Real-time Feature Detection in Mass Spectrometer Data

Hillman, Chris

Award date:
2018

Real-time Feature Detection in Mass Spectrometer Data

Christopher Hillman

Doctor of Philosophy

University of Dundee

2018

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am particularly grateful to Professor Mark Whitehorn who has been my mentor and supervisor for the past eight years whilst I studied part-time for MSc and PhD degrees. Mark has been a constant source of advice, support and encouragement and has always made himself available whenever I needed him. Under Mark’s supervision I have learned the true meaning of research and developed many skills. Amongst these skills, being able to communicate complex ideas effectively in writing and being able to think critically about others work have helped me in business and academic life. Without Mark’s influence, I would never have embarked on the journey that led me to write this thesis. Thanks also go to Andy Cobley who has been my second supervisor through most of the PhD process. Andy’s knowledge of the University of Dundee and the academic system in general have been invaluable. Andy has also been a major source of inspiration and support concerning all things technical. My thanks also go to Dr. Karen Petrie for stepping in as my supervisor at the end of process and providing support and guidance to ensure that I could submit this thesis.

The support and help from Professor Angus Lamond and his team in the life sciences laboratory at the University of Dundee have made a major contribution to the success of this research. Professor Lamond and his team have provided me with all the data and domain expertise that I needed, they have always been available for help, guidance and a cup of tea in the kitchen! I hope to continue to meet them on a regular basis.

I must also thank all my family and friends who have put up with me being absent, vacant and generally unavailable for the past six years while I was engrossed in the research and particularly in the final eighteen months while I was writing up the thesis. Finally, the biggest thanks go to my wife Heather, considering that we have been married for twenty-five years and I’ve been a part-time student for eight of them gives an idea of the sacrifices she has made so that I could reach this point. Without her support, encouragement and drive I could never have reached the end and completed this work.

DECLARATIONS

CANDIDATE’S DECLARATION

I, Christopher Hillman, hereby declare that I am the author of this thesis; that I have consulted all references cited; that I have done all the work recorded by this thesis; and that it has not been previously accepted for a higher degree.

SUPERVISOR’S DECLARATION

I, Mark Whitehorn, hereby declare that I am the supervisor of the candidate, and that the conditions of the relevant Ordinance and Regulations have been fulfilled.

SUPERVISOR’S DECLARATION

I, Karen Petrie, hereby declare that I am the supervisor of the candidate, and that the conditions of the relevant Ordinance and Regulations have been fulfilled.

ABSTRACT

Proteomics has become an essential component of systems biology in the quest for personalised medicine. Each of us has a unique biology and can respond in different ways to medical treatments. By analysing the complete set of proteins present in humans the field of life sciences is moving closer to the goal of being able to recommend specific drugs to specific individuals thus greatly enhancing the probability of a cure.

Extensive pre-processing of the complex files created by mass spectrometers during proteomics experiments is required before it is possible to gain any insight from them. A typical mass spectrometer run may produce forty thousand scans of data each containing around two thousand data points. If this data were to be laid out in a relational database schema, it would equate to eighty million rows of data per experiment. In a laboratory containing multiple machines running several times a day, this figure quickly reaches into the tens of billions of rows, which is a very significant amount of data to process. Many tools currently exist to carry out this processing but most focus on batch-based workloads where the mass spectrometer finishes its analysis, and then the data is processed on a file-by-file basis. The processing time can vary from hours to days leading to a substantial time lag before the results of the experiment can be examined. In addition, life science laboratories often carry out this work on the local storage of PC hardware creating a significant data management problem.

This research investigates the potential for processing proteomics data in near real-time using a parallel system. The focus is on the feature detection part of the mass spectrometer processing pipeline and how this could become part of an architected cloud-based or on premise solution. The experimentation involves using the MapReduce framework to enable running the feature detection algorithm in parallel on a horizontally scalable cluster of servers. Systems tested include Hadoop, Flink and Spark in both a batch and real-time streaming mode.

The work shows that it is possible to detect features in the mass spectrometer data using an “intra-file” parallelism. The term intra-file means that a data file is split into sections, which are then processed independently on the cluster and is a vital part of enabling feature detection in a streaming fashion. This is a major differentiation between this research and most current processing methods, which process complete files in a serial fashion. This work highlighted that it is highly relevant to consider the laboratory as an Internet of Things. This involves the data streaming from the mass-spectrometers in real-time to a central computing platform where the data processing is completed with contemporary open-source technology. Consequently, the research described in this thesis points towards the adoption of a distributed cluster-based architecture which will allow the processing of mass spectrometer output in real-time as it is generated. Making the results available as soon as the experiment has completed allows life scientists to iterate over a problem faster which will lead to quicker paths to insights.

LIST OF PRESENTATIONS GIVEN

1. 25th April 2012: Co-presenting work on Proteomics processing with Professor Mark Whitehorn at Teradata Universe conference in Dublin
2. 28th June 2012: Presented MapReduce methods for processing mass spectrometer data to the Hadoop User group, London
3. 23rd August 2012: Big Data training for Teradata, used mass spectrometer data processing as an example of the differences between SQL and MapReduce code
4. 17th, 18th January 2013: Lectures on Hadoop, MapReduce and the Hadoop Eco-system to Data Science MSc students
5. 21st February 2013: Presented Data Science Discovery with MapReduce and SQL at the Data Science meetup, London
6. 5th March 2013: Presented general session on Data Science using mass spectrometer processing as an example at the Hortonworks Hadoop breakfast briefing, London
7. 20th March 2013: Presented general session on Data Science using Mass spectrometer processing as an example at the Hadoop Summit, Amsterdam
8. 17th April 2013: Presented processing mass spectrometer data processing in Hadoop at Teradata Universe conference, Copenhagen
9. 25th April 2013: Presented work so far at the PhD Symposium, Dundee
10. 11th November 2013: Strata Conference London, Presentation on MapReduce techniques, use Proteomics as an example of efficient shuffling of data around a cluster
11. 13th December 2013: Birkbeck College, University of London guest lecture to MSc in cloud computing students on MapReduce concepts and uses with scientific data, using mass spectrometer data as an example.

12. 11th February 2014 – Teradata Universe Dubai
13. 12th December 2014: Birkbeck College, University of London guest lecture to MSc in cloud computing students on MapReduce concepts and uses with scientific data, using mass spectrometer data as an example
14. 15th, 16th January 2015: Lectures on Hadoop, MapReduce and the Hadoop Eco-system to Data Science MSc students
15. 26th April 2015: Presented work in progress at the PhD Symposium, Dundee
16. 13th October 2015: Presented real-time mass spectrometer data processing with Apache Flink at Flink Forward conference, Berlin.
17. 13th, 14th January 2016: Lectures on Hadoop, MapReduce and the Hadoop Eco-system to Data Science MSc students
18. 19th April 2016: Presented real-time processing of mass spectrometer data at Teradata Universe conference, Hamburg.
19. 24th May 2016: Presented work in progress at the PhD Symposium, Dundee
20. 19th April 2016: Presented real-time processing of mass spectrometer data at Teradata Partners conference, Atlanta.
21. 5th December 2016: Presented real-time processing of mass spectrometer data at IEEE Big Data conference, Washington
22. 17th, 18th January 2016: Lectures on Hadoop, MapReduce and the Hadoop Eco-system to Data Science MSc students
23. 24th May 2017: Presented work in progress at the PhD Symposium, Dundee

PUBLICATIONS

1. Hillman, Chris; Ahmad, Yasmeen; Cobley, Andrew; Whitehorn, Mark "Near Real-Time Processing of Proteomics Data Using Hadoop." *Big Data,* vol. 2, no. 1, pp. 44-49, 2014

   Publication based on the research described in chapter 5 (algorithm) and chapter 6 of this thesis

2. Hillman, Chris; Cobley, Andrew; Petrie, Karen; Whitehorn, Mark " Real-Time processing of proteomics data. The internet of things and the connected laboratory" *2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8,* pp. 2392-2399, IEEE, 2016.

   Publication based on the research described in chapters 6 and 7 of this thesis

ABBREVIATIONS

AMC	Aster Management Console
API	Application Programming Interface
BAR	Backup and Recovery
BLOB	Binary Large Object
BSP	Bulk Synchronous Parallel
CPU	Central Processing Unit
CQL	Cassandra Query Language
DDL	Data Definition Language
DLL	Dynamic Link Library
EPD	Enclyclopedia of Protein Dynamics
ESI	Electo-Spray Ionisation
ETL	Extract Transform Load
FIFO	First in First out
FLOPS	Floating Point Operations per Second
FT-MS	Fourier Transform Mass Spectrometry
GFS	Google File System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HDFS	Hadoop File System
HPC	High Performance Compute
IDE	Integrated Development Environment
JIT	Just in Time
JSON	Javascript Object Notation
LC-MS	Liquid Chromatography Mass Spectrometry
LDBC	Linked Data Benchmark Council
MIAPE	Minimum Information about a Proteomics Experiment
MISD	Multiple Instruction, Single Dataset
MPI	Message Passing Interface
MPP	Massively Parallel Processing
MS/MS	Tandem Mass Spectrometry
MS2	Tandem Mass Spectrometry
M/Z	Mass to charge ratio
PaaS	Platform as a Service
PTM	Post Translational Modification
PVM	Parallel Virtual Machine
RAM	Random Access Memory
RDBMS	Relational Database System
RDD	Resilient Distributed Dataset
RT	Retention Time
SaaS	Software as a Service

SILAC	Stable Isotope Labelling with Amino Acids in Cell Culture
SIMD	Single Instruction, Multiple Dataset
SLA	Service Level Agreement
SPC	Storage Processing Council
SPEC	Standard Performance Evaluation Corporation
SQL	Structured Query Language
SSD	Solid State Drive
TOF	Time of Flight
TPC	Transaction Processing Council
TPP	Trans-Proteomic Pipeline
YARN	Yet Another Resource Negotiator
YCSB	Yahoo Cloud Serving Benchmark

1 INTRODUCTION

1.1 PARALLEL COMPUTING

The challenges of ever-increasing data volumes, the complexity of data types and high rates of data ingestion have led to horizontally scalable compute clusters achieving a high profile in academia and business; the whole field has recently been known simply as Big Data. Many data storage and processing challenges exist with scientific data, including proteomics [1], that are directly relevant to Big Data. Another term, the "Internet of Things" is used to describe connected devices and shares many challenges with Big Data, for example, large complex data sets generated in a short space of time [2]. As more machines and devices emitting sensor and other data are connected via networks, the challenge of processing and storage continues to increase year on year.

The concepts of parallel distributed computing have had a long history [3]. Scaling in computer terms can be either “scaling vertically” also known as “scale up” or “scaling horizontally” also known as “scale out”. Vertical scaling involves adding faster CPUs with more cores, faster memory and larger storage to a single computer. Horizontal scaling involves adding more computers which are connected via a network. Connected clusters of computers require software specifically designed to support parallel computation and several key frameworks, for example Message Passing Interface (MPI) and Bulk Synchronous Parallel (BSP) [4], have been popular and used in supercomputers such as the Cray and general purpose High Performance Compute (HPC) environments. More recently, the focus on parallel computation using clusters can be traced to the Google file system (GFS) and the MapReduce processing model described in Dean and Ghemawat’s paper [5] published in 2004. This line of research led to the creation of Hadoop by Doug Cutting and the resulting eco-system that has built up around the core Hadoop engine. Hadoop has since become a major player in cluster computing in both business and academic environments. The MapReduce framework is distinguishable from parallel processing frameworks such as MPI and BSP in that it operates in a “shared nothing” environment. Shared nothing means that individual computation tasks operating in parallel cannot communicate with other tasks and must operate in complete isolation.

This thesis argues that a modern life sciences laboratory could benefit from an Internet of Things architecture where the instruments in the laboratory are connected to a central processing cluster, either locally hosted or cloud-based. The ability to see results from experiments in real-time as the measurements are taking place has many advantages including quality control and comparison to previous experiments. In addition, simply seeing results faster allows for more iterations and more time for experimental design.

In many cases, it is far from simple to take an existing process that runs on a PC and attempt to create a parallel algorithm capable of running on a distributed cluster. Often new algorithms need to be designed to allow efficient parallel processing, a problem compounded by a shared nothing environment where only part of the data is visible to an individual task. Other issues that arise when exploiting systems such as Hadoop include the availability of skills to program and support them [6].

1.2 PROTEOMICS

Proteomics can be defined as the large-scale study of protein properties, such as expression levels (how much of the protein exists in the sample), modifications and interactions with other proteins. By studying proteins and their properties it is possible to gain a deeper understanding of how proteins should function in healthy cells compared with diseased cells. This knowledge is essential if better healthcare and personalised medicine are to become a reality. Personalised medicine allows for the best treatment to be administered to a patient with a far greater chance of success and fewer contraindications. Proteomics is a key part in providing this personalisation [7]. The goal of mapping out the entire human proteome is a more complex task than that of the mapping the human genome, which has already been achieved. The number of proteins is estimated at between one hundred thousand, and one million [8], many times more than the number of genes. The proteome is also dynamic, constantly changing in response to environmental factors. Identification and quantification of the proteome using techniques such as liquid chromatography-mass spectrometry (LC/MS) allows Life Scientists to investigate how different environments and compounds affect the protein expression in cells [57][9].

The main instrument used to measure the type and quantity of proteins in cells during proteomics experiments is a mass spectrometer. Mass spectrometers work by ionizing molecules to induce a charge, which causes them to be attracted to a detector [66][10]. Complete protein molecules are too large to be processed by most mass spectrometers therefore the proteins are broken up by a chemical process into smaller pieces called peptides. It is these peptides that are detected by the machines and recorded in the output. The larger, heavier peptides travel more slowly than the smaller, lighter ones and thus the different types of molecule separate out and arrive at the detector at different times.

1.3 FEATURE DETECTION

A typical experimental sample takes between two and four hours to process and on completion the mass spectrometer produces data in the form of spectra. These spectra require extensive processing before Life Scientists can interpret the results. Currently most spectra are processed using PC-based software and the time to produce the output can be measured in days. The processing itself involves several distinct steps; this thesis focuses on the first step, feature detection (also known as peak-picking). The reason for this focus is that the second step, peptide identification, has already been the subject of published research, described in Chapter 3, and several parallel solutions exists, for example the Hydra solution published by Lewis et al. [79]. Feature detection in mass spectrometer data is a smaller task than the later step of peptide identification, however the feature detection process still takes a significant amount of time to complete.

Feature detection involves finding the peaks in spectra produced during the individual mass spectrometer scans (which are taken approximately five times a second) and then matching these peaks over time [11]. These features (or peaks) represent the peptide molecules; the goal of the feature detection step is to produce a list of the mass and abundance of all the peptides in the biological sample. This peptide list is then used as the input to the protein identification step of the data processing.

The parallel feature detection algorithm developed during this research has been benchmarked for speed of processing and accuracy of output against MaxQuant. MaxQuant is a software package created and maintained by the Max Planck Institute [12]. It is a leader in the field of proteomics data processing and the MaxQuant package compares favourably with other processing software [13].

1.4 PROBLEM STATEMENT

Proteomics is an important branch of life sciences as mapping the human proteome and how it responds to environmental factors is at the forefront of medical advances [14].

Currently many researchers store and process data on personal desktop and laptop computers, which causes two significant problems. Firstly, they must wait for the results, this can mean hours or even days of delay before results are processed and available. Secondly, they become involved with complex data management tasks such as data movement, duplication, reprocessing, backup and recovery.

Mass spectrometers produce data in a vendor proprietary format commonly known as a RAW file. Since these files typically utilise a complex binary format, the proteomics community has introduced several standard file-formats for mass spectrometer output to facilitate storage and data-exchange. The mass spectrometer RAW files include both mass-to-charge and abundance (known as intensity) data; this data is the basis of feature detection calculations and contains information about the molecular weight and quantity of the molecules detected by the mass spectrometer. The RAW files also contain a great deal of meta-data regarding the conditions of the experiment itself. A recent and widely accepted data-exchange format is called mzML [15]. mzML has proven very stable and useful, however, it is an XML based format, which makes it inefficient for parallel processing. This is because distributing XML files on a cluster either means keeping the file intact on a single node or providing some mechanism to ensure the file structure is not compromised when it is split into sections for distribution. This complication is not present when using simple file formats with one record per line. To address this, a new data format has been designed during this research, see Chapter 5, Section 5.3 for details. This new format does not contain the meta-data and presents the mass and intensity information in a simple row based tab delimited format which makes it ideal for processing on a distributed cluster.

In a recent paper, major contributors to the MaxQuant software [16] discuss the relative benefits of a compute cluster over the use of fast “gaming-style” PCs where the unit of parallelism is at the RAW file level (the results from a single experiment are contained in a single mass spectrometer RAW file). However, in this thesis it is argued in Chapter 7 that a real-time architecture engineered specifically for the task of feature detection has many benefits including a faster processing time. It is also argued in Chapter 6 that the unit of parallelism for processing should be at the scan level and not at the file level. This more detailed level of parallelism allows the exploitation of the resources available in larger clusters and near real-time processing. The proposed solution not only addresses speed issues but also architectural issues such as minimizing data movement and manual tasks.

1.5 MAJOR CONTRIBUTIONS

In summary, the major contributions to knowledge of this thesis are as follows:

1. A parallel implementation of the feature detection algorithm used in the MaxQuant software written using Java.
2. Validation that intra-file processing in parallel is possible (as opposed to other research which uses the file as the unit of parallelism). This is important as intra-file processing makes real-time stream processing a possibility. In this context one file equates to one run of a mass spectrometer and one experiment.
3. A proposed file format suitable for parallel processing of proteomics data. This is required as the current XML-based file format, mzML, is a poor solution for intra-file parallel processing.
4. A thorough investigation of using a parallel algorithm for feature detection on a compute cluster. This includes a detailed understanding of the constraints on parallel tasks in a proteomics context, the limitations of batch processing and the implementation of a stream processing architecture.

Additionally, the following resources have been made available to the research community

- All the source code for the algorithm and implementations using HDFS, HBase, Cassandra, Spark and Flink.

1.6 THESIS STRUCTURE

Chapter 2 reviews the concepts and history of parallel computing including shared-nothing processing and the development of MapReduce and Hadoop. The chapter concludes with a discussion on system performance benchmarking.

Chapter 3 discusses the field of proteomics; it describes the process of experimentation using mass spectrometers and the resulting data sets. Also, algorithms for feature detection and current software in use are described. This chapter also includes a description of the challenges of processing proteomics data.

Chapter 4 details the current application of clusters and parallel processing in proteomics and critically reviews other research in this area. This chapter discusses the use of Cloud and Big Data technologies for proteomics data processing and concludes with a summary of the background and related work and the main research question.

Chapter 5 defines the proposed methodology for parallel feature detection. The environments used for development, testing and benchmarking are also explained. The method for validating results and testing for accuracy is specified along with the data sets used in the benchmarks.

Chapter 6 describes the proposed parallel algorithm and its implementation in Java. This chapter introduces the limitations of batch processing proteomics data on a parallel cluster. The results from benchmarking the feature detection are presented.

Chapter 7 proposes a real-time processing architecture using an adapted version of the parallel algorithm introduced in chapter 6. The changes required to move from a batch process to a stream process are detailed and results from experiments using stream processing engines are also presented.

Chapter 8 summarises the contributions and findings of this thesis and lists recommendations for future work.

2 PARALLEL COMPUTING

2.1 CHAPTER SUMMARY

This chapter presents a brief history of parallel computing including the introduction of MapReduce and shared-nothing cluster-based processing. The focus is parallel, cluster-based processing as opposed to other types of parallel processing such as multi-core or GPU-based.

Section 2 discusses parallel processing

Section 3 Introduces MapReduce and the Hadoop eco-system

Section 4 describes the process of benchmarking computing systems and includes a discussion specific to benchmarking parallel systems.

2.2 PARALLEL PROCESSING

To understand the need for and the evolution of parallel computing techniques in computer science, it is necessary to briefly review the general history of computers and their architecture and how these have developed over time. John Von Neumann's seminal report in 1945 defined the modern stored-program architecture for computers [17]**.** By making the only distinction between data and instructions that operate on data is the use to which each is put, this report formed the basis of how modern-day computers work. Note that there is some controversy over who first defined the architecture but this review does not cover this detail. The implementation of this design has led to a phenomenon known as the "*von Neumann bottleneck*" [18] which occurs when all of the data and the operations performed on the data must be moved between CPU and memory one byte at a time. In fact, the x86 family of processors use a modification of the von Neumann architecture called the Harvard Architecture, which allows a CPU to be both reading an instruction and accessing data in memory at the same time. This simultaneous activity is not possible with a strict von Neumann design since the instructions and the data share the same system bus. Along with the Harvard Architecture, changes to von Neumann's original model to overcome the bottleneck include the introduction of cache memory (a quantity of high-speed memory very close to the CPU), which has led to increases in speed.

Aside from the architecture, continuous improvements have been made to chip and component design, which led to Gordon Moore's observation in his 1965 paper [19] that "the number of transistors on integrated circuits doubles approximately every two years" – now known as "Moore's Law". This prediction has proved remarkably accurate and until recently increasing amounts of faster memory and faster CPUs could be relied upon to provide significant increases in speed. This phenomenon is known as frequency scaling and was the dominant force in CPU speed increases up to the cancellation of Intel's Tejas and Jayhawk processor projects in May 2004 [20]**.** The termination of these projects can be seen as the point at which frequency scaling reached a practical limit. However, currently Moore's law still holds approximately true because manufactures have started including two or more CPUs on the same chip, known as multi-core CPUs. These allow multiple instructions to be run at the same time using a programming technique called multi-threading, a form of parallel execution. However, program code needs to be written explicitly to take full advantage of the threads and hardware available.

The concepts of parallel computing have a long history. In a Turing lecture in 1988, Cocke stated “The search for future scientific computing performance has to concentrate on gross parallelism” [3]**.** A definition of Parallel Processing is *"the concurrent manipulation of data elements belonging to one or more processes solving a single problem"* [21]**,** the concept being that multiple CPUs can each process part of the data, with the resulting output combined into a single answer. A certain class of parallel computing is known as massively parallel processing (MPP); this involves multiple CPUs processing the workload simultaneously and communicating via a network. Early attempts at MPP computing include the ILLIAC IV [22] from 1971. With up to 256 processors the ILLIAC IV allowed a high degree of parallelism and was used by NASA for computational fluid dynamics problems. Other examples of early MPP machines include the cosmic cube designed and built at Caltech in the early 1980’s [23]. The cosmic cube research introduced the concept of nodes, which are computers networked together into a cluster. The method by which nodes communicated contributed to the design of the message passing interface (MPI) protocol, which has become a standard for parallel processing. Flynn describes four classifications of processing [24], [25] that have become known as Flynn’s taxonomy. The classes are based on the number of concurrent instructions and the number of available data streams. For instance, in a single instruction, multiple data set (SIMD) architecture a single instruction is applied to multiple sets of data in parallel. Whereas in a multiple instruction, single data set (MISD) architecture, multiple instructions run on a single set of data in parallel. The SIMD architecture is well suited to a set of problems know as “naturally”, “pleasingly” or “embarrassingly” parallel where a subset of the data can easily be processed independently of the rest of the dataset.

In the book “*Computer Architecture: a quantitative approach*” [26] Paterson describes the performance growth in PC architecture based computers versus mainframes or supercomputers such as those built by Cray Research Inc. He details how CPU speed increases in line with Moore’s law allow cheaper commodity hardware to rival the performance of the larger, far more expensive machines. Utilising such hardware in a network, the class of parallel computing known as cluster computing has become very popular. Cluster computing entered the mainstream with the introduction of the parallel virtual machine (PVM) software in 1989 [27]. NASA experimented with this technology in 1993 and this research led to the creation of the Beowulf cluster [28]. The Beowulf architecture is a set of requirements for high-performance clusters using Unix-style operating systems and PC-based architecture running libraries such as MPI and PVM. Beowulf allows a cluster of machines to operate as a single machine where the individual nodes typically have no screens, keyboards or input devices and are controlled from a central machine via remote logins. Beowulf clusters have been adopted in academia and commercial sectors as a way of providing enough computing power at a low price to process complex scientific problems [29].

In contrast with cluster computing where the nodes in the cluster will reside in the same physical location, distributed computing is a class of parallel computing where the nodes are separated physically and connected by a wide area network such as the internet. An example of distributed computing is that performed by SETI where subscribers' home PCs are used to process data when they are not in use for other purposes [30].

The computer architectures and programming techniques described require the processing components to be connected to each other so that messages can be passed between them. There are many ways to connect processors in an MPP system, for example Mesh, Binary Tree, Hypertree and software frameworks such as Message Passing Interface (MPI) and Bulk Synchronous Processing (BSP) can be used to manage the messaging. Interconnections between processing units and passing messages between them result in an overhead, which limits the total speedup possible. Many have studied the behaviour of parallel systems and several prominent theories exist that quantify the speed increase of a process running in a parallel rather than a serial fashion. One of the earliest is Amdahl's law [31]; this states that the maximum speedup of a parallel process is limited by its sequential fraction. Amdahl's law is based on the assumption that processes have some parts that cannot be run in parallel. Regardless of the number of processors working on the problem, the total time taken to run the complete process is restricted by the total time taken to execute the serial part of the process. During a later re-evaluation of Amdahl's work by Gustafson [32], known as Gustafson's law, it is argued that rather than calculating speedup using a fixed data size, as Amdahl assumes, programmers tend to use all of the resources available in a system to process an amount of data in a fixed time. This means that larger and larger clusters are required to keep the processing time fixed as the amount of data increases. This research will evaluate the pre-processing steps for mass spectrometer output to identify parallel and sequential portions and calculate the maximum possible speedup.

Another aspect of parallel computing is that of scheduling; once a task is split into pieces that can run on separate nodes, controlling the order and location of these pieces must be done by a scheduling system. One method is prescheduling which decides where each piece of the process is run before execution, depending on what data is where in the system: early parallel systems used this approach. In contrast, self-schedulers allocate the next piece of code to execute to the next available node, which results in a higher utilisation of a system.

An important consideration for cluster architectures is how to manage the replication of data around the nodes. If a data read occurs before data is fully replicated then an inconsistency may occur as one node reports a different value to another node. Eric Brewer developed Brewer's or CAP theorem [34] to describe parallel architectures in terms of three areas:

- Consistency – Every data read should receive the most recent data or return an error
- Availability – Every data request receives a response but there is no guarantee that the response is the latest data available. The response must be valid and not an error
- Partition Tolerance – The system continues to operate in the event of a network failure or messages simply being dropped between several nodes

Brewer stated that where data partitions exist only one of consistency or availability can be guaranteed, leading to the common “any two from three” description of CAP theorem. Relational databases will choose consistency, opting for an error message rather than returning anything other than the most recent data. NoSQL systems such as Cassandra (described in Section 8 of Chapter 5) will choose availability, therefore returning the latest version of the data that is available rather than an error. Developments such as Google Spanner claim to be able to guarantee all three states because of the definition of “availability”. A system can be highly available without being one hundred percent available [141].

2.3 MAPREDUCE AND HADOOP

Dean and Ghemawat published a paper defining the MapReduce style of implementing a parallel processing framework in 2004 [5]. MapReduce is a general-purpose framework allowing large-scale computations to be run in parallel across a large cluster of computers. The framework is based on the simple notion that a process can be split into one of two basic operations map and reduce. A map task processes data one record at a time. The other type of operation is a reduce task which takes multiple rows of input and can perform aggregation type operations [35]. MapReduce was defined with some key concepts in mind:

- Reliability when running on commodity hardware
- Simplifying the task of taking a process and running it in parallel.
- Scalability to handle exponential increases in website traffic

The use of a map task to process data record by record has led to the term “schema on read” or "schema-last" [36], which describes the process of reading in data in a raw, unchanged format and using the map task to apply a relevant schema on the data at run time. This contrasts with a “schema on write” or “schema-first” style of data management that is very commonly found with relational databases. Schema on write describes the method of defining a fixed schema and transforming data to fit this schema as it is loaded.

MapReduce code is generally executed on clusters run in a “shared nothing” environment where each node has its own CPU, memory and disk space and the connections between nodes are handled by a TCP/IP network. This configuration makes the task of adding more nodes to an existing cluster a simple exercise. Since no reconfiguration of existing nodes is required, the cluster can easily be scaled out to handle larger workloads or process a given workload in a shorter time. In this way, the programmer is abstracted from the details of scheduling and distributing parallel processes and dealing with node failures as these are handled entirely by the system. The basic unit of data in the MapReduce framework is a key-value pair; these simple structures consist of an identifier (the key) and some data (the value). An example of a key-value pair could be when processing a standard text file with end of line characters, in this case, the key is the byte offset into the file, and the value is the actual text up until the next line break.

The value part of the key-value pair can be a single piece of data or something much more complex such as a binary object. In the MapReduce framework, the passing of key-value pairs occurs between different stages of the pipeline, for example between a map task to a following reduce task. Note that the framework prohibits the passing of information between tasks at the same stage of the processing pipeline, for example between two map tasks.

This may seem restrictive at first, but this simplicity is a factor, along with fault-tolerance, that has led to the success of the model. Removing the need to code for the interactions between tasks by message passing speeds up development and relieves some of the burden of understanding parallel systems for the programmer. Another key feature of MapReduce programming is the distribution of program code. In a conventional system, a processing unit receives data from the network and executes the code. In the MapReduce framework, the code is distributed to the node where the data resides to reduce data movement across the network. Processes are dynamically scheduled, and due to the data replication and distribution, each task has several options as to where it could run to ensure all data is processed. This self-scheduling allows some flexibility in case of a slow running machine or process. Note that MapReduce is a framework for executing program code in parallel, as are frameworks such as MPI, whereas Beowulf and PVM are systems for creating clusters of commodity hardware.

Hadoop is a parallel processing system based on the MapReduce framework, created by Doug Cutting, an employee of Yahoo [37] and more recently, Cloudera. Hadoop was designed to be run on cheap commodity hardware. Since commodity hardware, particularly disk drives, can have high failure rates, it is necessary to replicate and distribute data across the cluster to achieve reliability. Therefore, when data is loaded it is split into blocks (typically large blocks in multiples of 64Mb). The default behaviour is to replicate these blocks three times with two copies held on the same rack and one on a separate rack. This operation means that two of the servers in the cluster containing replicas of the same data block could fail completely, and all data would still be available. Hadoop very quickly gained popularity as a platform to process large data sets that do have a fixed schema. While it can be said that the use of Hadoop in a business setting is still in the early stages of adoption, its popularity continues to grow [38]. Since its original inception, Hadoop has undergone much development, and a considerable amount of effort has gone into overcoming some of the perceived shortcomings of security, accessibility and recovery. The next chapter reviews the current use of Hadoop, MapReduce and parallel computing in general for processing proteomics data.

2.4 BENCHMARKING

Before any testing could be carried out to determine the best platform for the proposed parallel feature detection algorithm, a set of rules had to be defined. These rules specify what was measured and how the measurement should take place. In computing terms, these rules and the tests that are performed and measured are collectively called a benchmark. The intention was to provide a quantitative measure of relative performance between platforms along various dimensions to make a comparison between them. General computing benchmarks exist such as Whetstone and Dhrystone [39] which address specific computing performance areas such as Floating Point Operations per Second (FLOPS).

More specific benchmarks exist for relational databases that are well established such as those detailed by the Transaction Processing Council (TPC) which cover various database operations such as Decision Support [40]. These are widely used and are accepted by many vendors and customers of relational database software [41]. Other notable database benchmarks include the Standard Performance Evaluation Corporation (SPEC) and the Storage Performance Council (SPC), each aimed at providing a measure of the performance of different aspects of database operations.

Benchmarks seek to provide a level platform for testing the efficiency of different solutions and provide rules to cope with various areas. These include [42] :

- Hardware specification such that different platforms can be evaluated fairly
- Explicit instructions for implementing the benchmark. The benchmark should not test how well the code was written, or the data model designed, as this is not generally a factor of the platform being tested.
- The input data for the benchmark
- The data model and any relevant schema
- Specific queries or workloads designed to test certain parts of a system
- The number of runs for each workload and how results are obtained and reported

The benchmarks mentioned above provide many instructions for setting up the testing including architecture designs and data models for table definitions and artificial workloads with which to run the tests. These rules attempt to ensure that the benchmark is fair and measures the intended element of the system. Certain database vendors will release benchmarks of their own, but these are often regarded as more for marketing purposes than for serious testing as the tests carried out maybe biased towards the strengths of the platform in question. Questions have also been raised over the practice of vendors tuning their products and specifying systems well in excess of what would be needed in a real-life situation specifically to gain artificially high scores in benchmark tests; a practice known as Benchmark escalation [43]. Examples of the areas that are commonly included in database benchmark tests include:

- Scalability (adding or reducing components)
- Availability
- Redundancy
- Security
- Disk I/O
- Timings of certain tasks specified in the benchmark

2.4.1 BIG DATA BENCHMARKS

The benchmark situation is much less clear and well defined for cluster-based parallel systems, which can be labelled as big data processing systems. As these systems are recent innovations in data management and processing, existing database benchmarks which relate to transaction processing or business intelligence queries may not be relevant. Older benchmarks exist such as the NAS parallel benchmarks [44] aimed at supercomputers. There are also more recent benchmarks such as Rodinia designed for heterogeneous clusters and newer forms of hardware, for example, GPU processing [45].

Attempts have been made to introduce benchmarks that are more relevant to the specific properties of big data processing. The TPC has produced a new benchmark

[42] that can be freely downloaded from the TPC website. This new benchmark is designed to test Hadoop systems and provide a measure for hardware, operating systems and Hadoop file systems. It provides performance and availability metrics. Other benchmarks are being designed to focus on specific use cases of big data technology, for example, the Linked Data Benchmark Council (LDBC) is producing a set of benchmark tests specifically to measure the performance of graph algorithms in Social Network processing. The state of big data Benchmarking and its future direction were discussed at a workshop in 2011 [47] organised by the Centre for Large Scale Data Systems from the San Diego Supercomputer Centre. An interesting point raised was that traditional relational databases are usually over specified when used to run benchmarks. In other words, the system is far more powerful than one that would be utilised in a real-life situation, whereas for a big data system the reverse is often true, i.e. the system used to benchmark can be smaller than a production system. Thus, an element of scalability is an important part of the benchmark results. One reason for this under-specification could be that it is hard to provide realistic workloads that cover the three elements of volume, velocity and variety [48], which are often used to identify big data, as noted by Ren et. al [49].

