$  - earliest  $t_C$ 
10.    if  $R_j$  is unassigned:
11.      assign  $P_i$  to  $R_j$ 
12.      mark  $P_i$  as mapped
13.    else:
14.       $P' =$  task already assigned to  $R_j$ 
15.      if sufferage  $P' <$  sufferage  $P_i$ :
16.        unassign  $P'$ 
17.        assign  $P_i$  to  $R_j$ 
18.        mark  $P_i$  as mapped
19.    update  $t_C$ 

```

Figure 4.6: Sufferage mapping heuristic.

4.2.1. Scheduling for Data Distribution

When a workflow contains many large data sets, data distribution becomes a principal factor in efficiently scheduling the tasks onto the computational resources. Congestion and link speeds contribute to a situation where resources wait longer for the required data than they spend processing that data, resulting in an inefficient use of the resource and extending the makespan. This issue is a known problem with very large data projects, but is also important at smaller scales, as the important factor for efficiency is the ratio between data transfer time and compute time.

Data grids are built around Grid computing resources to help alleviate these issues, by increasing the availability of data by efficiently replicating data to storage close to the computational resources which will process them. However, these systems address a different scenario to that which is being presented here. Data Grids exist at the cluster or institutional level, and move data between sites. They typically hold long term data stores, and expect an amount of long term data reuse. The scenario here uses data replication at the machine level, and expects no data to be pre-existing on resources at the beginning of the execution. This difference is an important one with respect to the algorithms and approaches which can be adopted for this situation. For instance, it is important for both the data location and machine capabilities to contribute to scheduling decisions.

```

1. for each tasks  $P_i \in P$ :
2.   for each resource  $R_j \in R$ :
3.      $t_{C_{ij}} = t_{E_{ij}} + t_{T_d} + R_j$  (Based on fastest available data
      sources)
4. mark all  $P_i \in P$  as unmapped
5. for each idle resource  $R_j \in R$ :
6.    $g = \{\}$ 
7.   for each tasks  $P_i \in P$ :
8.     if  $t_{C_{ij}} \leq \min(t_{C_i}) * 1.1$ :
9.       add  $t_{C_{ij}}$  to  $g$ 
10.  if  $g \neq \{\}$ :
11.     $p, r = \min(g)$ 
12.    mark  $p$  as mapped
13.    mark  $r$  as running
14.    update  $tC$ 
15. for each idle resource  $R_j \in R$ :
16.    $g = \{\}$ 
17.   for each task  $P_i \in P$ :
18.     skip the first  $P_i$ 
19.     if  $R_j = P_i$  resource:
20.       add  $t_{C_{ij}}$  to  $g$ 
21.   if  $g \neq \{\}$ :
22.      $p, r = \min(g)$ 
23.     mark  $p$  as mapped
24.     mark  $r$  as running
25.     update  $tC$ 

```

Figure 4.7: Neglected matching heuristic maps tasks to resource when a resource is near, but not strictly, optimal for the task.

The final heuristic presented here, which will be referred to as the *neglected* heuristic (Figure 4.7), is designed to provide efficient execution of tasks, avoiding the situation where some tasks are neglected because their data are not widely available on the computational

resources. Instead of just making greedy selections based on the objective function, the neglected algorithm uses the a second round of mapping to recruit currently non-optimal resources and populate them with the required data. The objective function is arranged such that for each task the executing resources are sorted from shortest makespan to longest. Then, for each available resource, each task is considered for mapping if that mapping is within 10% for all the possible mappings of that task. Then, the shortest makespan is selected from this short-list of tasks. Following this a second round of mapping occurs, this time for each available resource the task is only considered if the resource is not the first choice based on makespan. From this list of tasks the smallest makespan is chosen.

The first step of the Neglected algorithm is similar to the greedy algorithm, but with a relaxed selection criteria, as it assigns tasks to the resource that minimises the task's makespan. However, many tasks may find a small subset of hosts as the most desirable, thus there will be contention for those resources. The relaxed selection criteria, of within 10% of the best available makespan for the task, was selected as it is considered as an acceptable worst case trade-off. Further work could be done to tune the selection of this value. The second step of Neglected attempts to map tasks to resources that are not that's tasks shortest makespan. In a data dominated workflow this would likely be a resource that does not have all the required datasets. By mapping these tasks, the availability of these datasets is increased, in turn increasing the number of potential parallel tasks.

Sufferage is a variation on the greedy algorithm, and attempts to prevent worst case task mapping, but does not attempt to resource starvation due the data distribution overheads. It would be possible to use the Sufferage allocation mechanism for the first step of Neglected, but this investigation is not addressing this variation. Finally, the compute-first algorithm is only concerned with the execution time component of the makespan, and is not data allocation aware at all.

Neglected, Sufferage, greedy and compute-first are compared throughout the remainder of this chapter.

4.3. Validation of systems via simulation frameworks

A systematic approach is needed to evaluate a particular scheduling algorithm or approach, compare it to others or measure its performance. In these situations it is desirable to simulate the entire system of machines and networks which compose the scenario in software. Simulation allows statistical approximations of the characteristics of the components of the system to be used in testing the algorithm of interest. Due to the statistical nature of the approximations, a number of runs of any simulation must be done to build up confidence in their outcomes. Further, comparisons with some real world measurements are also required to help validate that the simulations are representative of a real system.

There are a number of techniques for simulating systems like those discussed. Their properties can be broadly broken down into:

- Static or dynamic – does the system change with time
- Continuous or discrete – is the system constantly changing, or are there distinct events which signify the change
- Deterministic or stochastic – does the system rely on sources of randomness

One of the most widely used techniques in network simulation is discrete event simulation (DES)[97]. DES simulations have a global clock and calendar of future events. At the outset the calendar is populated with user provided events, such as starting jobs or data transfers. Then, as these events are processed by moving the clock to the start time of each event and invoking the specified elements of the simulations. The processing elements may then add their own future events to the calendar, such as the completion of a data transfer based on link conditions and data size. The simulation proceeds in this way until the exit condition is reached, or there are no more future events. Important information can then be obtained from the trace of this execution, such as system responses to certain events, total execution time, etc.

Quite complex scenarios can be simulated, such as complex network topologies with background network and machine load, system and network failures, and addition of new

resources. These real world situations would be very difficult to arrange, orchestrate and then monitor, especially in the repeatable way which is required for this kind of research.

There is already a significant body of work done on simulating scheduling systems, including for Grid style resource allocation and usage. Here we will discuss some of these, and select an existing framework which we will use to test our scheduling algorithms, and compare them with existing contenders.

GridSim [17] is a Grid system simulator which models many layers of the Grid. It is built on SimJava, a discrete event simulation framework for Java. The power of GridSim comes from the number of components modelled. From the bottom level machines and the network links between these, including simulated background traffic, quality of service (QoS), and network routing. On top of this computational resources support advanced reservation, allocation, and workloads based on recorded traces or simulated distributions. More recently data grid elements were provided which include replica catalogues, and data movement management. After evaluating other simulation packages including GSSim[98], DGSim[99] and GangSim[100], GridSim was selected for this project. Contributing factors to this decision included the use of Java which makes integration with the real workflow system components easier, and the complementary work in [45] and [61] which were performed using GridSim.

4.3.1. Calibration of the simulation

The selected simulation software, Grid Sim toolkit version 4.2beta, provides a number of different network models and configurations.. These include packet based and flow (stream) based network models as well as various “background traffic” simulations. The difference between these network models is important, as they may impact on the accuracy of the simulations, and the time it takes to compute them.

When a simulator entity, for instance the master or slave, needs to communicate it does so by using the simulated network. It creates a packet, which has a payload, size and

destination. This packet is sent to the output link of the entity, where it is passed along routers and links until it reaches its destination. When the packet based model is used the single packet is broken up into many smaller packets which can be sent along the network, and these are independently handled by the networking entities. When the final packet is received the destination entity is given the payload.

The flow based networking simply calculates the time the data would take to reach the destination, based on propagation delay and bottleneck bandwidths. When a flow is established the bottleneck bandwidth is reserved along the entire path, with existing flows being adjusted to accommodate these new flows.

Computationally the flow network model is much faster to run than the packet based model, as it does not involve creating many packets out of the data and handling them independently. This is important when simulations involve large data sets. However, the accuracy of the simulations needs to be tested to ensure that the network is performing as expected. To do this a simple network with one router and two entities was set up, using the standard network packet size of 1500 bytes. Data of different sizes is passed between the entities, and the transfer time recorded. This was repeated for the packet based and flow based network, and compared to the ideal transfer times.

Data size (bytes)	Flow network transfer time (s)	Packet network transfer time (s)	Ideal network (10⁷bps) transfer time (s)
10 ³	0.04	0.04	0.001
10 ⁴	0.05	0.10	0.008
10 ⁵	0.12	0.70	0.080
10 ⁶	0.84	6.70	0.800
10 ⁷	8.04	66.70	8.000

Table 4.1: GridSim network model comparison results.

The results of this experiment, seen in Table 4.1, demonstrate that the flow networking is much more accurate than the packet network model. In fact, the packet network model appears to be broken as the transfer time is in error by an order of magnitude. The

consequences of this network layer error would be to inflate the impact of data transfer time, and any experiment using this model would favour algorithms which concentrated more on efficient data placement than efficient computational placement.

Obviously, for the purpose of simulations in this thesis the flow network model will be used, as it is superior to the packet network in both computational time and accuracy. The discrepancies between simulated and ideal are acceptable for these purposes.

4.3.2. Simulation of Allocation Algorithm

Simulation was used to validate the performance of the proposed Neglected algorithm against the competing algorithms. The simulations compare a number of exemplar network topologies and workflows which cover a range of observed and plausible experiments performed in cheminformatics.

The simulation is built using the Grid Sim toolkit, version 4.2beta. The components of the network are modelled using the link and router components provided by Grid Sim, with the topologies under investigation being inspired by actual site layouts. Computing elements are subclasses from the Grid Sim GridResource classes. GridResources represent an abstract computing element, comprised of one or more machines, each with one or more processing elements or CPUs. As these scheduling algorithms are used to assign work to specific CPUs, in this simulation each computing element will only consist of one machine with one or more CPUs.

GridResources are designed to process abstract Grid jobs called Gridlets. A Gridlet is a package of work which takes a certain number of instructions to complete, requires input data of a certain size, and produces further data of another size. This input data are copied from the master before the job starts, and the output data are copied back at the end of the job.

The implemented algorithms are designed to operate on a bag of tasks which, as discussed earlier, is a primitive unrolled representation of the workflow being run. As the

algorithms do not directly deal with complex workflows a class, Workflow, is provided which presents the tasks available for running. It deals with task dependency, updating the list of runnable tasks on the completion of a task. It also provides the dependency information for more advanced algorithms to utilise.

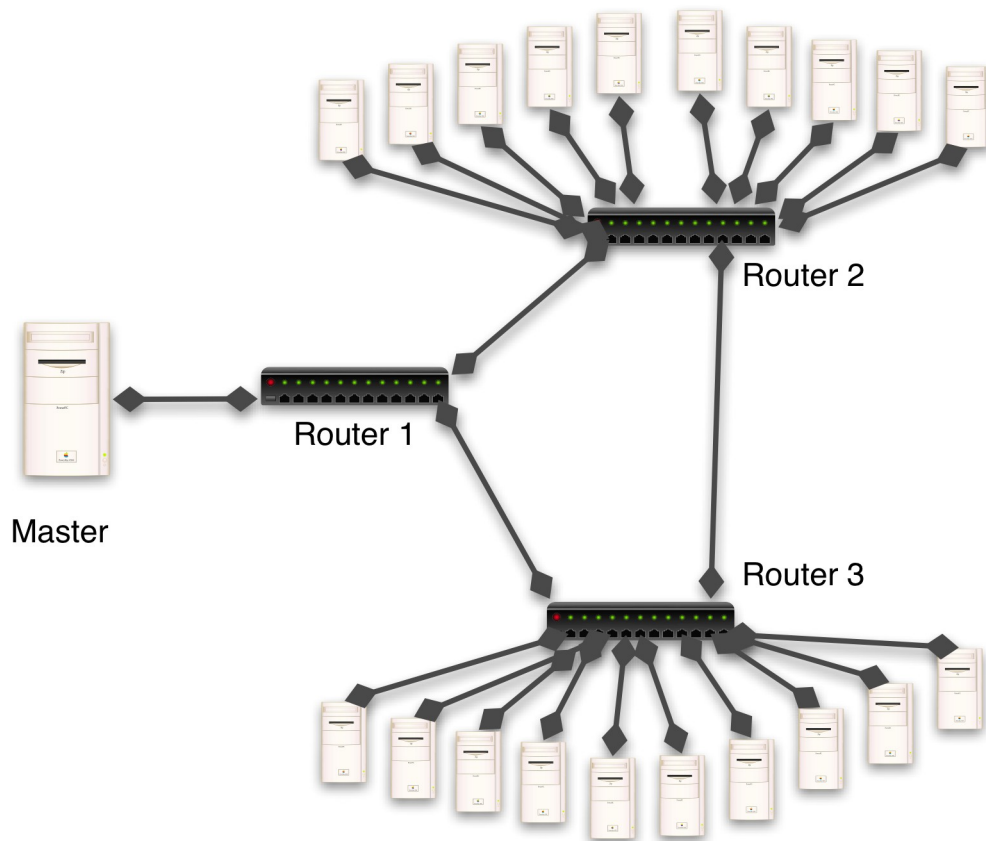


Figure 4.8: Configuration of compute resources connected via a network used in scheduling simulation.

The network scenario (Figure 4.8) considered here will vary the processing speed, number of processors and their position in the network topology. Network links are all 100Mbps with 1500byte packet size, as are the interconnect between routers. For each of the simulation runs the machines are randomly assigned a router, such that there are 10 hosts connected to Router 1, and 5 to each of Router 2 and Router 3. The host configurations are:

Count	CPUs	CPU Speed
2	1	100MIPS
2	1	120MIPS
2	1	140MIPS
2	1	160MIPS
2	1	180MIPS
1	2	100MIPS
1	2	120MIPS
6	2	140MIPS
1	2	160MIPS
1	2	180MIPS

Table 4.2: Host configurations for simulation executions.

The workflow scenario (Figure 4.9) being used to investigate the performance of these scheduling algorithms will consist of 20 data files ranging from 1GB to 20GB (*datafile*). The workflow will be a parameter sweep across those data files, executing a number (5) of *operators* of different classes on each file. The run time of each operator will be different, but will remain constant within the class of operator. An additional parameter *repeat* will be swept as well, which represents the repeated re-execution as occurs in cross validation. This parameter will be varied to control the number of tasks present in the system. Similarly, the *operator* run times will be varied to adjust the ratio of compute time to data transfer time.

```

1.  for each datafile
2.      for each operator
3.          for each repeat
4.              execute operator on datafile

```

Figure 4.9: Workflow used in simulation.

This scenario covers a range of variability within scientific workflows, and is useful for creating a heterogeneous pool of independent tasks. The workflow situations covered by this scenario include situations where the data transfer is expected to dominate the compute time, and vice-versa, and where there are many operators of varying run times operating on the same data.

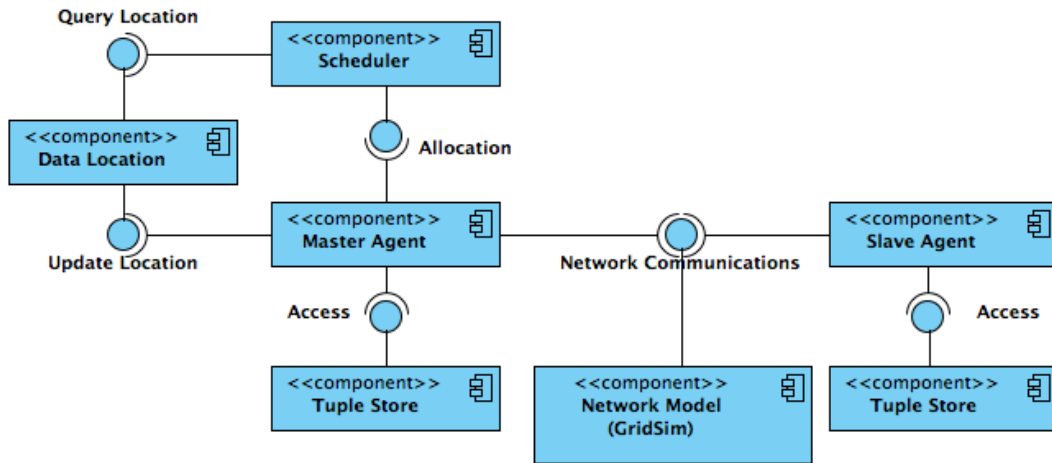


Figure 4.10: Component diagram of simulation experiment software.

The data transfer system used in these simulations is not the data grid functionality within GridSim, rather it is a new data management system implementation which is designed to more naturally fit the tuple data model of the data mining workflow system developed in Chapter 3. Each resource in the network has a tuple store attached to it, which can be queried by other resources to retrieve data. When a tuple is replicated between resources that replication is registered with the Master node's data location service. This allows the Master to quickly access the locations of all the data within the system and allows the scheduling algorithms to use this information to estimate data transfer times. The network information is mapped out using the GridSim's ping packets which include information about the bottleneck bandwidth.

When a Slave is assigned a task, the data requirements are handed to the local tuple store to retrieve. The task allocation specifies the remote stores to be used when retrieving data, as it is the responsibility of the scheduling algorithm to determine the best replica sources. Once all the data are available on the slave resource the task is passed to the Slave Resource's local scheduler to execute in space share mode, which process tasks using a first come first served model.

Once tasks are complete the Slave signals the Master that the task has completed, returning task metadata such as the amount of CPU time the task required to execute. The

scheduler can then use this information to build a model of the time each class of task takes to run. It also marks this task as complete, consulting the workflow segmenter if any new tasks can be released, and then invoking another iteration of the workflow scheduler.

4.4. Experimental Results

The four scheduling strategies discussed in Section 4.2. were evaluated using simulations. The experiment recorded the makespan, system efficiency, and final data distributions across the parameter space. From the data distribution it is possible to calculate the total data transferred, and present this as a percentage of the worst case scenario where all data are transferred to all nodes, termed transfer saturation. The fixed parameters are the data file sizes and locations, the machine speeds, and the network. The variable parameters are the lengths of the operations, which are 1, 2, 10 and 100 times the base operators and number of iterations over each data set-operator pair, which are 10, 25, 50 and 100. This translates into the total number of tasks in the workflow, with each datafile requiring the same number of times, for the same average length of time. The locations of the machines in the network were permuted for the 20 runs of each scenario to randomise for small network variations and machine ordering.

The normalised values in Table 4.3 show the makespan of each of the four scheduling algorithms for variations in the number of tasks, with the four different operator run times and different numbers of data files. The compute-first algorithm and the greedy algorithm both increase in relative makespan as operator runtime or number of tasks increases, with there being little difference between the two. This observation agrees with the observations made in [61], where their greedy algorithm only slightly outperformed compute-first. Sufferage performs much better than compute-first and greedy over the same parameter space, particularly in the data dominated scenario. The performance of the Neglected algorithm is similar to Sufferage, but it particularly out performs Sufferage when there are less tasks to average out poor allocations.

Op. Length	Tasks	Files	Compute-first	Greedy	Neglected	Suffrage	
x1	25	1	1.00	1.07	1.05	1.11	
		10	18.28	17.43	9.16	10.67	
		20	49.85	40.27	17.24	36.13	
	50	1	1.00	1.03	0.83	1.04	
		10	17.21	19.04	8.72	7.23	
		20	53.65	37.46	15.60	28.01	
	75	1	1.00	1.06	0.97	1.05	
		10	14.82	19.28	9.09	8.11	
		20	45.61	37.91	17.12	19.63	
	100	1	1.00	0.98	0.87	0.97	
		10	12.39	16.21	8.55	7.80	
		20	35.19	33.88	16.38	15.06	
	x5	25	1	1.00	1.18	1.00	1.19
			10	7.89	8.88	7.42	7.13
			20	18.17	20.48	13.97	14.91
50		1	1.00	1.05	0.72	1.06	
		10	6.38	7.51	5.99	5.88	
		20	13.16	14.76	11.60	11.38	
75		1	1.00	1.11	0.85	1.08	
		10	7.62	8.45	7.22	7.13	
		20	15.82	16.77	14.13	13.89	
100		1	1.00	1.03	1.00	0.99	
		10	8.21	8.94	7.84	7.73	
		20	16.87	17.69	15.47	15.28	
x10		25	1	1.00	1.26	1.00	1.22
			10	7.88	8.21	7.30	7.72
			20	15.06	15.08	14.06	14.22
	50	1	1.00	1.10	0.69	1.13	
		10	6.11	6.19	5.78	5.94	
		20	11.98	11.90	11.40	11.34	
	75	1	1.00	0.97	0.75	1.01	
		10	6.59	6.63	6.31	6.43	
		20	13.02	12.84	12.54	12.48	
	100	1	1.00	1.03	0.99	1.02	
		10	8.01	8.08	7.82	7.80	
		20	15.87	15.83	15.47	15.49	
	x100	25	1	1.00	1.26	1.00	1.31
			10	8.31	8.34	7.66	8.31
			20	15.42	15.61	14.51	15.47
50		1	1.00	1.13	0.67	1.10	
		10	6.05	6.14	5.68	6.07	
		20	11.57	11.43	11.32	11.40	
75		1	1.00	1.02	0.77	1.01	
		10	6.79	6.75	6.52	6.76	
		20	13.25	13.17	12.96	13.15	
100		1	1.00	1.00	0.95	0.99	
		10	7.83	7.81	7.71	7.77	
		20	15.44	15.41	15.25	15.35	

Table 4.3: Normalised makespan for varying operator run times and repeat values.

A second observation is that the compute-first mapping is outperformed by the greedy when there are only a small number of data dominated tasks, but as the tasks become more CPU dominated, or the number of tasks increases, the compute-first mapping performs better.

To understand the differences between these algorithms the final data distributions were analysed and plotted in Table 4.5 and Table 4.6. These diagrams plot the data sets as rows, and machines as columns, where a solid box indicates the presence of the data set on that machine. The patterns shown are typical examples taken from a single execution of the simulation. It can be seen that as the number of tasks increases the compute-first and greedy algorithms migrate all data sets to all hosts. In contrast, the Neglected algorithm avoids unnecessary replication of data sets, preferring to extend the availability of neglected data sets rather than to focus on data sets which are already being processed elsewhere. This means that all the tasks are completed with a reduced need to transfer data. This is also expressed in Table 4.4 as transfer saturation – the percentage of data (in bytes) that has ended up being copied to each node.

This behaviour is explained by the way tasks are assigned to resources. Under Neglected a resource is assigned tasks P_i if that task can be completed within a given margin of the best resource for running P_i . In a data dominated scenario t_i^C will be heavily influenced by t_{ij}^T , meaning resources will favour tasks which require the least relative transfer time. This will include tasks that have most of the required data, or tasks for which there are no resources with the required data.

Op. Length	Tasks	Files	Compute-first	Greedy	Neglected	Sufferage	
x1	25	1	82%	81%	83%	80%	
		10	55%	56%	37%	40%	
		20	54%	45%	18%	42%	
	50	1	100%	100%	100%	100%	
		10	98%	82%	44%	46%	
		20	99%	64%	19%	53%	
	75	1	100%	100%	100%	100%	
		10	100%	93%	46%	49%	
		20	100%	72%	21%	50%	
	100	1	100%	100%	100%	100%	
		10	100%	98%	47%	50%	
		20	100%	78%	21%	44%	
	x5	25	1	82%	81%	82%	79%
			10	68%	65%	41%	45%
			20	66%	58%	18%	53%
50		1	100%	100%	100%	100%	
		10	98%	90%	47%	52%	
		20	97%	86%	20%	59%	
75		1	100%	100%	100%	100%	
		10	100%	95%	48%	53%	
		20	100%	94%	21%	61%	
100		1	100%	100%	100%	100%	
		10	100%	97%	50%	52%	
		20	100%	97%	21%	62%	
x10		25	1	82%	80%	82%	82%
			10	71%	66%	41%	47%
			20	71%	68%	17%	52%
	50	1	100%	100%	100%	100%	
		10	97%	87%	48%	56%	
		20	97%	86%	21%	64%	
	75	1	100%	100%	100%	100%	
		10	100%	93%	49%	57%	
		20	100%	91%	23%	66%	
	100	1	100%	100%	100%	100%	
		10	100%	96%	52%	56%	
		20	100%	94%	24%	67%	
	x100	25	1	82%	82%	83%	82%
			10	68%	66%	42%	50%
			20	71%	68%	17%	54%
50		1	100%	100%	100%	100%	
		10	97%	87%	47%	55%	
		20	97%	84%	21%	64%	
75		1	100%	100%	100%	100%	
		10	100%	93%	49%	54%	
		20	100%	91%	22%	65%	
100		1	100%	100%	100%	100%	
		10	100%	96%	52%	55%	
		20	100%	93%	23%	66%	

Table 4.4: Experiment data transfer saturation for the four scheduling algorithms, across the parameter space of operator size, task size and file count.

	Compute-first	Greedy	Neglected	Suffrage
25				
50				

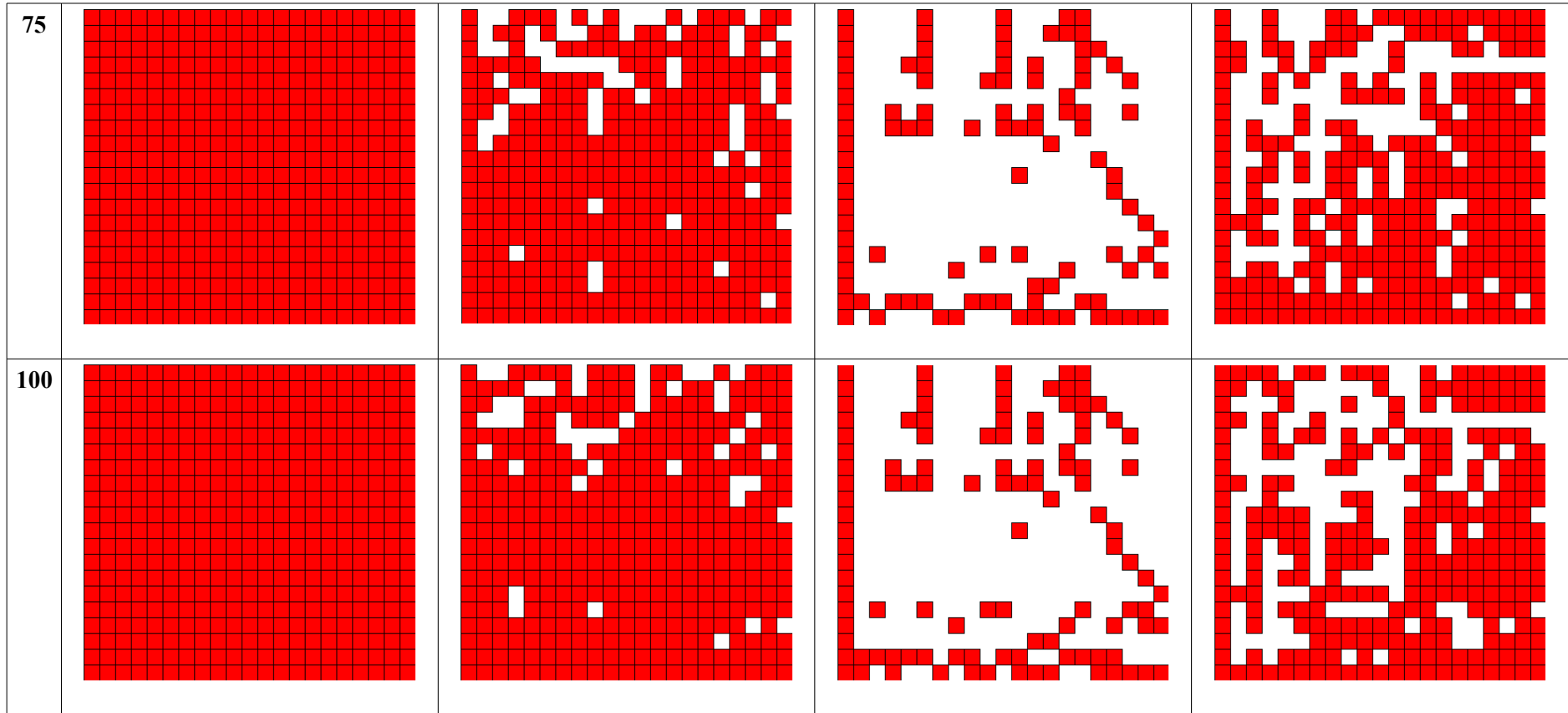
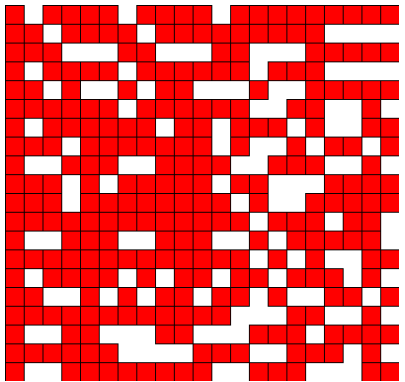
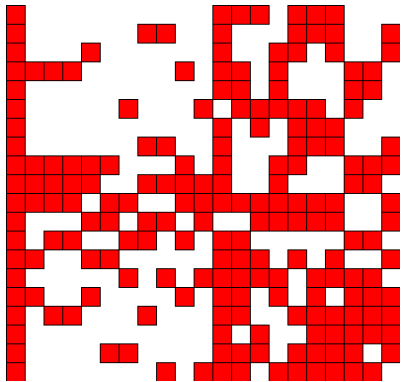
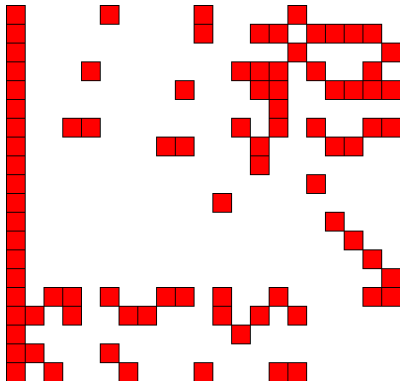
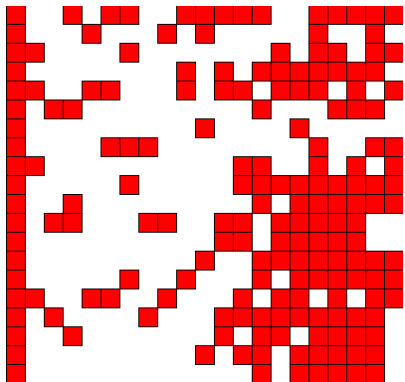
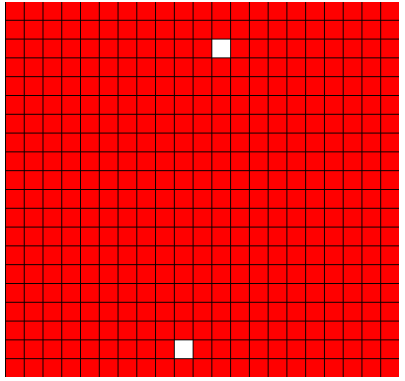
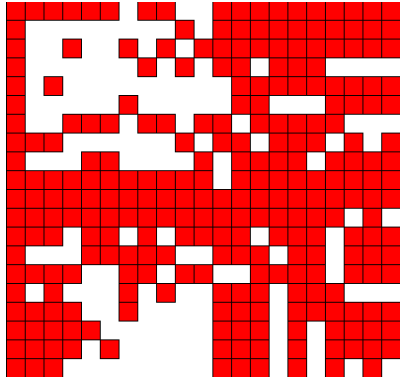
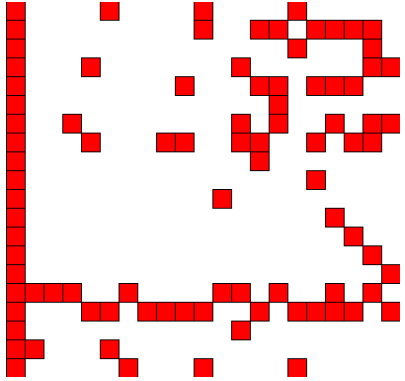
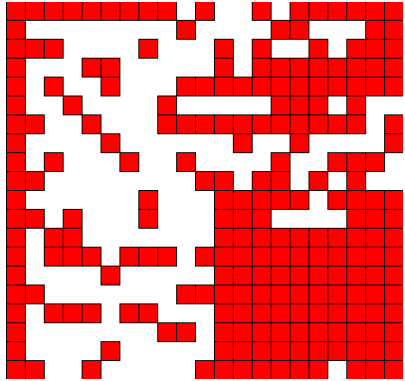


Table 4.5: Final data placements for scheduling algorithms vs iterations (compute dominated)

	Compute-first	Greedy	Neglected	Sufferage
25				
50				

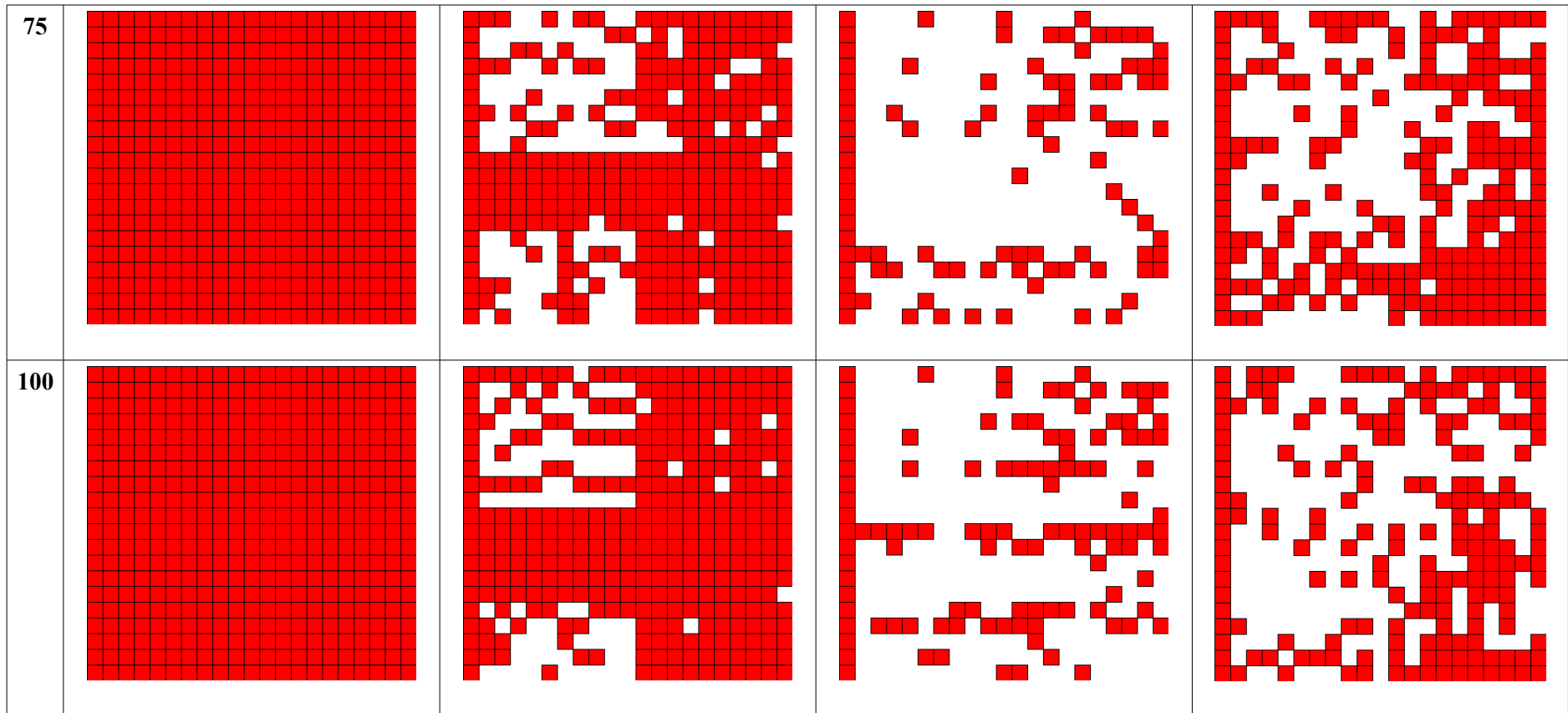


Table 4.6: Final data placements for scheduling algorithms vs iterations (data dominated).

Figure 4.11, Figure 4.12, Figure 4.13, and Figure 4.14 show a comparison of the CPU efficiencies of the various scheduling methods under increasingly CPU dominated workflows. All scheduling algorithms fail to efficiently use the CPUs in the data dominated scenario, as the data transfer time dominates the job run times. This is a worst case scenario, and demonstrates a situation where there is an excessive number of compute resources assigned to the application. This is best illustrated by the Neglected scheduler which isolates 3 resources to perform a majority of the computational work. As the workflows become more CPU intensive the Neglected scheduler outperforms the other schedulers in efficiency, until the workflow is completely CPU dominated. When the workflow is completely CPU dominated the contribution of data transfer to the makespan becomes insignificant, and the Compute-First scheduler slightly exceeds the performance of the Neglected scheduler.

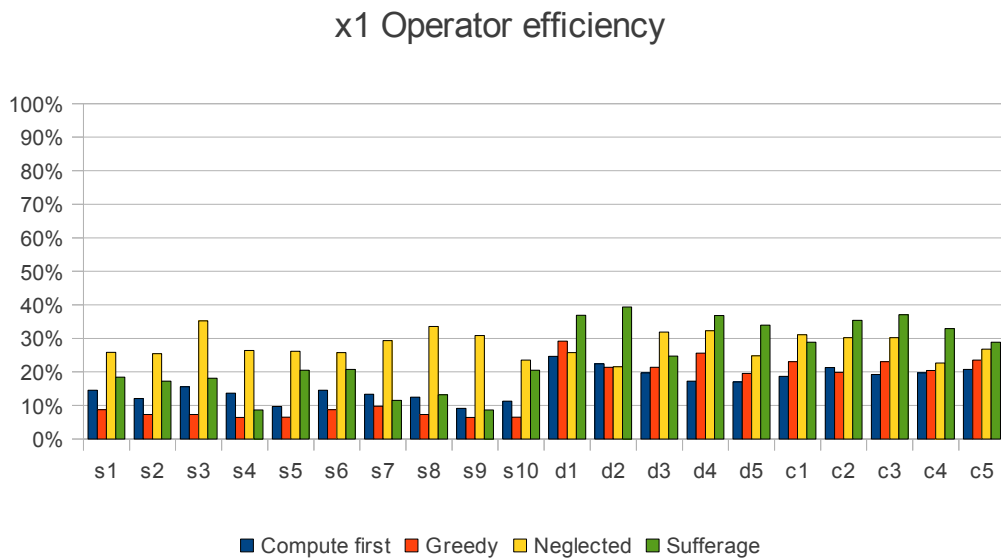


Figure 4.11: CPU Usage Efficiency with increasing CPU dominance (x1).

x5 Operator efficiency

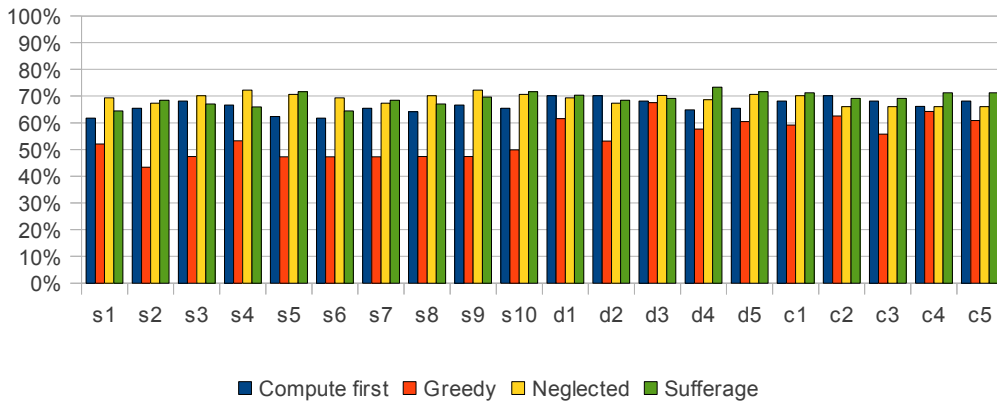


Figure 4.12: CPU Usage Efficiency with increasing CPU dominance (x5).

x10 Operator efficiency

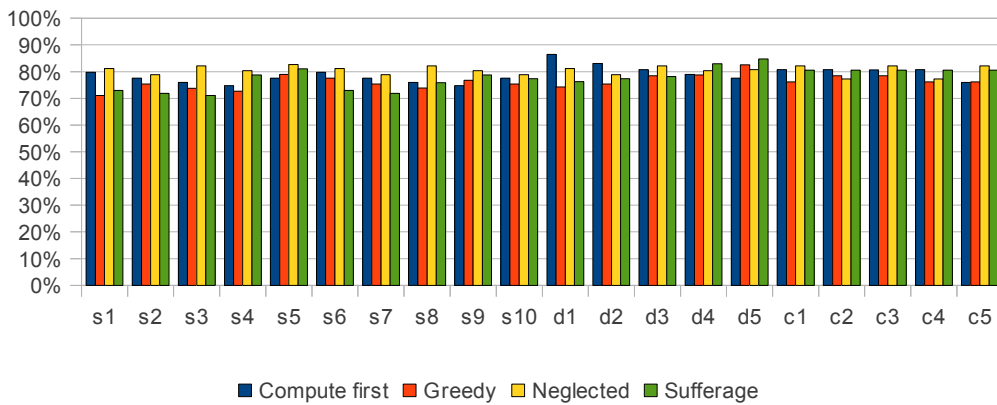


Figure 4.13: CPU Usage Efficiency with increasing CPU dominance (x10).

x100 Operator efficiency



Figure 4.14: CPU Usage Efficiency with increasing CPU dominance (x100).

4.6. Utilisation of Resource Parallelism

Resources with multiple CPUs or CPU cores have a distinct advantage when executing data intensive workflows, as the data transfer penalty can be immediately exploited by multiple tasks. Scheduling for resource parallelism is achieved here using a secondary scheduling algorithm which is invoked immediately after a task is assigned to a resource. This secondary scheduler uses a minimum time selection across all tasks based on their completion time on the host as if all the newly requested data are now available. To demonstrate this an experiment was conducted using a data dominated workflow run across 20 CPU cores. The distribution of the cores was varied from 20 individual resources (0 parallel hosts), to 10 dual-core resources (10 parallel hosts).

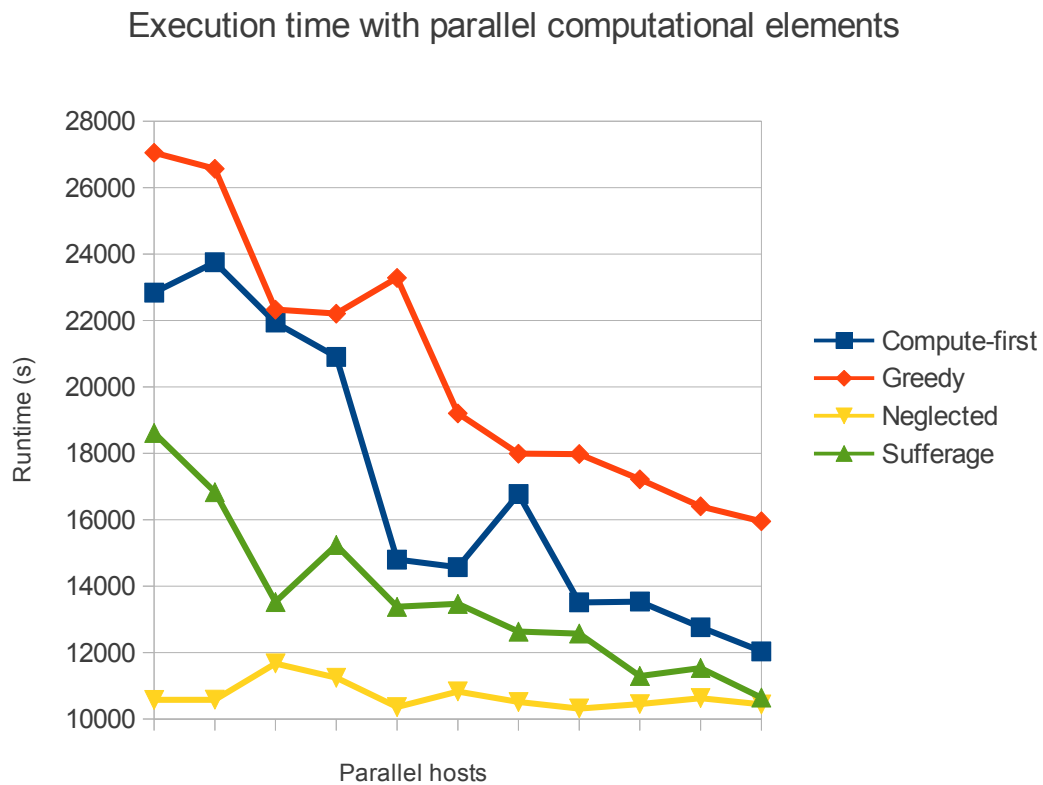


Figure 4.15: Execution times for varying numbers of parallel arrangements of 20 computational elements, running a data-intensive workflow.

The results, seen in Figure 4.15, show that the Greedy scheduler benefited significantly by having parallel hosts. While the Neglected scheduler also benefited significantly, it