The proposed BigBench big data benchmark [50] sets out a framework based on the TPC tests which includes a data model, specific queries and synthetic data. Yahoo has produced specifications for two benchmarks, YCSB [51] and YCSB++ [52]. These frameworks specify testing areas critical to parallel clusters such as the ability to scale-out, elasticity (scaling to more nodes in a cluster while the cluster is in use) and high availability. Recognising that certain trade-offs are made when designing a system, they also test read and write performance as separate measures.

Most of the systems used in this research provide simple benchmark tests as part of their distribution. These include the dfsio and terasort benchmarks for Hadoop [142] and the Cassandra stress test. For many, the terasort benchmark is a standard way of testing the performance of a cluster [53]. This benchmark simply measures how fast a system can sort a terabyte of data and it is supported by most platforms. Terasort is also part of the BigBench big data benchmark discussed above and itself based on the work of Jim Gray [54].

2.5 CONCLUSION

Parallel computing is very important for processing large complex datasets and has had a long history. While some early methods of parallel computing involved complex programming, and required a low-level understanding of parallelism, MapReduce programming abstracts much of the detail away into two processing primitives called mappers and reducers, described fully in Chapter 4, Section 4. The MapReduce style of parallel processing was presented in a paper by two Google engineers in 2004 [5], which led to the creation of Hadoop by Doug Cutting [37]. Hadoop has been widely adopted by businesses and academia as a means to process complex workloads in parallel and has several benefits including:

- Reliability when running on commodity hardware due to built-in redundancy
- Simplifying the task of taking a process and running it in parallel.
- Horizontal Scalability

The whole field of parallel processing and complex data sets has recently been known simply as Big Data and this term has also been applied to compute clusters and software processing frameworks. In order to test and compare Big Data systems benchmarks are being developed to compare relative performance. These benchmarks differ from older tests used to benchmark relational database systems in that they are designed to test the three main elements of a Big Data system, the volume of data, the velocity with which the data arrives and the variety of data formats that can be processed. As explained in Chapter 3, the data created by mass spectrometers has characteristics matching all three of these elements.

3 PROTEOMICS

3.1 CHAPTER SUMMARY

This chapter presents a literature review regarding current methods for processing the data output from mass spectrometers during proteomics experiments.

Sections 2 and 3 provide background information describing proteomics and the use of mass spectrometers.

Section 4 describes the challenges related to data processing and feature detection from the spectra.

Section 5 discusses the feature detection process and its place in a more general proteomics processing pipeline.

3.2 PROTEOMICS

As with genomics, where the field of functional genomics evolved to investigate the function of genes following the mapping of their sequence in an organism's DNA [55], proteomics can be thought of as an offshoot of functional genomics where the Proteome is the entire set of proteins expressed by the genome of an organism. The goal of proteomics is to map the proteome and explain the function of each of the proteins it contains [56]. Proteomics is, therefore, the study of protein properties within an organism. It has become an essential component in the field of systems biology, which aims to describe biological systems by integrating detailed data from diverse domains. Important properties of the proteins studied include quantities (expression levels), modifications and interactions with other proteins [57]. This work is important because proteins constitute the principal structures of an organism and play a pivotal role in the basic functional and structural framework of all cellular life. By studying proteins and their properties and by building up a view of the Proteome, it is possible to gain a deeper understanding of how proteins should function in a healthy organism. This knowledge is used in areas such as comparing healthy cells with diseased cells and exploring the effect of drug treatments.

Mapping the human proteome is a vast and complex task. To put this into perspective, the human genome is estimated to contain around thirty thousand genes [58] and with the discovery of new information this number is under constant review. In contrast, the human proteome may contain over a million proteins and is also under constant review. Following their synthesis, proteins can also undergo post-translational modification (PTM) [59]. A PTM is a chemical change to the protein structure through the addition of, for example, phosphates. Importantly the current understanding of PTMs, of which there are over three hundred different types, is that they are not predicted or governed by the genome, which means that proteomics experiments are necessary to identify and quantify them. The addition of PTMs to the proteins greatly increases the number of identifications required.

The human proteome project [60] is a global project designed to map the entire set of proteins found in humans protein set. Barriers to completing the mapping task include the lack of understanding of the underlying genes, the sheer number of proteins and possible modifications, quality of experimental data and the sharing of information between organisations. Hood et al. [9] also make one of the first references to moving data resources to a cloud-based environment. The potential benefits indicated include managing large-scale data analysis and sustainable data repositories. Identifying the need for collaborative data repositories that can be accessed online in the early stages of proteomics led to the establishment of the open-access protein sequence database SWISS-PROT [8]. The goals of SWISS-PROT include the standardisation and sharing of the results of proteomic experiments, minimising the redundancy of information caused by submissions from many institutions and also simplifying the integration with other databases.

Typical proteomics research involves taking cells from an organism and preparing a sample in such a way as to remove as many contaminants as possible while retaining as many of the proteins present as possible; contaminants include the materials used in the culture of cells and other laboratory contaminants [61]. A further complication is that the quantity of a protein expressed varies enormously. While some proteins occur in small amounts, others are more abundant and found in large quantities, which can make it difficult to detect low-abundance proteins [62]. Following the culture and purification of a sample, the most accurate method used for analysis is Mass Spectrometry, which is fast and allows a high throughput of samples. The discussion of standard instruments and experimental techniques is carried out in the next section of this review

3.3 MASS SPECTROMETRY

In the context of proteomics, a mass spectrometer is an instrument used to measure the mass and abundance of molecules in cell samples. Physicist Joseph John Thomson conceived mass spectrometers in the late 19th century [63]. While researching the nature of cathode rays, Thomson noted that they mark the paths of charged particles, which could be deflected by a magnetic field towards a detector. Francis Aston continued this research and built the first machine, which at the time was called a mass spectrograph, winning the Nobel Prize for chemistry in 1922 [64]. Continuous improvements and modifications have occurred, resulting in the modern instruments we see today. Recent advances in mass spectrometry have led to "next-generation proteomics" where the resolution and accuracy of the device allow for the measurement of tens of thousands of molecules. Comparing this to the measurement of hundreds of molecules possible with machines built before the turn of the century demonstrates the extent of the technical advances [65] and the consequent increase in data processing required.

Although there are several types of machine with different methods for processing and measuring samples, there is a common pattern to a proteomics experiment involving mass spectrometry. During an experiment, a mass spectrometer will perform different tasks to measure the abundance and mass of the molecules that it takes as input as depicted in Figure 1.

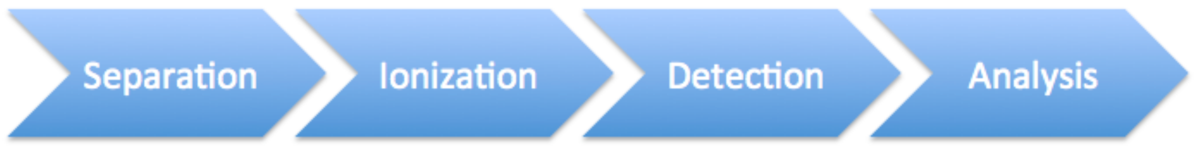


FIGURE 1 HIGH LEVEL PROCESS OVERVIEW - THE STAGES OF PROTEIN IDENTIFICATION

Separation is the process of simplifying a complex mixture of proteins by separating molecules from each other before introducing them into the mass spectrometer. A common method of separation used is liquid chromatography; when used in combination with mass spectrometry the technique is known as LC-MS. This technique involves passing the sample through a column, which is a narrow tube containing a gel material that partially and differentially binds to molecules meaning that they migrate through the column at different speeds depending on their properties [66]. In this way, a liquid chromatography stack feeds a stream of output molecules into the mass spectrometer that are sorted according to their properties.

The next step is ionisation, which is the process of charging the molecules so that an electromagnetic field can attract them. Electrospray ionisation (ESI) is a very common technique used in contemporary mass spectrometers due to its ability to ionise heavy molecules [67]. Note that all the data employed in this research has been output from mass spectrometers using ESI.

Following ionisation, the machine takes a scan of the ions it contains at regular intervals. The scan is taken by applying an electromagnetic force to the ions, which pushes a selection of them into the instrument's mass analyser. It is important to note here that each of the scans is a sample of the contents of the original input. It is inevitable that some of the molecules are not measured as they flow through the machine between scans. For this reason, there can be difficulties in reproducing experimental results exactly. The point at which the scan is taken is known as the retention time (rt).

Analysis of the ions is carried out by measuring their quantity and mass to charge ratios. This analysis is done inside the mass analyser of which there of four main types in common use. These are the Quadrupole, which uses four charged rods to trap ions in an electromagnetic field, the oscillation of molecules between these rods allows the mass to charge ratio to be measured. The Orbitrap (or ion trap) captures ions in a similar way to the Quadrupole. It is a highly accurate detector that can detect ions at a very high resolution [68]. Other types of analyser are the time of flight (TOF) and Fourier Transform Ion Cyclotron (FT-MS) [66]**.** Figure 2 shows a schematic of a Q Exactive plus mass spectrometer, which uses an Orbitrap ion detector in conjunction with a quadrupole.

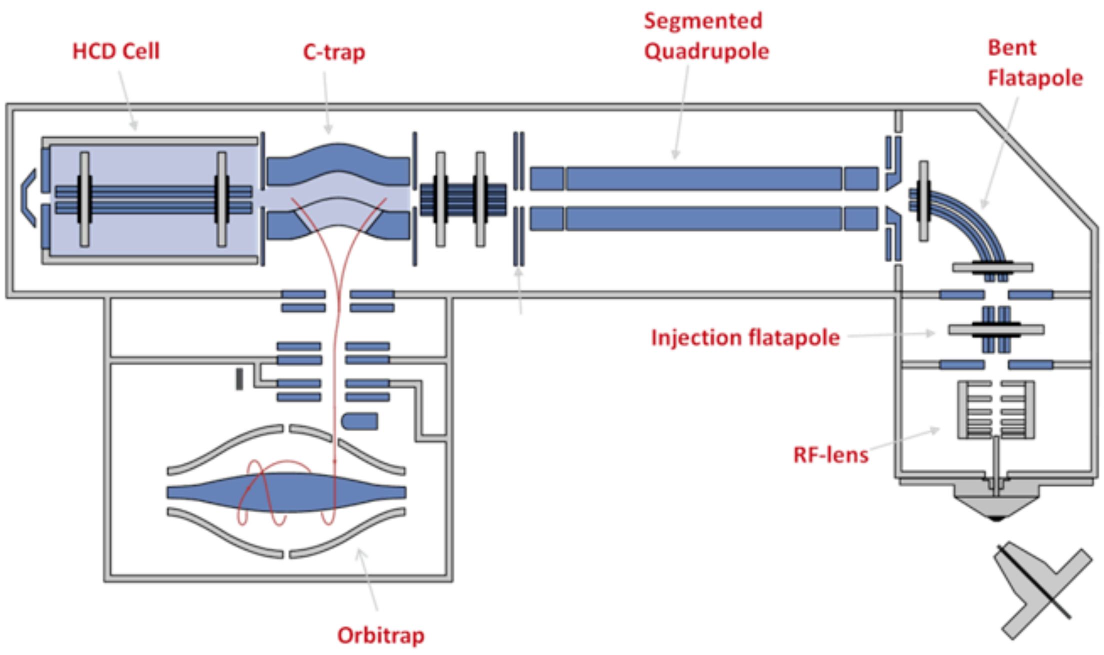


FIGURE 2 SCHEMATIC OF A Q EXACTIVE MASS SPECTROMETER - SOURCE THERMO SCIENTIFIC [139]

Isotope “labelling” is a quantitation method of analysing ions from different samples in the same experiment. Using this method cells from, for example, a diseased and a treated cell could be analysed and measured under the same conditions and at the same time. One method for labelling Isotopes is called Stable Isotope Labelling by Amino Acids (SILAC) as described by Ong and Mann [69] and used extensively at the University of Dundee. SILAC experiments involve incorporating the amino acids arginine and lysine into samples before the liquid chromatography phase of the experiment, therefore, providing a mechanism for four different samples (no treatment, lysine, arginine and both lysine and arginine). The incorporation of the amino acids increases the mass of the ions by a known amount. In this way, the isotopes, otherwise called peptides, can be identified, which leads to the identification of the proteins themselves [70].

The sample processing techniques and the equipment described above do not allow the determination of the volume of complete proteins required by most proteomics experiments. Complete proteins are large molecules which mass spectrometers have difficulty in measuring. The large complete protein molecules will need to be isolated and measured one at a time, a process that is known as a protein-centric or “top down” approach. If the experiment requires the measurement of the existence and abundance of proteins on a large scale, then it is necessary to break proteins into smaller pieces before introducing them to the mass spectrometer. Cutting the large protein molecules into smaller pieces is accomplished by using a digestive enzyme that breaks the protein molecule chains in a predictable way. The resulting smaller molecules are called peptides. The process of deducing the parent protein by detecting these peptides is called “bottom-up” proteomics. The bottom-up approach allows the analysis of large proteins and very complex samples.

Many experiments will combine the bottom-up approach with a process called tandem mass spectrometry, also known as MS/MS or MS2 [71]. The tandem mass spectrometry technique involves using two mass analysers in the same machine: effectively the machine contains two mass spectrometers acting in sequence. The first mass spectrometer operates as described previously; molecules enter the instrument from the liquid chromatography phase, are ionised through ESI and are measured by the ion detector in the mass analyser. At this point, the mass spectrometer selects specific ions for further analysis following the measurement phase. The usual criterion for selection is the abundance of particular ions, although the mass spectrometer can be set up to select specific mass to charge ratios if required. The selected ions (also known as precursors) are fragmented into smaller molecules by colliding them with a neutral gas in a process called collision-induced dissociation. The resultant fragments pass into the second mass analyser where the ion mass to charge ratios and abundances are again measured and reported; these scans are known as MS level two scans. The results from MS/MS are used in the later data processing stages to increase the confidence in the correct identification of proteins.

The Lamond Laboratory, School of Life Sciences, Centre of Gene Regulation and Expression, University of Dundee uses mass spectrometers manufactured by Thermo Scientific. These include the LTQ Orbitrap XL, LTQ Orbitrap VELOS and Q Exactive (Detailed specifications of the Thermo Scientific machines can be found online https://www.thermofisher.com/uk/en/home/industrial/mass-spectrometry/liquid-chromatography-mass-spectrometry-lc-ms/lc-ms-systems.html# last accessed 04/12/2017). The data utilised in this research has been provided by the Lamond laboratory and contains the results of actual proteomic experiments (further described in the methodology chapter). Figure 3 is a photograph of the laboratory showing the rows of mass spectrometers in use.

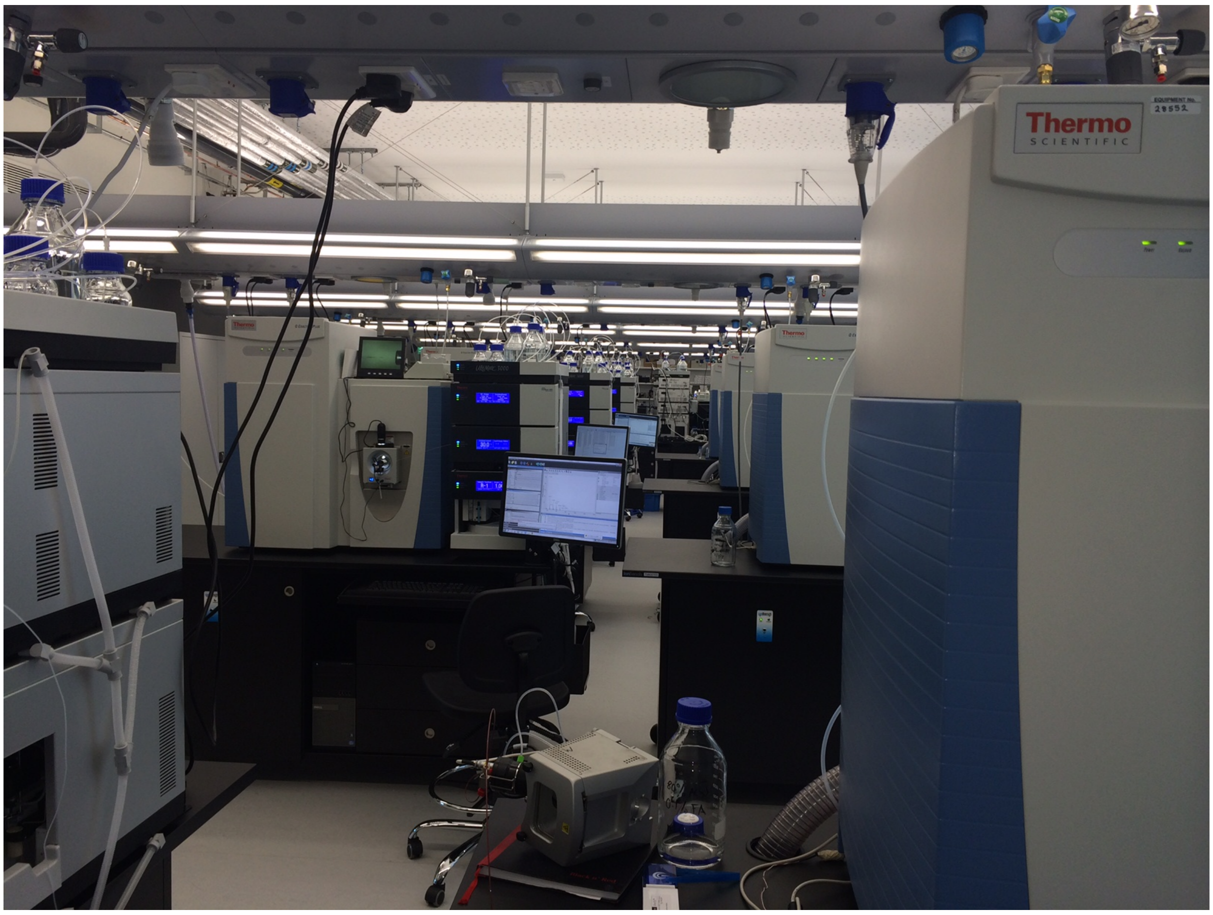


FIGURE 3 MASS SPECTROMETERS IN THE LAMOND LABORATORY, UNIVERSITY OF DUNDEE

The data that is output, when the mass spectrometer processing completes, is in the form of spectra. These can be plotted as shown in Figure 4 and consist of the mass to charge ratio (x-axis) and ion abundance (y-axis). At this stage in the process we do not know the charge of the ions, and therefore the measurement is the mass to charge ratio and not the actual mass. Mass to charge ratio is notated as m/z and is measured in a unit called a thomson.

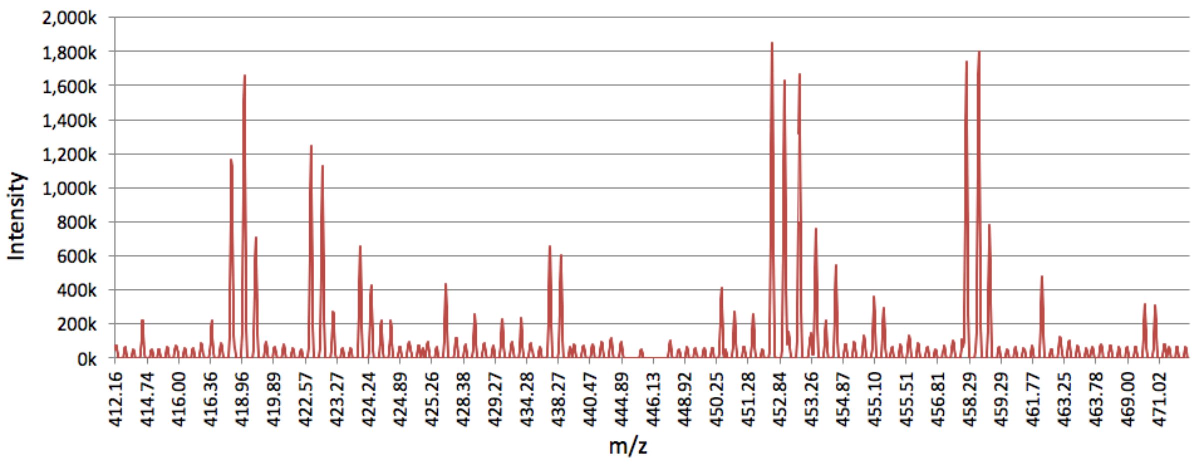


FIGURE 4 A SECTION OF A SINGLE MASS SPECTROMETER SPECTRUM

As mentioned, a scan is taken of the ions present in the mass spectrometer at regular intervals with each scan producing spectra as shown in Figure 4. Scans are recorded at a rate of around 5 per second producing up to 40,000 scans for a two-hour experiment. For the Thermo Scientific instruments, this data is output in a proprietary binary format called a RAW file. The binary RAW formats are efficient regarding data compression, however, they require specific software from the vendor to allow the data to be analysed. In fact, each of the instrument manufacturers uses a different proprietary format for their data output, which causes problems for sharing results and for any processing software, which needs to include parsers for specific formats. For this reason, amongst others that are discussed in Section 3.4.1, an XML-based format called mzML is commonly used to store the output from proteomics experiments [15]. Conversion software is available to create mzML files from the vendor specific format. Section 3.4.1 of this thesis contains a detailed described of the mzML format.

3.4 DATA PROCESSING CHALLENGES

During a typical proteomics experiment, a mass spectrometer will produce a large volume of data. The binary RAW files generated by the Thermo Scientific instruments are typically greater than one gigabyte and in a highly-compressed vendor specific format. Extensive processing of the spectra is necessary to identify proteins and gain any insights from experiments. Processing the data consists of several stages as below, each of which presents a series of challenges to the typical life sciences researcher and laboratory. This series of processing steps is referred to as the data processing pipeline [72]

- File parsing
- Filtering
- Feature Detection
- Alignment
- Normalisation
- Protein identification (database lookup)
- Results sharing

This research focuses on the feature detection part of the processing pipeline. Section 3.5 describes the feature detection process and algorithms in detail, but in essence, the reason for feature detection is to identify the ions in the spectra by mass, charge and abundance while avoiding false positives [72]**.**

3.4.1 DATA FILE FORMAT

As stated previously, a common output file format for mass spectrometers is required to provide a standard for the sharing of data. In response to this need, the proteomics community has produced specifications for several file formats. At the time of writing this thesis, the latest of these is called mzML. mzML is an XML format based on the consolidation of two previous formats (mzXML and MzData). The mzML format has been widely accepted and very stable with little revision since its inception in 2008 [15]. The mzML specification document describes the objectives of the format, which include comprehensive support for instrument output and the sharing of results and best practice. The XML specification uses a dictionary of keywords, which are included in the XML tag as attributes when required. The full specification and dictionary are linked here (last accessed 24/11/2016) http://psidev.cvs.sourceforge.net/viewvc/psidev/psi/psims/mzML/controlledVocabulary/psi-ms.obo.

The specification is very comprehensive and attempts to accommodate any details that specify a proteomics experiment. These include metadata such as the instrument used and its operational parameters, the cell sample analysed and any software used in pre-processing the output. Adding some complexity is the use of 32 or 64bit base64 encoded strings to hold the data. Data held in base64 strings requires a decoding step before use, but it does allow a high degree of accuracy to be kept while maintaining a compressed format [15]. Figure 5 presents a high-level schematic of the mzML format, this includes a header containing the metadata and the body section containing the actual data from the mass spectrometer scans. The format also allows for the base64 strings of scan data to be compressed using the Zlib algorithm. Zlib compression of the binary spectra data results in a decrease in the size of the file by a factor of two. Zlib is a well-known compression algorithm that usually exists natively on UNIX or Linux-based systems and is available in standard Microsoft Windows-based compression and decompression programs.

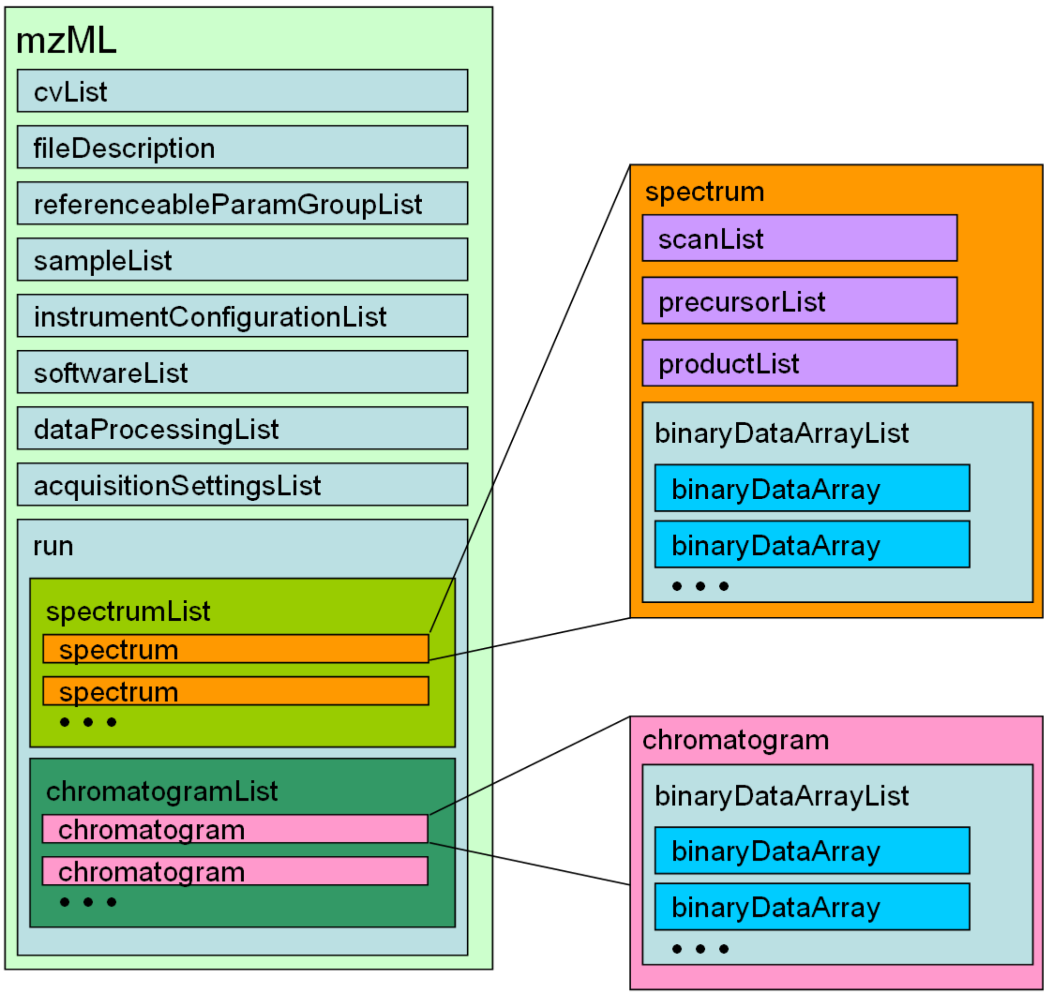


FIGURE 5 HIGH LEVEL SCHEMATIC OF THE XML-BASED MZML DATA FILE FORMAT

Even with this compression, the verbose nature of XML formats means that the size of the data file will increase significantly from that of the original binary format. The description of the files used in this research, the largest of which is 7.5 Gb, is in Chapter 5 Section 9. One final important point on the mzML format is that files formatted as XML are not well suited to parallel processing. For parallel processing to be effective, the data must be distributed evenly across the processing nodes, and It is a complex task to split an XML file into discrete units without breaking the format by splitting between XML tags. Note that the fast, efficient processing of data was never one of the prime target objectives for the mzML format. For this reason, the development of a new file format based on mzML was necessary. The successful design and testing of this new file format was an important step to enable the development and benchmarking of the parallel feature detection algorithm. Chapter 5 contains a detailed discussion of this format, which is designed to allow distribution of the scans around a cluster of computing nodes.

3.4.2 DATA VOLUME

The volume of data produced by proteomics experiments grows year on year for several reasons. Firstly, newer instruments have an increased level of sensitivity and can record more detailed data. For example, new devices can perform a further degree of fragmentation going beyond the MS/MS two-stage processing that is commonplace; this is done to increase the confidence in protein identification further. An increase in data volume also occurs at the second level of fragmentation. The machines that produced the data used in this research provide the level 1 scans as full profile spectra as opposed to the level 2 scans which were output as centroids. A centroid is a single point at the centre of a peak representing the mass to charge ratio in a spectrum (Figure 6) [12].

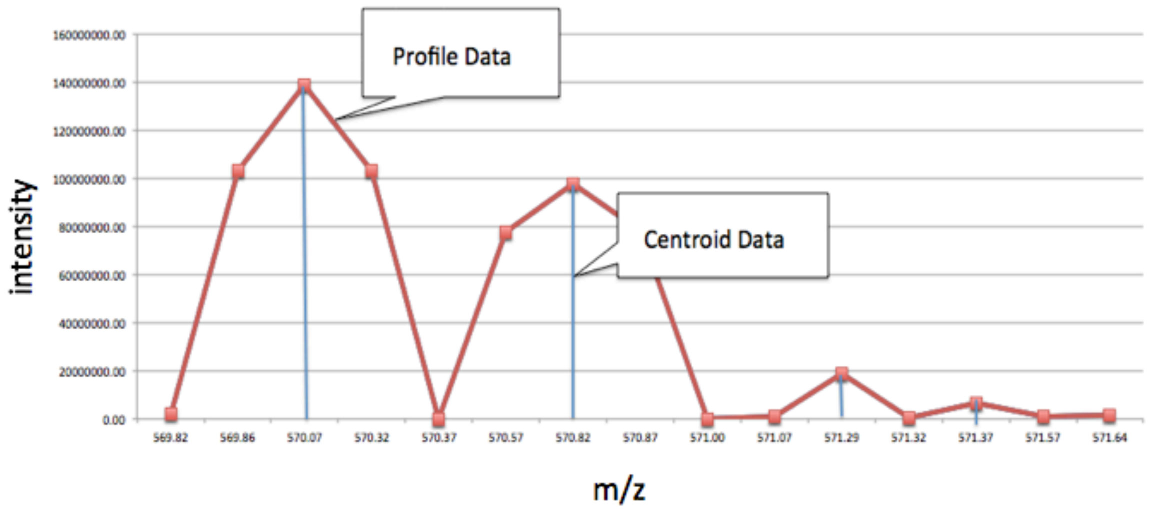


FIGURE 6 CENTROID VS PROFILE SPECTRUM DATA

However, more recent machines produce the data for level 2 scans as profiled spectra data, this substantially increases the amount of data output. Also, as laboratories add more devices more experiments are run simultaneously with a corresponding growth in data.

A large volume of data causes issues that include the cost of storage; an effective platform will need to provide a low cost per terabyte for storage while still providing fast online access to data files. It can also be the cause of a data management issue, as data must transfer from the computer attached to the mass spectrometer to a separate processing environment, and then the results move to a shared area where they can be analysed. This potential data management problem is one of the issues that this research targets for mitigation. As previously noted the processing of RAW mass spectrometer output is a complex, time-consuming task. This processing time needs to be reduced so that the researchers can analyse results as quickly as possible and allow research to progress in an iterative fashion.

A further issue is that of the workflow; life science researchers rely heavily on computer scientists or bioinformaticians to provide them with data that they can understand [73]. An ideal system would abstract the complexity of RAW file processing away from the Life Scientists.

3.4.3 COMPLEXITY

Producing a list of proteins and their abundance from a binary RAW file is a complex process, and importantly it is open to some interpretation and error. It is usual to find that different algorithms and software packages will produce different results from the same input file [74]. It is worth noting that on the discovery of new information or experimental techniques, it is entirely possible that reprocessing of the data files would be necessary. This requirement for reprocessing emphasises the need for a fast, simple solution to the issues of data management and processing. The potential need to reprocess data has raised the question of what should be stored on the completion of an experiment. Martens et al. [75] argue that the only acceptable data is the original output from the mass spectrometer in the vendor’s binary format. Certainly, given that reprocessing may well identify new centroids, simply storing the centroid data is insufficient.

A further challenge is that by the very nature of the way a mass spectrometer processes data (in scans which are snapshots of the ions in the machine at that time) the data will be incomplete. Furthermore, MS/MS scans are typically only from the largest, most abundant peptides as, currently, even the most advanced mass spectrometers cannot fragment every peptide [76].

3.4.4 FEATURE DETECTION

The purpose of the algorithm tested and benchmarked on parallel clusters in this research is to detect features in the mass spectrometer output spectra. This process is also called "Peak Picking" because the algorithm scans the spectrum and selects peaks in the data; these peaks represent the ions detected by the mass spectrometer during a scan. A feature in this context consists of a set of peaks aligned across time that correspond to a single eluting peptide [77]**.** Features and how they are made up of peaks are described fully in the next section of this chapter. The feature detection process results in a list of peptides and includes their mass, charge, abundance (known as intensity in this context), the retention time at the highest point of the peak and can also include the start and end retention times. A recent review [16] of the performance of computational analysis of mass spectrometer data describes feature detection as the second most time-consuming part of the process, with the most time consuming being the database search, which is also outlined in the next section**.** The review [16] also states that the feature detection phase benefited the most from parallelization. A detailed description of the steps required for feature detection can be found in Section 3.5 of this chapter.