appears that the greater emphasis on data location selection had already made a greater contribution.

4.7. Summary

The focus of this chapter has been the scheduling problem, as it relates to the efficient execution of data mining workflows. The scheduling problem is known to be NP-complete meaning optimal solutions to the problem are not computable in polynomial time, and are therefore not possible in practical terms. However, the application of heuristics to the scheduling problem is known to produce acceptable results.

Applying the master-slave paradigm as the distributed computing framework introduced a range of existing heuristics which could be applied to the scheduling problem. A review of these heuristics led to the selection of the Greedy algorithm as the benchmark scheduler, due to its wide acceptance and application in the fields of Grid and distributed computing. The characteristics of the Greedy algorithm were established across a range of workflow parameter combinations, and particular observations were made which led to the development of a new scheduling heuristic, Neglected, which seeks to optimise the execution schedule of data intensive applications. The performance of Neglected was compared against Greedy, and through the experimental results it could be seen that the relative makespan performance varied from significant improvement to approximately equal. However, in all cases the final data distribution of Neglected showed a significant reduction in the data transfer volume. This in itself is significant when consideration is given to the cost of data transfer.

The work presented in this chapter has implications for many data intensive distributed computing applications, through its potential to reduce the execution makespan, and by reducing the data transfer costs. Further research into its application to execution scheduling using commercial resource may yield opportunities to minimise both the compute-time and data-transfer costs of an execution.

Chapter Five

5. Predictive modelling for human intestinal absorption of chemical compounds

To be useful, new pharmaceutical drugs need to be effective on their bio-chemical target, and also need to be efficiently transported to the site where they are required. While there may be many candidate compounds that effectively target a particular condition, if they are not properly absorbed and transported then they will ultimately be ineffective. And while it may be possible to test a drug on a given target in isolation with some level of ease, it is many times more difficult to test a drug's transport through the many complex systems of the human body by any experiment other than clinical human trials. Clinical trials come at great cost in terms of time, money and patient risk, so need to be used sparingly. It is possible however to treat the absorption, distribution, metabolism and excretion (ADME) scenario described here as a Quantitative Structural-Property Relationship (QSAR) problem, and through the use of robust data mining tools there is the possibility of producing accurate models for predicting the transport performance of a drug [101].

The ADME problem space is typical of many cheminformatic problems as, due to expense, there are typically relatively few experientially measured absorption rates, while there are many more candidate predictors available to describe the drugs. Predictors will include molecular descriptors from calculated and measured data sets of candidate compounds. Robust methods are required to reliably tease out relationships between these predictors and the ADME response variables due to the dimensionality of the data set, and should include robust cross-validation to improve the confidence in the predictive performance of the generated model [102].

This chapter presents a study comparing a number of popular data mining techniques, used to produce a predictive model for the human intestinal absorption of various pharmaceutical drugs. This time consuming process is aided by the use of distributed

computing, and demonstrates the utility of the eScience methodology to pharmaceutical drug development. In particular it demonstrates the workflow language and distributed execution framework, along with a prototype of a provenance collection system.

5.1. Introduction

The effectiveness of orally administered pharmaceuticals is a combination of the suitability of the drug to work at the target site, and the efficiency of the transport mechanism via the stomach, through intestinal absorption, and then via blood pathways. This transport efficiency can ultimately determine the usefulness of the drug, so accurately modelling the pathway prior to clinical trials can reduce the cost of pharmaceutical research by filtering poorly transported drugs before they are synthesised. To construct this model requires a data set consisting of clinically measured absorptions of pharmaceutical molecules.

From here there are several approaches to the modelling problem, including QSAR descriptor based models, substructure searching, and similarity matching. The QSAR approach, which is used here, requires that a set of numerical and categorical descriptors of the molecules be calculated from a 2D or 3D model of the molecule, using various methods. This creates a new data set which can be used by a data mining method to fit a model. There are many existing data mining methods which can be applied in this situation, such as PLS, SVM, CART, etc. The existing body of knowledge around these methods gives confidence and understanding to the models they produce.

Alternative approaches include substructure searching and similarity matching, typically utilising the 2D or 3D structure of the molecule directly. These approaches either require a database of known active substructures to test against, or require a database of full molecules and an algorithm to compute their similarities. Both these methods are known to produce good results, but require specialised algorithms to perform the core comparison work. Using numerical descriptors enables the use of general purpose data mining methods, and will be

applied in this investigation.

5.2. Materials and methods

The goal of data mining is to produce a predictive or descriptive model of a system. The system under investigation here is the absorption of a molecule through the human intestine, into the blood. Through clinical experiments a data set has been developed of molecules and their absorptions.

This data set is presented by Hou in [103] contains 552 compounds and their human intestinal absorption (HIA) value. The compounds are presented as 3D models in SDF format – a file format used for the interchange of molecular structures. As seen in Figure 5.1 the responses are heavily skewed to the high end of the scale, which may have consequences for any model fitting the data.

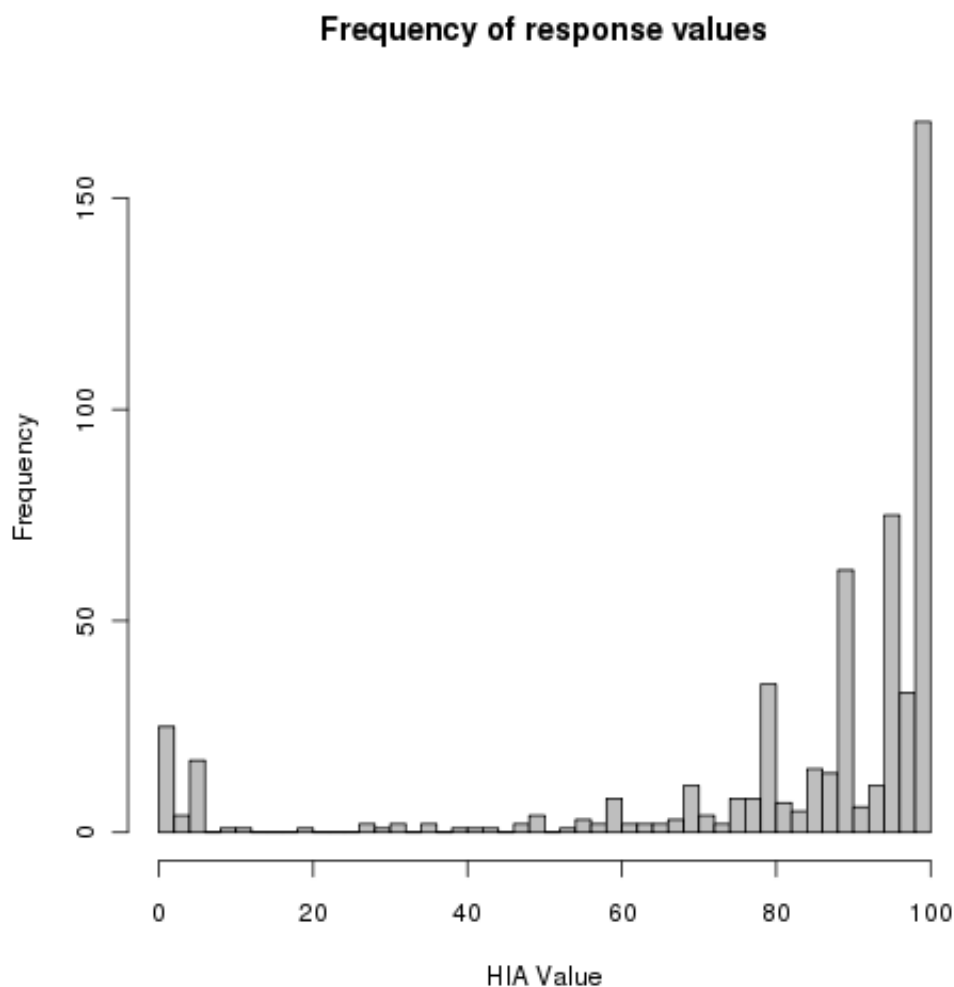


Figure 5.1: Frequency of response variables in Hou data set.

This exact data set has been analysed several times, and will be the focus of this investigation. In [103] Hou used a genetic algorithm to select from a set of descriptors, then using MARS, available in Cerius²[104]. This produces a linear model which contains spline functions. Using this model they achieved an r^2 of ~ 0.81 on the test set. Following on from this Yan, Wang and Cai[105] calculated two separate descriptor sets using ADRIANA.Code[106] and Cerius². They performed variable selection using a genetic algorithm and PLS program genetic-PLS[107], selecting the best 6 descriptors of each data set. Each reduced data set was split into a single test and training set using a Kohonen's self-organising Neural Network (KohNN), and used to fit a Support Vector Machine (SVM) model and a Partial Least Squares (PLS) model. This was repeated using the 9 best descriptors from a combined descriptor set.

The results of this can be seen in Table 5.1. It should be noted that the two investigations reported in this table used different test/training sets, making direct comparison harder.

Descriptors	Method	Results (testset R)
ADRIANA.Code	PLS	0.83
ADRIANA.Code	SVM	0.87
Cerius ²	PLS	0.83
Cerius ²	SVM	0.89
ADRIANA.Code + Cerius ²	PLS	0.83
ADRIANA.Code + Cerius ²	SVM	0.88
See Hou <i>ibid</i>	GA+MARS	0.9

Table 5.1: Previous performance results for predictive models based in the HIA data set.

It is clear from this discussion that there is a wide selection of descriptor sets available, in addition to a wide selection of techniques available to analyse those descriptors. Selecting the appropriate descriptor set, along with appropriate learning technique is required to achieve good results. And variations in the test/training set makes it hard to directly compare results. However, the type of geometric optimisation performed on the model before it is passed to the descriptor calculating application can affect the values of the calculated

descriptors and thus the performance of the techniques. In nature molecules will change shape to achieve their lowest energy shape. Estimations of these shapes can be made, and the accuracy of these estimations will impact on the values of some calculated descriptors.

The more complex the molecule is, the harder it is to calculate what this shape will be. As with descriptor calculations, there is a number of algorithms and applications available to optimise a molecule, which will not always produce the same results. The origins of the 3D models in this investigation are not clear either from the original paper, or the SDF file metadata. For the purposes here they will be assumed to have been optimised with an unknown algorithm. For comparison in this context, the models will be re-optimised with the CORINA[108] software, then these molecular models will be used as inputs to calculate the suite of descriptors provided by the eDRAGON[109] software, which calculate physical, topological and electrical attributes of molecules. In addition, the two descriptor sets provided by Yan, et al. will be also used in comparison.

In contrast to those investigations discussed above, this investigation employs a different validation technique, monte carlo cross validation (MCCV)[32]. MCCV offers a more representative estimation of the predictive power of the model, based on a correlation coefficient which is calculated and averaged over many equal partitions of the data set into test and training subsets[29]. The drawback of this technique is that it requires many models to be fitted, which can be computationally challenging if the model takes a significant amount of time to compute.

This investigation will compare three popular QSAR models, the linear technique SIMPLS[110], and the non-linear techniques SVM[35] and Random Forests[34]. These techniques are selected due to their acceptance within the QSAR community, and because they can be used for prediction and interpretation. In the case of PLS this is a matter of inspecting the latent variable, for SVM Ustun, et al[111] have produced an inspection technique, and random forests can be interpreted by inspecting the individual trees in the forest.

Due to the high dimensionality of the data set two variable reduction techniques will be applied to remove unimportant variables, improving predictive performance. This will include using the genetic algorithm with PLS method to select the most promising 10 variables, and also by using random forests to again select the 10 most important variables.

GA-PLS restricts the total number of attributes, p , in any model so it is within the limits of what PLS can fit. Many models, m , are initially constructed using a random subset of attributes, and are evaluated for their cross validated performance. Selection based on performance weighted probability occurs with n of the m models surviving to form the parent population. This selection technique is used in preference to the top n , to reduce the rate at which the pool becomes homogeneous. Pairs of parent models are then randomly selected, and their attributes, genes, are combined to result in a new model. Mutation then occurs on randomly selected models, with two random genes being flipped. The combined parent and child pool is combined and evaluated, and the process starts over. The stopping criteria was 100 generations. Other parameters were selected as follows: $p = 10$ to allow PLS to clearly and efficiently fit the model, $m = 500$ to give enough models for each attribute to be considered at least once, $n = 100$. This model was implemented using the *pls* module in R 2.7.1.

SVM benefits from variable reduction, but as it works on the separation of feature space by a hyper plane it is more robust to noise variables than MLR. Further, SVM-R better captures non-linearity as it uses a kernel function to project the predictors onto an m -dimensional feature space, which transforms the non-linear relationships into linear relationships in the feature space. This model was implemented using the “e1071” 1.5-19 library in R 2.7.1.

Random forests is an ensemble technique which combines many small trees, allowing it fit non-linear responses in a robust way. Each of the m models is created using a bootstrap sample of the available training set. At each split of the tree n try variables are considered as the splitting criteria. The default value for n try is $p/3$, and in this investigation a range from

$p/3$ to $p^{2/3}$ will be evaluated, with m values of 500, 600, 700, 800, 900 and 1000 being used. This model was implemented using the “randomforest” 4.5-28 library in R 2.7.1.

Of interest in this investigation is the overall performance of the method in predicting the HIA value of the molecule, and the computational time required to compute the model. By using a distributed computing framework it is possible to robustly evaluate the performance of computationally intensive methods, and further evaluate if there is significant value in committing the resources in this class of problem.

5.2.1. Experiment

To ensure consistency across the comparisons the following workflow will be used. The experiment starts with a data set which has been collected and published independently of this experiment. The data set consists of N compounds 3D structures' and laboratory measurements of their human intentional absorption characteristics. It is assumed that the data was collected in a way consistent with best practice, and this itself would require provenance data to confirm. Unfortunately, without access to the original group this information will need to be assumed.

The format of this initial data set is a set of SDF 3D model files. This data mining investigation requires molecular descriptors of each of these compounds to be calculated. This requires using an application called eDRAGON and supplying the original data file to produce outputs containing the descriptors. In this case there are several options for computing the descriptors, and the outputs are split into different files with extra data preceding the first records in each file. These files are combined into a single data file suitable for the data mining component of this experiment. Next the data set is cleaned of any constant variables, and highly collinear variables ($r > 0.9$) leaving 47 predictor variables. The two other data sets were used as supplied.

Monte carlo cross validation is used to validate the process of variable selection and model building, so for consistency the test sets are generated at the start of the experiment so

that each combination of variable selection and model building uses the same set of testsets. Then, for each testset, data set, and variable selection technique combination the variable selection process is executed. Following this, for each testset, data set, variable selection technique and model combination the predictive model is computed, and predictive performance calculated.

5.3. Results and discussions

Discussing the results of the investigation some of the uncertainties of the data need to be highlighted. The primary issue is that the finer details of how this data was collected are not available. In particular, it is not clear what procedure was used to measure the HIA for each compound reported here. Possibilities would include measuring the availability of the compound found in the blood or body fat, or measuring the quantity of the compound which is excreted, and thus not absorbed. Further it is not clear how many times each absorption experiment was performed, and how an aggregate HIA value was arrived at. This uncertainty requires that the confidence in the accuracy of the data set be considered in the analysis of the results.

The second artefact of this data set is that the distribution of response variable is heavily biased towards high HIA values, compared to low HIA values, at a ratio of approximately 10:1. This reduces the confidence that a representative sample of low HIA yielding compounds are available to fully model mechanisms which regulate the HIA value.

ADRIANA.Code GA-PLS	Cerius2 GA-PLS
Nrule5 (X..ViolationsRo5)	Nrule5 ()
Hdon (HDON)	Nrot (ROTLBOND)
LogS (LOGS)	LogP (LOGP)
MW (WEIGHT)	Hdon (HBOND_DO)
TPSA (TPSA)	Jurs-FNSA3 (Jurs_FNSA_3)
Acorr_Sigchg_3 (X2DACorr_SigChg_3)	Jurs-RPCG (Jurs_RPCG)

Table 5.2: Top 6 most frequently selected variables by data set using GA-PLS as reported by Hou.

Where Cerius2 and ADRIANA.Code have the same descriptors the following discrepancies were observed:

1. Hdon for Mesna (ADRIANA.Code = 1 ; Cerius2 = 2)
2. Hacc from ADRIANA.Code does not equal HBOND_AC from Cerius2
3. No Rule of 5 violations were not available from Cerius2, despite this being reported.
4. Molecular weights do not match by an average of 0.0023

The results of the variable selection process using Random Forests and GA-PLS are shown in Table 5.3 and Table 5.4 respectively. All three variable selection sets for Random Forests contained a TPSA and LogP values, which ended up in the selected set of descriptors. Another similarity between the original GA-PLS variable selector and the Random Forest variable selector is the inclusion of the HDON and HBOND_DO descriptors from the ADRIANA.Code and Cerius2 descriptors respectively. As explained by Hou in [103] TSPA and the hydrogen donors are highly correlated, and can possibly explain the electrostatic surface charges of molecules which prevents absorption.

EDRAGON Random Forests	ADRIANA.Code Random Forests	Cerius2 Random Forests
TPSA.Tot.	TPSA	ALOGP98
ALOGPS_logP	HDON	LOGP
T.O..O.	XLOGP	HBOND_DO
BAC	X2DACorr_TotChg_4	Jurs_TPSA
T.N..O.	LOGS	FOCT
ALOGPS_logS	X2DACorr_TotChg_2	Jurs_FNSA_2

Table 5.3: Top 6 variables selected using random forests variable importance measure.

Comparisons between the original GA-PLS variables (Table 5.2) and the variables selected inside the MCCV (Table 5.4) reveals a reordering of some of the selected variables, and the replacement of others. In the ADRIANA.Code data set Nrule5, MW and Acorr_Sigchg_3 were replaced with IsViolatingRu5, MW and Acorr_PiChg_2; and in the Cerius2 data set Nrule5 and Jurs-FNSA3 were replaced with ALOGP98 and Jurs-FNSA2.

While, comparisons between the GA-PLS and Random Forests within MCCV ADRIANA.Code differ Acorr_TotChg_2 and ACorr_TotChg_4 are replaced with IsViolatingRo5 and Acorr_PiChg_2; and Cerius2 Jurs_TPSA and FOCT are replaced with ROTLBOND and Jurs_RPCG. The eDRAGON data set shows even greater variability between selection techniques with, T.O..O, BAC and ALOGPS_logS replaced with PCD, MW and Mor01M. As there is at least a 50% crossover of the most important variables between each selection technique it is possible that these account for most of the predictive power of the models, and the remaining variables contribute little to the model.

EDRAGON GA-PLS	ADRIANA.Code GA-PLS	Cerius2 GA-PLS
ALOGPS_logP	LOGS	ROTLBOND
TPSA.Tot.	IsViolatingRo5	HBOND_DO
PCD	XLOGP	LOGP
MW	TPSA	Jurs_RPCG
Mor01m.	HDON	ALOGP98
T.N..O.	X2DACorr_PiChg_2	Jurs_FNSA_2

Table 5.4: Top 6 variables selected using GA-PLS across the MCCV variable selection.

The results of the combinations of data set and modelling method are shown in Table 5.5 and Table 5.6. The results include the training set and test set values for R^2 and RMSE, as well as the standard deviation of these values across the population of MCCV results. When comparing these results to previously reported results it is important to acknowledge the different cross validation techniques used. Previous investigations used a single training/test split, which may have led to an overestimation of the true predictive performance. The more robust MCCV validation technique, as argued by Konovalov, et al. [29], gives a more accurate estimation of performance.

data set	Method	R ²	R ² prediction	RMSE	RMSEP
EDRAGON	PLS	0.61 (0.09)	0.45 (0.14)	21.82 (2.09)	27.68 (5.16)
EDRAGON	SVM	0.82 (0.02)	0.76 (0.03)	16.12 (0.98)	18.78 (1.26)
EDRAGON	RF	0.96 (0.01)	0.77 (0.03)	8.55 (0.69)	18.11 (0.95)
ADRIANA.Code	PLS	0.69 (0.03)	0.68 (0.04)	19.90 (0.92)	20.78 (1.08)
ADRIANA.Code	SVM	0.85 (0.03)	0.74 (0.04)	15.01 (1.32)	19.20 (1.33)
ADRIANA.Code	RF	0.95 (0.01)	0.76 (0.03)	8.84 (1.06)	18.35 (0.97)
Cerius2	PLS	0.68 (0.05)	0.66 (0.04)	20.32 (1.14)	21.14 (1.05)
Cerius2	SVM	0.85 (0.03)	0.76 (0.04)	15.05 (1.10)	18.77 (1.47)
Cerius2	RF	0.96 (0.01)	0.76 (0.04)	8.52 (0.75)	18.13 (1.20)

Table 5.5: Mean predictive performance of HIA data set descriptors using PLS, SVM and RF predictive models using variables selected by Random Forest.

data set	Method	R ²	R ² prediction	RMSE	RMSEP
EDRAGON	PLS	0.58 (0.13)	0.42 (0.12)	22.28 (2.63)	28.07 (6.54)
EDRAGON	SVM	0.81 (0.03)	0.74 (0.04)	16.64 (1.33)	19.51 (1.45)
EDRAGON	RF	0.96 (0.01)	0.74 (0.04)	8.95 (1.16)	18.81 (1.40)
ADRIANA.Code	PLS	0.55 (0.13)	0.51 (0.15)	22.82 (2.50)	24.12 (2.77)
ADRIANA.Code	SVM	0.87 (0.02)	0.73 (0.03)	14.38 (1.06)	19.65 (1.37)
ADRIANA.Code	RF	0.97 (0.01)	0.77 (0.03)	7.87 (0.70)	17.79 (0.97)
Cerius2	PLS	0.66 (0.12)	0.63 (0.12)	20.49 (2.40)	21.55 (2.71)
Cerius2	SVM	0.87 (0.02)	0.76 (0.03)	14.27 (0.97)	18.79 (1.39)
Cerius2	RF	0.97 (0.01)	0.77 (0.03)	8.07 (0.56)	17.89 (1.10)

Table 5.6: Mean predictive performance of HIA data set descriptors using PLS, SVM and RF predictive models using variables selected by GA-PLS.

Regardless of the data set used there is a clear pattern of predictive performance between PLS, SVM and random forests. In all situations the mean RMSEP of random forests outperforms PLS and SVM. As well, the standard deviation of RMSEP for random forests is also smallest for all data sets and variable selection methods, with one exception, indicating that the performance of random forests is the most stable for these data sets across the MCCV iterations.

Overall, the best predictive performance measured by R² was achieved using random

forests with the EDRAGON descriptors and RF variable selection, random forests using ADRIANA.Code descriptors and GAPLS variable selections, and random forests using Cerius2 descriptors and GAPLS variable selection. The standard deviation of each of these values is 0.03 or 0.04, indicating that they are all effectively equivalent. Measured by RMSE the model produced the same combinations of methods performed best, and again, the standard deviation of the RMSE indicates that the methods are equivalent.

5.4. Workflow and provenance

The workflow for this experiment shown in Figure 5.2 is expressed here as the entire investigation, first generating the testsets (1-3), then performing variable selection for each group of test sets (4-8), and finally computing the predictive models (9-15). In this investigation all work is performed using R or MATLAB, using the modules described in Chapter 5.2.

This workflow was executed using a number of resources available from the James Cook University High Performance and Research Computing department. These included 2 small AMD Opteron 2356 SMPs each of which contributed 8 cores, and 2 AMD Opteron 2593 cluster nodes that each contributed 2 cores. The workflow was executed from the cluster head-node as it was the only point which could directly connect to all the resources involved in the execution.