3.4.5 DATABASE LOOKUP

The final stage of the data processing pipeline is to look up the identified peptides in a database of known protein to peptide relationships. This search is a complex process and involves some degree of probability, as a certain peptide is likely to be an integral part of many different proteins [78]. Several attempts at creating a parallel process to complete this stage of the pipeline have already been successful. These include parallel tandem, a parallel implementation of the X! Tandem search engine using the message passing interface (MPI) style of programming and parallel virtual machines (PVM) both of which are described in Section 5.2 of this thesis. A more recent example of parallelization is Hydra [79], which uses Hadoop and the MapReduce programming framework to create a parallel proteomics search engine. The research presented in this thesis does not attempt to recreate the database lookup but instead suggests ways in which the output from the parallel feature selection can be provided as the input to a parallel database lookup system.

3.4.6 SHARING OF RESULTS

Once the proteins in the sample have been identified it is usual practice to share the results in an online repository such as SWISS-PROT [8]. There are several reasons for this including reproducibility and to aid collaboration between facilities working towards a common aim. If other researchers want to recreate parts of workflows that have already been completed then having data accessible in a standard format means that research can be repeated and checked in a controlled manner. Other proteomics data repositories such as PRIDE provide an archive for data used to support scientific publications [80].

3.5 OVERVIEW OF FEATURE DETECTION ALGORITHMS

There have been several comparative studies on the proposed algorithms for feature detection. For example, Katajamaa and Oresic [72] reviewed the filtering and feature detection algorithms available at the time (2007) and the different software packages using them. They conclude that as new algorithms are designed it is necessary to test them as part of the whole processing pipeline and that open source is a way to address this.

Note that early attempts to produce peak lists had to contend with more complexity than is needed with the data that is output from mass spectrometers today. These include steps such as baseline correction [81] which is not required now as the mass spectrometer performs this processing on the data before it is output. Also due to the availability of high-resolution data, peak picking based on profiles such as the wavelet technique [82] used in the TOPP open-source software [83] have given way to feature detection techniques. Profile-based approaches are more accurate on low-resolution data where feature detection is relatively more error prone resulting in missing values in the output [77]. Due to the inherent data reduction in the process, feature detection is less computationally expensive than profile-based approaches and leads to a correspondingly faster processing time [77].

An early algorithm design that describes the steps for feature detection sets out three stages, 1) identify peaks within a single scan 2) smooth peaks over time and identify peaks that appear over multiple scans 3) assemble peaks into isotope groups [84]. An isotope in this context is a variant of a particular chemical element, which has a different mass due to the incorporation of a different form of carbon. For example, a peptide that incorporates carbon-12 will have a different mass from the same peptide incorporating carbon-13 or carbon-14. These mass changes are predictable and are used to distinguish actual peptides from contaminants and noise in the data. A recent study into the software tools and algorithms available [85] contains a similar description of feature detection and also describes the software MaxQuant as a well-established program for the analysis of quantitative data.

This research focuses on the feature detection algorithm as employed in the MaxQuant software, which in outline follows similar steps to those described by Bellew et al. [84] as mentioned above. MaxQuant is described as "*A quantitive proteomics software package designed for analysing large-scale mass-spectrometric data sets*" http://www.biochem.mpg.de/5111795/maxquant (accessed 1st March 2017) and is freely available to download and use. Professor Juergen Cox at the Max Planck Institute of Biochemistry, Germany, created MaxQuant and his team of researchers continues to maintain and add features to it. The on-going collaboration between Cox and Matthias Mann has ensured that MaxQuant is widely used in the proteomics community and the original paper [12] has several thousand citations. Mann has been a leading researcher in this field, pioneering the SILAC method of protein quantitation [69] and together with Ruedi Aebersold wrote one of the first papers detailing a workflow using mass spectrometers for proteomics experiments [66]. Researchers in the Lamond Laboratory use MaxQuant extensively to analyse the results of their experiments. The RAW data used for benchmarking purposes in this research was obtained from The Lamond Laboratory, School of Life Sciences, Centre of Gene Regulation and Expression, University of Dundee and has also been previously processed using MaxQuant.

Cox and Mann published extensive supplementary material alongside the original MaxQuant paper [12]. This supplement outlines the main parts of the algorithm used to process mass spectrometer spectra without giving all the details of the exact implementation. The information in the supplement has been used to design the parallel algorithm in this research. In 2011, The Lamond Laboratory at the University of Dundee undertook a collaborative project with Teradata Ltd. to recreate the MaxQuant workflow using a relational database and the SQL scripting language in a project called Spectracus. Following the movement of the Spectracus project into a testing phase, the focus in the Lamond Laboratory has changed to an open-source strategy. Although this work has not reached a production stage and no papers have been published regarding it, reference was made by the author to the project documentation and laboratory notes when researching ways to run a feature detection algorithm in a parallel fashion.

Figure 7 presents a view of a 3D view of all the peaks in a complete experiment, it is possible to see the intensity of the features as well as their distribution. Note how the distribution of the peaks is highly skewed towards certain ranges of m/z and retention time. The scale of Figure 7 is such that the entire dataset is shown in a single figure. The remainder of this section describes the algorithm used to identify these peaks as described in the MaxQuant supplement [135]

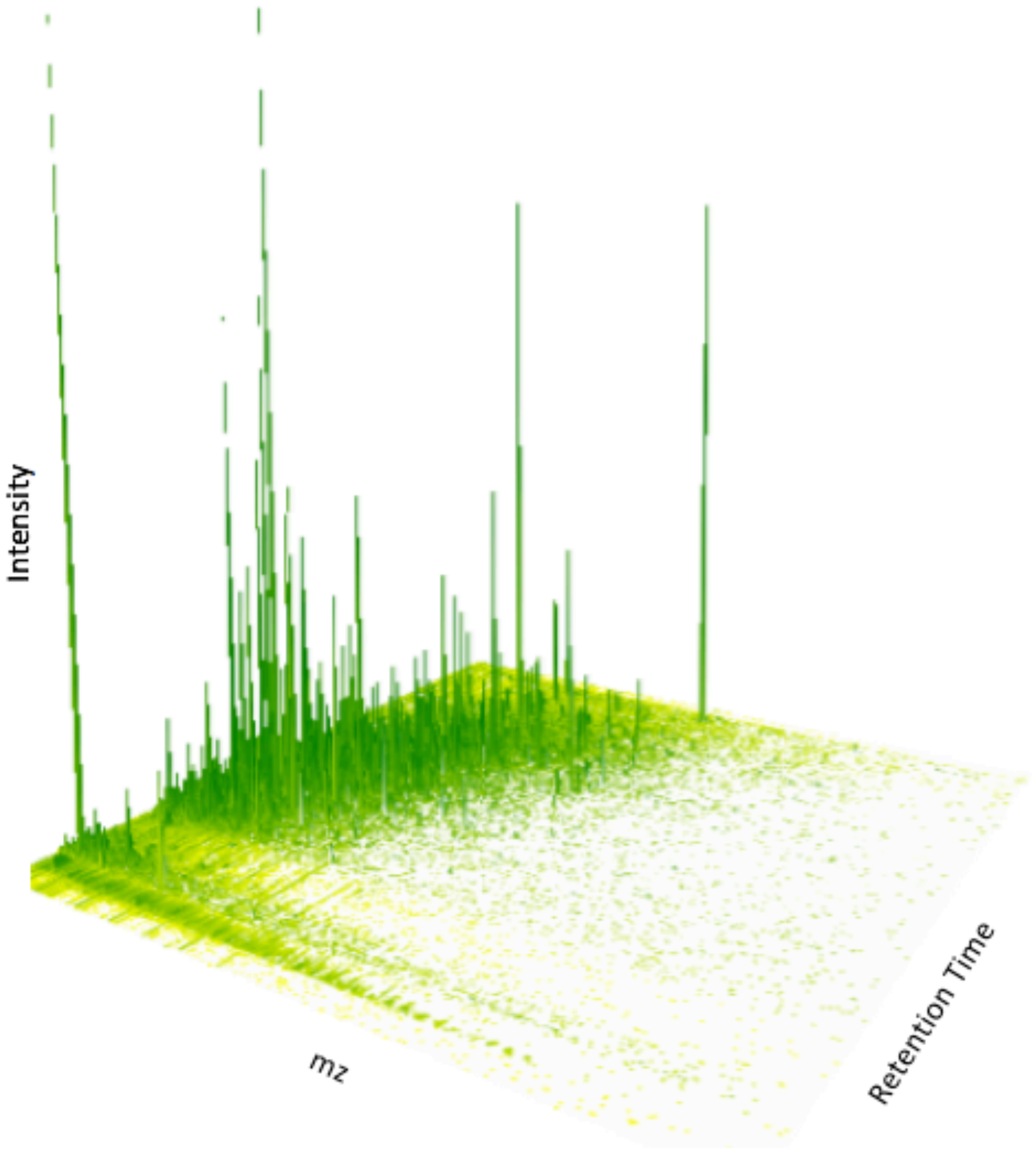


FIGURE 7 ILLUSTRATION OF THE COMPLETE SET OF PEAKS (FEATURES) IN A 3D VIEW

3.5.1 DETAILED DESCRIPTION OF THE FEATURE DETECTION ALGORITHM

The steps required by the feature detection algorithm are as follows:

1) 2D peak picking

The two-dimensional (2D) name of this process comes from the fact that peaks are detected in the mass to charge ratio and intensity dimensions. The algorithm works on a single mass spectrometer scan at a time. Figure 8 shows a section of a spectrum with the mass to charge (m/z) values on the x-axis and the intensity (which equates to the abundance of the ions detected) on the y-axis. By convention the scan is represented as a continuous line rather than individual data points. As mentioned, the unit of measurement for the m/z ratio is a thomson; at this stage in the process the value of the electrical charge held by the ions in the spectra is not known and therefore their actual mass cannot be calculated

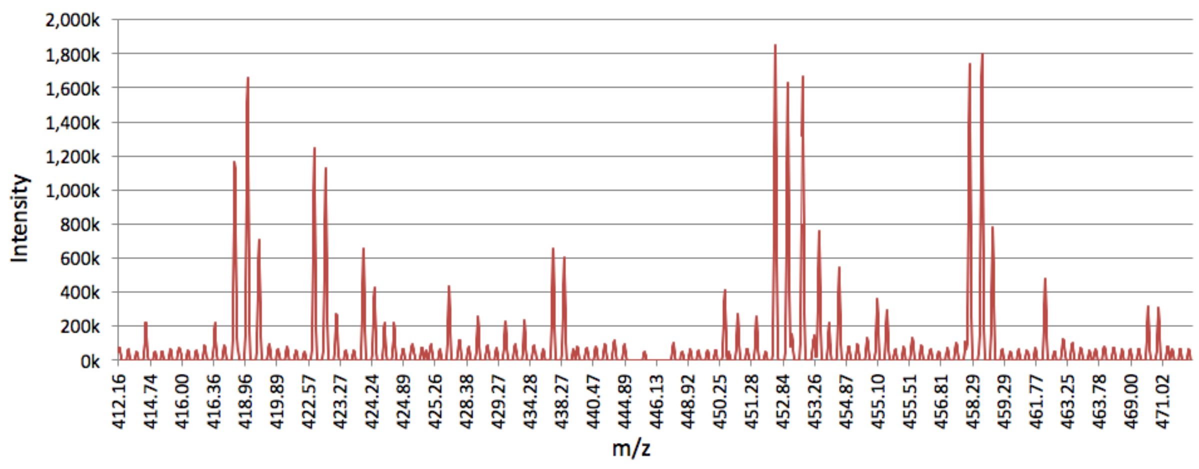


FIGURE 8 A SECTION OF A SINGLE MASS SPECTROMETER SCAN

Figure 9 Shows an example of a single peak taken from a scan, the data points that make up the peak are shown in Table 1.

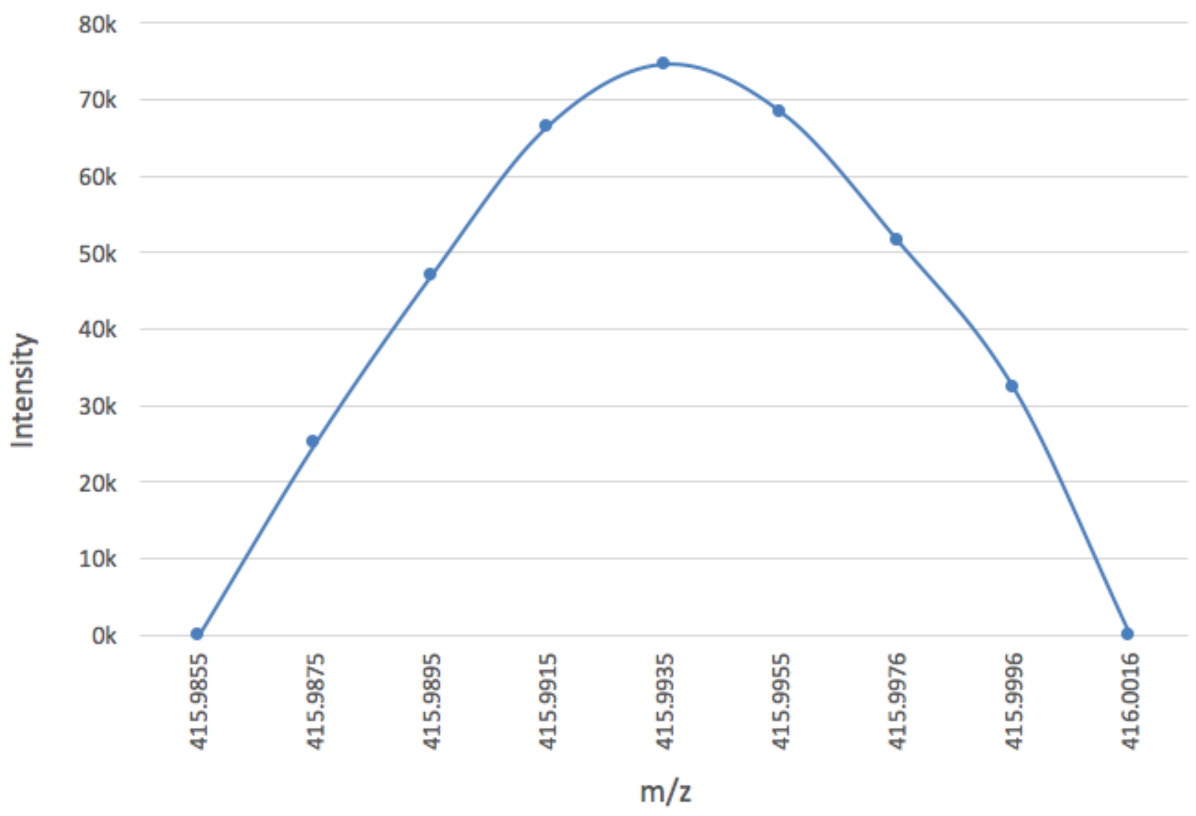


FIGURE 9 EXAMPLE 2D PEAK

m/z	intensity	difference between current m/z and previous m/z
415.98547	0.000	
415.98750	25057.193	0.00203
415.98950	47017.250	0.00200
415.99152	66487.530	0.00202
415.99353	74497.980	0.00201
415.99554	68222.600	0.00201
415.99756	51487.445	0.00202
415.99957	32179.443	0.00201
416.00160	0.000	0.00203

TABLE 1 DATA POINTS FOR EXAMPLE 2D PEAK

The difference between the current and previous m/z values in Table 1 shows the sampling rate of the mass spectrometer. Note that while theoretically each scan will contain the same number of data points, in practice the mass spectrometer performs a data reduction step and does not report all zero values between peaks. A complication in the peak detection process is that peaks can overlap.

shows overlapping peaks where the intensity does not return to zero after reaching the apex of the peak but instead rises to another apex with the relevant data in Table 2.

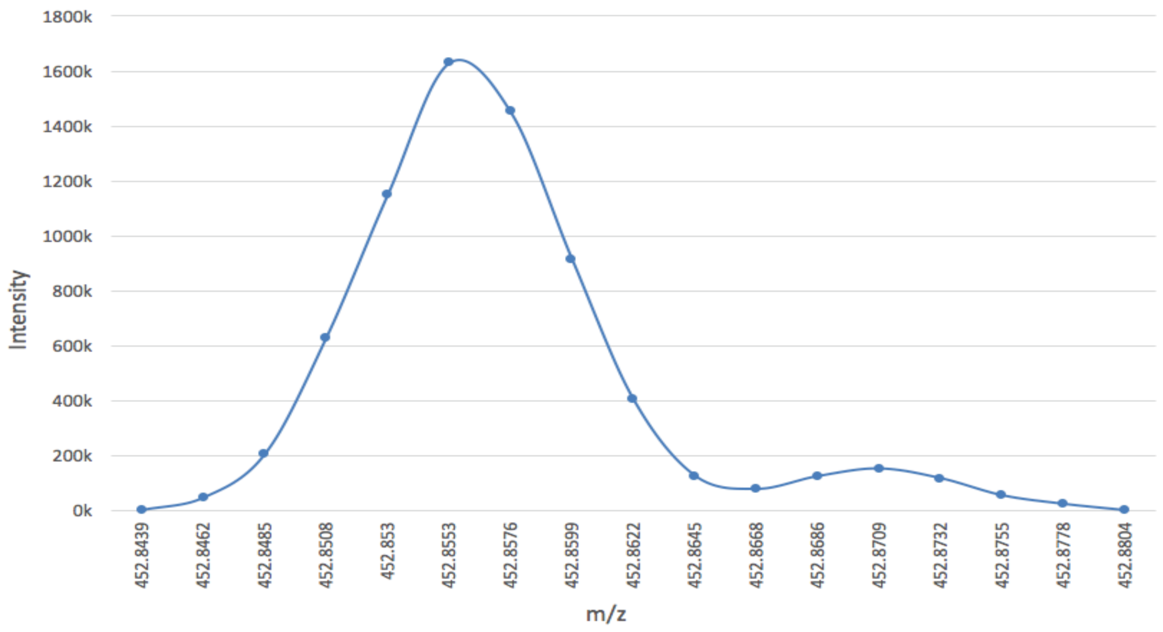


FIGURE 10 EXAMPLE OVERLAPPING 2D PEAKS

m/z	intensity	difference between current m/z and previous m/z
452.84390	0.000	
452.84620	45549.650	0.00230
452.84848	204779.620	0.00228
452.85077	629295.300	0.00229
452.85303	1150425.400	0.00226
452.85532	1630558.100	0.00229
452.85760	1452616.100	0.00228
452.85990	915715.560	0.00230
452.86218	404217.440	0.00228
452.86447	125762.070	0.00229
452.86676	77428.920	0.00229
452.86862	124085.400	0.00186
452.87090	151868.830	0.00228
452.87320	116575.590	0.00230
452.87550	54405.645	0.00230
452.87778	22953.500	0.00228
452.88040	0.000	0.00262

TABLE 2 DATA POINTS FOR EXAMPLE OVERLAPPING PEAKS

Figure 11 shows an illustration of the peak detection process (without the complication of the overlapping peaks), using a simple slope detection algorithm to identify all the data points that constitute a single peak.

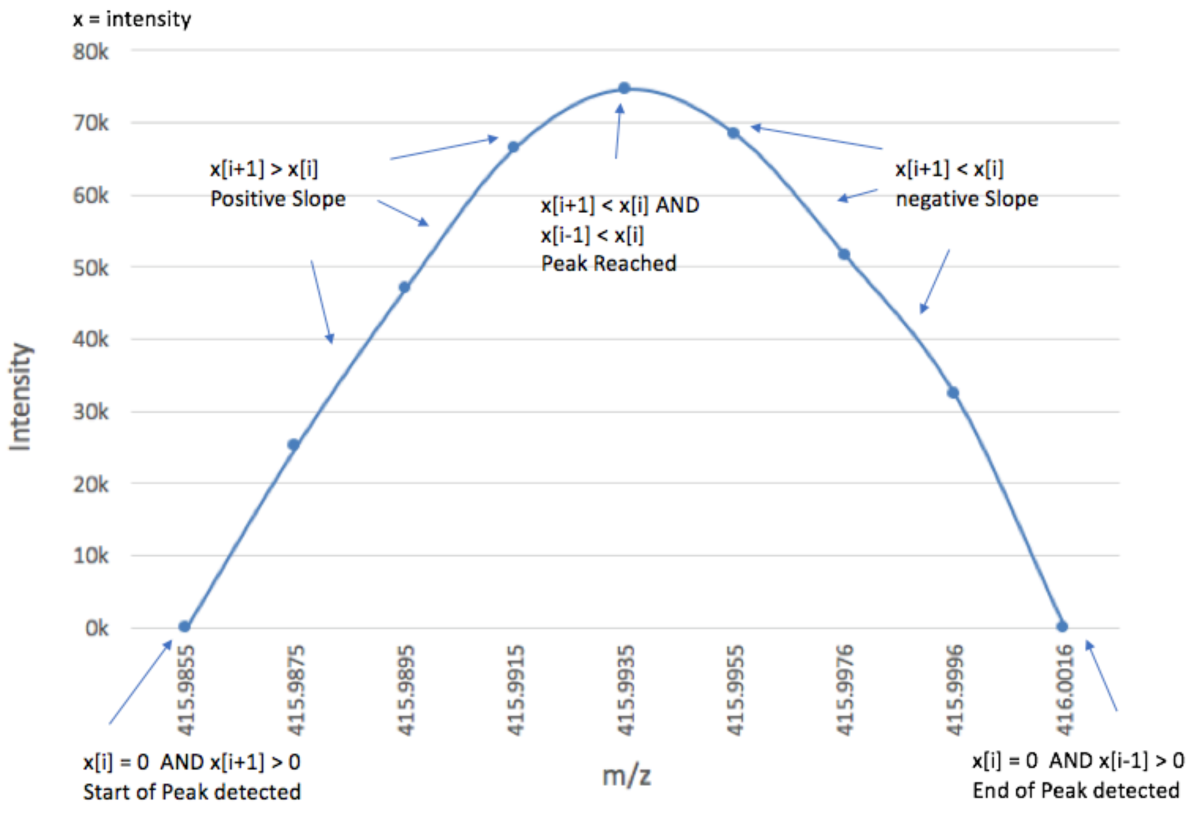


FIGURE 11 ILLUSTRATION OF PEAK DETECTION

The pseudocode below details the first part of the 2D peak picking process as defined by MaxQuant. This step iterates through the array of mz and intensity values from a single scan and groups together data points that belong to the same peak. Any overlapping peaks are flagged and assigned separate peak identifiers. The groups of date points making up a peak are then processed to calculate intermediate metrics to be passed to the next part of the process. The result of this step is a peak identifier, Weighted Peak m/z (defined in the pseudocode), the sum of the intensity values and the maximum intensity for each 2D peak in the scan. The weighted peak m/z value corresponds to the centroid of the peak as discussed in Chapter 3, section 4.2

Description: Identify peaks and calculate peak metrics
Input: array of mz values, array of intensity values, scan number
Output: for each peak; peak identifier, weighted peak mass, sum of intensities, maximum intensity, retention time
Parameters Used: (See table 3)
nf: 2D noise filter, used to remove low intensity data points from a peak = 10% of maximum peak intensity

```
1  SET position counter to 0
2  SET peak index to 0
3  WHILE position counter < length of mz and intensity arrays
4    //Find the start of a peak
5    IF (current intensity is > 0 AND previous intensity = 0) OR overlap flag is set
6      WHILE current intensity is > 0
7        SET the 2D peak ID for this postion to the peak index
8        IF current intensity > previous intensity
9           AND previous intensity 2 steps back > previous intensity
10         //Overlapping Peak found, store Intensity
11         //and start new Curve for next Iteration
12         SET the overlap flag
13         BREAK out of this loop
14       ELSE
15         INCREMENT the position counter
16       ENDIF
17     END WHILE
18   END IF
19   //end of peak found
20 INCREMENT the peak index
21 INCREMENT the position counter
22 END WHILE
23 //Reiterate over mz and intensity arrays to create intermediate metrics
24 FOREACH peak index
25   FOREACH mz, intensity
26     IF current intensity > nf
27       SET maximum intensity
28       SET minimum intensity
29       SET sum of intensities for this peak
30       SET sum MZ by intensity = SUM(mz*intensity)
31       SET weighted peak mass = sum MZ by intensity / sum of intensities
32     ENDIF
33   END FOREACH
34 END FOREACH
```

Parameters

Several parameters are defined in the feature detection process. Table 3 lists the parameters and the reasons for choosing the default values. The values have been chosen either from best practice outlined in the literature, or from experimental observation. The following details the parameters, their settings, symbols as used in the pseudocode and the reference for the value of the setting

TABLE 3 PARAMETERS USED IN THE FEATURE DETECTION ALGORITHM

Symbol	Description	Setting	Reference
nf	2D noise filter, used to remove low intensity data points from a peak. Setting is a percentage of the maximum intensity of a peak	10%	MaxQuant
pht	Minimum ratio of intensities between two 3D peaks in a isotopic envelope	60%	MaxQuant
sm	Mass of a sodium molecule, used to modify the range of masses that qualify as a match	+/- 0.0109135	MaxQuant
cc	Correlation coefficient for matching 3D isotopic peaks	0.6	MaxQuant
Mmt	Mass matching tolerance, used in conjunction with sm to calculate the range of masses that qualify as a match	7ppm	MaxQuant and Thermo

The next step in the 2D peak picking process is to identify peaks that fall within an Isotopic envelope. The envelopes are created when the incorporation of carbon atoms of different molecular mass into the ions causes a predictable pattern. An example of an Isotopic envelope is shown in Figure 12. The envelopes are identified once an initial pass of the spectra picks out the individual weighted peaks. The m/z shift between the isotopes in the envelope depends on the electrical charge of the ion and through the measurement of this change in m/z and a lookup table such as that in Table 4, the charge of the ion is calculated and therefore its true mass can also be calculated by multiplying the m/z value by the charge.

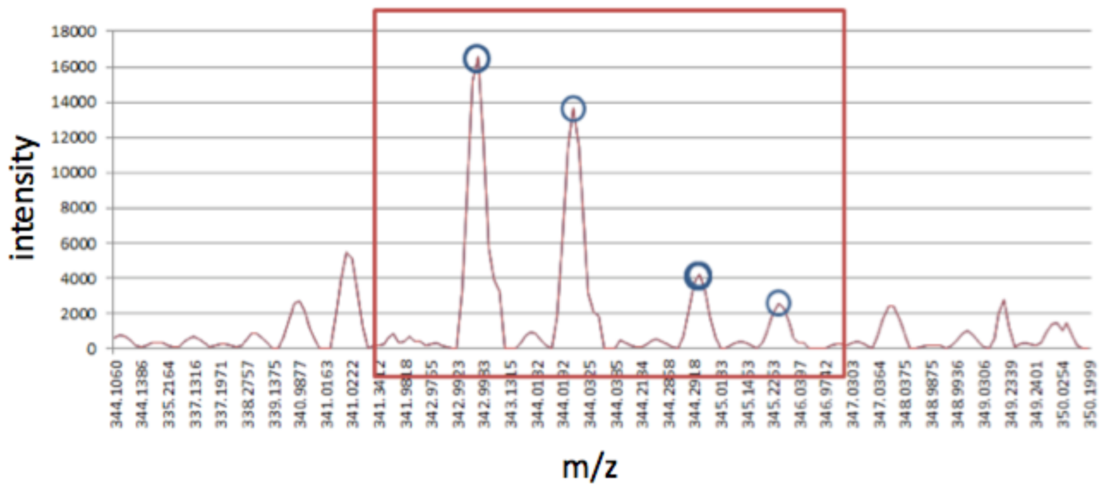


FIGURE 12 PEAKS WITHIN AN ISOTOPIC ENVELOPE

The values in Table 4 are created using an “averagine” model [77]. Here the term averagine refers to a theoretical peptide the properties of which are calculated from the average composition of peptides at a given mass. The m/z shifts are reported by convention to eight decimal places.

TABLE 4 MASS SHIFT BY CHARGE FOR ISOTOPIC ENVELOPE CALCULATION

Charge	Mass Shift (thomsons)
2	0.50143432
3	0.33428955
4	0.25071716
5	0.20057373
6	0.16714477

The following pseudocode details the 2D Isotopic envelope detection process as used by MaxQuant. The result of this step is a list of 2D Weighted Peak m/z, summed Intensity and charge values for each 2D peak that formed part of an isotopic envelope. To count as a valid envelope it must be formed of at least three 2D peaks. It is important to note at this point that the 2D isotopic envelope detection step is used to identify the charge of the ions and also as a filter to remove any of the peaks that are not included in an isotopic envelope. The individual 2D peaks are passed to the 3D peak picking process and are not aggregated into a single mono-isotopic peak at this stage.

Description: Identify Isotopic envelopes of 2D peaks and calculate charge
Input: peak identifier, weighted peak mass, sum of intensities, maximum intensity, retention time
Output: peak identifier, weighted peak mass, sum of intensities, maximum intensity, charge, retention time
Parameters used:
pht: Minimum ratio of intensities between two 3D peaks in a isotopic envelope=60%
sm : Mass of a sodium molecule=+/- 0.0109135

```
FOREACH 2D peak identifier in input
  SET outer loop weighted peak mass
  SET outer loop sum of intensities
  SET outer loop maximum intensity
  FOREACH  2D peak identifier in input
    SET inner loop weighted peak mass
    SET inner loop sum of intensities
    SET inner loop maximum intensity
    IF the inner and outer loops are not at the same point
       AND outer loop weighted peak mass - inner loop weighted peak mass > 0
       AND outer loop weighted peak mass - inner loop weighted peak mass <= 1
       AND inner loop maximum intensity > pht * outer loop maximum intensity
             SET wpm difference = outer loop weighted peak mass - inner loop weighted peak mass;
         IF wpm difference > 1-sm AND wpm difference < 1+sm SET charge to 1
//the decimal values to calculate charge are from the "averagine" table 4
         ELSEIF wpm difference >= 0.50143432 - sm
           AND wpm difference <= 0.50143432 + sm SET charge to 2
         ELSEIF wpm difference >= 0.33428955 - sm
           AND wpm difference <= 0.33428955 + sm SET charge to 3
         ELSEIF wpm difference >= 0.25071716 - sm
           AND wpm difference <= 0.25071716 + sm SET charge to 4
         ELSEIF wpm difference >= 0.20057373 - sm
           AND wpm difference <= 0.20057373 + sm SET charge to 5
         ELSEIF wpm difference >= 0.16714477 - sm
           AND wpm difference <= 0.16714477 + sm SET charge to 6
         SET mass = charge * outer loop weighted peak mass
             OUTPUT 2D peak identifier, outer loop weighted peak mass,
                    outer loop sum of intensities, mass, charge
  END FOREACH
END FOREACH
```

2) 3D peak picking

Following the 2D peak picking and isotope identification, the next step in the feature detection process is search for chains of 2D peaks that align in a narrow band of m/z in order of increasing retention time. The addition of the further dimension of retention time (rt) into the calculation is the reason for the three-dimensional (3D) naming. A peptide will present as ions in the scans over a period of time, starting off with a small amount rising to a peak and then tailing off. Figure 13 shows a selection of the 2D peaks that were output from the 2D peak picking process. When the 2D peaks are arranged in this way it is possible to see the 3D peaks as chains of 2D peaks. To be grouped into the same chain, 2D peaks must have the same charge and the m/z values lie within a +/- 7ppm window, this is known as "peak alignment". Pseudocode for the 2D peak alignment step is shown below.

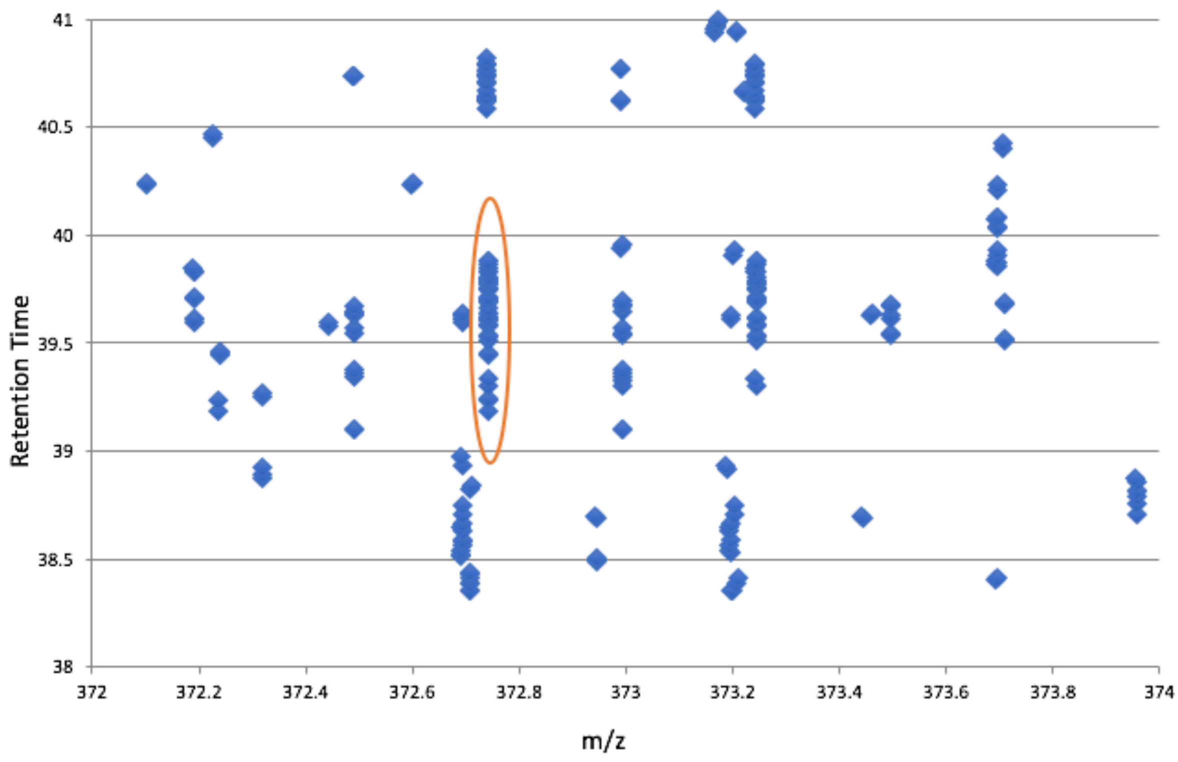


FIGURE 13 2D PEAKS WITH AN EXAMPLE CHAIN OF 2D PEAKS CIRCLED

(note there are several chains in this figure but only one is circled for illustrative purposes)

Description: Create chains of 2D peaks ordered by retention time.
Input: 2D peak identifier, weighted peak mass, sum of intensities, maximum intensity, charge, retention time
Output: 2D peak chain identifier, 2D peak identifier, weighted peak mass, sum of intensities, maximum intensity, charge, retention time
Parameters used:
mmt: Mass matching tolerance = 0.000007

```
1 rt = retention time
2 NESTED SORT list of input 2D peaks by retention time and then by weighted peak mass within Retention Time
3 FOREACH 2D peak identifier
4   FOREACH 2D peak identifier
5     IF retention time for outer 2D peak is within 30 seconds of retention time for inner 2D peak
6       //fast forward to next possible match in inner loop
7       //performance enhancement to prevent excessive looping -
8      WHILE outer loop mass < inner loop mass + 0.05 and inner loop rt = current rt
9        IF outer loop charge != inner loop charge
10        OR outer loop mass < inner loop mass * mmt
11         INCREMENT inner loop
12    END WHILE
13    IF outer loop wpm < inner loop mass + (inner loop mass * mmt)
14      SET match flag to found
15      FOREACH of the next 4 2D Peaks check for another match
16        IF other matches found choose closest mass to the outer loop 2D mass
17      END FOREACH
18      SET 2D peak chain identifier for the matched 2D peak
19    END IF
20   END IF
21   IF matchflag is set to found break out of inner loop and move to next point in outer loop
22  END FOREACH
23  INCREMENT the 2D peak chain identifier if we had matches before but now don't to start a new 3D peak.
24 END FOREACH
```

Once the 2D peaks have been linked to form chains, they present as a time series as seen in Figure 14. The process from this point is similar to that followed in the 2D peak picking step; that is individual peak detection and isotopic envelope identification. An additional step is necessary at the start of the process and this is to smooth the intensity values to prevent splitting an eluting peptide into several sections due to noise in the data. Figure 14 shows an example of this effect, without smoothing many false peaks would be found. The method for smoothing, based on the MaxQuant algorithm, is to apply a window mean filter of +/- 1 scan width [135]. Many other ways of smoothing data exist such as Gaussian Filters, Continuous Wavelet Transformation and Discrete Wavelet Transformation [147]. However, as the moving average window mean filter is specified in the MaxQuant supplement, this was used in the algorithm developed here.