```

1. foreach id in inputdata
2.   generate_testsets(data_file_name=id.data_file_name,
   testsets=testsets)
3.   write to testsets(inputdata=id.data_file_name,
   testset=testsets)
4. foreach id in inputdata
5.   foreach ts in testsets[inputdata=id.data_file_name]
6.     foreach vs in variable_selectors
7.       _${vs.variable_selector}(data_file_name=id.data_file_name,
   testset=ts.testset, variables=variables)
8.     write to variable_selected(variables=variables,
   variable_selector=vs.variable_selector,
   inputdata=id.data_file_name, testset=ts.index)
9. foreach id in inputdata
10.  foreach ts in testsets[inputdata=id.data_file_name]
11.    foreach vs in variable_selectors
12.    foreach vars in
   variable_selected[inputdata=id.data_file_name,
   variable_selector=vs.variable_selector,
   testset=ts.index]
13.    foreach m in methods
14.      _${m.method}(data_file_name=id.data_file_name,
   variables=vars.variables,
   testset=ts.testset,
   model_file_name=model)
15.    write to filestore(method=m.method,
   testset=ts.index,
   variable_selector=vs.variable_selector,
   model=model,
   variables=vars.variables)

```

Figure 5.2: Workflow for HIA investigation.

Operation	Total Time (s)
Generating testsets	2s
Computing Random Forest variable selection	2,422s = 40.4m
Computing GA-PLS variable selection	26,238s = 437.3m
Computing PLS model	381s = 6.35m
Computing SVM model	427s = 7.1m
Computing Random Forest model	1,327s = 22.1m
Total (linear time)	30,797s = 513.3m

Table 5.7: Amount of time (serial) spent performing each step of the workflow.

The parallel execution took 27.9 minutes (1,674s) to compute, with a majority of the time being spent computing the GA-PLS variable selection in MATLAB. A speedup factor of 18.40 was obtained across the 20 cores involved, which demonstrates a high efficiency parallel execution. Table 5.7 shows a breakdown of the total amount of time spent in each

step of the execution and the total compute time. The time required to fit the predictive models increased with predictive performance of those models, with random forests exhibiting the best predictive performance and requiring the longest time to fit, followed by SVM and finally PLS. This illustrates an important point regarding the trade off between execution time and performance of predictive models. While the time penalty for using an extremely computationally expensive method such as GA-PLS may not be justified by its performance relative to other method, there are also situations where investing extra time fitting a model will produce a significantly improved model, as demonstrated by random forests in this experiment. This illustrates the importance of properly understanding and evaluating the methods used to ensure they are appropriate for the situation. This evaluation of method may also take into account other considerations such as the degree to which the model can be interpreted or visualised.

5.5. Conclusion

Screening of potential pharmaceuticals for their suitability to human use at early stages of their development offers an excellent opportunity to reduce the costs and development times for new drugs. Cheminformatic QSAR methods are an extremely useful tool in this endeavour, however the large number of potential input data sets, data mining methods and the need for robust validation can make the exercise computationally prohibitive. In this chapter it was found that some computationally expensive methods are required to produce high-quality results, while others may not prove so valuable. In order to usefully evaluate and then use candidate methods distributed computing was utilised. Distributed computing allowed a highly parallel workflow to be executed quickly, using the resources available.

The ability to execute the workflow quickly allowed the experiment to include many computationally expensive methods, and also allowed a far more robust method of cross-validation to be employed. Future work on these data sets should include the expansion of the input data set to include a wider range of molecules which includes a better spread of

HIA values. This in itself may improve the performance of the candidate models, and may also allow the most important contributors of human intestinal absorption to be more clearly identified. This experiment reinforced the results of previous experiments which found TSPA and hydrogen donor bonds, which contribute to the polarisation of molecules are significant contributors to human intestinal absorption. By refining future data sets it may be possible to calculate new descriptors which are related to TSPA and HDON which better capture the properties responsible. However, this will surely be another computationally expensive operation, and will again employ distributed computing and eScience methods.

Chapter Six

6. Conclusions and Future Work

The use of information technology (IT) in scientific investigations is now commonplace, and with its use has come improved ways of managing, organising and processing data. The use of IT has become necessary due to the increasingly large and complex nature of data sets being collected during scientific investigations, as well as the growing libraries of these data sets. These libraries are often available not only to the original researcher, but to the wider research community through government and institutional policy, and groups like the Open Science movement. Great opportunities and challenges are posed by the analysis of these data sets, and through the use of modern analysis techniques such as data mining it is possible to unlock the information within them.

The project reported in this thesis specifically explored the application of workflows for expressing experiments, the collection of provenance data for organising investigations, and distributed computing for reducing the time-to-solution of experiments. The project builds on and contributes to the fields of eScience, distributed computing, and data mining to in turn improve the uptake and effectiveness of IT in research.

6.1. eScience: Workflow and Provenance

The eScience paradigm has developed to provide tools and techniques that assist in handling the process of analysing large and complex data sets, and so large scale data mining benefits from advances in eScience.

At an abstract level the process of data mining is relatively straight forward. It involves the preparation of the data sets, the repeated application of a data mining method, and summarising the results from that process. Variations on this process are common, and mostly involve the repeated testing of different combinations of methods and data sets. Practitioners invest large amounts of time exploring parameter spaces, and developing new

methods to improve on previous results. It is possible to abstract away the high level data mining process from the actual method implementation, and this approach was demonstrated in this project using an extended parameter sweep workflow language.

Capturing configuration of the experiment using a workflow language allows the practitioner to interchange methods and data set while ensuring the actual process remains consistent. It also has the benefit of exposing the flow of data and parameters, which allows the workflow to be broken into discrete tasks. These tasks can then be independently executed using distributed computing, the principal benefit of which is to decrease the makespan of the experiment. Quicker solutions enable the investigator to perform increasingly more thorough experiments, reduce costs, and tackle larger problems.

Another benefit of expressing an experiment as a workflow is that it forms documentation of the experiment process, and assists in the collection of provenance information. Provenance information describes where data comes from, and in the context of an experiment it describes, what processes and transformations have occurred on the data.

6.2. Distributed Computing

Distributed computing is the process of using independent computers, connected via a network, to solve a single computational problem. The application or workflow being executed must be broken down into discrete tasks which are either independent, or have all their data dependencies available. Tasks are executed on the resources in the system, with data either transferred or reported centrally so that it becomes available for the user, or other tasks in the system.

The amount of time taken between starting the first task and the final task being completed is affected by the order that tasks are run, and where they are executed. This is known as a scheduling problem and to solve it many aspects of the tasks, computers, data networks and data need to be considered. In a simple scenario where there is very little data and fast networks, only the size of the tasks and the speed of the resources are important, and

strategies exist for scheduling such tasks in quite an efficient manner.

When the size of data of a computational problem become significant network bandwidth and data location become critical to scheduling decisions. Many large scale schedulers use simple objective functions for a greedy algorithm to produce a schedule of tasks to run on resources. Extending the greedy algorithm, which optimises for job runtime, to include a data transfer term fails to produce a satisfactory schedule. This is because the greedy algorithms take a short sighted view of the problem in order to solve the larger problem. In this case, tasks mapped by a greedy approach tend to be those requiring data sets that are already prevalent within the network, as these may be individually quicker to execute.

A new scheduling algorithm, Neglected, is described in this thesis that better evaluates the location of data, and prevents the unnecessary transfer of data. The consequence of this is to reduce the runtime for the application or workflow, and also to use the computational resources more efficiently.

The Neglected algorithm was evaluated against the greedy algorithm, and others, in a number of scenarios with varying sizes of data, thus investigating different impacts from the data location problem. Neglected outperformed the greedy scheduling in all cases, with significant improvement when the data sizes were large. Thus, Neglected can be used as a general replacement for the greedy algorithm when scheduling distributed systems.

6.3. Benefits to Data Mining Applications

The research questions addressed in this thesis resulted from the application of eScience and Distributed computing to the data mining application area. This need was identified due to long execution times (makespan) when contemporary data mining techniques were applied to large data sets. The strong similarities between data mining and the eScience paradigm meant that common eScience techniques could be readily applied, such as the use of workflows and the collection and cataloguing of provenance information. The resulting research questions surrounded the workflow language, and the scheduling of the workflow

onto distributed resources. The data mining applications, which performed a number of data mining investigations, used the R and MATLAB languages to perform the actual computations. These were reported to demonstrate the application of the eScience paradigm, and emphasise the importance of applying these techniques to computer based experimentation. The case study involved the analysis of a data set of human intestinal absorption of pharmaceuticals, and has been previously analysed in several papers. This particular study is pertinent as the data supplied is a molecular model, which is then processed to produce the numerical input to the actual data mining methods. Without careful reporting of each step of this process it is impossible to reproduce the results reported previously, thus to properly evaluate the merit or deficiencies of the reported analysis.

6.4. Outcomes and Contributions

This thesis makes several contributions to the fields of eScience and data mining. These are summarised as follows:

1. This thesis provided an analysis of some typical data mining workflows and developed a general template for these investigations that captures the common constructs of the data mining process. The template exposes parallel computational components of the workflow, and enables the understanding of the computational and data requirements for supporting the execution of these investigations
2. This thesis presented a workflow model that addresses the requirements developed from the general template of data mining, and the design and implementation of a workflow engine to execute this workflow model.
3. This thesis developed a contemporary master-slave scheduling algorithms to efficiently schedule tasks derived from the workflow. Simulation studies conducted using the GridSim toolkit, investigating the performance of the scheduling algorithms on a number of different network topologies.

6.5. Future Work

From the contributions of this thesis it is possible to improve the baseline processes of many practitioners working with data mining and *in silico* experiments. Adoptions of such changes is typically problematic due to issues around legacy applications and engrained practices. Facilitating the integration of legacy codes in the workflow overcomes the former issue, and the increased computational scalability which provides real incentives to adopt an e-Research methodology such as the one presented here overcomes the latter issues.

6.5.1. Resource Scheduling

Data transfer and location is a critical component when scheduling data intensive workflows on heterogeneous resources. Future research topics in this field could include researching the impact of data queuing and transport schemes and their interactions with various task scheduling algorithms. For instance, data transfers can either be centrally coordinated, or each slave agent could locate its own data sources. Data transfers could be queued to give maximum bandwidth to a single transfer, or bandwidth could be shared with multiple concurrent transfers. Queuing could occur at the sender or receivers end of the transfer, as outgoing and incoming bandwidth are equally important. Further, using multiple data sources concurrently such as what is done with the BitTorrent protocol may provide a more reliable transfer time optimisation than a single source.

Similar to the potential investigation into data transfer scheduling, research into the impact of data transfer estimates would provide an important insight into the stability of a scheduling algorithm. Improving mechanisms for estimating data transfer times, and possibly providing limits on the estimated transfer time could provide a more robust scheduling outcome.

The provision of tasks from the workflow also offers many opportunities for optimisation research. In particular, improving the way the task segmenter handles dependencies would allow the segmenter to gradually release tasks as input data becomes available, instead of

only segmenting complete iterator groups. This would then allow critical path analysis to be performed on the tasks, improving the availability of tasks and reducing task starvation.

Improvements to the system's implementation from a research project to a more robust product could allow this system to be applied to more real world application, and would inform future research paths involving the workflow language and provenance systems. This may require improvements and extensions to the data handling system, workflow debugging, and language integration to allow the early detection of invalid workflows and the testing of operators in isolation. The testing of operators in isolation would also provide important quality control information by executing test cases on each operator to validate that the operator performs as expected.

6.5.2. Applications

The project documented in this thesis provides a framework for performing eScience experiments, utilising distributed resources, and addresses a number of problems around these topics. This work can now be applied to a wider range of real-world situations that will support researchers in their scientific investigations. The workflow language separates the applications being used in an investigation from the process that the investigation will follow. This high level view of the investigation could be constructed and managed via a web portal type environment, and by also moving the researchers data to institutional repositories, the entire investigation process would become an “online workbench”. This model of application delivery is becoming popular, particularly for some business applications such as email and document editing, as it becomes accessible from anywhere and not reliant on the practitioners resources.

Moving eScience investigations into a framework like the one described in this thesis allow the practitioner to utilise a larger number of computational resources than would be available had they simply continued using stand alone applications. Grid and Cloud resources, as described in Chapter 2, are becoming more common in research institutions,

and more commonly available on the internet in general. If it is easy to utilise these resources then they can provide a cost effective way of improving the throughput or time-to-solution of investigations. Cloud resources in particular may not provide any guarantees on the bandwidth between resources, or the computational power at the resource, so to utilise them effectively the scheduler needs to be able to respond to the real-world measurements of bandwidth and processing power. The Neglected algorithm is a good candidate for scheduling such resources.