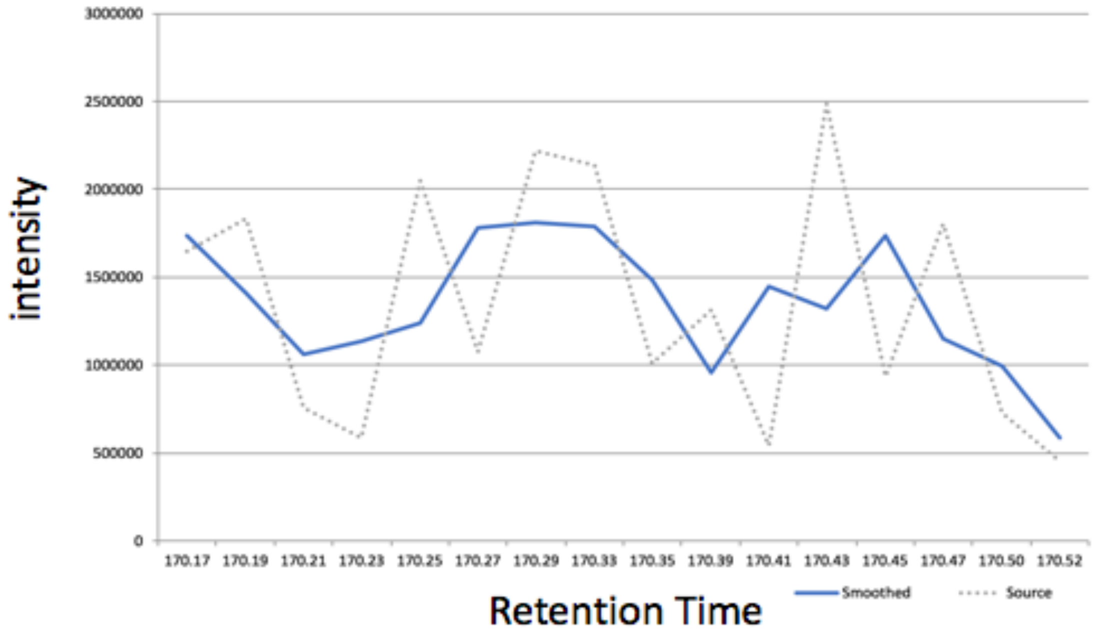


FIGURE 14 ILLUSTRATION OF SMOOTHING IN 3D PEAK PICKING

The following pseudocode details the process for smoothing the intensity values of the 2D peaks making up to 2D peak chains.

```
Description: Smooth intensities of 2D peaks that make up 3D peaks
Input: 2D peak chain identifier, 2D peak identifier, weighted peak mass, sum of intensities, charge, retention time
Output: 2D peak chain identifier, 2D peak identifier, weighted peak mass, smoothed intensity, sum of intensities, charge, retention time

SORT 2D peaks in order of Retention Time
  FOREACH 2D peak chain
    FOR 2D peak in the current 2D peak chain
      IF this is the start of a new 2D peak chain
        SET smoothed intensity = (current intensity + next intensity) / 2
      ELSE IF this is the last 2D peak in a 2D peak chain
        SET smoothed intensity = (current intensity + last intensity) / 2
      ELSE this is the a 2D peak in the middle of a 2D peak chain
        SET smoothed intensity = (current intensity + next intensity + last intensity) / 3
    END FOREACH
    OUTPUT 2D peak chain identifier, 2D peak identifier, weighted peak mass, smoothed intensity, sum of intensities, charge, retention time
  END FOREACH
```

Following the smoothing process, the 3D peaks can be identified in a similar manner to that in which the 2D peaks and overlaps were identified in an earlier step. The major difference between the two procedures is that the smoothed intensity values do not fall to zero between 3D peaks, this means that in effect all peaks are treated in the same way as overlapping peaks in the 2D process. The following pseudocode details the process for identifying individual 3D in the 2D peak chains.

Description: Identify the 3D peaks from the smoothed data
Input: 2D chain identifier, 2D peak identifier, weighted peak mass, smoothed intensity, charge, retention time
Output: 3D peak identifier, 2D peak identifier, weighted peak mass, smoothed intensity, charge, retention time

```
FOREACH 2D point in the input
  IF previous 2D chain identifier <> current 2D chain identifier
    //start of new chain must be start of new peak
    INCREMENT 3D peak identifier
    IF current smoothed intensity < next previous intensity
       OR current smoothed intensity = next previous intensity
       SET slope indictor to positive
    //Due to the partition strategy it is possible to start a chain with a negative slope
    ELSE SET slope indictor to negative
    END IF
  ELSE
    IF previous smoothed intensity < next smoothed intensity
       OR next smoothed intensity = previous smoothed intensity
       SET slope indictor to positive
       IF current smoothed intensity < previous smoothed intensity
          INCREMENT 3D peak identifier
    ELSE IF next smoothed intensity < previous smoothed intensity
         SET slope indictor to negative
         IF current smoothed intensity < next smoothed intensity
           INCREMENT 3D peak identifier
    END IF
  END IF
END FOREACH
```

Following the process described above, the individual 3D peaks and the 2D peaks of which they they are comprised have been identified. Now, isotopic envelopes can be detected in a similar way to the 2D peak picking step with the exception that a correlation coefficient is calculated to check that the candidate 3D peaks have a similar shape before they can be classed as matching in a 3D Isotopic envelope. This method of detecting the 3D Isotopic envelopes is taken from the MaxQuant description of the algorithm. The pseudocode below details the process for identifying 3D isotopic envelops.

Description: Identify the 3D peaks from the smoothed data
Input: 3D peak identifier, 2D peak identifier, weighted peak mass, smoothed intensity, charge, retention time
Output: ISO envelope identifier, 3D peak identifier, 2D peak identifier, weighted peak mass, smoothed intensity, charge, retention time
Parameters used:
pht: Minimum ratio of intensities between two 3D peaks in a isotopic envelope=60%
sm : Mass of a sodium molecule=+/- 0.0109135
wpm = weighted peak mass
rt = retention time

```
FOREACH 3D peak identifier in input
  FOREACH 3D peak identifier in input
    IF position in outer loop <> position in inner loop
      //Match on wpm
      AND outer loop wpm - inner loop wpm >= 0
      AND outer loop wpm - inner loop wpm <= 1)
      //Match on rt window between first and last 2D peak in the 3D peak
      AND outer loop last rt >= inner loop rt
      AND outer loop first rt <= inner loop rt
      //Match on Charge
      AND outer loop charge = inner loop charge
      //Match on Intensity
      AND inner loop smoothed intensity > pht * outer loop smoothed intensity
        SET wpm difference = outer loop weighted peak mass-inner loop weighted peak mass;
        IF wpm difference > 1 - sm AND wpm difference < 1.0 + sm SET charge to 1
          ELSEIF wpm difference >= 0.5014343200 - sm AND wpm difference <= 0.5014343200 + sm SET charge = 2
          ELSEIF wpm difference >= 0.3342895500 - sm AND wpm difference <= 0.3342895500 + sm SET charge = 3
          ELSEIF wpm difference >= 0.2507171600 - sm AND wpm difference <= 0.2507171600 + sm SET charge = 4
          ELSEIF wpm difference >= 0.2005737300 - sm AND wpm difference <= 0.2005737300 + sm SET charge = 5
          ELSEIF wpm difference >= 0.1671447700 - sm AND wpm difference <= 0.1671447700 + sm SET charge = 6
        END IF
        IF wpm difference calculation found a match
          //calculate the correlation coefficient between current 3D peak
          //and the next 3D peak in the potential isotopic envelope
          FOREACH potential isotopic envelope
           FOREACH 3D peak in current potential isotopic envelope
           FOREACH 2D peak in current 3D peak
             SET normA = normalised intensity for current 2D peak in current 3D peak
             SET normB = normalised intensity for current 2D peak in next 3D peak
             normAnormB = normAnormB + (normA * normB)
             normAnormA = normAnormA + (normA * normA)
             normBnormB = normBnormB + (normB * normB)
           END FOREACH
           corelation coefficent = normAnormB / square root of (normAnormA * normBnormB)
          //Write out matched points if correlation coefficient is greater threshold
          IF corelation coefficient >= pht
           true isotopic envelope found
           OUTPUT ISO envelope identifier, 3D peak identifier, 2D peak identifier, weighted peak mass, smoothed intensity, charge, retention time
          END IF
        END IF
     END IF
   END FOREACH
 END FOREACH
```

Finally, the individual 3D peaks making up the Isotopic envelopes are combined into a single mono-isotopic peak with a single value for the mass, intensity and retention time of the peptide. The mass and retention time are taken from the first 3D peak in the isotopic envelope and the intensity is the sum of the intensity values from all 3D peaks in the envelope. This information will feed into the later stages of the data processing pipeline, resulting in identified proteins. The following pseudocode details the process for creating the final mono-isotopic peaks

```
Description: Aggregate component values into mono isotopic peaks within an istopic envelope
Input: ISO envelope identifier, 3D peak identifier, 2D peak identifier, weighted peak mass, smoothed intensity, charge, retention time
Output: mono-iso peak identifier, mass, intensity, charge, retention time

FOREACH Isotopic Envelope
  FOREACH 3D peak in the current isotopic envelope
    FOREACH 2D peak in the current 3D peak
      sum weighted peak mz = weighted peak mz + (current weighted peak mz * current smoothed intensity)
      sum weighted peak rt = weighted peak rt + (current weighted peak rt * current smoothed intensity)
      sum intensity = sum intensity + current smoothed intensity
    END FOR EACH
      weighted 3D peak mz = sum weighted peak mz / sum intensity
      weighted 3D peak rt = sum weighted peak rt / sum intensity
  END FOR EACH
  weighted peak mass = weighted 3D peak mz for first 3D peak * charge
  weighted peak rt = weighted 3D peak rt for first 3D peak
  intensity = sum intensity for all 3D peaks in envelope
END FOR EACH
```

3.6 CONCLUSION

The aim of proteomics is to understand the full complement of proteins expressed by an organism and the interactions between them. Proteomics is an important field of study in life sciences and is an essential component of systems biology. By integrating detailed data from different domains, it is possible to describe biological systems, the benefits of which include the development of personalised medicines. The study of proteins is a complex task involving many steps, the most common technique used to measure the presence and abundance of proteins in a sample is mass spectrometry.

During a process known as “bottom-up proteomics”, the large protein molecules are cut into smaller pieces, called peptides, using an enzyme. The peptides are detected by the mass spectrometer and output in the form of spectra consisting of mass to charge ratios and intensities (abundance of peptide ions). In order to identify the original proteins, present in the experimental cell sample, complex processing is required. The first step in this processing pipeline is feature detection and the algorithm for feature detection described in this thesis is based on the algorithm as implemented by the MaxQuant software first described by Bellew [84]. Current methods of processing mass spectrometer data in serial can be slow, one potential way of speeding up the process is through parallel processing. The next chapter reviews related work in this area.

4 PARALLEL COMPUTING FOR PROTEOMICS

4.1 CHAPTER SUMMARY

This Chapter discusses the current state of parallel computing for processing proteomics data.

Section 2 gives a detailed description of MapReduce and the capabilities of Hadoop.

Section 3 describes current research into parallel solutions for feature detection and the proteomics pipeline.

Section 4 discusses the use of cloud and big data solutions.

Section 5 concludes the literature review chapters and presents the main research question of this thesis.

4.2 HADOOP AND MAPREDUCE

As discussed in Chapter 2, Section 2.3, the paper "*MapReduce: Simplified Data Processing on Large Clusters*" [5] introduced a programming framework called MapReduce which has its roots in functional programming languages such as Lisp [35]**.** The framework consists of two main operations, a map and a reduce, and operates by the passing of key/value pairs. The process flow is as follows [5]:

- Input data is divided into a series of "splits" that are distributed around the nodes of the cluster.
- The map task takes an input split in the form of a key/value pair and emits a set of intermediate key/value pairs.
- The intermediate key/value pairs are moved around the cluster so that all pairs with the same key are placed on the same node, this phase is called the shuffle.
- Following the data movement in the shuffle phase, the reduce task takes an intermediate key and the set of values for that key as input and emits a set of key/values pairs consisting of the intermediate key and the final result values.

The following notation is used to describe the map and reduce tasks [35] where k and v are in the input keys and values, k1, v1 are the intermediate keys and values and v2 is the output values.

```
map (k, v) -> set(k1, v1)
reduce (k1, list (v1)) -> set (k1, v2)
```

Note that a key and a value can take many forms, for instance, it is common for the value to be a dataset formed of delimited values that are parsed and operated on in the task itself. Two other operations exist, the custom partitioner and the combiner. The custom partitioner takes an intermediate key as an input and outputs a partition index which is used to distribute data to the reduce steps. A custom partitioner is used to override the standard hashing method of data distribution and provide a fine degree of control over which reduce task the intermediate key/value pairs from the map tasks are sent to. The combiner is a performance enhancement operation used to reduce the amount of data moved around the cluster and is an aggregation operation, often called a “map-side reduce”. By combining values with the same key within a map task, fewer key/value pairs are emitted and therefore less data needs to be shuffled around the cluster. The custom partitioner and combiner are described as follows.

```
    Partition(v1) -> (k1)
    Combine set(k1,v1) -> (k1, combined_v1)
```

One of the aims of MapReduce is to exploit the locality of data in a shared nothing system. The parallelism is aligned with the distributed storage of data sets over the cluster so that the use of the internal cluster network is limited, as much as possible, to the shuffle redistribution phase. The input data is partitioned into “splits” which are distributed around the cluster and these splits are processed in parallel [5]. The canonical example of a MapReduce program is the word count problem, in this use case the words in a number of documents are counted in parallel using the map and reduce tasks. The following pseudocode describing the word count problem is provided in the original Dean and Ghemawat paper [5]

```
map(String key, String value):
// value: document contents
for each word w in value:
EmitIntermediate(w, "1");

reduce(String key, Iterator values): // key: document name
 // key: a word
// values: a list of counts
int result = 0;
for each v in values: result += ParseInt(v);
Emit(AsString(result));
```

Figure 16 shows the map reduce process in operation for the word count problem. A MapReduce job follows the steps from left to right. The input documents are split into sections, here they are represented as individual records but could be complete documents. The sections are distributed around the cluster and the map task operates on the splits by parsing the them and emitting a single word and a 1. At this point a combiner could be used to reduce the amount of data emitted. For example, the map task with the data “<it,1>, <it,1>, <of,1>” could emit “<it,2>, <of,1>” if a combiner was used but this extra step is not included in the diagram. The output from the map task is distributed during the shuffle phase so that all output records with the same key, in this case the word itself, are passed to the same reduce task. The reduce task then aggregates the count of the words and outputs the result.

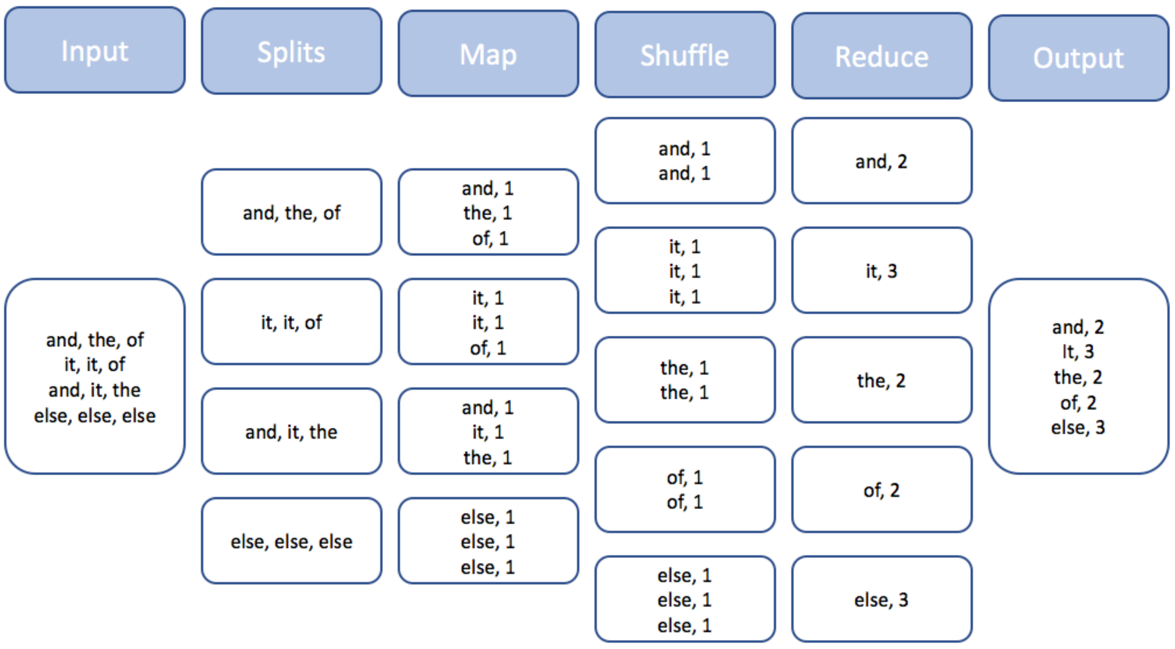


FIGURE 15 WORD COUNT EXAMPLE OF A MAPREDUCE JOB

As mentioned above, in practice, values do have to be simply counts of the occurrence of a particular key, nor indeed do they have to be numeric at all. In addition, a key does not have to be useful as part of the operation performed by a map or reduce task. An example of this would be in the case of proteomics data where the key could be the scan number and the value could be a concatenated list of all of the values in the scan that are relevant.

Subsequent to the publication of Dean and Ghemawat’s paper [5] an Apache Foundation project called Hadoop was developed to implement the concepts introduced. Hadoop includes the MapReduce framework and also a distributed file system, the Hadoop Distributed File System (HDFS), which is fault tolerant and designed to run on commodity hardware to keep the costs low [86]. The architecture and implementation details of Hadoop are described in Appendix A.

Hadoop has been adopted in the scientific community as a means of parallel processing for large, complex datasets in a large variety of domains. Some examples of its use are:

- Parallelisation of the BLAST DNA sequence alignment algorithm [143]
- “Smart City” operation, using Hadoop as a repository and processing platform for sensor data from water distribution systems [144]
- Processing of astronomical images, in this case Hadoop is used to process data from astronomical surveys allowing them to be “stitched together” to turn multiple, partially overlapping images into a single image [145]
- Analysing climate change data, large amounts of binary files are parsed and analysed using Hadoop [146]

4.3 CURRENT USE OF PARALLEL COMPUTING FOR PROTEOMICS

The adoption of Hadoop and MapReduce by the business community has been rapid. Companies are evaluating Hadoop as an alternative to traditional relational databases for reasons of cost, the amount of data that can be stored and processed and also about the types of analysis that it can perform [38]. In reviewing the published literature, this adoption has not been widespread in the field of proteomics as very few articles that cite the original MapReduce paper refer to MapReduce as a method of processing the data from mass spectrometers. A wider search to find papers related to the parallel processing of proteomics data by any method reveals that parallelisation, in general, is not being widely adopted by the proteomics community.

The papers related to parallel processing and proteomics fall into several groups.

1) Directly mention the processing of mass spectrometer data using a parallel technique, of these most are concerned with the final protein identification stage of the pipeline. Some do use the MapReduce framework whereas others use MPI or other methods of parallelization for example, [79] [87].
2) Mention proteomics in passing or in an introduction and no further reference is made to proteomics or mass spectrometers for example, Kasson et al. [88].
3) Focused on Cloud Computing rather than specifically on parallel algorithm implementations. These papers are concerned with the use of Cloud Computing to enable cost-effective ways of managing large volumes of data in a fault tolerant environment for example, [89] [90].
4) Focused on the implementation of an algorithm and mention proteomics as a possible use case. For example, “An effective and efficient parallel approach for random graphs generation over GPUs” mentions the use of a graph algorithm to process large-scale biological networks and references protein interactions as an example [91].

Following the 11th Annual Bioinformatics open source conference in 2010, a review published by Taylor lists the applications of MapReduce in Bioinformatics [92]. In this review, Taylor makes a single reference to proteomics stating that the author started a pilot project in August 2010 to build a data repository for transcriptomics and proteomics for storing vast amounts of data from mass spectrometry-based proteomics experiments. This reference refers to storing data on the Hadoop distributed file system (HDFS) but there is no mention of processing the RAW file data using MapReduce in the Hadoop environment. It also notes that the sizes of the datasets that require processing are increasing to levels where traditional methods of analysis on non-parallel computing platforms cannot produce results in a reasonable time frame. Matsunaga et al. [93] make references to the use of MapReduce in genomic sequencing, and its capabilities in facilitating the parallel processing of distributed subsets of input data. However, there is nothing specifically related to proteomics. Dudley et al. [73] examined the programming skills required by today's Bioinformatics practitioners along with statistical languages such as R and more conventional languages such as Java and Python. They note that many bioinformaticians are experimenting with MapReduce and use the work of Matsunaga et al. [93], referenced above, as an example.

A relatively recent paper [79] reviewed the use of Hadoop and MapReduce as part of the proteomics workflow, but it is concerned with the searching of identified spectra against existing reference databases. This process is the protein identification or database lookup part of the workflow described in section 3.4.5. The paper discusses the implementation of an algorithm written specifically to run on Hadoop and the benefits of scalability, flexibility and reliability. This work is important as it focuses purely on the use of Hadoop to process mass spectrometer output, but it does not provide any detail on feature detection from the RAW data. The work of Lewis et al. [79] could be pipelined together with the parallel feature detection algorithm developed during this research to produce a complete parallel process as discussed in a later chapter.

Another example of a parallel protein identification algorithm running on Hadoop uses previously identified peaks (as opposed to identifying the peaks itself) so that a search can commence, this time using a standard input format of MGF and DTA file types [94]. MGF and DTA file types are standard file types used to share proteomics data in a common format. Another review of the use of distributed computing in proteomics [95] contains many details of the types of system available, for example, grid computing, cloud computing and GPU-based systems. However specific details are not given on the implementation of data processing on these systems. An early reference to parallelizing a proteomics data process is the conversion of the Tandem protein identification engine to a parallel version called X!Tandem [96]. This system is implemented using MPI and the Parallel Virtual Machine (PVM) architecture. The research concludes that there is a significant possible reduction in the time required to identify proteins by performing searches in parallel.

Wang et al. [97] describe a process to identify peptides and proteins in parallel using cluster hardware and the MPI framework. They discuss the challenges of the ever-increasing size of the data sets involved and include detailed timings. An interesting observation is that the processing time is dependent on the number of peptides in the spectra; this means that although two spectra may contain the same amount of data, if one contains more features (which equate to peptides), it will take longer to process. This phenomenon is discussed in more detail in the results chapters of this thesis where there is a discussion on the difference between data skew and processing skew in the cluster.
A significant paper [87] discusses a parallel protein identification method using MapReduce and Hadoop in a similar way to the Hydra system proposed by Lewis [79]. Importantly it uses mzML format files as the input to the system and discusses distributing files around a cluster by splitting them into "daughter files". mzML is an XML-based format developed by the Institute of Systems Biology [15] and is a popular standard file format. Vendor specific mass spectrometer RAW files are converted into mzML before processing. Mohammed et al. [87] describe one of their large mzML files as being 1.3 Gb. This is small in comparison with those produced by the Lamond Laboratory at the University of Dundee, where files regularly reach 15 Gb. To be able to process the individual scans, this single file must be split into a series of “daughter files” as identified by Mohammed et al. A further reference to this method of splitting the input files across map tasks to exploit the “embarrassingly parallel” nature of processing individual scans is made by Kalyanaraman et al related to peptide identification via a search method [33]. This work also provides some information on timings compared with a non-parallel implementation and the reduction of total processing time from weeks to hours. Hillman et al. [98] discuss a

hybrid system using the HDFS system in the Hadoop framework to store the scan data from mzML files and a relational database to store the dimensional information in a standard third normal form schema. The work also includes the description of an example architecture, and some experimentation undertaken using MapReduce jobs to pick precursor ions from the mzML files and an example of data quality checking using PIG, a data manipulation language that is part of the Hadoop eco-system.

As previously stated, the volume of data produced by mass spectrometers is increasing as the instruments are becoming more sophisticated. The increase in volume is creating a data management challenge for those involved in proteomic studies. This situation mirrors what is happening in many other areas of life sciences such as genome sequencing. Wandelt et al. [99] describe the issues of data management in sequencing and propose the use of a cloud-based MapReduce solution. Fusaro et al. [100] discuss the use of Amazon's web services and conclude that there is a substantial up-front effort required to make an existing application suitable for parallel processing in a Hadoop system and that learning and using the MapReduce framework can be challenging. Neuhauser et al. [16] argue that although a modified version of MaxQuant running on a cluster outperforms a desktop PC, the results show that a high specification PC with solid state disks, multiple cores and a large amount of RAM can perform more than adequately for most laboratories. They recommended that rather than invest in a cluster, the PC-based approach is optimal for laboratories with medium to large data volumes. However, this thesis asserts that speed is not the only reason for choosing a central cluster-based approach for data processing. In addition to speed there are other benefits discussed in later chapters including data management, quality control, removing the burden of processing tasks from researchers and governance over the algorithms and parameters used to process the data. The use of Amazon's EC2 Cloud Computing platform is also documented with the Myrna project [92], an application used in genomics for calculating differential gene expression. Wu et al. [101] introduce a "Life Science Gateway" as a layer on top of cloud-based services where individual jobs are managed as part of a complete workflow process. The focus of this paper is managing Bioinformatics analysis running in Hadoop and does not mention the data from mass spectrometers. Hodgkinson et al. [102] also discuss the importance of cloud-based services for Bioinformatics researchers. A message passing protocol is proposed which allows the creation of workflows using hardware clusters in the cloud; one of the workflows discussed is computing protein interactions. They also mention the need to rewrite code to take advantage of the MapReduce programming framework. A review of big data, Hadoop, cloud computing and their use in genomics [103] concludes that cloud computing and big data technologies have a significant role to play in the future of life sciences. The reasons for this are stated as increases in the amount of data and increases in the number of genes and proteins that have not yet been characterised. This review also points to challenges in implementing such solutions including the high-level of Java expertise required by software developers.

Gunarathne et al. [90] present three Biomedical applications which are all related to genomics that can be easily run in parallel and therefore represent a good use case for MapReduce. They include a review of Cloud services such as Amazon EC2 and Microsoft Azure and detailed performance comparisons. Mao et al. [104] describe another workflow system called GreenPipe which is Hadoop-based and designed to be used in the cloud. They comment that although Hadoop has been proven to be successful in handling large datasets, to do this the researcher must become very skilful at MapReduce programming before they can run analyses. Prahalad et al. [105] propose a system called “Phoenix” which attempts to provide a cloud-based platform as a service layer for all ‘omic’ disciplines including proteomics although all the examples given are in the context of genomics. Wruck et al. [106] approach the issues of data management related to ‘omic studies from the point of view of open access and data sharing and also the open standard MIAPE (minimum information about a proteomics experiment) and its importance in proteomics experiments. Niemenmaa et al. [107] introduce a Java library for the processing of BAM files (Binary Alignment/Map), a format used in bioinformatics for such tasks as detecting differential gene expression. This library was written with the intention that it would be used in cloud-based Hadoop clusters acting as an integration layer between Hadoop and analytic applications.

From reviewing the published literature, it is clear there has been little research into the parallel processing of proteomics data and what there is has been focused on the protein identification part of the pipeline and not specifically on feature detection. Also, few studies involve the use of scalable clusters of commodity hardware, cloud-based infrastructure or MapReduce as a solution.

4.4 USE OF GPUS FOR INCREASING PERFORMANCE.

Other means of executing program code in parallel exist, such as using Graphic Processing Units (GPU). Although a GPU’s primary purpose is in processing the instructions necessary to provide complex graphical output, for instance in the gaming industry, the use of GPUs for general parallel processing is possible. The manufacturer Nvidia released the CUDA language for programming GPUs in 2007 [149] and more recently the OpenCL cross-platform language has become the standard [150]. GPU clusters have enabled a machine learning technique based on neural networks known as “Deep Learning” to create accurate predictive models due to the very high levels of parallelism that can be achieved [151]. For instance, a single Nvidia Titan X GPU contains three-thousand five-hundred and eight-four cores [152]**.** Once combined into clusters, an extremely high level of parallelism can be achieved. However not all processing problems are a good use case for GPU processing, for instance problems that are not “embarrassingly parallel”, that is where the workload cannot be simply split into a large number of parallel tasks with no need for communication between the tasks [153] In addition GPUs run at a slower clock speed than CPUs, for example the Nvidia Titan X runs at 1.4GHz as opposed to the Intel core I7 CPU running at 4.2GhZ meaning that serial parts or parts of the process where the input data cannot be split into small enough sections will run more slowly. A further complication is the extra task of translating the feature detection algorithm into the OpenCL language. Given these considerations it was decided that GPU processing and indeed other forms of hardware acceleration represented a separate area of research that could be followed at a later stage. The purpose of this research is therefore to concentrate on proving that parallel feature detection is actually possible in the first instance and given that it is, can it be carried out in near real-time using an algorithm coded in a MapReduce style on a scalable cluster.

4.5 PROTEOMICS, CLOUD AND BIG DATA

The term "Big Data" has gained popularity in recent years, it is used to describe data sets that create a storage or processing problem or both. The parameters of volume, velocity and variety have been used to define big data, that is where the size of a dataset (volume), the speed the data arrives for processing (velocity), and the complexity it contains and/or the variability of the format (variety) are the cause of problems with its storage or processing. Complexity can mean the quantity of processing required, the data format, the lack of structure (for example, free text) or all three. Hadoop is a key solution for processing and storing these big data datasets due to the use of commodity hardware (resulting in a low cost per terabyte for storage) and the flexibility of the MapReduce programming framework. While according to Gartner the hype around big data has begun to recede (https://www.gartner.com/doc/3115022/demise-big-data-lessons-state last accessed 19th August 2017), the management of massive complex datasets is a growing problem for many organisations. Data from proteomics experiments fits into all three of the big data dimensions. There is a large volume of data, it arrives quickly in a large laboratory with multiple mass spectrometers, and it is in a complicated format that requires a lot of processing.

In the commercial world, there is an increasing focus on cloud computing as a basis for the solutions to big data problems. Cloud computing is a name given to a broad set of scenarios where the underlying premise is that computing space and processors are rented rather than owned. There are various models to choose from, including the use of a platform as a service (PaaS) [108], where the cloud provider supplies a bare system, and the user has to setup everything else to software as a service (SaaS) [108] where the cloud provider provides a fully managed software package. Solutions such as salesforce.com, Google Apps and Cisco Webex are all examples of SaaS. Access to cloud-based systems is via the Internet or a private connection if security is a concern. There are several styles such as public and private cloud which determine whether servers are shared with others or not. Large cloud computing providers such as Amazon and Microsoft offer data centres around the world so that a user can comply with local data distribution and privacy laws.

Cloud computing has several advantages for companies wishing to use a distributed cluster of servers for parallel computing:

- The task of maintaining a complex computing cluster is turned over to the cloud provider.
- It is simple to expand or contract the number of nodes and therefore the storage and processing capacity of the cluster as loads dictate.
- The cloud provider can handle security and the latest software patches, leaving the users to concentrate on their area of expertise.
- Data management, including backup, disaster recovery and access are all part of the same service.

An analysis of cloud use for analysis of proteomics data [109] investigates the advantages of using Amazon’s EC2 system and concludes that important factors in favour of a cloud-based solution are the costs, no need for maintenance and no need to physically house a large cluster. Cloud CPFP [89] is a cloud-based application for proteomics data processing, which is built on existing tools from a suite of software called the Trans-Proteomic Pipeline (TPP). It contains software applications to cover most areas of the workflow, although the authors concede that parallelisation of the tools is limited. A more recent application of cloud technology is from the Proteocloud application which uses several well-known protein identification solutions such as X!Tandem to create a cloud-based pipeline for protein identification [110]. Given the availability of open-source tools for proteomics data processing, parallel processing and the ease of access to cloud-based resources, this research investigates the use of a cloud-based central processing cluster as part of an architected solution to proteomics data processing. This is compared to current PC-based solutions where life scientists often must handle data movement and processing themselves and the processing itself is done in an offline batch-mode. A literature search shows that recent parallel processing frameworks such as MapReduce, Spark and Flink, and alternatives to Relational Databases such as Cassandra and HBase have not been widely researched or used for proteomics data processing. Therefore, this research will concentrate on this area of parallelism and not older style frameworks such as MPI and BSP.

Since the release of Hadoop, an eco-system of related software has grown up around it to address shortcomings of Hadoop itself and to add functionality. For example, Ambari is a management system used to simplify the maintenance and management of a Hadoop cluster without resorting to command line interaction. New processing frameworks have been developed to extend the limited number of operations allowed in MapReduce and to make it easier for developers to code. Starting with platforms such as Pig and Hive, developed by Yahoo and Facebook to provide a simpler abstraction layer sitting on top of the MapReduce framework, the eco-system now includes others such as Spark and Flink which extend the capabilities and facilitate parallel programming on a cluster. Clusters of connected commodity hardware are becoming increasingly common and are to be in found in many companies and university laboratories [111]. The cluster landscape is split between

- Commodity clusters running using the Hadoop eco-system,
- Traditional high-performance computing (HPC) clusters which operate older style parallelisation such as PVM and MPI,
- Standalone, purpose-built appliances and cloud-based environments.

Many of the big computer vendors such as IBM, Oracle and Teradata now produce Hadoop appliances, which are supplied as an optimised set of nodes and interconnects pre-installed with Hadoop and the tools necessary to operate it. This has been in response to the degree of expertise and maintenance required to build and maintain a commodity cluster. While this may be true in industry where skilled resource is scarce and expensive, it would be reasonable to assume that a University with an established computer science department would have access to many computing software and hardware resources. Vendors such as Hortonworks and Cloudera have been established, offering their own distributions of Hadoop and support packages for customers wishing to build their own clusters whilst still having access to support.

With the increase in data volume and the rate at which it arrives, real-time processing is now a focus for many companies. Data sources coming in a continuous stream from social media feeds, such as Twitter, to data from sensors on machines have brought about the need for systems that can process in real-time. Processing frameworks such as Spark and Flink that can operate on top of a Hadoop system include real-time processing capabilities. The Internet of Things is a current industry trend describing the ever-increasing number of devices connected directly to computing units. A survey of examples of the Internet of Things does not list scientific data as one of the uses [2]. This thesis argues that it should. One of the goals of this research is to understand whether it is possible to treat a life sciences laboratory as an Internet of Things with the mass spectrometers connected to a central cloud-based cluster that processes the output data in real-time.

Some research on real-time acquisition has been completed using a branch of the MaxQuant code named MaxQuant Real-Time. This work uses a set of code libraries provided by Thermo Scientific, the manufacturer of the mass spectrometers that produced the data used in the research. The computer that is attached to the mass spectrometer itself carries out the processing in real-time, as opposed to a remote cluster [112]. The researchers made some modifications to the standard MaxQuant Feature Detection algorithm including relaxing the correlation values required for a 3D peak, and code to re-evaluate 3D peaks as the real-time data stream adds more data one scan at a time. The author of this thesis presented an original view of a real-time Internet of Things solution for proteomics processing at the IEEE conference on big data, 2016 [113]. The solution presented used Apache Flink running on a Hadoop cluster for the processing. This thesis expands on that research and provides more detail on a wider range of software packages and comparisons of processing time along with a comparison to batch-mode processing.

4.6 CONCLUSION AND RESEARCH QUESTION

The adoption of Hadoop has been rapid in commercial and academic areas as a system for implementing parallel processing. Amongst the expected benefits of implementing Hadoop are reduction in cost, parallelisation of complex and large datasets and scale. Research into using a MapReduce-style of parallel programming on a Hadoop system for processing mass spectrometer data is scarce. Several key papers focus on the protein identification section of the proteomics pipeline leaving the initial feature detection as a serial process. in addition to the benefits listed above, the implementation of a parallel feature detection algorithm can lead to a near real-time system using a cloud architecture, which would bring the following additional benefits to a Life Sciences Laboratory:

- The task of maintaining a complex computing cluster is handed over to the cloud provider.
- It is simple to expand or contract the number of nodes and therefore the storage and processing capacity of the cluster as loads dictate.
- The cloud provider can handle security and the latest software patches, leaving the users to concentrate on their area of expertise.
- Data management, including backup, disaster recovery and access are all part of the same service.

To realise these benefits, it is necessary to first show that feature detection can be carried out in parallel and therefore the main research question addressed in this thesis is “Can feature detection in proteomics data be performed in near real-time using a parallel algorithm on a horizontally scalable compute cluster?”. To answer the research question, the first step is to create a parallel version of the feature detection algorithm that is able to exploit the resources of a compute cluster. Given the scarcity of research in this area, the MapReduce style of parallel programming has been chosen and Chapter 5 provides the details of how this has been implemented as map and reduce tasks. Note that for the reasons stated in Section 4.4 of this chapter the research concentrates on creating and validating the parallel algorithm and does not involve hardware, for example GPU, acceleration.

5. METHODOLOGY

5.1 CHAPTER SUMMARY

To address the research question, “Can features in mass spectrometer be detected in near real-time?” this study evaluates various systems using a parallel feature detection algorithm. This chapter describes the tools employed during this research, the algorithm that was selected and coded to run using the MapReduce framework and the overall methodology for testing and benchmarking on selected platforms.

Section 2 explains the validation of results against those produced by existing, widely used, processing software.

Section 3 discusses the use of different input file formats and the effect that these have on the efficiency of the processing and storage of mass spectrometer data.

Section 4 explains the reason for choosing the programming language Java for coding the parallel algorithm. This Section also includes a discussion of alternative programming languages.

Section 5 details in full the development environment used, along with operating systems, software versions and the hardware and virtual environments used to test the accuracy of the code and the efficiency of the selected platforms. Following this, **Sections 6 and 7** discuss the details of the test and performance benchmarking environments.

Section 8 contains details of the various systems used in this research. It describes the different processing frameworks and several data storage mechanisms that are able to interface with MapReduce.

Section 9 describes the data files used in the benchmarking process.

Section 10 describes current benchmark methodologies for big data systems and the exact methods chosen to benchmark during this research. There is also a listing of the variables collected during the testing and validation phases along with ways of capturing and reporting benchmark metrics.

5.2 VALIDITY AND ACCURACY

The data used in the development and testing phases of the research was created from real proteomics experiments carried out in the Lamond Laboratory. Section 5.9 of this Chapter gives a detailed description of the data files and their sources.

Several software packages (described in more detail in Section 3.5 of Chapter 3) were used to obtain the mass and intensity values of the 3D peaks (or features) in the data.

- Maxquant
- Proteowizard
- The Spectracus project (Lamond Lab, Centre for Gene Regulation and Expression)

The results from the algorithm created by this research were validated by comparing them to the outputs from the systems named above. In line with standard practices among proteomics researchers, the metrics used to verify the results were precision and recall [74], [77], [114]. Chapter 6 presents the results of the benchmark testing using a batch-based process and gives a more detailed explanation of the precision and recall metrics. Chapter 7 presents the results of a streaming process.

5.3 DATA FORMATS FOR PARALLEL PROCESSING

As stated, to openly share results vendor RAW files are commonly converted to mzML files, and there are many software packages available to do this. A frequently used package is Proteowizard [115]. For Thermo Scientific instruments, the conversion software provided as part of this package is called msconvert and uses a Windows DLL library provided by the manufacturer to decode the RAW file. The decoded data is then written out to a new file in mzML format. Table 6 in Section 5.9 of this chapter displays a list of the test files used in this research and the elapsed time taken to convert between RAW and mzML formats.

The design of mzML incorporated certain principles including comprehensive support for instrument output, the sharing of results and best practice. XML is a good format for sharing data because it is self-describing and the mzML schema contains many metadata fields for detailing experimental conditions; however, there are downsides to using XML. One is the growth in file size caused by the space taken up by the XML tags which define the structure of the file. For the files used as test data in this research, the growth has been approximately five-fold. Note that much of this growth is due to the conversion from the binary RAW format to the text-based mzML format. This can be seen in Table 6, Section 5.9, that shows the size of the test files in various formats. For example, file PT2441S1FP1A1.RAW is 2,704Mb, the mzML file is 3.09 times larger, the SCMI format file that does not contain XML tags (defined in this later in this section) is 2.87 times larger and the Avro format file is 1.96 times larger.

A further downside is the difficulties XML causes when processing files in a parallel fashion. Files need to be split into pieces for distribution around a cluster of machines. This splitting is hard to do with XML as the files need to be split in the correct places or the XML start and end tags will not match and the file will be invalid. Complex parsers are required to allow XML to be used effectively on a distributed system [140]. For this reason, different file formats have been researched to assess their suitability for the parallel processing of mass spectrometer data.

The mzML file contains metadata detailing the experiment; it includes the parameter settings of the mass spectrometer and the type of machine used. This information is not needed when performing the feature detection and protein identification steps of the data processing pipeline. Therefore, a data format used for parallel processing can ignore this metadata and store just the scan data plus a small number of additional fields that provide context for the scan data. The scan data is held in two base64 encoded strings, one for the mass to charge values and one for the intensity values. The fields required for feature detection are as follows:

- **Scan identifier**, an integer field numbering the mass spectrometer scans in order of retention time, starting at 0
- **msLevel**, an integer field with 1 being the first level of scans and a 2 indicating a level 2 scan. Level 2 scans are created when a particular peptide is picked from a level 1 scan and further broken down into smaller molecules which are then measured in a second ion detector inside the mass spectrometer.
- **Retention Time**, a decimal field, the time since the start of the experiment at which the scan was taken.
- **precursorIonMz**, a decimal field for level 2 scans only, this field contains the mass to charge value of the peptide picked to be broken down further from the level 1 scan
- **precursorIonIntensity**, a decimal field for level 2 scans only, this field contains the intensity of the peptide picked to be broken down further from the level 1 scan. Intensity in this context means the number of ions detected
- **precursorIonCharge**, an integer field for level 2 scans only, this field contains the value of the electric charge of the peptide picked to be broken down further from the level 1 scan
- **MZAarray**, a text field containing the base64 encoded mass to charge values
- **intensityArray**, a text field containing the base64 encoded intensity values

Initial investigations used a simple text format named the SCMI format (SCan, Mass, Intensity) with one record per scan and the various fields separated by tab characters. This format is extremely easy to split and distribute around a cluster, and the HDFS file system in Hadoop can natively do this. A file copied into HDFS will be split into chunks (64Mb chunks by default), which are then distributed to the nodes in the cluster; this process and HDFS itself are explained in more detail in Appendix A. This text-based file format proved to be very efficient as described in detail in the results chapter. However, with a file format such as this, the benefits of a self-describing format are lost because the text format requires a definition document so that programmers will know what data is in each of the columns. Apart from XML, other self-describing formats include JSON, Parquet and Avro. These formats are popular as ways to store complex data and are also suitable for parallel processing. The following sections give a brief description of these file formats.

In response to the complexity of accessing data stored in XML-based formats such as mzML, Rost et. al researched a method for random access to mzML data using the existing OpenMS libraries as a base [148]. Rost et. al benchmarked their tool against the ProteoWizard libraries and noted a significant performance increase. In relation to the research presented in this thesis, the tool developed by Rost et. al could be used to decode the mzML files into the proposed SCMI format, however following continued research into this area it was decided to not use mzML as an intermediate format and decode the RAW binary files directly into the SCMI format as the data was copied onto the processing cluster. This means that the issues of XML complexity and difficulties with file distribution are avoided.

5.3.1 JSON

JavaScript Object Notation (JSON) is a self-describing lightweight data-interchange format. It is deemed lightweight as it adds less to the overall files size as would a format such as XML. The general JavaScript Programming Language specification [116] gives a detailed description of the format. The basic structure of a JSON file is plain text. This means that it is a simple task to sample and investigate a file's contents without complex parsers. A JSON file comprises key-value pairs where the key is the description of the field and the value contains the contents of the field. The JSON specification allows for two principal structures

- Objects - an unordered collection of key/value pairs
- Array - an ordered list of values

The following extract shows the simple flat file version of the mzML file (note the mzarray and intensity array fields have been shortened for display purposes).

scan	mslevel	rt	precursormz	precursorintensity	precursorcharge	mzarray	intensityarray
0	1	0.00680	0	0	0	oXXPWVmodUCh	ro1lqHVAUV6i0W
1	1	0.01223	0	0	0	2dX6h1QKJ9ruF	xWiJWyodUB5s7b
2	1	0.01669	0	0	0	PAfQBVmodUDsF	6h1QJ3ero1lqHV
3	1	0.02115	0	0	0	Q1Un41iodUDrW	hUnX6h1QJQYt
4	1	0.02561	0	0	0	2T3V3liodUBHQ	sMiX6h1QLX+s2Z
5	2	0.03952	362.22271	26164.71875	2	irEWTEvASEC+	EDWJ59ngoRMQI

This next extract shows the same data formatted as a JSON file.

```
[
  {
    "scan": 0,
    "mslevel": 1,
    "rt": 0.0068095369,
    "precursormz": 0,
    "precursorintensity": 0,
    "precursorcharge": 0,
    "mzarray": "oXXPWVmodUCh",
    "intensityarray": "ro1lqHVAUV6i0W"
  },
  {
    "scan": 1,
    "mslevel": 1,
    "rt": 0.01223972,
    "precursormz": 0,
    "precursorintensity": 0,
    "precursorcharge": 0,
    "mzarray": "2dX6h1QKJ9ruF",
    "intensityarray": "xWiJWyodUB5s7b"
  },
  {
    "scan": 2,
    "mslevel": 1,
    "rt": 0.01669872,
    "precursormz": 0,
    "precursorintensity": 0,
    "precursorcharge": 0,
    "mzarray": "PAfQBVmodUDsF",
    "intensityarray": "6h1QJ3ero1lqHV"
  },
  {
    "scan": 3,
    "mslevel": 1,
        "rt": 0.02115777,
        "precursormz": 0,
        "precursorintensity": 0,
        "precursorcharge": 0,
        "mzarray": "Q1Un41iodUDrW",
        "intensityarray": "hUnX6h1QJQYBmt"
    },
    {
        "scan": 4,
        "mslevel": 1,
        "rt": 0.02561677,
        "precursormz": 0,
        "precursorintensity": 0,
        "precursorcharge": 0,
        "mzarray": "2T3V3liodUBHQ",
        "intensityarray": "sMiX6h1QLX+s2Z"
    },
    {
        "scan": 5,
        "mslevel": 2,
        "rt": 0.039521102,
        "precursormz": 362.222717285156,
        "precursorintensity": 26164.71875,
        "precursorcharge": 2,
        "mzarray": "irEWTEvASEC+AG",
        "intensityarray": "EDWJ59ngoRMQI+3+cq"
    }
]
```

Note that JSON files are record-based and therefore suitable for distribution on a compute cluster. JSON files are more lightweight than XML but still verbose compared with a plain record-based text file or Avro due to the key/value tag structure.

5.3.2 AVRO

Apache Avro was designed by Doug Cutting, the original developer of Hadoop. Avro files have a schema which is only stored once at the start of the file. Storing the schema once in this way achieves some level of compression over a JSON or XML formatted file. The data itself is stored in a binary format for further compression. Avro was designed so that files could be easily split into chunks for distribution on a cluster. To convert data into Avro format, a schema must first be created, following this, standard code libraries can be used for the conversion process. An Avro schema is represented as a JSON file detailing the fields in the file. The following JSON string shows the Avro schema for the mass spectrometer output data used in this research

```
{
        "type": "record",
        "namespace": "scmitest.avro",
        "name": "massspecdata",
        "fields":[  {"type": "int", "name": "scan"},
                    {"type": "int", "name": "mslevel"},
                    {"type": "long", "name": "rt"},
                    {"type": "long", "name": "precursormz"},
                    {"type": "long", "name": "precursorintensity"},
                    {"type": "int", "name": "precursorcharge"},
                    {"type": "string", "name": "mzarray"},
                    {"type": "string", "name": "intensityarray"} ]}
```

It should be noted at this point that the choice of file format for parallel processing is between XML, SCMI, JSON and Avro files. The original vendor specific binary RAW files were dismissed as a poor choice due to several factors including the complicated method of extracting the scan data, the reliance on the use of a proprietary .DLL file for decoding the binary date (windows based operating systems only) and the difficulty in distributing a RAW file on a cluster. Of the remaining choices, XML and JSON are discarded for reasons of verbosity and difficulty in data distribution for XML. This leaves the Avro and SCMI files as candidates both of these file formats store the data arrays for m/z and intensity values as Base64. Of the systems being benchmarked only HDFS stores data as files, Cassandra, HBase and Aster (described further in Chapter 5, Section 8) all have the concept of tables, although the implementation and definition of a table differs between them. Using the text base SCMI file in HDFS means that all of the systems have a similarly structured data source and makes the task of debugging far simpler as opposed to needing to use an Avro decoder to extract the specific scan data Base64. Note that this does not affect performance but does result in an increase in disk space usage.

5.3.3 PARQUET

Parquet is a relatively new format introduced in 2013. Being similar to Avro it uses a schema to self-describe its contents. However, whereas Avro stores data in rows, Parquet is a columnar file format. This can lead to high level of compression and can be a very efficient way to store data. Columnar formats are typically effective when files contain many columns and when processing tasks involve selecting subsets of columns. In the case of feature detection from mass spectrometer scans, the opposite is true. The files contain only the eight columns listed above and each scan is read completely with no sub-selection. For this reason, Parquet was not considered as a suitable format for the experimentation. However, it should be noted that Parquet is becoming a standard format for data when using the Spark processing framework (described in detail in Appendix A). This is partly due to its adoption by Databricks one of the chief supporters and contributors to the Spark codebase. Databricks has included enhancements to the Spark code that make Parquet file handling very simple and fast.

5.3.4 FILE CONVERSION

The creation of a custom conversion program written in Java allowed the conversion of a mzML file to a simple text-based format. One potential issue was the extra time taken to convert the data. However, any process using a cluster needs the data to be copied onto the cluster first. To exploit this, the custom conversion process was extended to copy the data to the cluster and convert it to the text-based format as part of the copy. This custom copy program minimised the amount of time that was added to the whole process. Chapter 5 Section 3 details the timings for converting the text files from RAW format and copying them to cluster in one step.

Initial experiments were carried out by converting RAW files to mzML using the proteowizard msconvert program and then custom code used to convert from mzML to the text-based format. However, since the mzML format is not actually a requirement for the data processing pipeline, the conversion from RAW format to mzML is also not required. This means that it is possible to further refine the process and remove the RAW to mzML conversion step. Therefore, a time saving was achieved by modifying the custom conversion and copy program to produce the required text format file directly from the vendor-specific RAW file.

5.4 CODING

The MapReduce style of parallel programming is used extensively in this research and all but one of the systems tested operate within the Hadoop eco-system. The Hadoop and the MapReduce frameworks are themselves written using the Java programming language. It is possible to code MapReduce jobs in other languages using the stream interface which receives input from the Linux stdin and outputs to stdout. However, for this research it was decided to use Java, partly to align with the researcher's knowledge of computer languages and partly to use the native libraries which include a variety of input and output readers as opposed to the stream interface which can only read and write text-based formats via stdin and stdout. There are many benefits to using Java including the operating system independence of the final code and the significant number of pre-written third-party code libraries that exist to speed up development. Note that early versions of Java were notoriously slow compared to pre-compiled languages. However, since the introduction of "Just in Time Compilation" (JIT) in versions 1.2 and 1.3, Java performance has significantly improved.

Python is a popular scripting language widely used in the analytics industry**.** Another scripting language is R, which is very popular amongst statisticians and in the Data Science community. There are libraries available for proteomics data processing written using a variety of programming languages, the list below is an example of some of the frequently used packages.

- OpenMS 2.0 [117] implements a proteomics processing pipeline using C++ and integrates with the open-source workflow tool, Knime. The Knime integration allows a researcher to build a graphical flow of the processing and schedule it to run. The graphical flow is a beneficial tool for helping to overcome some of the data management challenges.
- PyOpenMS [118] is a Python-based version of the original OpenMS library [119]
- Bioconductor [120] is a set of proteomics data analysis functions available in an R library.
- Dinosaur [121] is a recent development of a processing pipeline using the programming language Scala.

Scala is the main language for the processing framework Spark, explained in detail in Section 5.8.6 of this Chapter. Based on Java, Scala is gaining popularity as a language for scalable parallel programming. All the systems and libraries mentioned above are open source with freely available code.

Proteomics data processing packages are available with open source code in various languages. However, none of those explicitly addressed the task of developing a parallel feature detection algorithm. Therefore, the decision was made to code this algorithm in Java as part of this research. The third-party libraries incorporated into the code are the Apache commons libraries for base64 decoding and the Hadoop libraries themselves; all other code is original and written during this research. Java version 1_8.0_77SE was used for the development and testing. Note that the use of a pre-compiled language, such as C++ can result in faster run times than Java, where the code is compiled at the time of execution. Therefore, it may be possible to further speed up the feature detection process by implementing the parallel algorithm in such a compiled language.

5.5 MAPREDUCE IMPLEMENTATION OF THE FEATURE DETECTION ALGORITHM

The parallel algorithm for feature detection used in this research has been designed based on previous work by Bellew [84] and the additional material supplied as support to the original MaxQuant paper [12]. The serial implementation of the algorithm is described in Chapter 3, Section 3.5.1. The following describes the implementation of the algorithm using the MapReduce framework.

The serial algorithm is comprised of two major parts, 2D peak picking and 3D peak picking. In essence the 2D peak picking algorithm takes a single scan of mass spectrometer data, processes it and outputs a set of 2D peaks. The 3D peak picking algorithm, takes groups of 2D peaks, processes them and outputs a single 3D peak for each group of 2D peaks. This section describes how the 2D peak picking algorithm can be represented as a map task that transforms a single scan into a set of 2D peaks and the 3D peak picking algorithm can be represented as a reduce task that takes a set of 2D peaks and “aggregates” them into a 3D peak. The implementation is complex as there are many steps to the 2D and 3D peak picking algorithms. However, the whole process can be represented as a simple pattern of a map transformation followed by a reduce aggregation.

```
map (k, v) -> set(k1, v1)
reduce (k1, list (v1)) -> set (k1, v2)

k = scan identifier
v = data from mass spectrometer scan
k1 = partition number used to redistribute 2D peaks to reduce task
v1 = 2D peak values
v2 = 3D peak values

an additional custom partitioner step is used to calculate which
partition a particular 2D peak belongs to can be represented as

partition(v1) -> k1
```

In Chapter 4, the MapReduce framework was explained in terms of the canonical word count problem. For reference, the pseudocode for the map and reduce tasks is reproduced below.

```
map(String key, String value):
// value: document contents
for each word w in value:
EmitIntermediate(w, "1");

reduce(String key, Iterator values): // key: document name
 // key: a word
// values: a list of counts
int result = 0;
for each v in values: result += ParseInt(v);
Emit(AsString(result));
```

2D peak picking

To represent 2D peak picking as a map task, the input value becomes a delimited string of the data in a single record of the SCMI file format described in section 5.3 of this chapter with the key being the byte offset of the record in the file. The SCMI file format contains the following columns of data: scan, mslevel, rt, precursormz, precursorintensity, precursorcharge, mzarray, intensityarray. This delimited string of values represents a complete mass spectrometer scan taken at a particular retention time.

The 2D peak picking process represented as a map task is as follows.

```
map(String key, String value):
// key: byte offset of the record in the input file
// value: "scan, mslevel, rt, precursormz, precursorintensity, 
precursorcharge, mzarray, intensityarray"
for data in mzarray, intensityarray:
    detect the 2D peaks
EmitIntermediate(partition number, "weightedPeakMass, 
charge,retentionTime,sumIntensity, minimumMZ, maximumMZ");
```

The term “`detect the 2D peaks`” refers to the 2D peak picking process as described in Chapter 3. As this is an embarrassingly parallel task, it can be run as a map task without any changes to the algorithm as it already works on an individual scan basis. The map task outputs a list of 2D peaks found in the scan, with the key being the reduce partition number and the value being a concatenated string of the 2D peak metrics. The reduce partition number is calculated using a custom partitioner function based on the weightedPeakMass value and the number of partitions required. This calculation is described in Chapter 6, Section 6.4 where there is a full discussion on data distribution and partitions.

In operation, the map task reads scans of data from the text-based SCMI-format file using the standard Hadoop text file reader, which passes the byte offset of the start of the record as the key and the rest of the scan data in a text string datatype as the value. The value part of the key-value pair contains the components of the scan data as tab delimited values, these are parsed and read into local variables for processing. The mass to charge ratios and intensity values are decoded from the base64 format using the Apache commons library and stored in float arrays, The Java Double type is a floating point datatype represented in 64 bits, giving approximately fifteen precise digits. This is sufficient to accurately store the values required which range between zero and one thousand with up to eight decimal places. The method of reading in a complete record from a data source and using the program code to apply a schema to it is called schema-last or schema on read. In this case, the definition of the fields within a record are in the Map task programming code and applied to the data at run time. Modifications are made to this method when using a data source other than text files stored in HDFS, for example HBase or Cassandra. For data sources which already store the data with a defined schema, the input and output interfaces are modified to use the data types and definitions from that schema. This is described in more detail in Section 6.5 of this chapter where results are presented and discussed. As stated, all the information necessary to calculate the 2D peaks in a scan is contained within the scan itself. No communication of values between nodes is required at this stage and each scan can be processed completely independently.

This results in a process of intra-file parallelism, which has not been widely researched or applied.

Figure 16 shows the 2D peak picking process arranged as a map task, the input scans are split into sections, in the example shown the twelve input scans are split into four instances of the map task each with three scans. In practice each map will receive thousands of scans. The map task then process each scan in turn and outputs the intermediate keys, which equate to the partition number and 2D peak metrics, for distribution to the reduce tasks.

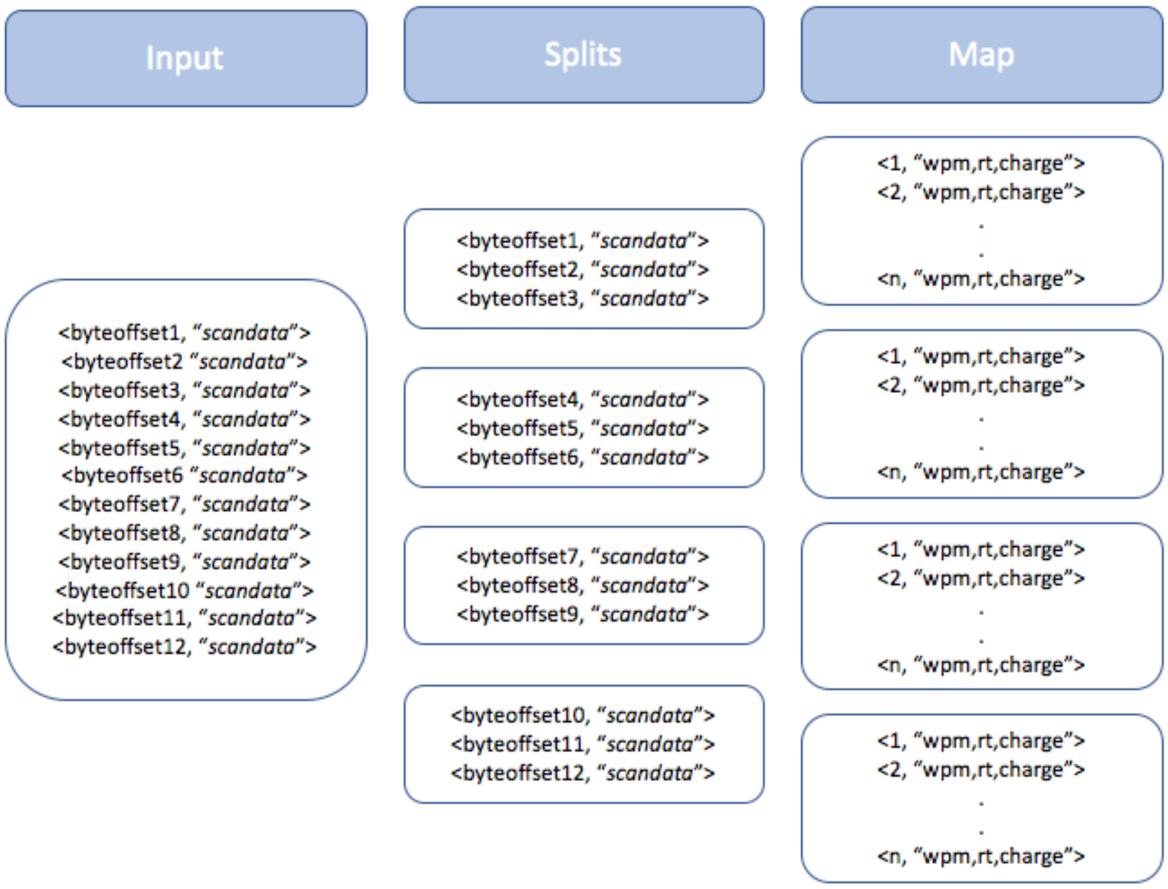

Input
Splits
Map
<byteoffset1, "scandata">
<byteoffset2 "scandata">
<byteoffset3, "scandata">
<byteoffset4, "scandata">
<byteoffset5, "scandata">
<byteoffset6 "scandata">
<byteoffset7, "scandata">
<byteoffset8, "scandata">
<byteoffset9, "scandata">
<byteoffset10 "scandata">
<byteoffset11, "scandata">
<byteoffset12, "scandata">
<byteoffset1, "scandata">
<byteoffset2, "scandata">
<byteoffset3, "scandata">
<byteoffset4, "scandata">
<byteoffset5, "scandata">
<byteoffset6, "scandata">
<byteoffset7, "scandata">
<byteoffset8, "scandata">
<byteoffset9, "scandata">
<byteoffset10, "scandata">
<byteoffset11, "scandata">
<byteoffset12, "scandata">
<1, "wpm,rt,charge">
<2, "wpm,rt,charge">
<n, "wpm,rt,charge">
<1, "wpm,rt,charge">
<2, "wpm,rt,charge">
<n, "wpm,rt,charge">
<1, "wpm,rt,charge">
<2, "wpm,rt,charge">
<n, "wpm,rt,charge">
<1, "wpm,rt,charge">
<2, "wpm,rt,charge">
<n, "wpm,rt,charge">

FIGURE 16 2D PEAK PICKING AS A MAP TASK

Shuffle

Skew for the reduce tasks occurs when one or more nodes receives a disproportionately high amount of data to process from the output of the map tasks. Depending on the processing task, the node or nodes with the extra data may continue processing after the other nodes have finished. This delays the completion of the whole job and results in inefficient use of the cluster. By default, the MapReduce framework redistributes data from the map tasks to the reduce tasks by applying the standard Java hash algorithm to the keys in the key-value pairs output from the mapper. In general this produces an acceptable data distribution and does not result in skew. However, during testing, it became apparent that this was not the case for proteomics data. To overcome this the MapReduce framework provides a task called a custom partitioner that allows the programmer a fine degree of control over how data is redistributed from the mappers to the reducers avoiding the problems with data skew. The proteomics data is highly skewed towards a certain range of mz values, as can be seen in Figure 17 which shows the distribution of the 2D peaks in file 561L1AIL00.RAW as an example. In addition to this the processing required to detect features in a certain range of data also depends on the ratio of information to noise.

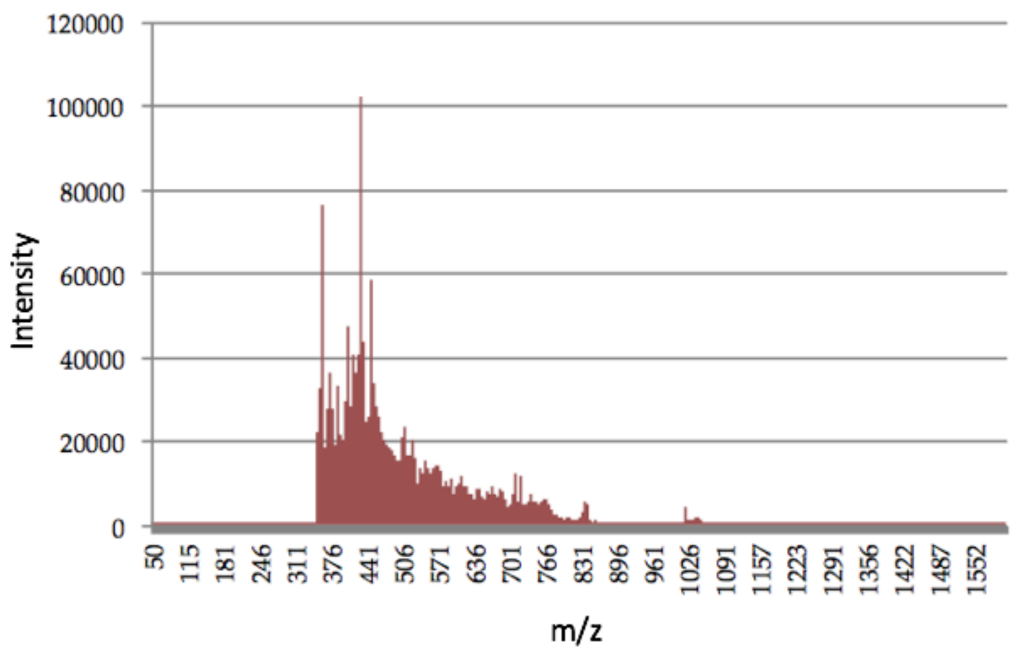


FIGURE 17 SKEWED DISTRIBUTION OF 2D PEAKS

The set of 2D peak values output from the map tasks must now be distributed to the reduce tasks so that processing can continue in a parallel fashion. To achieve this, the 2D peaks are grouped into partitions. The 2D peaks belonging to a particular partition will all be distributed to the same reduce task. In MapReduce terms this process is known as the shuffle. The partitions are made by grouping the 2D peaks by ranges of m/z values. Enough overlap of data must be provided at the split points so that if a partition split occurs in the middle of a feature, as shown in Figure 18, that the complete peak can be found either side of the split.