Cloud computing resources coupled with an “online workbench” provides a universally accessible platform for Cloud science, that can be accessed and monitored from anywhere, and can acquire resources when required, without the overhead of ownership and maintenance costs. An immediate audience for this service would be cheminformatics practitioners from whom this project's requirements were derived. As data sets are made available through national and institutional data repository projects, they can be automatically processed using the tools developed in this project, with the results analysed by practitioners to then validate and act upon the outcomes. This platform could then be extended to capture more of the practitioner's workflow, further improving the efficiency of their work.

Many other scientific domains practise methodologies that would benefit from the outputs of this project. Computational biology, criminology, physics and others utilise data mining techniques and workflows that could utilise this technology. Also disciplines that produce computational models, such as economics and climate sciences would be able to execute and manage more simulations as a result of the work presented here.

The development of a production system based on this work would require the consideration of several other real-world requirements which were beyond the scope of this work, such as:

1. Security and access control,
2. Fault tolerance, and

3. Data retention and backup

6.6. Concluding remarks

Information technology tools have improved the quality and volume of research that can occur, through automation, consistent and improved practices. The project presented in this thesis has contributed significant insights and advancements that will continue to improve the impact that IT has on many fields of research, and allow practitioners to exploit the contemporary IT tools available to them.

Nomenclature

Term	Meaning
Data mining	The process of identifying patterns and structures within data.
Experiment	The overall investigation including the process, input and output data.
Experiment process	The steps involved in reproducing the experiment.
data set	A collection of data relating to an experiment.
Variables	Attributes or data points in the data.
Practitioner/Domain practitioner	A researcher in a field.
Computational resource	A computer comprised of one or more computational elements.
Computational element	A CPU or CPU core within a computer that is used to execute a program.
Distributed computing	Using many geographically distributed computers to solve a single problem.
Network of workstations	Using standard desktop workstations to perform batch mode computations.
SMP	Symmetric multiprocessor, a computer with multiple identical CPUs.
Multi-core	A CPU chip which has multiple processors on it. Provides SMP on a single CPU chip.
Cluster	A group of identical computers connected by a high speed, low latency network. Often designed for high performance scientific applications.
Cheminformatics	The study of chemical systems using computational and informatics methods.
Master-slave	A distributed computing paradigm involving centralised command and control.
Master	The controlling node in a master-slave network.
Slave	One or more nodes which perform work in a master-slave network.
SVM	Support Vector Machine – a modern machine learning technique for regression and classification.
PLS	Partial Least Squares regression.
RMSEP	Root Mean Squared Error of Prediction.
HIA	Human Intestinal Absorption.
SDF	Structured Data File – A file format used for describing the 3D structure of molecules.
MLR	Multiple Linear Regression.
QSAR	Quantitative Structure-Activity Relationship.

MCCV	Monte-carlo Cross-validation.
MARS	Multivariate adaptive regression splines.
CART	Classification and Regression Trees.

References

- [1] J. Gasteiger and T. Engel, *Chemoinformatics: a textbook*. Weinheim: Wiley-VCH, 2003, p. 680.
- [2] T. Hey and A. Trefethen, "The Data Deluge: An e-Science Perspective," in *Grid Computing - Making the Global Infrastructure a Reality*, F. Berman, A. J. G. Hey, and G. C. Fox, Eds. Wiley and Sons, 2003.
- [3] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, Second. Morgan Kaufmann, 2005.
- [4] T. Hey and A. Trefethen, "e-Science and its implications.," *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, vol. 361, no. 1809, pp. 1809-25, Aug. 2003.
- [5] I. T. Foster, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Lecture Notes In Computer Science; Vol. 2150*, 2001.
- [6] I. Foster and C. Kesselman, Eds., *The Grid: Blueprint for a New Computing Infrastructure (The Elsevier Series in Grid Computing)*. San Francisco: Morgan Kaufmann, 1998.
- [7] I. Foster and C. Kesselman, Eds., *The Grid 2, Second Edition: Blueprint for a New Computing Infrastructure (The Elsevier Series in Grid Computing)*. San Francisco: Morgan Kaufmann, 2003.
- [8] H. Attiya and J. Welch, *Distributed Computing*, 2nd ed. John Wiley & Sons,

2004.

- [9] I. Foster and C. Kesselman, Eds., *Computational Grids: The Future of High Performance Distributed Computing*. San Mateo, CA: Morgan Kaufmann, 1998.
- [10] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets," *Journal of Network and Computer Applications*, vol. 23, no. 3, pp. 187-200, Jul. 2000.
- [11] O. H. Ibarra and C. E. Kim, "Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors," *Journal of the ACM (JACM)*, vol. 24, no. 2, 1977.
- [12] R. F. Freund et al., "Scheduling Resources in Multi-User, Heterogeneous, Computing Environments with SmartNet," in *Heterogeneous Computing Workshop*, 1998, pp. 184-199.
- [13] "Object Management Group - UML." [Online]. Available: <http://www.uml.org>. [Accessed: Dec-2009].
- [14] A. Z. Dudek, T. Arodz, and J. Gálvez, "Computational methods in developing quantitative structure-activity relationships (QSAR): a review.," *Combinatorial chemistry & high throughput screening*, vol. 9, no. 3, pp. 213-28, Mar. 2006.
- [15] A. R. Katritzky, V. S. Lobanov, and M. Karelson, "QSPR: the correlation and

- quantitative prediction of chemical and physical properties from structure,” *Chemical Society Reviews*, vol. 24, no. 4, pp. 279 - 287, 1995.
- [16] A. Höskuldsson, “PLS regression methods,” *Journal of Chemometrics*, vol. 2, no. 3, pp. 211-228, 1988.
- [17] R. Buyya and M. Murshed, “GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing,” *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1175-1220, Nov. 2002.
- [18] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya, “A toolkit for modelling and simulating data Grids: an extension to GridSim,” *Concurrency and Computation: Practice and Experience*, vol. 20, no. 13, pp. 1591-1609, Sep. 2008.
- [19] I. Atkinson et al., “CIMA Based Remote Instrument and Data Access: An Extension into the Australian e-Science Environment,” *2006 Second IEEE International Conference on e-Science and Grid Computing (e-Science'06)*, pp. 125-125, Dec. 2006.
- [20] J. G. Frey et al., “Combinatorial Chemistry and the Grid,” in *Grid Computing – Making the Global Infrastructure a Reality*, F. Berman, A. J. G. Hey, and G. C. Fox, Eds. Wiley and Sons, 2003, pp. 945-962.
- [21] P. Murray-Rust, “Open Data in Science,” *Serials Review*, vol. 34, no. 1, pp. 52-64, 2008.

- [22] V. Barnett and T. Lewis, *Outliers in Statistical Data*, vol. 3. Wiley, 1994, p. 604.
- [23] C. Goble, C. Wroe, and R. Stevens, “The myGrid Project: services, architecture and demonstrator,” in *UK e-Science All Hands Meeting 2003*, 2003, pp. 595-602.
- [24] “myGrid website.” [Online]. Available: <http://www.mygrid.org.uk>. [Accessed: Jan-2010].
- [25] “GEODISE,” 2010. [Online]. Available: <http://www.geodise.org>. [Accessed: Jan-2010].
- [26] R. D. C. Team, “R: A Language and Environment for Statistical Computing,” 2009. [Online]. Available: <http://www.r-project.org>. [Accessed: Dec-2008].
- [27] I. Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Boston, MA, USA: Addison-Wesley, 1995.
- [28] T. Hastie, R. Tibshirani, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2001.
- [29] D. A. Konovalov, N. Sim, E. Deconinck, Y. Vander Heyden, and D. Coomans, “Statistical confidence for variable selection in QSAR models via Monte Carlo cross-validation.,” *Journal of chemical information and modeling*, vol. 48, no. 2, pp. 370-83, Feb. 2008.
- [30] B. Efron, “Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation,” *Journal of the American Statistical Association*, vol. 78, no.

382, p. 316, 1983.

- [31] R. Kohavi, "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 1995, pp. 1137-1143.
- [32] J. Shao, "Linear Model Selection by Cross-Validation," *Journal of the American Statistical Association*, vol. 88, no. 422, pp. 486 - 494, 1993.
- [33] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123-140, 1996.
- [34] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [35] V. N. Vapnik, *The nature of statistical learning theory*. Springer, 1995.
- [36] D. Weininger, "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules," *Journal of Chemical Information and Modeling*, vol. 28, no. 1, pp. 31-36, 1988.
- [37] Y. L. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *ACM SIGMOD Record*, vol. 34, no. 3, p. 31, Sep. 2005.
- [38] R. Stevens, J. Zhao, and C. Goble, "Using provenance to manage knowledge of in silico experiments.," *Briefings in bioinformatics*, vol. 8, no. 3, pp. 183-94, May 2007.
- [39] M. Greenwood et al., "Provenance of e-Science Experiments - Experience from

- Bioinformatics,” in *Proc. UK e-Science All Hands Meeting 2003*, 2003, pp. 223-226.
- [40] I. Foster, J. Vockler, M. Wilde, and Yong Zhao, “Chimera: a virtual data system for representing, querying, and automating data derivation,” *Proceedings 14th International Conference on Scientific and Statistical Database Management*, pp. 37-46, 2002.
- [41] I. Foster, “The virtual data grid: a new model and architecture for data-intensive collaboration,” *15th International Conference on Scientific and Statistical Database Management*, no. 2, p. 11, 2003.
- [42] “OpenPBS.” [Online]. Available: <http://www.pbsworks.com/>. [Accessed: Aug-2009].
- [43] “XGrid: High Performance Computing for the Rest of Us.” [Online]. Available: http://developer.apple.com/hardware/drivers/hpc/xgrid_intro.html. [Accessed: Aug-2009].
- [44] D. Thain, T. Tannenbaum, and M. Livny, “Distributed computing in practice: the Condor experience,” *Concurrency and Computation: Practice and Experience*, vol. 17, no. 2-4, pp. 323-356, Feb. 2005.
- [45] R. Buyya, D. Abramson, and J. Giddy, “Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid,” in *The 4th International Conference on High Performance Computing in Asia-Pacific Region*, 2000, pp. 283-289.

- [46] R. Rew and G. Davis, "NetCDF: an interface for scientific data access," *IEEE Computer Graphics and Applications*, vol. 10, no. 4, pp. 76-82, 1990.
- [47] D. Thain and M. Livny, "Parrot: Transparent User-Level Middleware for Data-Intensive Computing," in *In Workshop on Adaptive Grid Middleware*, 2003.
- [48] R. Housley, W. Ford, W. Polk, and D. Solo, "RFC-2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile," 1999. [Online]. Available: <http://www.ietf.org/rfc/rfc2459.txt>. [Accessed: Oct-2011].
- [49] "Amazon Elastic Compute Cloud (Amazon EC2)." [Online]. Available: <http://aws.amazon.com/ec2/>. [Accessed: Dec-2009].
- [50] R. Buyya, D. Abramson, and J. Giddy, "A Case for Economy Grid Architecture for Service Oriented Grid Computing," in *Proceedings of the 10th Heterogeneous Computing Workshop*, 2001, pp. 776-790.
- [51] B. Allcock et al., "Data management and transfer in high-performance computational grid environments," *Parallel Computing*, vol. 28, no. 5, pp. 749-771, Apr. 2002.
- [52] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC storage resource broker," in *Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research*, 1998.
- [53] K. Ranganathan and I. Foster, "Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications," *High Performance Distributed Computing*, p. 352, 2002.

- [54] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems," *Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99)*, pp. 30-44, 1999.
- [55] S. Sahni, "Scheduling Master-Slave Multiprocessor Systems," *IEEE Transactions on Computers*, vol. 45, no. 10, 1996.
- [56] D. P. Anderson, "Public Computing: Reconnecting People to Science," in *Conference on Shared Knowledge and the Web*, 2003.
- [57] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: an experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, 2002.
- [58] A. L. Beberg, D. L. Ensign, G. Jayachandran, S. Khaliq, and V. S. Pande, "Folding@home: Lessons from eight years of volunteer distributed computing," in *Proceedings of the 2009 IEEE International Symposium on Parallel Distributed Processing*, 2009, pp. 1-8.
- [59] J. Y.-T. Leung, "Introduction and Notation," in *Handbook of Scheduling. Algorithms, Models and Performance Analysis*, 1st ed., J. Y.-T. Leung, Ed. Boca Raton, FL, USA: CRC Press, 2004.
- [60] J. Pineau, Y. Robert, and F. Vivien, "The impact of heterogeneity on master-slave scheduling," *Parallel Computing*, vol. 34, no. 3, pp. 158-176, Mar. 2008.
- [61] S. Venugopal, "Scheduling Distributed Data-Intensive Applications on Global

- Grids,” PhD Thesis, University of Melbourne, 2006.
- [62] J. Yu, “QoS-based Scheduling of Workflows on Global Grids,” PhD Thesis, University of Melbourne, 2006.
- [63] S. Venugopal, R. Buyya, and L. Winton, “A grid service broker for scheduling distributed data-oriented applications on global grids,” *Proceedings of the 2nd workshop on Middleware for grid computing* -, pp. 75-80, 2004.
- [64] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, “Dynamic mapping of a class of independent tasks onto heterogeneous computing systems,” *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107-131, 1999.
- [65] M. Pinedo, “Off-line deterministic scheduling, stochastic scheduling, and online deterministic scheduling: A comparative overview,” in *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, CRC Press, 2004.
- [66] J. Ullman, “NP-Complete Scheduling Problems,” *Journal of Computer and System Sciences*, vol. 10, pp. 384-393, 1975.
- [67] P. C. Carvalho, R. V. Glória, A. B. de Miranda, and W. M. Degraive, “Squid - a simple bioinformatics grid,” *BMC bioinformatics*, vol. 6, p. 197, Jan. 2005.
- [68] J. Wang, “Soap-HT-BLAST: high throughput BLAST based on Web services,” *Bioinformatics*, vol. 19, no. 14, pp. 1863-1864, Sep. 2003.

- [69] M. S. Pérez, A. Sánchez, V. Robles, P. Herrero, and J. M. Peña, "Design and implementation of a data mining grid-aware architecture," *Future Generation Computer Systems*, vol. 23, no. 1, pp. 42-47, 2007.
- [70] P. T. Domenico Talia, "Weka4WS: a WSRF-enabled Weka Toolkit for Distributed Data Mining on Grids," in *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2005, pp. 309-320.
- [71] P. Brezany and I. Janciak, "GridMiner: A Framework for Knowledge Discovery on the Grid - from a Vision to Design and Implementation," *Architecture*. University of Vienna, 2004.
- [72] P. Brezany, I. Janiciak, and A. M. Tjoa, "GridMiner: An advanced support for e-science analytics," in *Data Mining Techniques in Grid Computing Environments*, W. Dubitzky, Ed. Chichester, UK: John Wiley & Sons, Ltd, 2008, pp. 37-56.
- [73] D. Abramson, C. Enticott, and I. Altintas, "Nimrod / K: Towards Massively Parallel Dynamic Grid Workflows Grid Programming in Kepler," *IEEE SuperComputing*, no. November, pp. 1-11, 2008.
- [74] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, "Kepler: An Extensible System for Design and Execution of Scientific Workflows," in *Scientific and Statistical Database Management*, 2004, pp. 423-424.
- [75] "Kepler Project," Feb-2007. [Online]. Available: <http://kepler-project.org/>.

[Accessed: Jul-2007].

- [76] D. Hull et al., “Taverna: a tool for building and running workflows of services.,” *Nucleic acids research*, vol. 34, no. Web Server issue, pp. W729-32, 2006.
- [77] T. M. Oinn, “Talisman—rapid application development for the grid,” *Bioinformatics*, vol. 19, no. 1, pp. 212-214, 2003.
- [78] “Rapid Miner website,” Dec-2008. [Online]. Available: <http://rapid-i.com/>.
- [79] R. C. Gentleman et al., “Bioconductor: open software development for computational biology and bioinformatics,” *Genome biology*, vol. 5, no. 10, p. R80, Jan. 2004.
- [80] “Extensible Markup Language (XML).” [Online]. Available: <http://www.w3.org/XML/>. [Accessed: Jan-2010].
- [81] “OASIS Web Services Business Process Execution Language (WSBPEL) TC.” [Online]. Available: <http://www.oasis-open.org/committees/wsbpel/>. [Accessed: Dec-2009].
- [82] A. Slominski, “Adapting BPEL to Scientific Workflows,” in *Workflows for e-Science*, London: Springer London, 2007, pp. 208-226.
- [83] H. Casanova, G. Obertelli, F. Berman, and R. Wolski, “The AppLeS Parameter Sweep Template: User-level middleware for the Grid,” *Scientific Programming*, vol. 8, no. 3, 2000.
- [84] D. Gelernter, “Generative communication in Linda,” *ACM Transactions on*

Programming Languages and Systems (TOPLAS), vol. 7, no. 1, pp. 80-112, 1985.

- [85] J. Postel, "RFC-793: Transmission Control Protocol," 1981. [Online]. Available: <http://www.faqs.org/rfcs/rfc793.html>. [Accessed: Jan-2010].
- [86] R. Fielding et al., "RFC-2616: Hypertext Transfer Protocol -- HTTP/1.1," 1999. [Online]. Available: <http://www.faqs.org/rfcs/rfc2616.html>.
- [87] J. Klensin, "RFC-2821: Simple Mail Transfer Protocol," 2001. [Online]. Available: <http://www.faqs.org/rfcs/rfc2821.html>. [Accessed: Jan-2010].
- [88] "XMPP Standards Foundation." [Online]. Available: <http://www.xmpp.org>. [Accessed: Jan-2010].
- [89] J. Bresnahan, M. Link, G. Khanna, Z. Imani, R. Kettimuthu, and I. Foster, "Globus GridFTP: what's new in 2007," in *GridNets 07 Proceedings of the first international conference on Networks for grid applications*, 2007, pp. 1-5.
- [90] I. Atkinson, A. M. Buckle, D. Groenewegen, N. Nicholas, A. Treloar, and A. Beitz, "ARCHER □ An Enabler of Research Data Management," in *2008 IEEE Fourth International Conference on eScience*, 2008, pp. 246-252.