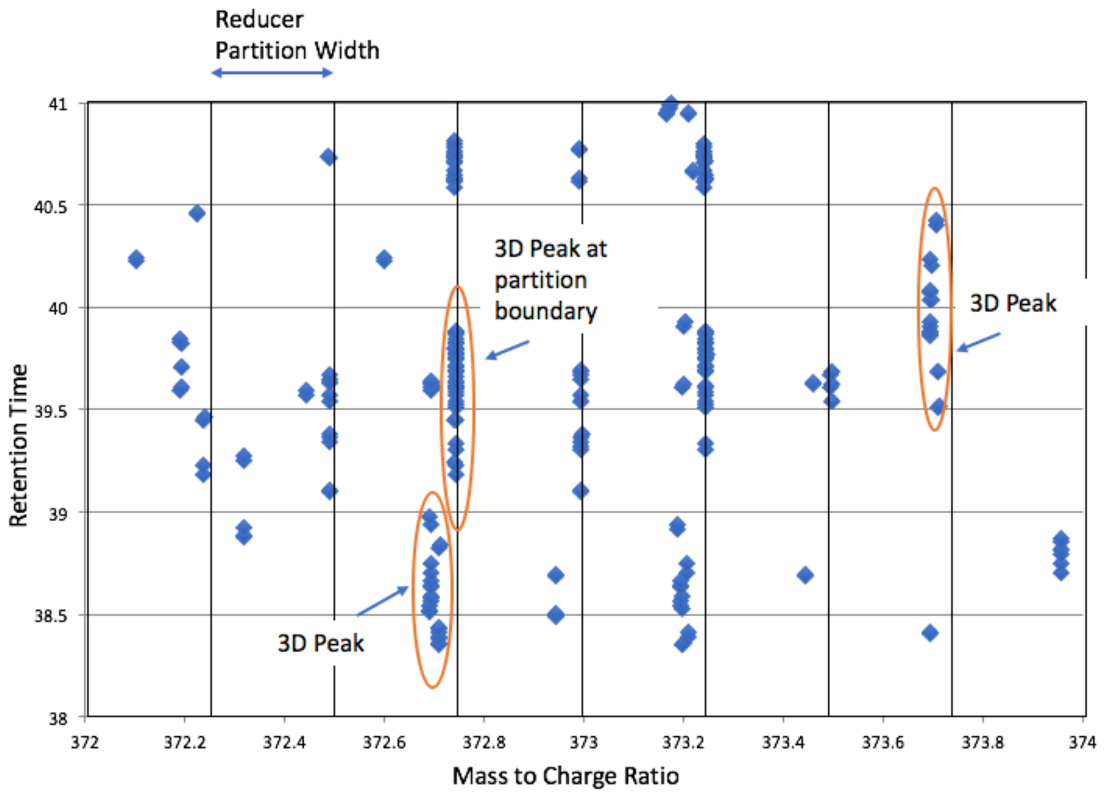


FIGURE 18 EXAMPLES OF PEAKS OCCURING AT PARTITION BOUNDARIES

This overlap strategy necessitates a de-duplication phase as a final step. A more complete description of the partition and overlap strategy can be found in Chapter 6, Section 6.4. A custom partitioner must be coded to provide an output key to each input value; this key is used to decide to which reducer the value is sent for processing. Initially this has been done by assuming a fixed number of reduce steps. The 2D peaks from the Map tasks are grouped into clusters with the aim that each cluster contains an equal number of peaks. It is acknowledged that this hardcoded partitioner while adequate is far from an optimum solution for this problem and this is discussed later in section 6.3.1 of Chapter 6. The following pseudocode describes the custom partitioner

```
Partition(String value):
// value: weightedPeakMass
calculate partition p
Emit (p);
```

3D Peak Picking

With the list of 2D peaks split into approximately even portions by weighted peak mass and an appropriate overlap added, the 3D peak picking process can now be completed in parallel. 3D peak picking adds the dimension of retention time to the mass and intensity dimension utilised in the 2D peak picking stage. The 3D peak picking step is comprised of several stages that are wrapped into a Reduce task in the MapReduce framework as shown in pseudocode below.

```
reduce(String key, Iterator values): // key: document name
 // key: a partition value
// values: concatenated string of "weightedPeakMass,
charge,retentionTime,sumIntensity, minimumMZ, maximumMZ"
for values in value:
    join 2D peaks in 3D chains
    smoothIntensity
    identify overlapping peaks
    identify isotopic envelopes
    calculate mono-isotopic peak values
Emit(finalKey, finalValue);
//finalKey: partition number (not required)
//finalValue: concatenated string of "charge, mass, intensity,
retention time"
```

The reduce tasks receive data from the map tasks as key-value pairs dictated by the custom partitioner. The input data consists of an integer key and a text value. The text value is a tab-separated list of the variables calculated during the 2D peak picking with each input record representing a single 2D peak. The input records are loaded into an arraylist of Java objects that enables them to be easily sorted using a custom compare class inside the object. The 2D peaks are first sorted in order of retention time (which equates to scan number) and then sorted by mass to charge ratio within each retention time before starting to loop through the 2D peaks checking to see if they are part of a 3D peak chain. From this point the operation within the reduce task follows the same flow as described for the 3D peak picking process in Chapter 3, Section 5. That is joining 2D peaks into 3D chains of peaks, intensity smoothing, overlapping peak detection, isotopic envelope detection and mono-isotopic peak calculation.

Figure 19 shows the 3D peak picking process arranged as a reduce task, the output from the map tasks is distributed to the reduce tasks in the shuffle phase. The reducers then detect the 3D peaks and output them as a list per reducer that needs to be joined to create the final output. In the diagram n = the number of reduce tasks.

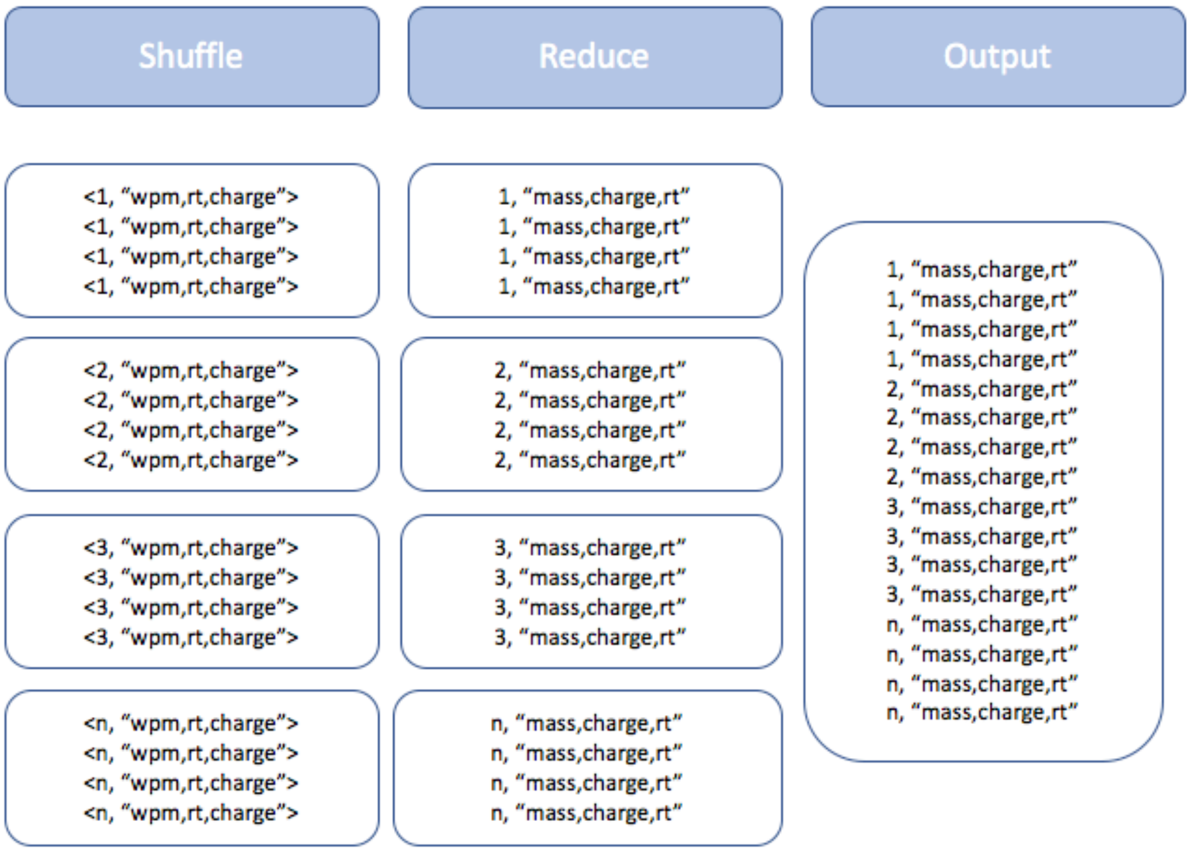


FIGURE 19 3D PEAK PICKING AS A REDUCE TASK

5.6 DEVELOPMENT ENVIRONMENT

5.6.1 CODE DEVELOPMENT

The Java code has been developed, tested and debugged using the Kepler release of Eclipse on a Macbook Pro laptop. Eclipse is an Integrated Development Environment (IDE) used for programming in a variety of languages. Eclipse provides a range of tools to aid the process of program code development, such as syntax checking and compiling and deploying Java packages as Java archive or "jar files" (so named because by convention the file extension is always “.jar”). As the Apple Macintosh operating system is based on BSD Unix, it is possible to run the Hadoop MapReduce framework on the local file system natively. A setup like this does not create a parallel processing environment as there is effectively only a single worker node, but it is very efficient for testing and debugging program code during development.

A plug-in exists that allows Aster SQL-MR packages (see Section 5.8.4 for the definition of SQL-MR and Aster) to be developed in Eclipse and easily deployed to a cluster from within the IDE by running an SQL-MR script. The plug-in makes iterative testing and deployment simple and efficient and means that a single IDE was used for development across all the systems included in the research.

5.6.2 SOURCE CONTROL

During the lifetime of this research, version control of the Java source code has been crucial. During the initial development phase of the MapReduce algorithm, testing the results against existing correct peak outputs involved many iterations over the development, testing and release cycle. The version control required a detailed version history of changes and to enable rollbacks to previous code releases. The version control system chosen was Git, originally developed by Linus Torvalds, the architect of the Linux operating system. Git is a distributed system where a master of the code resides in a repository on a remote server and developers have a copy of the repository on their local workstation. A pull request is made to receive the latest version of the code repository and to commit changes a push command is made. Other standard version control features such as branching, merging and cloning are available.

The GitHub hosting service was used to provide a remote repository. GitHub provides an online hosted service that allows the master repository to be accessed by anyone with correct access rights and an Internet connection. All the code written during this research is available at the URL https://github.com/chillman99/phd

5.7 TEST ENVIRONMENT

5.7.1 HARDWARE SPECIFICATION

Research into the performance of the parallel algorithm utilised several iterations of test hardware with the setup being refined on each iteration. The aim was to provide enough capacity to be able to process complete mass spectrometer output files and provide consistent running times to benchmark the results. Initial stages of the research utilised a test cluster built with re-purposed desktop personal computers from an office environment. A previous Master's degree project by the author of this thesis used the same test cluster as part of a discussion on the use of Hadoop in the context of a hybrid ETL process for proteomics data [122].

However, the eight personal computers used as the data nodes had single core 1.8 Ghz processors with one gigabyte of RAM and a single 80 gigabyte spinning disk drive. While good as a learning environment, this configuration did not have sufficient disk capacity or processing power to process the large files produced during a proteomics experiment, which as previously stated can reach more than seven gigabytes. Therefore, a new virtual cluster was constructed on a custom-built server using an eight-core AMD processor with 32 gigabytes of RAM and five individual drives. A single virtual machine was assigned to each of the drives to avoid access bottlenecks.

The results of initial benchmark tests using virtual machines showed a high degree of variation in run times. An investigation into the performance of the virtual cluster using spinning disk drives showed that there is a high degree of correlation between runtimes of code and how much data the drive contained before copying a virtual machine image onto it. One reason for this variation, is that on a traditional spinning disk hard drive, the data is stored starting at the outside of the platter and progressing inwards towards the centre. The data on the outside of the platter can be read more quickly as the rotational velocity is faster under the read-head at the outside than it is on the inside. As the disk fills, the later data is stored towards the centre of the disk where read times are slower. Another significant factor is that of fragmentation. Files on an empty drive can be written as contiguous blocks. In contrast, on a disk that already contains data it is likely that files will be fragmented. Fragmentation describes the situation where the data blocks are interspersed with blocks from other files that already existed on the disk and results in slower file read times. As stated, the results showed a significant variation between timings for running the process as recorded using the Linux “time” command, Hadoop dfsio and teradsort performance tests and I/O benchmarking software. Therefore, the host machine was upgraded to replace the original spinning disks with solid-state drives (SSD). The random-access times for SSDs are consistent across all parts of the drive and independent of how much data they contain. Changing to SSD drives resulted in far more consistent runtime across independent tests.

Table 1 shows a summary of the results obtained using three different disk drives: a Sandisk SDSSDP-128Gbm, a Seagate ST31000528AS and a Western Digital WD20EARX. The tests were carried out by copying a virtual machine on to freshly formatted blank drive for the “empty” readings and then copying the same virtual machine on to a drive having only 30Gb remaining space for the “full” readings. Two things of note here are firstly the virtual machine severely restricts the I/O of the drives, the SSD write speed is only about 20% of the speed when using the native operating system. Secondly the read, write and terasort measurements are much more stable on the SSD than the spinning drives and not dependent on how much data is currently on the disk. Each test was run three times and the results here are averages of the three readings. The results show that the 128Gb SSD drive was the fastest regardless of whether it was empty or full at the time of the test.

TABLE 5 COMPARISON OF I/O DATA FOR SSD AND SPINNING DISKS

Model	Type	Capacity	State	Native OS Read (MB/s)	Hadoop on VM Read (MB/s)	Native OS Write (MB/s)	Hadoop on VM write (MB/s)	Hadoop Terasort Write (hh:mm:ss)	Hadoop Terasort Process (hh:mm:ss)
Sandisk 128Gb SDSSDP-128G	SSD	128Gb	Empty	315.51	83.21	240.87	52.82	00:03:18	00:27:42
Sandisk 128Gb SDSSDP-128G			Full	316.01	83.13	240.63	52.43	00:03:15	00:27:25
Seagate Barracuda ST31000528AS	Disk	1Tb	Empty	114.55	46.53	99.84	40.28	00:03:56	00:33:49
Seagate Barracuda ST31000528AS			Full	109.01	37.98	65.21	37.38	00:04:02	00:54:16
Western Digital WD20EARX	Disk	2Tb	Empty	108.09	36.57	101.97	44.25	00:03:31	01:44:43
Western Digital WD20EARX			Full	106.78	23.06	100.84	32.27	00:04:05	03:25:33

5.7.2 OPERATING SYSTEM

The operating system installed on the host server running the test system was Windows 10. Thermo Scientific, the manufacturer of the mass spectrometers used at the University of Dundee, provides a single mechanism for reading the binary data in their RAW data formats. This implementation of this mechanism is a Microsoft Windows DLL file. It is necessary to call the routines in this DLL file in order to convert the RAW format into any other required format. This dependency on a DLL file is, therefore, the main reason for choosing a Microsoft Windows host. Without it, it would not be possible to read the RAW data, convert it into a suitable file format and test the algorithm.

To produce consistent run-times for replicates of experiments, non-essential services on the host machine were shut down. During the testing phases, the processes that automatically search for and install software updates were switched off. This precaution was taken both for Windows itself and other software packages such as Microsoft Office. As a further precaution, the host machine did not have access to the Internet.

5.7.3 ENVIRONMENT

The different cluster environments tested were installed as groups of individual virtual machines that together created a virtual cluster on the host. This virtual setup proved to be an efficient method of researching multiple systems and combinations thereof in a single test environment. The virtual machines were created using the VMware platform, specifically the free VMware Player software version 12.5. The allocation of a virtual hard drive fixed at 40 gigabytes in size, two CPU cores and four gigabytes of memory to each of the virtual machines gave a consistent base for each type of cluster. Here the aim was to prevent the timings of the experiments being affected by virtual machine maintenance activity, such as growing a virtual hard drive file behind the scenes. The virtual cluster created for use as a development proved to be an essential part of the process of changing the serial algorithm detailed in Chapter 3 into the MapReduce-based parallel algorithm detailed in Chapter 5. The laptop environment contained a single node “pseudo cluster” this means that all services and data run in the same place. So, although the code runs as MapReduce, in practice there is no parallelism. The virtual cluster allowed development and testing to be run on an actual parallel system without having to negotiate extended periods of access to a physical cluster. Issues such as data distribution, processing skew and limitations of processing large datasets where detected and tested on the virtual cluster where the nature of the single-node pseudo cluster means this wouldn’t have been possible. Using the virtual cluster also showed that two parts of the process were considerably slower than the rest when the size of the data increased. This was due to the way the algorithm was initially coded and would have had the same effect on the physical cluster, this is discussed further in Chapter 6, Section 6.3.3.

5.7.4 CREATION

Installing Ubuntu Linux 16.04.02 LTS as the operating system on a new virtual machine with the specifications given above created a template system for a virtual node. Cloning this template node ensured that all the nodes in the clusters had an identical setup. As with the Windows host, the disabling of automatic software updates along with non-essential services removed some possible causes of inconsistent run times. It was necessary to build and test systems with various configuration options. New clusters were formed during this investigation phase, by cloning the template node as many times as was needed (for example usually four worker nodes and a master node) and manually installing the cluster environment on them.

5.7.5 SCRIPTING

As the configuration of each of the systems was mastered, the creation of the virtual clusters was scripted using a software system called Vagrant. Following the development of a script, Vagrant allows the creation and deployment of virtual machines directly from a command line interface. The intention was to reduce the amount of effort needed in cluster maintenance between experiments and to make the experiments more reproducible. Using the scripting software ensured that the virtual machine was in the same state for each run of a feature detection process on a data file. However, following further research into benchmarking process methodology, it was clear that the system under test should not be tampered with between test runs. The only change allowed is the deletion of the data created from the previous test. This methodology meant that there was no need for scripted virtual machines. In fact, it was sufficient to have a template virtual machine with just a Linux operating system setup. This template virtual machine could be copied multiple times to create a new cluster that was then stored before being copied onto the test system for benchmarking. Further tests just required the clean cluster to be copied over, replacing the already tested virtual cluster.

Note that during this research a technology called “containers” and a software package called Docker have become increasingly popular [123]. Virtual machines are an abstraction of the physical hardware that allows a single machine to operate as many servers. Each virtual machine contains a complete copy of the operating system and any applications that are needed; these all need to be installed and maintained separately. A virtual machine is essentially a complete system and needs to be booted up at the start of a processing run, although it is possible to suspend the virtual machine to a disk file enabling a faster start up time. In contrast, containers are an abstraction at the application layer, which is one layer up from the operating system. Multiple containers can run on the same machine and share the same operating system; containers, therefore, take up much less space on disk than a virtual machine, start up almost instantly and do not need the same level of maintenance. Docker makes the whole task of creating and distributing complete environments simple and efficient. For this reason, it has begun to be used for the purposes of reproducible research [124] allowing researchers to distribute a complete environment for re-evaluation and further experimentation. Further reference to the use of containers and Docker to create an architected, automated, production system is made in the conclusion of this thesis.

5.8 PERFORMANCE TESTING ENVIRONMENT

Further environments were used to simulate the feature detection process running on full production hardware. These environments were used to produce the final timings reported in Chapter 6 and Chapter 7. Teradata Ltd. provided this hardware for the duration of the benchmarking phase of the research (note that the author of this thesis is employed by Teradata UK Ltd). The three systems consisted of:

- a three-node Hadoop cluster (running the Apache Hadoop distribution)
- a three-node Aster cluster
- a seven-node Aster cluster

Each cluster consists of two types of node, the accepted terminology for these types is “master” and “slave” nodes. The master node is the controller and distributes tasks and data to the workers, which are the nodes where the processing takes place. The terms “three-node” and “seven-node” refer to the number of slaves; each cluster also has a single master node. For performance testing, the Aster benchmarks were carried out on the three-node and seven-node Aster clusters. The benchmark tests on all other systems discussed below were completed on the three-node Hadoop cluster, with the relevant services being enabled as necessary.

The nodes in the performance testing systems were all identical, each containing dual eight-core Intel Xeon processors running at 2.6Ghz, with 512 Gb RAM for master nodes and 6-core Intel Xeon processors running at 2Ghz, with 256 Gb RAM and twelve 4Tb disk drives for the slave nodes.

5.9 SYSTEMS TESTED

5.9.1 TYPES OF SYSTEM

To address the main research question outlined in Chapter 1, a parallel feature detection algorithm, has been benchmarked using a cluster of connected compute nodes. A broad spectrum of technologies has been tested on the cluster, the technologies include a Distributed File System, a NoSQL database, a relational database, columnar storage, in-memory storage and stream processing. This is not an exhaustive list and does not, for example, include any form of hardware acceleration such as Graphics Processing Units (GPU) or Field Programmable Gate Arrays (FPGA). GPU-based acceleration is currently a major topic of research, partly due to the attention being given to Artificial Intelligence and Deep Learning [125]. However, the focus of this research remains on the implementation of the feature detection algorithm. Performance enhancement using hardware could be the subject of a different research project. Appendix A contains a detailed description of the systems benchmarked in this research. The following contains a brief overview of each system and explains its relevance to the research.

Hadoop

The base Hadoop distribution consists of a distributed file system called HDFS and the MapReduce processing engine. A large eco-system of open source technology has been built up around Hadoop including scheduling and resource allocation engines as well as other processing engines that replace MapReduce. The most recent allocation engine in Hadoop is called YARN, which has been adopted by many open source technologies as the default scheduling engine. Hadoop forms the basis of the parallel architecture used for most of the feature detection algorithm benchmarking in this research.

Cassandra

A class of database systems has emerged known collectively as “NoSQL” (an abbreviation of Not only SQL). These systems are used as alternatives to relational database systems and have been produced in response to the Big Data challenges outlined in Chapter 1, Section 1, such as scalability and the requirement to ingest data at high speed. As such NoSQL database capabilities could be directly relevant to the challenges presented by processing mass spectrometer data. Cassandra stores data in “column families”, which have columns and rows but in contrast to a relational database each row may have a different number of columns.

Aster

Aster is included in the research as an example of a relational database, it is based on the widely used open source Postgres database and uses ANSI standard SQL for querying data. What makes Aster particularly relevant to this research is the patented SQL-MapReduce engine, which allows MapReduce code written in Java to execute in parallel on the database nodes and access data directly from the database. The SQL-MapReduce engine allowed the Java code running on Hadoop to be compiled to run on the Aster platform with very few changes.

HBase

HBase is part of the Apache Hadoop ecosystem and is usually found ready configured on most Hadoop clusters as it is part of the base install. It runs on top of Hadoop’s HDFS file system and provides a column store in a non-relational format that is similar to the column families implemented by Cassandra. In contrast to HDFS, HBase allows fast, random access to data. Hadoop’s MapReduce framework can read and write directly from and to HBase tables.

Spark

Spark is a general-purpose parallel execution engine, initially developed at the University of Berkeley, that can run on a cluster using YARN to manage resources and scheduling. Spark operates in-memory with objects called Resilient Distributed Datasets (RDD). Spark has gained a large following in the business and academic world in recent years and has rapidly become a standard for parallel processing.

Flink

Flink is a relatively new software package, developed at the University of Berlin and is presented as an alternative execution engine to Spark. Superficially Spark and Flink are similar but when comparing them as Stream engines, used to capture and process data as it is produced, there are some pronounced differences that are explained in Chapter 7 where the results of the stream experiments are detailed.

Kafka

Unlike Flink and Spark, Kafka is not an execution engine. It does not process data and perform complex transformations on it. Instead, Kafka is a distributed publish-subscribe messaging system. Kafka is included here as it is used in the stream-processing experiments to simulate the behaviour of a mass spectrometer which instead of writing out its results to a file for processing, streams them out in real-time as they are produced.

5.9.2 COMBINATIONS OF SYSTEMS

The systems described above, or elements thereof, fall into three categories:

1. Data Storage
2. Processing Frameworks
3. Message Publishing

HDFS, HBase, Cloudera and Aster have data storage capabilities. In this category, data is loaded into the system's data store and persisted there. MapReduce, Spark, Flink and Aster's SQL-MR are data processing frameworks; here the code is executed to process the data in parallel reading from data stores. Kafka is in the third category of message publishing; it pushes data out for subscribers to consume in a streaming fashion. It is possible to combine the systems to evaluate how they work together and if there are benefits of doing so. Combinations explored are as follows:

- MapReduce with HBase
- MapReduce with Cassandra
- MapReduce with HDFS
- Aster as a standalone system
- Spark with HDFS
- Spark streaming with Kafka
- Flink with HDFS
- Flink streaming with Kafka

Figure 20 shows a complete cluster environment as used in this research to benchmark the performance of the algorithm. Each worker and the master node are running the services required for Hadoop, Flink, Spark, HBase and Cassandra. These systems and their components, for example “zookeeper” are described fully in Appendix A

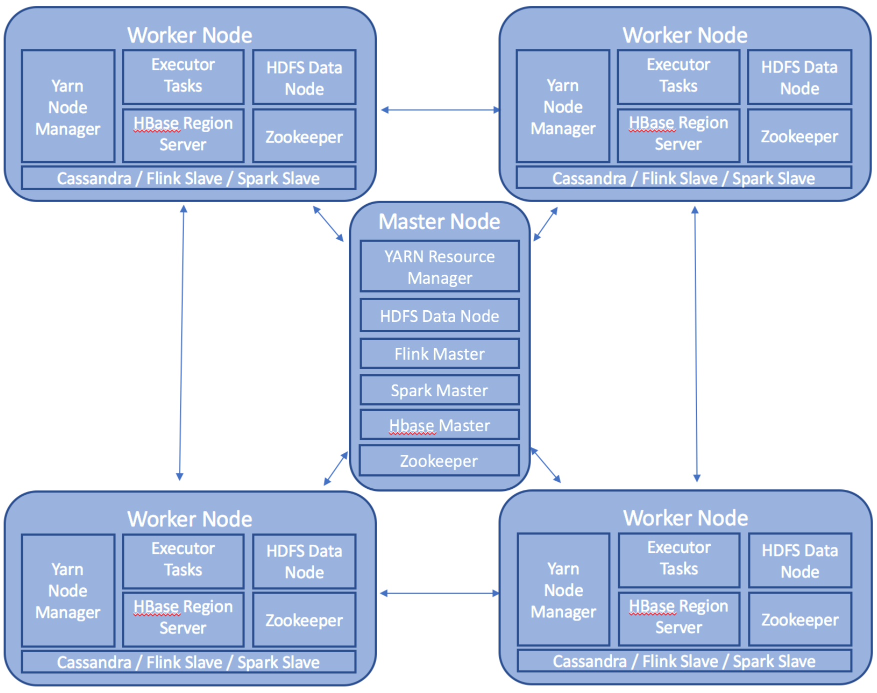


FIGURE 20 COMPLETE CLUSTER ENVIRONMENT

Note that performance testing with Aster was carried out on a separate cluster with the same physical configuration as the Hadoop-based systems

5.9.3 OTHER POSSIBLE CONFIGURATIONS

During this research, the Hadoop ecosystem has been constantly evolving with new software frameworks and distributions being developed. It has not been possible to test and benchmark the parallel algorithm with every new processing framework as it appears. However, consideration has been given to some of the major alternatives to those researched here. Notably Storm, Flume and Splunk as streaming frameworks, MongoDB, Redis and Redshift as database and NoSQL alternatives. However, the frameworks chosen for the complete test and benchmark research were chosen to be representative of the type of system required to design a complete architecture.

5.10 TEST DATA

Dr. Tony Ly from the Lamond Laboratory supplied the datasets used in validating the parallel algorithm. This data consisted of raw output files from mass spectrometers manufactured by Thermo Scientific. Specific models used were the LTQ Orbitrap XL, LTQ Orbitrap VELOS and Q Exactive. The data was created as result of proteomics experiments researching myeloid-specific gene expression, variations in protein abundance, isoform expression and phosphorylation at different cell cycle stages [131]. The complete set consists of eight files each related to different cell stages, with each file having two technical and three biological replicates, a total of forty-eight files. Note that not all the data files were processed during the benchmarking process as similar files (such as replicates) have a very similar processing time.

Standard practice for proteomics researchers is to run replicates of the experiments as a validation process. There are two types of replicate:

- A biological replicate involves rerunning the same cell sample but using a different mass spectrometer.
- A technical replicate involves running a different cell sample using the same mass spectrometer with the same parameters as the original sample.

This process of running replicates generates a significant amount of data that needs to be validated, adding to the complexity and length of time required for an experiment.

As mentioned in Section 5.2 of this chapter, the data had already been processed using the MaxQuant software. The results from MaxQuant were used to validate the output from the new parallel algorithm.

The files listed in Table 6 were used in the testing and benchmarking process.

TABLE 6 LIST OF FILES USED FOR TESTING AND BENCHMARKING

File Name	Size in Mb	number of scans
561L1AIL00.RAW	1,709	36245
561L1AIL01.RAW	1,743	36969
561L1AIL02.RAW	1,753	37180
561L1AIL03.RAW	1,750	37111
561L1AIL04.RAW	1,748	36329
561L1AIL05.RAW	1,732	36162
561L1AIL06.RAW	1,727	36181
561L1AIL07.RAW	1,718	36445
PT2441S1FP1A1.RAW	2,703	125550
371.RAW	1,404	37369
100312_EXP229_GFPIP_5.RAW	883	27381

As part of the process of benchmarking, each of the RAW files was converted into a mzML format using the msconvert software that forms part of the proteowizard tools. These files represent the types of output that is commonly processed in the Lamond Laboratory at the University of Dundee with the smallest RAW file creating a mzML of approximately 2.6Gb and the largest an mzML of approximately 8.4Gb. Following the conversion step, the files were loaded into the various cluster environments. Table 7 displays a list of file sizes in RAW, mzML, flat file and Avro formats and timings for conversions between them. The test set comprises files from four types of experiment, note that all the files with names beginning “561L1AIL” are from different stages of similar experiments. This set of test files represents a broad range of the type of experiments on the Human Proteome performed in the Lamond Laboratory. It is acknowledged that further research will be required to understand how the parallel system will perform on non-human experimental results.

TABLE 7 LIST OF FILE SIZES CONVERSION TIMINGS

File Name	RAW File Size Mb	mzML File Size Mb	Flat File Size Mb	Avro File Size Mb
561L1AIL00.RAW	1,709	5,128	4,962	3,388
561L1AIL01.RAW	1,743	4,704	4,541	3,101
561L1AIL02.RAW	1,753	4,614	4,454	3,042
561L1AIL03.RAW	1,750	4,821	4,658	3,180
561L1AIL04.RAW	1,748	5,490	5,318	3,632
561L1AIL05.RAW	1,732	5,426	5,255	3,589
561L1AIL06.RAW	1,727	5,409	5,237	3,577
561L1AIL07.RAW	1,718	5,393	5,221	2,565
PT2441S1FP1A1.RAW	2,703	8,376	7,766	5,305
371.RAW	1,404	3,873	3,743	2,028
100312_EXP229_GFPIP_5.RAW	883	2,604	2,499	1,707

File Name	Time to Convert RAW to mzML	Time to Convert mzML to Flat File	Time to Convert Flat File to Avro
561L1AIL00.RAW	00:09:52.49	00:02:28.22	00:00:57.00
561L1AIL01.RAW	00:10:31.73	00:02:34.70	00:00:52.57
561L1AIL02.RAW	00:10:15.38	00:02:43.87	00:00:52.19
561L1AIL03.RAW	00:10:13.27	00:02:27.66	00:00:53.99
561L1AIL04.RAW	00:10:09.75	00:02:28.05	00:01:02.35
561L1AIL05.RAW	00:10:19.17	00:02:29.81	00:01:01.05
561L1AIL06.RAW	00:08:43.51	00:02:23.83	00:01:00.41
561L1AIL07.RAW	00:08:43.38	00:02:21.89	00:00:59.65
PT2441S1FP1A1.RAW	00:09:38.00	00:03:13.04	00:01:42.76
371.RAW	00:07:32.11	00:01:33.09	00:01:01.78
100312_EXP229_GFPIP_5.RAW	00:03:23.00	00:01:03.15	00:00:27.84

This data shows that the Avro file format, which is easily distributed on a cluster for parallel processing, offers the potential to save disk space. In addition to the MaxQuant processing, some of the data had also been processed using a Teradata appliance hosting a relational database. The Teradata appliance processed the data using an algorithm created using the SQL query language. This SQL-based process had already undergone an extensive period of validation, although at the time of writing this thesis no publications have been produced from this work. The output from the Teradata system proved to be an excellent way of validating the results of the MapReduce algorithm as the relational database holds intermediate results for parts of the SQL process. Comparing the output of the MapReduce algorithm with these intermediate results allowed testing for accuracy and correctness at many points along the process. Inserting the results from the MapReduce system into the appropriate relational table and continuing the SQL process from that point meant that the final results could be checked accurately against the existing process. As the algorithm implemented by the Teradata system and the parallel algorithm implemented with MapReduce are both based on the algorithm used by MaxQuant, the results should all match to those obtained with MaxQuant.

5.11 CONCLUSION

The benchmarks carried out during this research were specifically for testing the performance of processing mass spectrometer data and not general-purpose processing. Therefore, to be relevant to mass spectrometer data it has been decided not to use synthetic data to benchmark the systems but rather to use specific workloads in the form of data files produced from actual proteomics experiments. This use of specific data sets means that the terasort test discussed in Chapter 2, Section 4 has not been utilised.

The benchmarks were initially run on the test environment using virtual clusters set up on a Windows 8 host machine. Once the process was tested and working as expected, the benchmarking was conducted on the performance testing environments described in Section 7 of this Chapter. The following items have been measured, with the results described in depth in Chapters 6 and 7:

Overall process timings

In keeping with standards from well-established benchmarks such as the TPC, timings were recorded by running the desired workload three times in succession and taking the slowest of the runs [42]. The only maintenance allowed between runs was the deletion of previous test data output. The time for each workload to run was measured using the Linux time command, which returns the elapsed, system CPU and User CPU time that the process takes to complete. Elapsed time is the metric used to report the results, which is also known as “wall time". Hadoop and the Aster system both include software called Ganglia; this is used to provide some metrics on the performance of the cluster while it is being used. During the benchmark phase of this research it was discovered that the elapsed time to complete the feature detection was all that was needed to compare the performance of the parallel algorithm, therefore results from Ganglia are not included. Appendix B presents details on the environment and scripts used to produce the results presented in this thesis so that they are reproducible.

6 BATCH PROCESSING

6.1 CHAPTER SUMMARY

This Chapter presents the results of experimentation using a parallel feature detection algorithm and a batch process with the test files listed in Table 6 (Chapter 5, Section 9).

Section 2 starts with an introduction to the problem from a technical point of view and reiterates the research questions. The data formats used for the experiments are described and timings given for loading data to the experimental systems. This section also includes a description of batch processing and its advantages and drawbacks.

Section 3 More detail is also presented on data and processing skew, performance tuning and any thresholds used. Finally, the various data models and schemas used are described.

Section 4 describes the experimentation and validation against other feature detection software is described. The effects of partitions in the reduce step are illustrated.

Section 5 presents the results of the experiments.

Section 6 contains a discussion on the findings and possible enhancements

6.2 INTRODUCTION

6.2.1 REAL-TIME

The meaning of the term real-time is unclear and needs qualification. True real-time processing is not realistic as there will always be a delay between data creation and data processing. Before experimentation takes place, the researcher must decide and specify the amount of delay that is acceptable for a process. This delay will be the constraint used to decide whether the system qualifies as real-time or not after benchmarking results have been obtained.

This chapter presents the results of running a parallel feature detection algorithm in a batch processing mode. Batch mode systems are not commonly considered to be capable of real-time processing; instead specialized streaming platforms such as those described in Chapter 7 are deployed. However, in the case of mass spectrometer data processing, it could be possible to obtain a sufficiently significant speed-up in processing time as to render a real-time system unnecessary. For instance, the speed-up is sufficient that further data processing and analysis is not delayed. This statement is based on the current time taken to process data which can range from hours to days for a full process. (Note that the complete process includes all stages of the data processing pipeline including the protein lookup stage that matches identified peptides to known sequences).

Currently laboratories process proteomics data on dedicated PC hardware. The data from the mass spectrometers is copied onto the processing PC, the processing is completed and then the data is copied to a shared network drive where it can be accessed by the researcher. If this process could be automated and the processing time reduced to several minutes by combining a parallel feature detection algorithm followed by a protein lookup system such as Hydra [79] on a cluster, then the speed-up and reduction in data management tasks would be beneficial to life scientists. The current time for feature detection using MaxQuant on a high specification PC (32Gb RAM, 8-core 3.6 Ghz SSD storage) is seventeen minutes for smallest file and more than fifty minutes for the largest of test files benchmarked, see Table 8 for the time to run only the feature detection process for each of the test files.

Setting a goal of reducing the elapsed time for feature detection by 95% results in a time of between fifty seconds to approximately two minutes forty seconds for the chosen test files. Conversations with proteomics researchers have confirmed that these timings would be acceptable because the processing time is reduced to the equivalent of the time taken to retrieve the results from the processing PC. Therefore, in the context of feature detection using batch processing, “real-time” is defined to mean completing the processing of the data in a short enough time so as not to hold up further operations on the dataset.

TABLE 8 MAXQUANT TIMINGS FOR FEATURE DETECTION

File Name	MaxQuant Feature Detection Time (HH:MM:SS)	95% Reduction in processing Time (HH:MM:SS)
561L1AIL00.RAW	00:31:05	00:01:31
561L1AIL01.RAW	00:32:23	00:01:37
561L1AIL02.RAW	00:29:56	00:01:30
561L1AIL03.RAW	00:31:05	00:01:33
561L1AIL04.RAW	00:32:32	00:01:38
561L1AIL05.RAW	00:32:33	00:01:38
561L1AIL06.RAW	00:32:02	00:01:36
561L1AIL07.RAW	00:31:30	00:01:35
PT2441S1FP1A1.RAW	00:53:24	00:02:40
371.RAW	00:19:54	00:01:00
100312_EXP229_GFPIP_5.RAW	00:17:05	00:00:51

6.2.2 DATA FORMATS

The widespread use of the mzML format for storing and transferring data and issues with using XML-based file for parallel processing necessitated the design of a new file format during the course of this research. The plain text SCMI format was used for the experimentation, it is created by parsing out the following fields from the RAW data file for each scan:

- Scan identifier
- msLevel
- Retention Time
- Precursor Ion mz
- Precursor Ion Intensity
- Precursor Ion Charge
- Mz array (a Base64 encoded float array)
- Intensity Array (a Base64 encoded float array)

The simplicity of a text-based file format for benchmarking and testing was preferred over a binary format such as Avro. In the text file, each record contains a complete set of data for a single mass spectrometer scan. The scan is the most granular level of data suitable for parallel processing and constitutes the "unit of parallelism". A complete scan contains features or peaks that represent the peptide molecules detected by the mass spectrometer. As explained in Section 6.3 of this chapter there are complications in the feature detection process that mean it is not possible to qualify a feature on its own as surrounding peaks need to be taken into consideration. These complications include overlapping peaks and isotopic envelopes. In an appraisal of parallel processing Gunther [132] describes how the efficiency of a parallel process degrades if an attempt is made to increase parallelism by splitting the data into chunks smaller than the optimum for a particular case. In the case of feature detection, this would happen if the data within scans was to be split across parallel tasks. Splitting a complete scan into chunks and distributing these around a cluster would necessitate decoding the base64 encoded mz and intensity arrays first. Following this, the data processing would need to include a new step where the features detected on each node are checked against each other to look for incomplete peaks that occurred where the data split occurred. (Section 6.3 of this chapter contains more details on this subject but in a different context of 3D feature detection). It would also be necessary to match all the detected peaks together along the mz dimension to detect Isotopic envelopes and ascertain the charge of the peptide ions. The example of splitting the data within scans does not increase the complexity of the algorithm to any great extent, it just introduces an extra step that requires data movement between nodes on the cluster, hence the decrease in efficiency and subsequent increase in processing time.

6.2.3 BATCH PROCESSING

The term “Batch Processing” in a computing context refers to a mode of executing programs in an automated and non-interactive way. Batch processing has a long history, the name originating from the days when computer programs were created using punched cards. To execute program code, an operator fed cards into the computer in batches. The absence of any interaction with the process as it is executing is a key component, any parameters, inputs, and outputs must be predefined before a batch process starts.

If a life sciences laboratory contains one or two mass spectrometers, then processing data in an interactive mode is still a possibility. Researchers can be responsible for monitoring their mass spectrometer experiment and moving the output RAW data file to a processing machine. The next steps could then be to commence a job using software such as MaxQuant, waiting for completion, then moving the result files to the researcher’s own personal computer for analysis. As laboratories become more complex with more mass spectrometers, the interactive mode of working becomes less desirable. Fenyo and Beavis [133] describe data management challenges in proteomics and categorize experiments into three sizes, small, medium and grande. Noting the need for automated systems to acquire and analyze data large-scale (grande) experiments. As an example of an experiment where automated batch-processing becomes a necessity, Fenyo and Beavis cite a large-scale experiment identifying protein content in the yeast Saccharomyces cerevisiae. This experiment analysed a total of 15,683 samples, identifying over 35,000 proteins. This would be extremely time-consuming and difficult to do using the manual method described at the start of this paragraph.

A common use of batch processing is to complete work in a defined “batch window”. A batch window is a period of downtime in computer processing cycles where the computers are being underused for interactive processing. An example is in an information management cycle where transactional systems are used throughout the day to record and process transactions. During the nightly window, batch processes extract data from the transactional systems, transform it and load the results into a data warehouse system for analysis the next day. This process is known as Extract, Transform, Load (ETL). Another example would be the monthly billing cycle for a credit card company, here data is stored during the active month and then a batch window, perhaps over a weekend, is used to process the monthly customer bills. The processing requirements for proteomics experiments loosely fit this pattern, and it could be possible to set up a nightly batch processing cycle to process that day's experimental output overnight and have results ready for the next morning. This method could certainly address some of the data management challenges such as data movement and availability of processing machines. However, the intention of this research is to provide results in as close to real-time as possible. This means that there is a requirement for a dedicated batch processing system.

Batch processing environments will include a scheduling system to ensure efficient running. Scheduling has been studied extensively both for computer-based batch processing and for manufacturing processes [134]. A major concern is balancing a system to ensure the full utilization of all resources, therefore obtaining maximum throughput. This balancing is of particular importance in non-heterogeneous environments, which may contain a mix of newer and older compute nodes with different processing speeds or where tasks have different processing requirements. For example, in a general batch processing system, some computing processes are many times smaller than others. Large resource-intensive jobs can monopolise the system, effectively blocking smaller jobs from running. A job queue holds the stream of jobs and frameworks such as Spring Batch exist for this purpose. For Hadoop-based systems, the original option was a simple first in first out (FIFO) scheduler which processed the jobs in the order in which they were submitted. Current versions of Hadoop include two scheduling methods:

- A fair scheduler which seeks to assign resources to jobs such that, on average over time, each request gets an equal share of the available resources.
- A capacity scheduler, which instead of using resource pools, as the fair scheduler does, uses a queue-based system. Here each queue is configured with a guaranteed quota of resources and jobs are assigned to queues based on submission parameters.

Note that other external frameworks exist such as Airflow and Oozie, but these have not been investigated in this research.

The proteomics processing pipeline consists of several steps with varying resource requirements. Differences in the complexity of the sample processed by a mass spectrometer and the length of processing time result in varying file sizes with varying processing times, this can be seen in Section 6.5 of this Chapter where the batch processing results are presented. On a large cluster dedicated to processing proteomics data, a capacity-based scheduler could be used to ensure that larger, more complex files could be consistently allocated more resources than smaller, simpler ones.

The increasing complexity of proteomics experiments and the increasing number of mass spectrometers has the potential to create a data management problem. The implication is that researchers need to spend a disproportionate amount of time on processing and handling data instead of experimenting and analyzing of results. A dedicated batch processing cluster automatically fed by the mass spectrometers could be used to completely disconnect researchers from the data management and processing tasks involved in turning output spectra into data that be analyzed. A capacity scheduler would ensure that large jobs do not affect smaller jobs while still ensuring that large jobs get the resources they need to complete as quickly as possible. As previously noted it is important to decide what is an acceptable time for process completion, firstly to be acceptable to researchers and secondly to conclude whether the terms real-time or near real-time are appropriate.

6.2.4 SYSTEM AND SOFTWARE

The hardware and software environments used for the experiments detailed in this chapter are described in detail in Chapter 5 and Appendix A. The performance testing environment has been used for all the experimentation except where noted. For example, performance tuning was mostly carried out using the development environment. As the development environment was far slower than the performance testing environment, the exaggerated differences between tests helped to identify slow parts of the process that required tuning.

6.3 IMPLEMENTATION DETAILS

6.3.1 PEAK DISTRIBUTION

Different experiments produce files with different peak distributions. The type of cell used in the experiment will determine which proteins are present for detection by the mass spectrometer. The distribution of the detectable features affects the way in which the data must be distributed between the map and reduce phases of the code execution, known as the shuffle phase. As mentioned in the previous section, the MapReduce framework allows the use of a custom partitioner function to direct the mapper output to the right reducer. In the current code, the rules for the custom partitioner have been tuned manually for each of the test files to produce a distribution of an equal number of 2D features in each partition. It is recognised that this is not ideal and not an acceptable solution for a production system. The correct way to tackle this task is to use an approach similar to that of Hadoop’s total order sort function (further explained in Chapter 6, Section 6.6.1), which runs the map task then takes a sample of the output to create an efficient shuffle strategy on the fly. However, the manual tuning method is manageable for the number of files used in the benchmark experiments and is a constant factor across all the systems tested. Taking this into account and the fact that this thesis is about research into whether the feature detection algorithm can be run in parallel and near real-time, the combination of a working algorithm suitable for benchmarking and theoretical investigation into the effects of parallelism have provided answers without the need for a one hundred percent production-ready system.

The figures on the following pages (Figure 21, Figure 22, Figure 23, Figure 24) were created using MaxQuant and illustrate the different distribution of peaks in the test files. Files 371.RAW and PT2441S1FP1A1.RAW show very different peak distributions. 371.RAW is a manufactured sample with a small number of known molecules used to test the setup and accuracy of mass spectrometers.

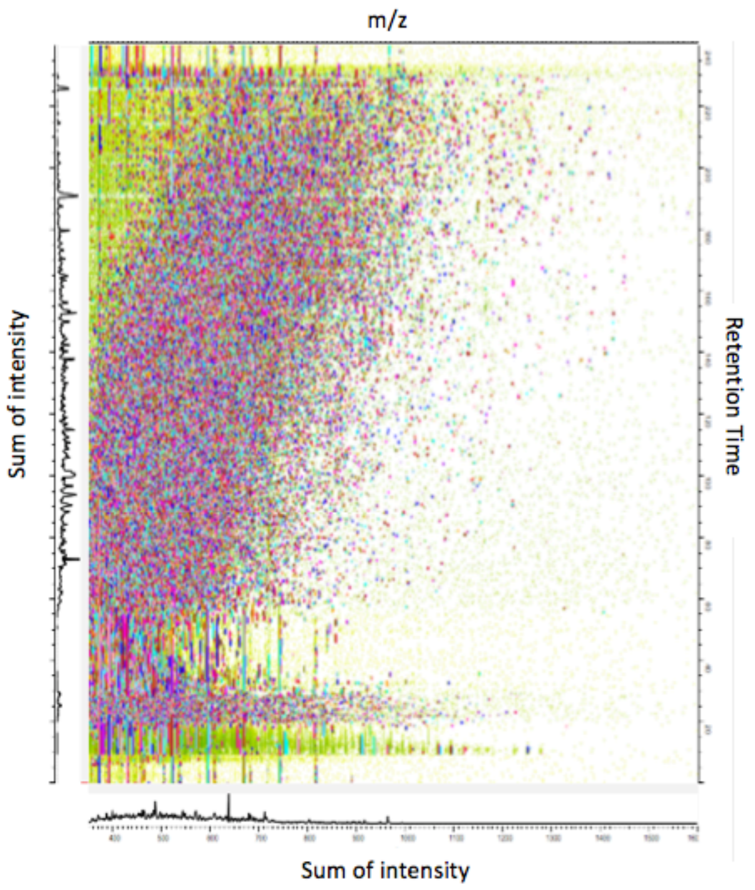


FIGURE 21 PEAK DISTRIBUTION IN FILE PT2441S1FP1A1.RAW

Files 561L1AIL001.RAW and 561L1AIL07.RAW show similar distributions as they are derived from the same cell, however differences do exist as they are taken from the cell at different stages in the cell lifecycle.

Note that in these figures showing the distribution of the data, the colours indicate the presence of the isotopic envelopes of 3D peaks that we detected as features in the data. At this level of zoom, that is the entire data set, the colours are not useful apart from to distinguish the overall pattern of feature distribution. If the images were to be zoomed in to show only a few isotopic envelopes then the colours could be used to distinguish which peaks belong to which envelopes. The darker colours represent high intensity isotopic envelopes of peaks with the light yellow coloured pixels being background noise in the data.

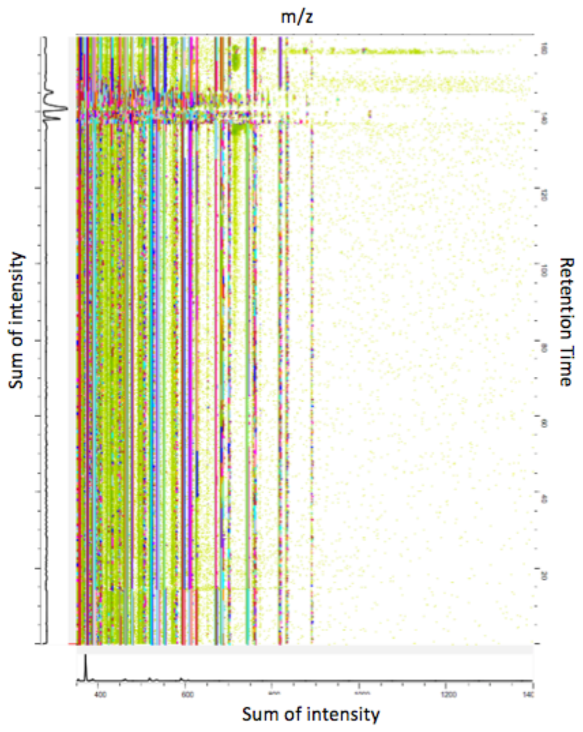


FIGURE 22 PEAK DISTRIBUTION IN FILE371.RAW

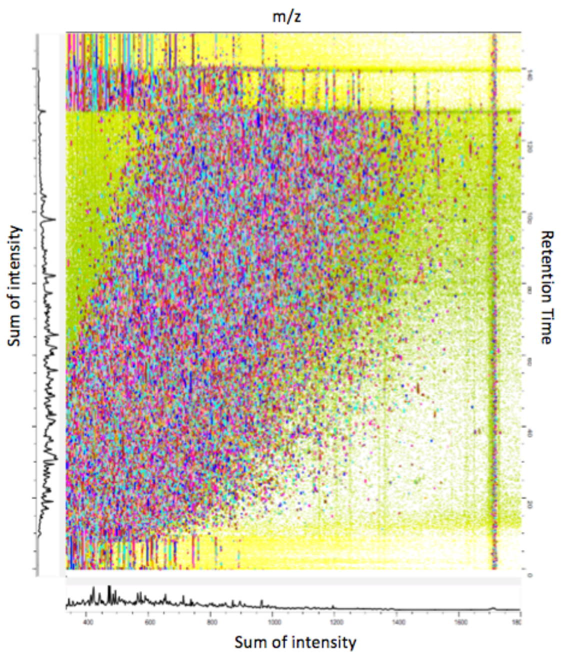


FIGURE 23 PEAK DISTRIBUTION IN FILE 561L1AIL01.RAW

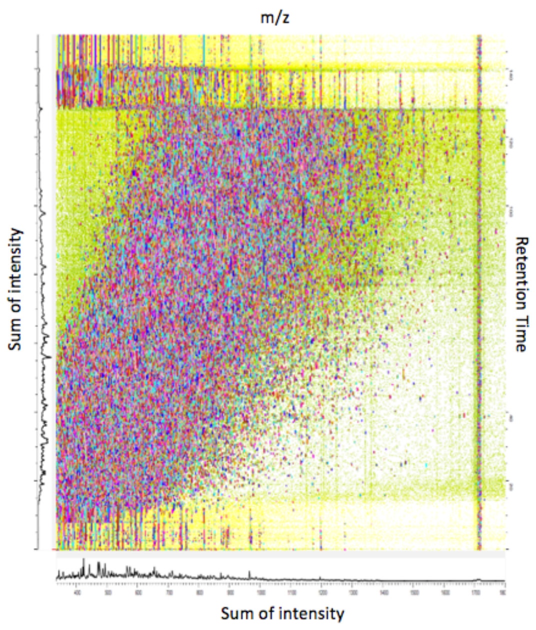


FIGURE 24 PEAK DISTRIBUTION IN FILE 561L1AIL07.RAW

The distribution of the features in the data files create a problem with the skew on the cluster which differs from the usual data skew. Data skew is a common occurrence when attempting to process data on a massively parallel system; the solution is to ensure that each of the nodes in the system contains approximately the same quantity of data. This approach is used with a relational database where tasks such as aggregation will take the same time on different nodes if the data is distributed evenly.

Evenly distributing the data is not sufficient to balance the workload across the cluster with the mass spectrometer files. The figures (Figure 21, Figure 22, Figure 23, Figure 24) above show that individual mass spectrometer scans do not contain the same number of peaks. It is usual that scans at the start of the process with a low retention time contain fewer peaks than scans that occur in the middle of the process, with the number of peaks tailing off towards the highest retention times. As the algorithm can skip sections of data with no discernible features, there is an issue of processing skew. In other words, the volume of data is less important than the amount of information that the data contains. A scan with a low retention time contains more noise than features and is processed relatively quickly compared to a scan containing more features. In this case, the most efficient data shuffling can be achieved in the same way as described except that instead of sampling the output for mass ranges, the output should be sampled for the total number of features contained.

6.3.2 COMPARISON WITH MULTI-THREADED SOFTWARE

MaxQuant is capable of multi-threaded processing and provides a parameter to control the number of threads used. To monitor the effect of altering the number of threads in use, just the feature detection part of the MaxQuant process was run on a test file repeatedly. On each execution, the number of threads was incremented from one to the maximum of eight. The maximum number of threads is eight because the PC used for tested had eight cores. As an additional test, the amount of RAM available on the test machine was changed while keeping the number of threads constant; this was done to investigate the effect of memory available. The results in Table 9 show that the feature detection part of the MaxQuant process does not speed up with the number of threads allocated. This implies that the feature detection process is not running in a parallel fashion and is not taking advantage of the number of threads available to it. The tests also show that the process does not speed up when adding more RAM and therefore that the process is not memory-bound. The fact that the processing was being distributed across the cores was verified by viewing Windows Task Manager which displays the percent usage of each core graphically.

TABLE 9 MAXQUANT MULTI-THREADED BEHAVIOUR

File Name	Number of Threads	Time to Complete
561L1AIL01	1	00:32:13
561L1AIL01	2	00:32:02
561L1AIL01	3	00:32:13
561L1AIL01	4	00:32:06
561L1AIL01	5	00:32:13
561L1AIL01	6	00:32:08
561L1AIL01	7	00:32:06
561L1AIL01	8	00:31:55

The same tests were then carried out using the open source software called Dinosaur [121]. Written in the Scala programming language, Dinosaur implements a similar feature detection algorithm as MaxQuant. The Dinosaur source code is freely available in a GitHub repository. The test results in Table 10 show that the Dinosaur feature detection process does speed up with an increase in the number of threads allocated to the process. Therefore, a degree of parallelism is being employed but the results in Table 10 show this parallelism does not produce a linear decrease in completion time as parallelism increases.

TABLE 10 DINOSAUR MULTI-THREADED BEHAVIOUR

File Name	Number of Threads	Time to Complete
561L1AIL01	1	00:30:37
561L1AIL01	2	00:29:32
561L1AIL01	3	00:27:49
561L1AIL01	4	00:26:59
561L1AIL01	5	00:26:47
561L1AIL01	6	00:26:18
561L1AIL01	7	00:26:16
561L1AIL01	8	00:26:09

The documentation for Dinosaur [121] mentions this and describes a method of windowing being used to achieve the parallel processing. Despite this feature, Dinosaur is not engineered to run on a cluster. Instead, it is intended to be run as a PC-based process in a similar fashion to MaxQuant.

6.3.3 PERFORMANCE TUNING

While writing and testing the parallel feature detection algorithm in Java, several rounds of performance testing and enhancement were carried out. This performance tuning was done to identify bottlenecks in the process and produce as efficient an algorithm as possible given the technology employed. A development environment was used as described in Chapter 5, Section 5. All of the performance testing was carried out using a basic Hadoop system with the data residing in HDFS. The performance testing involved using samples taken from the test files and timing each part of the process before making changes to the code and retesting the process to check for faster processing times. It was evident from the start that the 2D feature detection was very fast and needed little improvement. As a single scan can be processed in a very small amount of time, the metric used to test the process was the number of scans per second per map task. Table 11 shows the results for the test files and the 2D process.

TABLE 11 TIMINGS FOR 2D FEATURE DETECTION

File	Elapsed Time (secs)	Number of Map tasks	Minimum Map Time	Maximum Map Time	Average Map Time	Number of Scans	Number of Scans per Mapper	Number of Scans per Second per Map
100312_EXP229	23	19	9	16	10	27381	1441	144.11
371	12	29	6	10	9	37369	1289	143.18
561L1AIL00	20	30	9	29	12	36245	1208	100.68
561L1AIL01	20	30	9	29	12	36969	1232	102.69
561L1AIL02	21	30	9	28	12	37180	1239	103.28
561L1AIL03	21	30	8	29	12	37111	1237	103.09
561L1AIL04	20	30	9	29	12	36329	1211	100.91
561L1AIL05	20	30	8	27	12	36162	1205	100.45
561L1AIL06	20	30	9	29	12	36181	1206	100.50
561L1AIL07	20	30	9	29	12	36445	1215	101.24
PT2441S1FP1A1	25	40	7	16	15	125550	3139	209.25

YARN is responsible for providing the resources for each job based on the workload and allocates the number of map tasks to each job as it sees fit. This table shows that the difference between the minimum and maximum map time is small in terms of elapsed time but that in some cases the maximum time to complete a map task is three times that of the minimum time. This difference indicates that there is skew on the cluster as some of the map tasks take longer to process than others.

As the process is embarrassingly parallel at the scan level, the theoretical minimum time to complete the 2D feature detection is the time needed to complete scan requiring the most processing plus the fixed overhead of the time needed to instantiate the map tasks. Using the performance testing cluster as described in Chapter 5, Section 7, the time taken to instantiate a map task is in the order of several seconds, which equates to between ten percent and 30 percent of the overall map task timings. Note that other systems benchmarked in this research could further reduce this time as they do not have the same overheads as the MapReduce framework. Also note that to achieve this theoretical minimum time, a cluster with the same number of map task slots as there are scans in the file is required. Given that the largest file benchmarked here as 120,000 scans this would be a very large system indeed.

3D feature detection presented more of a challenge than the 2D process. Several of the steps were initially very time consuming: these were the linking of the 2D peaks into chains occurring in a mz window over time and the final 3D Isotopic window calculation. The linking of the 2D peaks into chains was initially an O(N^2) process due to a nested loop used in the Java code to create the links (N=the number of 2D peaks). This was replaced by using a Java arraylist and a method of scoping down the area in which a match was possible using the biological properties of the molecules. The rules employed to make this matching process more efficient are as follows:

- Flag points in the outer loop as they are checked, no need to recheck.
- Maintain a list of all matched points and skip them in future iterations.
- If the next point is in the same scan as the outer loop, fast forward to the start of the next scan.
- Confine search for 3D peak to a window of thirty seconds [76].

The next slowest process was the final Isotopic envelope matching, this involved using a correlation coefficient to check that two curves were a similar shape before allowing a correct match to be declared. Here, as with the 3D chaining process, switching from Java arrays to ordered arraylists, along with implementing some targeted rules provided an increase in speed. Overall the performance tuning produced an increase of approximately 6.5 times over the initial implementation. Table 12 lists some examples of the timings before and after the tuning.

TABLE 12 RESULTS OF PERFORMANCE TUNING THE JAVA CODE

Task	Elapsed Time (HH:MM:SS)			
	Round 1	Round 2	Round 3	Round 4
Create chain of 3D peaks	00:41:29	00:24:41	00:07:17	00:07:21
Peak Smoothing	00:00:19	00:00:19	00:00:17	00:00:19
3D feature detection	00:02:02	00:02:12	00:02:04	00:02:06
Isotopic Envelopes	00:10:50	00:05:33	00:04:20	00:01:02
Complete Process	00:54:40	00:32:45	00:13:58	00:10:48

During the performance tuning, the 2D peak count and 3D peak count were checked to ensure that the accuracy of the algorithm remained constant. The performance tests were carried out on the test environment using a subset of 5000 scans from the test files. In Table 12, the columns Round 1 to Round 4 show the difference in timings of the individual parts of the process as various changes were made, some examples of the changes are as follows:

- Round 1
  - The original implementation.
- Round 2
  - Used Java arraylists of objects creating chains of peaks.
- Round 3
  - Reduced the amount of looping needed to find the next 2D peak in a 3D chain, as described above.
- Round 4
  - Used a similar method as in Round 3 to reduce the amount of looping needed in the Isotopic envelope detection

6.3.4 CODE QUALITY

It is acknowledged that further improvements to the efficiency of the algorithm could be made by a Java performance expert.

To measure the general quality of the code written during this research, software called SonarQube was used. SonarQube produces a score called “Technical Debt” [136]. Technical debt is measured in the number of hours required to bring the code up to the correct standard. It involves for instance removing unused variables, identifying and removing code that is not executed, incorrect use of loops, incorrect use of datatypes and adherence to style guidelines such as those published by the Oracle Corporation, “The CERT Oracle Secure Coding Standard for Java” [137]. An initial analysis of the approximately six-thousand lines of Java code written during this research revealed a technical debt of four days, which could be reduced to zero by a Java expert aided by the SonarQube report in Appendix C

6.3.5 SCHEMAS

For benchmarking, files stored in the HDFS file system were formatted as plain columnar text format. This format was suitable for MapReduce, Spark and Flink when reading and writing the data to HDFS itself. For the other data storage systems tested, namely Aster, Hbase and Cassandra, it was necessary to design a schema into which to load the data. The design of these schemas varied by system but in all cases the result is a simple replication of the text file design with no need for multiple tables. Appendix D contains a description and sample code of the schemas that were implemented.

6.4 EXPERIMENTATION

6.4.1 OVERVIEW OF THE EXPERIMENTATION

The experimentation carried out during this research was split into several distinct parts:

- Testing the feature detection algorithm for accuracy compared to the output from the Max Quant software.
- Testing the feature detection algorithm execution on the various environments already described.
- Testing the feature detection algorithm for compatibility between the execution engines and the different storage layers.
- Performance testing using the different frameworks and storage layers already described.

6.4.2 WHAT WAS TESTED

The most important metric collected during performance testing was simply the total execution time. Intermediate measures were also collected including total map time, total reduce time, map tasks per second, average reduce time, max and min reduce times, Total shuffle time and the number of bytes processed.

During the tests to validate the algorithm output the mass, intensity and charge of the 2D and 3D peaks were collected for comparison with MaxQuant. The full data set resulting from all of the testing activities can be downloaded from the github site associated with this thesis https://github.com/chillman99

6.4.3 HOW RESULTS WERE COLLECTED

The principal method of capturing the performance data during the benchmarking was the Linux time command. When placed in front of any other command it will return the total execution time when the command finishes. For the benchmarks carried out in this research the total time returned was recorded. This is also referred to as the wall-clock time or total run time. The Aster system timings were gathered from the Aster Management Console (AMC), which is a web based tool for administering the Aster system. The time tests were run with the verbose option on. Amongst other data, this option reports “context switches” which gives an indication of whether the process was interrupted during execution.

For the Hadoop based system, the Hadoop web console was used to find the total time for all Map tasks, the total time for all reduce tasks along with the average timings of the individual map and reduce tasks. The various processing platforms benchmarked in this research also provide web based consoles, these are shown in Appendix E.

6.4.4 VALIDATION

The results from the batch run benchmarks were validated against output obtained from MaxQuant. The list of test files were processed using MaxQuant on a high specification “gaming” personal computer consisting of an eight-core CPU, six solid-state drives and thirty-two GB of 133Ghz memory. The timings for these tests are reported in Table 8 (Section 6.2.1). The output from these tests included a list of detected peptide masses, charge states and intensities, and it is this data that was used to validate the output of the parallel algorithm.

To make a comparison with the results from the parallel algorithm, the outputs from MaxQuant and the parallel algorithm were loaded into a relational database. The relational table structure made it a simple task to compare the masses, charge states and intensities detected using the SQL language. An accuracy score compared to MaxQuant was derived using the precision and recall metrics explained in Chapter 5, Section 10.

6.4.5 DISCUSSION ON PARTITION TESTING

Partitions were briefly introduced in section 6.3.2 of this chapter, this section includes a more detailed description of partitioning and its effect on performance and accuracy and also how the need to partition the data for distribution around the cluster limits the speedup possible from parallelism.

In the map step of the algorithm, a scan is the smallest unit of parallelism that allows shared nothing processing. In other words, to produce the 2d peaks each scan can be processed completely independently of any other scan. If a smaller unit of parallelism was chosen (that is intra-scan processing), then some communication between tasks would be needed to find complete peaks and isotopic envelopes. Figure 25 shows a scan, keeping the scan as the unit of parallelism means that a single map task will read in all the data in the scan and detect any 2D peaks and isotopic envelopes contained in it.

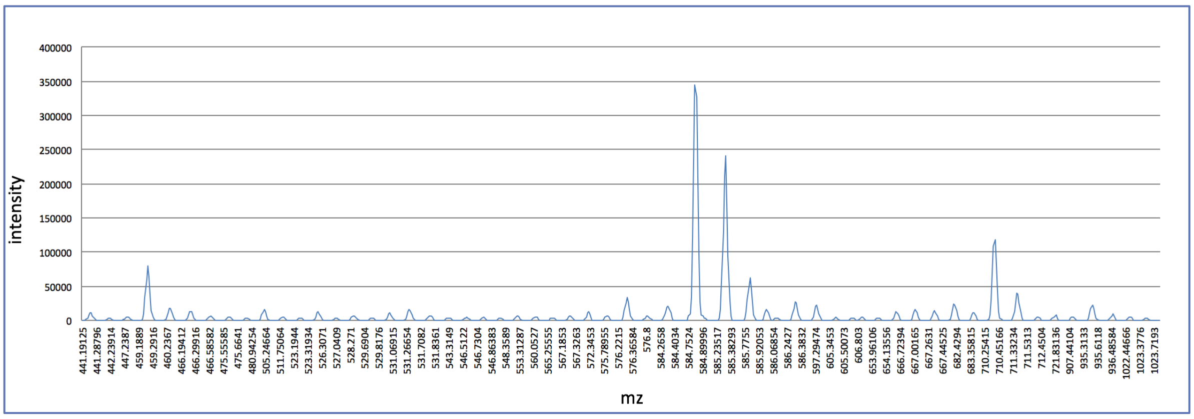


FIGURE 25 SCAN WITHOUT PARTITIONS

Figure 26 shows the same scan but here the dotted lines represent partitions of data. Partition A would be processed by a different map task than Partition B; if an isotopic envelope of 2D peaks started in Partition A and continued in Partition B the map task would need to communicate results between them to detect this.