- [91] A. Sulistio and R. Buyya, "A Dynamic Job Grouping-Based Scheduling for Deploying Applications with Fine-Grained Tasks on Global Grids," *Reproduction*, vol. 44, 2005.
- [92] E. Heymann, M. A. Senar, E. Luque, and M. Livny, "Adaptive Scheduling for

- Master-Worker Applications on the Computational Grid,” in *Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, 2000, no. 99186, pp. 214-227.
- [93] O. Beaumont, a Legrand, and Y. Robert, “The master-slave paradigm with heterogeneous processors,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 9, pp. 897-908, Sep. 2003.
- [94] J. Brest and V. Žumer, “A Simple Method for Dynamic Scheduling in a Heterogeneous Computing System,” *Journal of Computing and Information Technology*, vol. 10, no. 2, pp. 85-92, Jun. 2002.
- [95] C.-H. Hsu, T.-L. Chen, and G.-H. Lin, “Grid Enabled Master Slave Task Scheduling for Heterogeneous Processor Paradigm,” in *Grid and Cooperative Computing - GCC 2005*, 2005, vol. 3795, pp. 449-454.
- [96] R. Buyya, “Economic-based Distributed Resource Management and Scheduling for Grid Computing,” PhD Thesis, Monash University, 2002.
- [97] J. Banks, J. Carson, B. L. Nelson, and D. Nicol, *Discrete-Event System Simulation*, 4th ed. Prentice Hall, 2004, p. 624.
- [98] K. Kurowski, J. Nabrzyski, A. Oleksiak, and J. Weglarz, “Grid scheduling simulations with GSSIM,” in *International Conference on Parallel and Distributed Systems*, 2007, pp. 1-8.
- [99] A. Iosup, O. Sonmez, and D. Epema, “DGSim: Comparing Grid Resource Management Architectures through Trace-Based Simulation,” *Lecture Notes*

In Computer Science; Vol. 5168, 2008.

- [100] C. L. Dumitrescu and I. Foster, "GangSim: a simulator for grid scheduling studies," *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.*, pp. 1151-1158, 2005.
- [101] A. M. Davis and R. J. Riley, "Predictive ADMET studies, the challenges and the opportunities.," *Current opinion in chemical biology*, vol. 8, no. 4, pp. 378-86, 2004.
- [102] M. D. Wessel and S. Mente, "ADME by Computer," in *Annual Reports in Medicinal Chemistry*, 36th ed., A. M. Doherty, Ed. 2001, pp. 257-266.
- [103] T. Hou, J. Wang, W. Zhang, and X. Xu, "ADME evaluation in drug discovery. 7. Prediction of oral absorption by correlation and classification.," *Journal of chemical information and modeling*, vol. 47, no. 1, pp. 208-18, 2007.
- [104] "Cerius2 version 4.10L." [Online]. Available: <http://www.accelrys.com/>. [Accessed: Aug-2009].
- [105] A. Yan, Z. Wang, and Z. Cai, "Prediction of Human Intestinal Absorption by GA Feature Selection and Support Vector Machine Regression.," *International journal of molecular sciences*, vol. 9, no. 10, pp. 1961-1976, Oct. 2008.
- [106] "ADRIANA.Code." [Online]. Available: <http://www.molecular-networks.com/>. [Accessed: Aug-2009].
- [107] R. Leardi, "Genetic algorithms applied to feature selection in PLS regression:

- how and when to use them,” *Chemometrics and Intelligent Laboratory Systems*, vol. 41, no. 2, pp. 195-207.
- [108] “CORINA: Generation of 3D Coordinates.” [Online]. Available: networks.com/software/corina. [Accessed: Jan-2009].
- [109] I. Tetko et al., “Virtual computational chemistry laboratory--design and description.,” *Journal of computer-aided molecular design*, vol. 19, no. 6, pp. 453-63, 2005.
- [110] S. de Jong, “SIMPLS: An alternative approach to partial least squares regression,” *Chemometrics and Intelligent Laboratory Systems*, vol. 18, no. 3, pp. 251-263, 1993.
- [111] B. Ustun, W. J. Melssen, and L. M. C. Buydens, “Visualisation and interpretation of Support Vector Regression models.,” *Analytica chimica acta*, vol. 595, no. 1-2, pp. 299-309, 2007.
- [112] A. Krogh and J. Vedelsby, “Neural Network Ensembles, Cross Validation, and Active Learning,” *Advances in Neural Information Processing Systems*, pp. 231-238, 1995.
- [113] R. Shapire, “The Boosting Approach to Machine Learning: An Overview,” in *Nonlinear Estimation and Classification*, D. D. Denison, M. H. Hansen, C. C. Holmes, B. Mallick, and B. Yu, Eds. New York: Springer-Verlang, 2003.
- [114] C. W. Smyth, “Weighted tree-based cluster ensembles for high dimensional data,” PhD Thesis, James Cook University, 2007.

- [115] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression," *Annals of Statistics*, vol. 32, no. 2, pp. 407-499, 2004.
- [116] T. Back, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press.
- [117] H. Muhlenbein and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm i. continuous parameter optimization," *Evolutionary Computation*, vol. 1, pp. 25-49, 1993.
- [118] Y. N. Kaznessis, M. E. Snow, and C. J. Blankley, "Prediction of blood-brain partitioning using Monte Carlo simulations of molecules in water.," *Journal of computer-aided molecular design*, vol. 15, no. 8, pp. 697-708, Aug. 2001.
- [119] J. H. Friedman, "Multivariate Adaptive Regression Splines," *The Annals of Statistics*, vol. 19, no. 1, pp. 1 - 67, 1991.
- [120] E. Deconinck et al., "Boosted regression trees, multivariate adaptive regression splines and their two-step combinations with multiple linear regression or partial least squares to predict blood-brain barrier passage: a case study.," *Analytica chimica acta*, vol. 609, no. 1, pp. 13-23, 2008.

Appendix A - Optimising Ensemble Predictions

This appendix presents a data mining experiment that compares various data mining techniques. It is intended to be viewed as an example of one of the types of experiment that this thesis focuses on. The study involves using ensemble methods to predict the penetration of the blood brain barrier by various molecules. This is an important attribute for the development of pharmaceuticals, as an otherwise promising molecule may not be delivered to its intended site if it cannot cross the blood brain barrier. An ensemble method combines the predictions of many individual models to produce a single, hopefully more reliable output. This approach has been applied to trees [34], neural networks [112] and more generally on any model type using techniques such as bagging [33] and boosting [113]. For ensembles to perform well there needs to be diversity among the models within the ensemble, meaning that their errors or residuals should not be correlated. That is, they should not be making the same mistakes. This can be achieved by:

1. Using a variety of model types in the ensemble;
2. Supplying different starting parameters and random seeds to the models;
3. Sampling from the observations and variables;
4. A combination of the above 3 approaches.

Smyth and Coomans [42] compared many model weighting and selection techniques, including genetic algorithms, evolutionary strategies, lasso and quadratic programming. They found that using the model selection to impose parsimony, reducing the number of models, improves the predictive power of the ensemble by removing poorly performing models from the set.

In this study three ensemble techniques will be compared to evaluate their fitting performance on a large data set, and also their running times. These techniques will be Random Forests, bagged linear models with random variable selection, and a heterogeneous ensemble of linear models and trees, tuned by various means.

The ensemble will be constructed by a weighted, linear combination of the component models. The procedure for weighting the models will be drawn from [115], and involves splitting the data set into 3 segments, a model training set, a weighting training set and a test set. The models will be trained on the first training set, then the second training set was used to calculate the appropriate weights for each model. The testset is then used to determine the fitting coefficients for the models, R^2 .

A.1. Methods and materials

This investigation will compare the predictive and computational performance of a variety of ensemble models. The ensembles will include the well known random forests technique, a heterogeneous ensemble composed of linear models and tree models, and an ensemble of linear models analogous to random forests. The heterogeneous ensembles will be compared as is, and after post-processing using genetic algorithms (GA), a pruning technique, and Lasso with evolutionary strategies to impose parsimony.

A.1.1. Heterogeneous ensemble

The heterogeneous ensembles are M models, composed of simple MLR models and simple tree models. The MLR models are developed using a forward stepwise technique to limit the over-fitting, with a maximum number of variables set to 10. The particular stepwise technique used in this case will be Lasso, with the number of variables to use evaluated using internal cross validation. The tree models are developed using the rpart 3.1 recursive partitioning library in R to build regression trees. To establish diversity each model will be presented with a randomly selected subset of the available variables, $\frac{p}{10}$.

Mathematically, the ensemble is given by:

$$F(\mathbf{x}) = \sum_{i=1}^M (w_i f_i(\mathbf{x})) \quad (1)$$

where $f_i(x)$ is the output of the i^{th} model given the vector x , and w_i is a weighting given to the predictions of the i^{th} model. In the simplest case $w_i = \frac{1}{M}$, or the mean of the predictions is used. The post-processing techniques discussed next may adjust these weights to improve performance.

A.1.2. Lasso post-processing

Lasso is a constrained version of ordinary least squares regression which improves prediction of linear models through shrinkage, reducing the coefficients of that model some of which are reduced to zero. This can be solved using quadratic programming, adjusting the shrinkage parameter, t , to improve the model. Alternatively a Lasso solution can be achieved using the Least Angles Regression [115] which can produce the Lasso solutions over M steps.

For use as a post processing method Lasso is presented with a matrix of the predicted values from the ensemble models, and it will try to fit them to the response vector y . The algorithm will produce a set of solutions along the Lasso path, and the optimum value is selected using the cross-validated R^2 .

A.1.3. Evolutionary strategies post-processing

Evolutionary strategies (ES) use concepts from evolutionary theory to optimise a problem expressed in real numbers. Solutions are encoded as chromosomes, which have real valued genes representing the parameters of the problem. There needs to be a fitness function which can evaluate the potential solutions presented in the population. In general a ES will perform the following steps repeatedly, until an exit condition is reached:

1. Combine pairs of solutions to produce a offspring solution;
2. Randomly mutate genes in the solutions; and
3. Evaluate the performance of each solution and select from the pool for the next

generation.

There are many ways to achieve each of these steps, and to represent problem in the chromosome which are covered in more depth in [116]. Briefly, the parents and offspring may either exist in the same population, so a successful solution can live on, or the offspring can completely replace the parent generation, reducing the chance of getting caught in a local minima. The number of solutions in the child or combined population generally sets the number of parallel lines of enquiry, but with the penalty of increased compute time. Selection for reproduction can be random, weighted or ordered, resulting in slow, moderate or fast convergence. As well there are many strategies for computing the recombination and mutation operations.

Fitness is defined as the mean squared error (MSE) of the fitting, which is to be minimised. Recombination is achieved using line recombination [117], which produces offspring by randomly choosing a point, a , along a line drawn between the two parent values of each parameter. The interval, d , is set to 0.25 allowing the offspring to fall outside the range of the parent weights.

$$\begin{aligned} w_i^o &= w_i^{p1} a + w_i^{p2} (1-a) \\ a &\in [-d, 1+d] \end{aligned} \quad (2)$$

Two mutation techniques are used in this case. The first is the addition of a uniform random vector in the range of -0.001 and 0.001 to each solution. The second is a $p=0.05$ chance of any gene being set to zero. This is intended to help pressure the population to reduce the number of models active in the ensemble. At the end of the mutation step the weights are renormalised to sum to 1.

The stand alone evolutionary algorithms are run to 10,000 generations to allow time for convergence, with an early exit condition of 100 consecutive generations without improvement. Due to the random nature the technique is repeated 100 times to measure its average performance. This is a very long running technique, but has performed well in

previous experiments. However, it may be possible to shortcut the technique and supply it with a set of known good starting points in order to reduce the time to convergence via the early exit. To achieve this, half the starting population will be drawn from solutions around and including the optimum Lasso solution, with the remainder of the initial population being randomly selected as before.

A.1.4. Blood Brain Barrier (BBB) data set

The data set under investigation is a cheminformatics data set of blood brain barrier transport measurements from [29] Table S1 designated KS289-logBB. The data set consisted of structures expressed in the SMILES format, a textual representation of the

molecule structures, and logBB values where $logBB = \log\left(\frac{C_{brain}}{C_{blood}}\right)$, C_{brain} and C_{blood} are the concentration of the molecule in the brain and blood respectively [118]. The SMILES values were transformed into a 3D model using the CORINA [108] software, which was then used to compute values for chemical descriptors using the eDRAGON [109] interface to the Dragon Descriptor software. After removing the constant columns the data set has $n=298$ observations and $p=1502$ variables.

A.1.5. Friedman1000

The Friedman1000 data set refers to the synthetic data set from [119] set that is used to generate 1000 observations. This data set contains 10 variables distributed uniformly over the interval [0,1], with only 5 actually used to produce the response variable. This data set was generated using the *mlbench* library for R.

A.2. Results and discussion

The results of the four post processing techniques can be seen in Table 1. One distinct difference between the data sets which can be observed from these results is that the BBB data set was far more prone to overfitting than was the Friedman1000 data set. This result

was most probably due to the larger number of unimportant variables present in the BBB data set, which tended to dominate even when filtered via the bootstrapping of the variables when building the ensembles.

Method	BBB			Friedman1000		
	Training set	Testset	Runtime (s)	Training set	Testset	Runtime (s)
ES	0.72	0.23	311519*	0.85	0.83	58777
ES+Lasso	0.78	0.18	181068*	0.7	0.67	34164
SA	0.62	0.58	73.28	0.8	0.8	11.8
Lasso opt	0.84	0.62	330.65	0.88	0.86	74.45

Table 1: R^2 results of the ensemble post-processing techniques on the blood-brain barrier and Friedman1000 data sets.

Between the ES and non-ES post-processing it is clear that the ES tend to optimise too aggressively and make this overfitting phenomenon worse. This is particularly evident in the BBB data set. It can also be seen that the lasso post-processor performed best in both testset and training set performance. Lasso explicitly detects colinearity within the data set, excluding colinear components. Also, owing to its stepwise nature, only a subset of the weights will be non-zero through most of the optimisation. This ability to aggressively remove models from the ensemble is believed to be why the lasso approach outperformed the other approaches in both data sets. To confirm this, the experiment was re-run to inspect the model weightings.

In comparison to others who have analysed these data sets the best results compare favourably. The BBB data set was also used by Deconinck, et al. [120] who applied PLS and boosted regression trees to model the data. They achieved an R^2 of 0.64 using PLS, which only slightly outperforms the results here. It is also suggested by Abraham, et al, that the experimental error in the logBB measurement is approx 0.3, or 9.2%.

The Friedman1000 data set is a generated data set, so will differ every time it is used. However, its structure should remain relatively consistent and comparable. Smyth *ibid* used this data set in her original investigation, which inspired this work. In this she applied a

similar array of post-processing techniques, but not the ES + Lasso technique, which was the principal new work here. In her study she reported that ES performed best or equal best, with an R^2 of 0.77 average and 0.79 selecting the best case. In this investigation ES, lasso and simple average both outperform these reported results, but lasso outperforms ES 0.86 to 0.83. This margin is significant as Smyth reports ES outperforming lasso 0.77 to 0.72.

The ES+Lasso post-processor was intended to force the ES technique to an earlier finish time, without jeopardising the performance. However, the approach taken did not achieve this. Optimising the MSE appears to have overfitted the data, and forced it into a local minima through a biased starting position.

A.3. Conclusion

It is clear from the results presented in this investigation that attempts to bias the starting point of evolutionary strategies can achieve better runtime performance, but may jeopardise the predictive performance of the models. Further investigations are required to investigate techniques to reduce this bias, while still maintaining the computational performance gains.

It is also clear that an ensemble weighting strategy which has the ability to eliminate models can significantly improve the performance of the ensembles by improving parsimony, as expressed by Smyth.

Appendix B - Important Grid Agent Interfaces

B.1. IAgent.java

```
/**
 * The IAgent interface provides access to all the services
 * provided by the agent, that will be used by the workflow
 * execution threads.
 *
 * Possible implementations will return network enabled
 * and local-only communications.
 */
public interface IAgent {
    /**
     * Get the network address which the agent is listening on.
     * Can be null.
     * @return
     */
    public String getIpAddress();

    /**
     * Get the port the agent is listening on.
     *
     * Can be 0.
     * @return
     */
    public int getPort();

    /**
     * Returns the tuple store appropriate for this agent.
     * @return
     */
    public abstract ITupleStore getTupleStore();

    /**
     * Returns a map of operators that are available on this
agent.
     * @return
     */
    public abstract OperatorMap getOperators();

    /**
     * Returns the data access factory that is available from this
agent.
     *
     * The data access factory provides low-level access to the
tuple space
     * @return
     */
    public DataAccessFactory getDataAccessFactory();

    /**
     * Return the service interface for file movement between the
master
     * and the executing agent.

```

```

    * @return
    */
    public IFileMovement getFileMomentum();

    /**
     * Shuts the agent down, exiting the process.
     */
    public void shutdown();

    /**
     * Sets the experiment
     */
    public void setExperiment(Reader source) throws SAXException,
    IOException, WorkflowException;

    /**
     * Returns a populated status object
     * @return
     */
    public IPeerStatus getStatus();
}

```

B.2. ITupleStore.java

```

/**
 * The object implementing the ITupleStore provides universal access
 * to
 * data stored in the tuple space. Operations to get, put and locate
 * tuples
 * are provided by this service interface.
 * @author nigel
 *
 */
public interface ITupleStore {
    /**
     * Get size of tuple in store
     * @param storeName
     * @param index
     * @return
     */
    public long getTupleSize(String storeName, int index) throws
    DataException;

    /**
     * Returns a specific tuple.
     * @param storeName
     * @param index
     * @return
     */
    public Map<String, Object> getTuple(String storeName, int
    index) throws DataException;

    /**
     * Get the number of tuples in the particular store
     * @param storeName

```

```

    * @return
    */
    public long getLength(String storeName) throws DataException;

    /**
     * Append data to a particular tuple store.
     * @param storeName
     * @return the index at which this was inserted.
     */
    public int putTuple(String storeName, Map<String, Object>
data) throws DataException;

    /**
     * Get a reference to the underlying store provider.
     * @param storeName
     * @return
     */
    public IDataAccessInformation getTupleStore(String storeName);

    /**
     * List all the variables of this tuple store.
     * @param store
     * @return
     */
    public List<String> getTupleVariables(String store);

    /**
     * Find the tuples where the contents of the attribute
identified by field
     * matches the attribute supplied in criteria. This only
accepts int[] and String[]
     * @param storeName
     * @param field
     * @param criteria
     * @return
     */
    public int[] find(String storeName, String field, Object
criteria);
}

```

B.3. IFileMovement.java

```

/**
 * The object implementing the IFileMovement interface is intended
to
 * provide the ability to transfer files to and from the given agent
 * and master relative paths.
 * @author nigel
 *
 */
public interface IFileMovement {
    /**
     * Copies a file resource from local disk to the master's disk
     * @param local
     * @param remote
     * @throws IOException
     */
}

```

```

    */
    public abstract void uploadFile(String local, String remote)
        throws IOException;

    /**
     * Copies a file resource from the master's disk to local disk
     * @param remote
     * @param local
     * @throws IOException
     */
    public abstract void downloadFile(String remote, String local)
        throws IOException;
}

```

B.4. IPeerStatus.java

```

/**
 * IPeerStatus contains the basic information about a peer.
 * @author nigel
 */
public interface IPeerStatus {
    /**
     * Return the peers ID
     * @return
     */
    public Integer getId();

    /**
     * Get the total number of CPUs/threads
     * @return
     */
    public int getCpus();

    /**
     * Returns a string identifier for the CPU in use on this
agent.
     * @return
     */
    public String getCpuType();

    /**
     * Get the CPU speed.
     * @return
     */
    public int getCpuSpeed();

    /**
     * Returns the total amount of RAM in bytes
     * @return
     */
    public long getTotalMemory();

    /**
     * Returns the total number of threads active on this agent
     * @return

```



```
    */  
    public int getTotalThreads();  
  
    /**  
     * Get the number of CPUs which are idle.  
     * @return  
     */  
    public int getIdleThreads();  
}
```

Appendix C - Workflow XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://nigelsim.org/gridagents"
xmlns:tns="http://nigelsim.org/gridagents"
elementFormDefault="qualified">
  <element name="SIDAML" type="tns:sidamlType"></element>
  <complexType name="sidamlType">
    <sequence>
      <element name="operators" type="tns:operatorsType"
maxOccurs="1"
          minOccurs="0"></element>
      <element name="dataElements"
type="tns:dataElementsType"
          minOccurs="0" maxOccurs="1"></element>
      <element name="workflow" type="tns:workflowType"
maxOccurs="1"
          minOccurs="0"></element>
    </sequence>
    <attribute name="name" type="string"></attribute>
    <attribute name="version" type="string"></attribute>
  </complexType>
  <complexType name="operatorsType">
    <sequence>
      <element name="algorithm"
type="tns:algorithmOperatorType"
          maxOccurs="unbounded"
minOccurs="0"></element>
      <element name="callR" type="tns:rOperatorType"
maxOccurs="unbounded"
          minOccurs="0"></element>
      <element name="application"
type="tns:applicationOperatorType"
          maxOccurs="unbounded"
minOccurs="0"></element>
      <element name="java" type="tns:javaOperatorType"
maxOccurs="unbounded"
          minOccurs="0"></element>
    </sequence>
  </complexType>
  <complexType name="operatorType">
    <sequence>
      <element name="parameter" type="tns:parameterType"
maxOccurs="unbounded"
          minOccurs="1"></element>
      <element name="depends" type="string"
maxOccurs="unbounded"
          minOccurs="0"></element>
    </sequence>
    <attribute name="name" type="string"></attribute>
  </complexType>
  <complexType name="algorithmOperatorType">
    <complexContent>
      <extension base="tns:operatorType">
        <attribute name="version"

```

```

type="string"></attribute>
      <attribute name="library"
type="string"></attribute>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="rOperatorType">
    <complexContent>
      <extension base="tns:operatorType">
        <sequence>
          <element name="script"
type="string"></element>
        </sequence>
        <attribute name="script-uri" type="string"
use="optional"></attribute>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="applicationOperatorType">
    <complexContent>
      <extension base="tns:operatorType">
        <sequence>
          <element name="command"
type="string"></element>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="javaOperatorType">
    <complexContent>
      <extension base="tns:operatorType">
        <sequence>
          <element name="class"
type="string"></element>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="parameterType">
    <attribute name="name" type="string"></attribute>
    <attribute name="property"
type="tns:parameterPropertyEnum"></attribute>
    <attribute name="type" type="string"></attribute>
    <attribute name="direction"
type="tns:directionType"></attribute>
  </complexType>
  <simpleType name="directionType">
    <restriction base="string">
      <enumeration value="in"></enumeration>
      <enumeration value="out"></enumeration>
      <enumeration value="inout"></enumeration>
    </restriction>
  </simpleType>
  <simpleType name="parameterPropertyEnum">
    <restriction base="string">
      <enumeration value="data"></enumeration>

```

```

        <enumeration value="length"></enumeration>
    </restriction>
</simpleType>
<complexType name="dataElementsType">
    <sequence>
        <element name="data" type="tns:dataType"
maxOccurs="unbounded"
                minOccurs="1"></element>
    </sequence>
</complexType>
<complexType name="dataType">
    <sequence>
        <element name="format" type="tns:formatType"
maxOccurs="unbounded"
                minOccurs="1">
    </element>
        <element name="property" type="tns:propertyType"
maxOccurs="unbounded"
                minOccurs="0"></element>
    </sequence>
    <attribute name="name" type="string"></attribute>
    <attribute name="type" type="string"></attribute>
    <attribute name="role" type="string"></attribute>
    <attribute name="uri" type="string"></attribute>
</complexType>
<complexType name="formatType">
    <sequence>
        <element name="variable" type="tns:variableType"
maxOccurs="unbounded"
                minOccurs="1"></element>
    </sequence>
    <attribute name="type" type="string"></attribute>
    <attribute name="index" type="string"></attribute>
</complexType>
<complexType name="variableType">
    <sequence>
        <element name="value" type="string"
maxOccurs="unbounded"
                minOccurs="0"></element>
    </sequence>
    <attribute name="name" type="string"></attribute>
    <attribute name="type" type="string"></attribute>
    <attribute name="length" type="int"></attribute>
</complexType>
<complexType name="workflowElementType" abstract="true">
</complexType>
<complexType name="workflowContainerType" abstract="true">
    <complexContent>
        <extension base="tns:workflowElementType">
            <sequence>
                <element name="children"
type="tns:workflowElementType"
                    minOccurs="0"
maxOccurs="unbounded"></element>
            </sequence>
            <attribute name="name" type="string"
use="required"></attribute>

```

```

        </extension>
    </complexContent>
</complexType>
<complexType name="workflowType">
    <complexContent>
        <extension base="tns:workflowContainerType">
        </extension>
    </complexContent>
</complexType>
<complexType name="groupType">
    <complexContent>
        <extension base="tns:workflowContainerType">
        </extension>
    </complexContent>
</complexType>
<complexType name="iteratorType">
    <complexContent>
        <extension base="tns:workflowContainerType">
            <attribute name="data" type="string"
use="required"></attribute>
            <attribute name="from"
type="int"></attribute>
            <attribute name="to" type="int"></attribute>
            <attribute name="step"
type="int"></attribute>
        </extension>
    </complexContent>
</complexType>
<complexType name="executeType">
    <complexContent>
        <extension base="tns:workflowElementType">
            <sequence>
                <element name="map" type="tns:mapType"
maxOccurs="unbounded"
                    minOccurs="0"></element>
            </sequence>
            <attribute name="algorithm"
type="string"></attribute>
        </extension>
    </complexContent>
</complexType>
<complexType name="mapType">
    <attribute name="variable" type="string"></attribute>
    <attribute name="parameter" type="string"></attribute>
</complexType>
<complexType name="writeType">
    <complexContent>
        <extension base="tns:workflowElementType">
            <sequence>
                <element name="map" type="tns:mapType"
maxOccurs="unbounded"
                    minOccurs="0"></element>
            </sequence>
            <attribute name="data"
type="string"></attribute>
        </extension>
    </complexContent>

```

```
</complexType>
<complexType name="propertyType">
  <attribute name="name" type="string"></attribute>
  <attribute name="value" type="string"></attribute>
</complexType>
</schema>
```