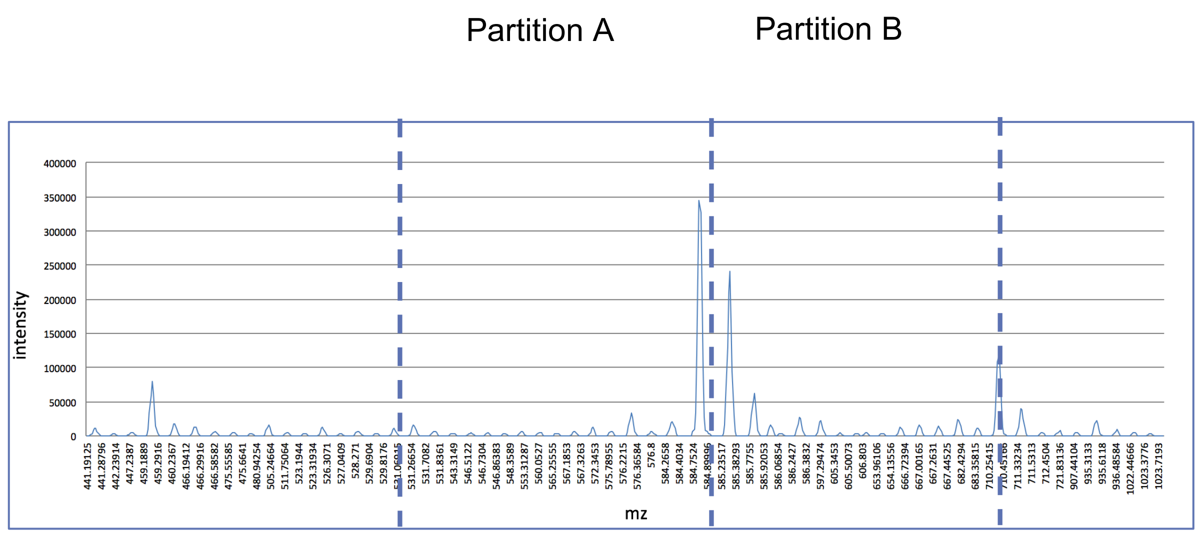


FIGURE 26 SCAN SPLIT INTO PARTITIONS FOR INTRA-SCAN PROCESSING

As there are between 30,000 and 125,000 scans in the test files used in this research, processing at the scan level allows a high degree of parallelism. The results displayed in Table 11 in Section 6.3.7 show that individual scans can be processed very rapidly, this coupled with a parallel cluster architecture allows the 2D peak maps to be completed in between nineteen and forty seconds for the benchmarked files.

The individual 2D peaks need to be arranged by retention time and iterated over to find 3D peaks and isotopic envelopes. This arrangement of 2D peaks ordered by retention time forms a continuous chain of peaks with no natural breaks. To process this peak chain in parallel, it must be broken into smaller pieces each of which can be processed by a separate reduce task. In this way running the 3D processing in parallel is a similar problem to that of processing the 2D peaks in an intra-scan fashion.

The construction of partitions creates a natural limitation on the amount of parallelism that can achieved and therefore limits the increase in speed that is possible.

6.4.6 EFFECT ON TIME TO PROCESS OF PARTITIONS AND REDUCERS

The size of the partitions and the number of reducers chosen for the 3D peak processing step directly affects the time taken to complete the process. Smaller partitions allow more reducers, which implies a higher degree of parallelism and therefore a faster processing time. However, the situation is more complicated and is not linearly scalable. The first complication occurs because of the size of the cluster and the number of reduce slots available. The number of nodes, the number of CPUs per node and the configuration of the cluster in the YARN framework all affect how many process slots are available for running reduce tasks. As an example, the three-node cluster used for performance testing during this research has forty-two slots available, having forty-two partitions would allow one partition to run in each slot. Moving to eighty-four partitions means that forty-two reducers run in parallel and another forty-two reducers are queued up waiting for slots to be available. This allows the cluster to process partitions which take less time to complete serially as slots free up due to processes completing, which potentially leads to a faster overall processing time.

A further complication occurs because the parallel algorithm contains a section of code with a nested loop giving a potential O(N^2) processing time where N equals the number of 2D peaks. The code has been enhanced with performance improvements to mitigate this but it still not a linear process. This means that smaller partitions are beneficial to reducing the time to process the data. The next section details the limitations on partition size brought about by the accuracy required.

6.4.7 SIZE OF PARTITIONS AND THE EFFECT ON ACCURACY OF RESULTS

As noted in the previous section, smaller partitions produce the best speed-up. This is partly due to the way that the parallel algorithm searches for 3D peaks in the reduce step. However, there are limitations to how small a partition can be due to the nature of isotopic envelopes, which is explained in Section 6.4.9. The way in which peaks and isotopic envelopes are constructed can be explained by researching the way that molecules are ionized and detected in a mass spectrometer. A 3D peak is made up of previously identified 2D peaks. The algorithm chains 2D peaks together into a 3D peak based on several rules:

- Increase in Retention Time
- One or two scans between 2D peaks
- A tolerance between the weighted mass of 2D peaks

Due to the different charge states of 2D peaks, the tolerance in weighted mass changes. That is the higher the charge state the higher the tolerance that is allowed when matching 2D peaks across time. As we do not know the charge states when setting the partition size, we need to account for the highest charge state to be detected. In this research this is set to +6 to align with the threshold used in the MaxQuant software.

To further complicate matters, we need to account for the presence of Isotopic envelopes amongst the 3D peaks for a further discussion on isotopic envelopes see Section 6.3.2 The partition must be of sufficient size for a complete isotopic envelope of 3D peaks to be included or the algorithm will not correctly detect them.

6.4.8 BOUNDARY PEAKS AND THE AFFECT ON PROCESSING

As the data is split into partitions it is inevitable that some peaks and isotopic envelopes will fall at the boundary between partitions. This leads to incorrect results as the isotopic envelopes will be incomplete.

To overcome the issue of peaks occurring at the boundary of partitions and the potential for incorrect identification of peaks and isotopic envelopes, an overlap of data between partitions was introduced. This means that there is some duplication of data processing but ensures that all the peaks are correctly identified. The algorithm was adapted to discard chains of 3D peaks where the intensity did not fall to zero before reaching the boundary as this indicated an incomplete peak and a post-processing de-duplication step introduced in case peaks were completely detected in adjacent partitions.

6.4.9 SIZE OF OVERLAPS

As with calculating the size of the initial partitions, the required size of the overlap between partitions can be calculated. The overlap of data needs to be sufficient to contain a complete isotopic envelope of 3D peaks. This overlap will ensure that all features are detected correctly. A solution to this is for the partition and the partition overlap at the boundary to be the same size. This is achieved by finding the mid-point of each partition and add the data either side of this midpoint to the preceding and following partitions, illustrated in Figure 27

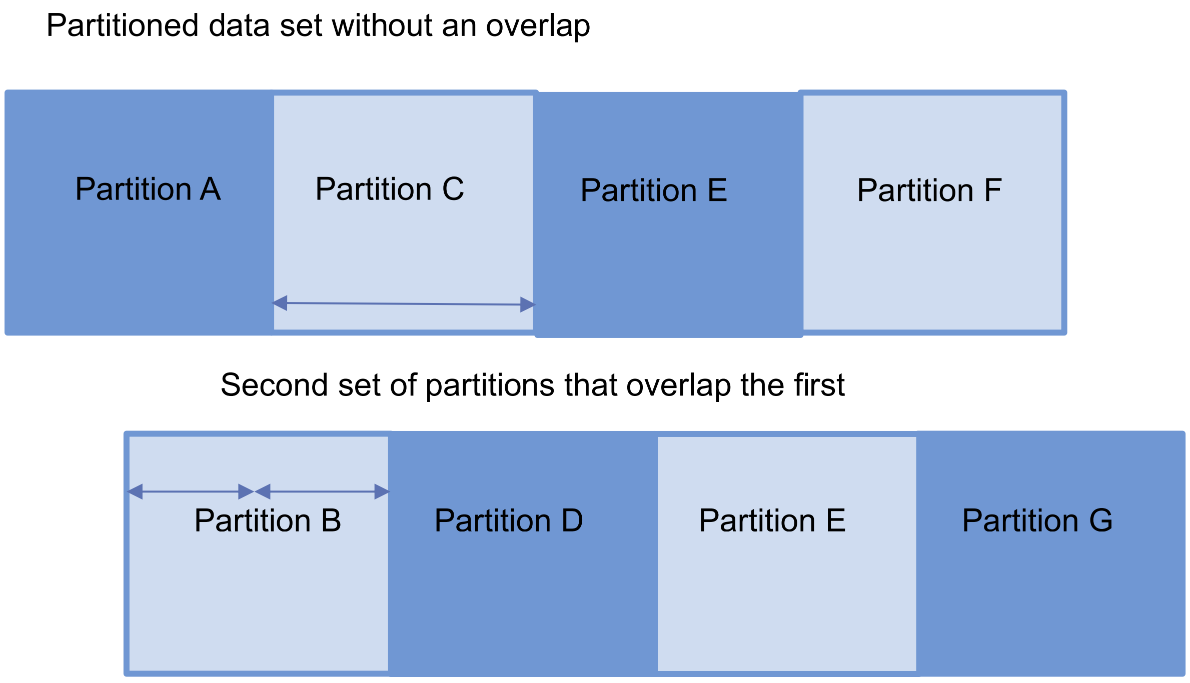


FIGURE 27 ILLUSTRATION OF OVERLAPPED PARTITIONS

The width of the partitions has been calculated by considering three factors:

1. The biological constraints, which impose a lower bound on the size of the partition as measured in thomsons. This can be calculated from the width of an isotopic envelope of 2D peaks. The entire envelope must fit into half of the full partition width for the overlap partition strategy to work correctly. See Figure 28 below for more detail.
2. The requirement to have small partitions to increase the speed of processing
3. The setup and available resources on the cluster

Creating the overlaps in the way described means that all the 2D peak data is processed twice in the reduce step. Although this is an overhead, it does ensure that the 3D peaks are correctly detected and allows the use of small partition sizes to make the most of the parallelism of the cluster and efficient use of the parallel algorithm. Figure 28 shows five isotopic envelopes of 2D peaks arranged in six overlapping processing partitions, in this case each partition equates to a single reduce task on the cluster. It can be seen that the isotoptic envelopes will be processed twice due to the overlapping partition strategy. A partition that is smaller than twice the width of a 2D isotopic envelope will result in envelopes being partially detected or not detected at all, as they will cross a partition boundary and not be picked up in the overlapping partitions.

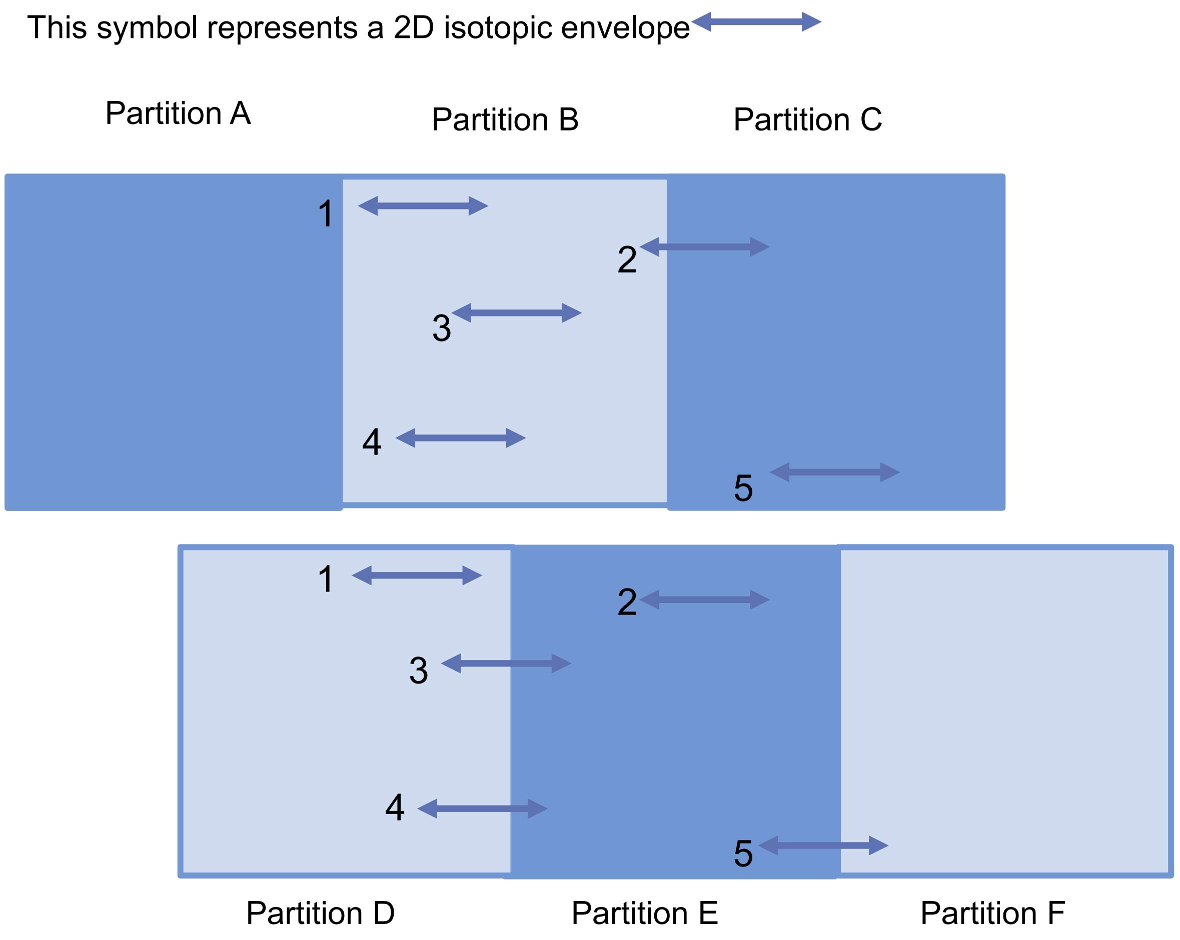


FIGURE 28 FIVE 2D ISOTOPIC ENVELOPES ARRANGED IN PROCESSING PARTITIONS

In the figure the two processing streams are arranged one above the other. The figure also shows that if a complete Istopic envelope fits within half of the partition window it will always be processed completely in at least one of the partitions in which it appears. The following explains this for each of the five isotopic envelopes displayed.

1) Envelope one fits entirely within the Partition A and Partition B and will be completely processed twice.
2) Envelope two occurs at the boundary of Partition B and Partition C and will be discarded by both. It fits entirely in Partition E and will be processed there.
3) Envelope three fits entirely in Partition B and will be processed there. It also occurs at the boundary of Partition D and Partition E and will be discarded by both.
4) Envelope four also fits entirely in Partition B and will be processed there. It also occurs at the boundary of Partition D and Partition E and will be discarded by both.
5) Envelope five fits entirely into Partition C and will be process there, it occurs at the boundary of Partition E and Partition F and will be discarded by both.

The theoretical maximum width of an 2D Istopic Envelope is calculated by taking the distance between subsequent peaks in the envelope and the number of peaks. The mass shift values in Table 4 show that for an ion with charge +2 the shift in mass between each peak in an Isotopic envelope is 0.50143432. At the point where the partition widths are calculated, the ion charge is unknown so the highest mass shift value is taken to ensure no features are missed.

6.5 RESULTS

6.5.1 TABLES OF RESULTS

The following tables present the results of the benchmarks. Firstly, the results are compared to MaxQuant, the overall timings are displayed plus the minimum and maximum times taken by the reduce tasks. The minimum and maximum times for the map tasks are not shown as they all completed in seconds and, due to the embarrassingly parallel nature of the 2D feature detection, do not constrain the overall completion time. For this reason, the majority of the discussion and analysis focuses on the 3D feature detection performed by the reduce tasks.

Table 13, Table 14 and Table 15 show the results from the benchmarks where the processing is carried out by Hadoop MapReduce. The results show that the batch process runs the fastest when data is stored in HDFS, followed by Cassandra and then HBase being the slowest. This is because the batch process simply reads all the data in and writes all the data out in a linear fashion. There are no queries of the data that would necessitate random access. This means that the storage mechanisms and indexing provided by HBase and Cassandra are an unrequired overhead compared to the simple file storage of HDFS. This is further discussed in Chapter 8.

TABLE 13 BENCHMARK HADOOP / HDFS

File	MaxQuant (HH:MM:SS)	Hadoop/HDFS (HH:MM:SS)	Minimum Reduce Time (HH:MM:SS)	Maximum Reduce Time (HH:MM:SS)
371	00:19:54	00:05:13	00:00:47	00:04:39
100312	00:17:05	00:04:58	00:00:32	00:04:21
561L1AIL00.RAW	00:31:05	00:11:40	00:02:10	00:09:57
561L1AIL01.RAW	00:32:23	00:12:18	00:02:02	00:10:18
561L1AIL02.RAW	00:29:56	00:12:16	00:01:59	00:09:34
561L1AIL03.RAW	00:31:05	00:11:11	00:02:12	00:09:25
561L1AIL04.RAW	00:32:32	00:12:02	00:02:23	00:10:12
561L1AIL05.RAW	00:32:33	00:12:03	00:02:43	00:10:19
561L1AIL06.RAW	00:32:02	00:12:30	00:02:26	00:09:53
561L1AIL07.RAW	00:31:30	00:11:30	00:02:11	00:10:20
PT2441S1FP1A1.RAW	00:53:24	00:15:49	00:02:22	00:15:02

The minimum and maximum reduce columns show the slowest and the fastest processing time for the reduce tasks. The difference between these times occurring with all files and systems indicates the presence of processing skew on the cluster, which is discussed in detail later in this chapter.

TABLE 14 BENCHMARK HADOOP / HBASE

File	MaxQuant (HH:MM:SS)	Hadoop/HBase (HH:MM:SS)	Minimum Reduce Time (HH:MM:SS)	Maximum Reduce Time (HH:MM:SS)
371	00:19:54	00:06:25	00:00:48	00:05:51
100312	00:17:05	00:06:08	00:00:34	00:05:31
561L1AIL00.RAW	00:31:05	00:13:03	00:02:19	00:12:20
561L1AIL01.RAW	00:32:23	00:13:35	00:02:13	00:12:50
561L1AIL02.RAW	00:29:56	00:13:20	00:02:23	00:12:35
561L1AIL03.RAW	00:31:05	00:12:46	00:02:25	00:12:00
561L1AIL04.RAW	00:32:32	00:13:58	00:02:40	00:13:14
561L1AIL05.RAW	00:32:33	00:13:52	00:03:12	00:13:09
561L1AIL06.RAW	00:32:02	00:13:39	00:02:27	00:12:52
561L1AIL07.RAW	00:31:30	00:13:58	00:02:14	00:13:12
PT2441S1FP1A1.RAW	00:53:24	00:19:20	00:02:28	00:18:33

TABLE 15 BENCHMARK HADOOP / CASSANDRA

File	MaxQuant (HH:MM:SS)	Hadoop/Cassandra (HH:MM:SS)	Minimum Reduce Time (HH:MM:SS)	Maximum Reduce Time (HH:MM:SS)
371	00:19:54	00:06:01	00:00:44	00:05:26
100312	00:17:05	00:05:44	00:00:31	00:05:06
561L1AIL00.RAW	00:31:05	00:11:54	00:02:04	00:11:08
561L1AIL01.RAW	00:32:23	00:12:23	00:02:03	00:11:34
561L1AIL02.RAW	00:29:56	00:12:00	00:02:03	00:11:06
561L1AIL03.RAW	00:31:05	00:11:35	00:02:12	00:10:44
561L1AIL04.RAW	00:32:32	00:12:37	00:02:27	00:11:47
561L1AIL05.RAW	00:32:33	00:12:34	00:02:50	00:11:43
561L1AIL06.RAW	00:32:02	00:12:19	00:02:25	00:11:31
561L1AIL07.RAW	00:31:30	00:12:39	00:02:06	00:11:52
PT2441S1FP1A1.RAW	00:53:24	00:17:23	00:02:12	00:16:35

Table 16 and Table 17 show the results from the Aster benchmarks on a three-node and a seven-node cluster respectively.

TABLE 16 BENCHMARK ASTER 3 NODE CLUSTER

File	MaxQuant (HH:MM:SS)	Aster 3 node (HH:MM:SS)	Minimum Reduce Time (HH:MM:SS)	Maximum Reduce Time (HH:MM:SS)
371	00:19:54	00:11:01	00:00:30	00:03:45
100312	00:17:05	00:10:50	00:00:27	00:03:39
561L1AIL00.RAW	00:31:05	00:26:40	00:01:45	00:09:24
561L1AIL01.RAW	00:32:23	00:24:38	00:01:36	00:08:49
561L1AIL02.RAW	00:29:56	00:20:26	00:01:30	00:07:24
561L1AIL03.RAW	00:31:05	00:21:35	00:01:35	00:07:18
561L1AIL04.RAW	00:32:32	00:25:36	00:02:04	00:09:02
561L1AIL05.RAW	00:32:33	00:23:16	00:02:17	00:08:09
561L1AIL06.RAW	00:32:02	00:21:14	00:01:42	00:07:20
561L1AIL07.RAW	00:31:30	00:22:46	00:01:40	00:07:55
PT2441S1FP1A1.RAW	00:53:24	00:33:50	00:01:39	00:12:07

TABLE 17 BENCHMARK ASTER 7 NODE CLUSTER

File	MaxQuant (HH:MM:SS)	Aster 7 Node (HH:MM:SS)	Minimum Reduce Time (HH:MM:SS)	Maximum Reduce Time (HH:MM:SS)
371	00:19:54	00:04:07	00:00:33	00:03:43
100312	00:17:05	00:04:02	00:00:25	00:03:35
561L1AIL00.RAW	00:31:05	00:10:00	00:01:46	00:09:21
561L1AIL01.RAW	00:32:23	00:09:20	00:01:38	00:08:43
561L1AIL02.RAW	00:29:56	00:07:57	00:01:31	00:07:21
561L1AIL03.RAW	00:31:05	00:07:57	00:01:37	00:07:22
561L1AIL04.RAW	00:32:32	00:09:46	00:02:03	00:09:07
561L1AIL05.RAW	00:32:33	00:08:49	00:02:14	00:08:14
561L1AIL06.RAW	00:32:02	00:07:59	00:01:41	00:07:29
561L1AIL07.RAW	00:31:30	00:08:31	00:01:38	00:08:00
PT2441S1FP1A1.RAW	00:53:24	00:12:39	00:01:46	00:12:03

The reason for the extra benchmark on the seven-node cluster is to illustrate a point regarding parallelism. The results from the three-node cluster show that Aster takes around twice the time to process the same file as the Hadoop/HDFS configuration. Whereas the seven-node cluster produces similar results, although slightly faster. This is because the Aster system has a fixed set of six virtual workers per node, which gives an overall eighteen-way parallelism on the cluster. In comparison, the three-node Hadoop cluster governed by YARN is configurable at runtime up to forty-eight processing slots. The Aster seven-node provides forty-two virtual workers and therefore a similar degree of parallelism as the Hadoop three-node cluster. The results show that given a similar degree of parallelism Aster performs faster on the larger files than the Hadoop-based systems.

Table 18 and Table 19 display the results from running the benchmarks on the in-memory execution engines, Flink and Spark, with the data stored in HDFS. Flink outperforms Spark for all the test files by a small amount. Both Flink and Spark outperform Hadoop MapReduce.

TABLE 18 BENCHMARK FLINK

File	MaxQuant (HH:MM:SS)	Flink (HH:MM:SS)	Minimum Reduce Time (HH:MM:SS)	Maximum Reduce Time (HH:MM:SS)
371	00:19:54	00:03:33	00:00:26	00:03:13
100312	00:17:05	00:02:03	00:00:19	00:01:46
561L1AIL00.RAW	00:31:05	00:10:40	00:02:02	00:10:55
561L1AIL01.RAW	00:32:23	00:11:03	00:02:02	00:11:30
561L1AIL02.RAW	00:29:56	00:10:19	00:02:06	00:11:21
561L1AIL03.RAW	00:31:05	00:10:11	00:02:07	00:10:22
561L1AIL04.RAW	00:32:32	00:10:56	00:02:20	00:11:15
561L1AIL05.RAW	00:32:33	00:11:02	00:02:43	00:11:14
561L1AIL06.RAW	00:32:02	00:10:40	00:02:27	00:11:41
561L1AIL07.RAW	00:31:30	00:11:06	00:01:55	00:10:47
PT2441S1FP1A1.RAW	00:53:24	00:14:03	00:02:02	00:13:41

TABLE 19 BENCHMARK SPARK

File	MaxQuant (HH:MM:SS)	Spark (HH:MM:SS)	Minimum Reduce Time (HH:MM:SS)	Maximum Reduce Time (HH:MM:SS)
371	00:19:54	00:03:51	00:00:34	00:03:31
100312	00:17:05	00:02:33	00:00:29	00:02:06
561L1AIL00.RAW	00:31:05	00:11:53	00:02:13	00:11:26
561L1AIL01.RAW	00:32:23	00:12:36	00:02:13	00:11:53
561L1AIL02.RAW	00:29:56	00:12:28	00:02:18	00:11:49
561L1AIL03.RAW	00:31:05	00:11:43	00:02:18	00:10:58
561L1AIL04.RAW	00:32:32	00:12:21	00:02:32	00:11:47
561L1AIL05.RAW	00:32:33	00:12:28	00:02:57	00:11:54
561L1AIL06.RAW	00:32:02	00:12:47	00:02:40	00:12:02
561L1AIL07.RAW	00:31:30	00:11:48	00:02:08	00:11:04
PT2441S1FP1A1.RAW	00:53:24	00:14:22	00:02:17	00:13:58

Table 20 presents the benchmarks side by side with the fastest times highlighted.

TABLE 20 BENCHMARK COMPARISON

File	Hadoop HDFS (HH:MM:SS)	Hadoop Hbase (HH:MM:SS)	Hadoop Cassandra (HH:MM:SS)	Aster 3 Node (HH:MM:SS)	Aster 7 Node (HH:MM:SS)	Flink (HH:MM:SS)	Spark (HH:MM:SS)
371	00:05:13	00:06:25	00:06:01	00:11:01	00:04:07	00:03:33	00:03:51
100312	00:04:58	00:06:08	00:05:44	00:10:50	00:04:02	00:02:03	00:02:33
561L1AIL00.RAW	00:11:40	00:13:03	00:11:54	00:26:40	00:10:00	00:10:40	00:11:53
561L1AIL01.RAW	00:12:18	00:13:35	00:12:23	00:24:38	00:09:20	00:11:03	00:12:36
561L1AIL02.RAW	00:12:16	00:13:20	00:12:00	00:20:26	00:07:57	00:10:19	00:12:28
561L1AIL03.RAW	00:11:11	00:12:46	00:11:35	00:21:35	00:07:57	00:10:11	00:11:43
561L1AIL04.RAW	00:12:02	00:13:58	00:12:37	00:25:36	00:09:46	00:10:56	00:12:21
561L1AIL05.RAW	00:12:03	00:13:52	00:12:34	00:23:16	00:08:49	00:11:02	00:12:28
561L1AIL06.RAW	00:12:30	00:13:39	00:12:19	00:21:14	00:07:59	00:10:40	00:12:47
561L1AIL07.RAW	00:11:30	00:13:58	00:12:39	00:22:46	00:08:31	00:11:06	00:11:48
PT2441S1FP1A1.RAW	00:15:49	00:19:20	00:17:23	00:33:50	00:12:39	00:14:03	00:14:22

Note that if the seven-node Aster system is excluded then Flink is the fastest for all the files. This table shows that the parallel algorithm does indeed provide a speed-up over the single-threaded algorithm. Although the execution times are faster than those obtained using MaxQuant, the timings do not show the 95% improvement that was outlined in the methodology chapter. The minimum and maximum timings for the reduce tasks show that there is still some skew in the processing that could be optimized and also that the reduce task timings are the constraint that prevent the process approaching a speed that could be classed as real-time.

The results in Table 20 show that, apart from the previously described issue with number of parallel slots available with Aster, the timings for all the runs with different processing frameworks and data storage layers are similar. The processing is carried using the algorithm coded in Java and the same set of libraries are used for all the systems. The differences in run-time are due to the differences in data access between file and non-file based systems, the way that the difference frameworks instantiate and control the parallel tasks and also the way they handle input and output data. Table 21 shows the fastest of the benchmark runs compared with the slowest non-file-based systems, which was Hadoop HBase and then compared to the slowest of the file-based systems, which was Hadoop HDFS.

TABLE 21 DIFFERENCES IN TIMINGS BETWEEN FLINK AND HBASE, FLINK AND HDFS

Flink	Hadoop HBase	Difference between Flink and Hbase	Ratio Flink time to HBase time	Hadoop HDFS	Difference between Flink and HDFS	Ratio Flink time to HDFS time
00:03:33	00:06:25	00:02:52	80.75	00:05:13	00:01:40	31.95
00:02:03	00:06:08	00:04:05	199.19	00:04:58	00:02:55	58.72
00:10:40	00:13:03	00:01:23	12.97	00:11:40	00:01:00	8.57
00:11:03	00:13:35	00:01:17	11.61	00:12:18	00:01:15	10.16
00:10:19	00:13:20	00:01:04	10.34	00:12:16	00:01:57	15.90
00:10:11	00:12:46	00:01:35	15.55	00:11:11	00:01:00	8.94
00:10:56	00:13:58	00:01:56	17.68	00:12:02	00:01:06	9.14
00:11:02	00:13:52	00:01:49	16.47	00:12:03	00:01:01	8.44
00:10:40	00:13:39	00:01:09	10.78	00:12:30	00:01:50	14.67
00:11:06	00:13:58	00:02:28	22.22	00:11:30	00:00:24	3.48
00:14:03	00:19:20	00:05:17	37.60	00:15:49	00:01:46	11.17
		Average Ratio	39.56		Average Ratio	16.47

The results in Table 21 show that for the smaller files where processing is already fast, the overhead caused by using a non-file-based data repository is large. The ratio of this overhead to the overall processing time falls as the processing time increases for different types of file. The same is true for using Hadoop with HDFS as a file-based data repository, the overhead is less than for HBase but still substantial for small files and decreasing for larger files. The reason the overhead decreases is that the map and reduce tasks have a fixed instantiation time of several seconds which does not change as data volumes increase. Also, HBase introduces a higher cost for accessing data row by row when compared to the simple file-based data repositories. Note this data access cost only holds true where sequential access is required. If the requirement changes to include random access to data then HBase, Cassandra and Aster would show improved performance over HDFS. The difference in times between Flink and Spark vs Hadoop HDFS show that Flink and Spark are faster when instantiating tasks compared to Hadoop and have less processing overheads.

The maximum reduce time as reported in Tables 11 to 17 is the constraining factor for all the systems tested here. The difference between the overall completion time and the maximum reduce time is under one minute (except for the three-node Aster system). This time difference is made up of several components including initialisation of the job, execution of map tasks, job completion and housekeeping tasks. Without further optimization of the processing skew and the algorithm code it would not be possible to reduce the execution time below that of the maximum reduce time regardless of the number of processing slots on the cluster. As discussed in section 6.4 it is not possible to parallelize the batch process beyond a certain point due the nature of the features that are being detected. This factor has led to researching processing the data in a streaming fashion as described in Chapter 7 and is discussed further in the final chapter.

6.5.2 VALIDATION

The results from the experiments were loaded into a relational database table and compared to the results from MaxQuant using the SQL language. Note that the output from all the systems benchmarked was identical in all respects, that is the number of features detected, mass, intensity and charge values all matched exactly. This was to be expected as the core algorithm code was used without any changes on each system. For this reason, the validation table only shows data for MaxQuant versus the parallel algorithm. The data for the parallel algorithm was obtained from the benchmarks using Aster, this was convenient because the output from Aster is a relational database table. To perform the validation, it was only necessary to load the MaxQuant data into a similar relational table. The two sets of results could then be compared with simple SQL queries.

Accuracy of results, judged by precision and recall metrics

To measure the accuracy of the parallel algorithm against the MaxQuant output, the precision and recall metrics were used. Recall is a measure of how many of the total number of true features were detected. Precision is a measure of how many of the detected features were true features.

Table 20 Shows a "confusion matrix" with fabricated data as an example to explain the metrics. A confusion matrix is a common device used to evaluate the performance of a binary classifier. There are four possible outcomes:

Predicted = true, Actual = true, result = True Positive
Predicted = true, Actual = false, result = False Positive
Predicted = false, Actual = false, result = True Negative
Predicted = false, Actual = true, result = False Negative

TABLE 22 EXAMPLE OF A CONFUSION MATRIX

	Predicted True	Predicted False
Actual True	100	10
Actual False	20	200

For the purposes of this thesis, the True Positives are the features that have been correctly identified by MaxQuant and also by the parallel algorithm being tested. The False Positives are features detected by the parallel algorithm that were not detected by MaxQuant. The False Negative area are features detected by MaxQuant that the parallel algorithm did not find. True Negatives would be features that MaxQuant detected that were not in fact actual features but as we are not testing the accuracy of MaxQuant here, this section will be null. All counts of features are at the complete file level. The definitions of Precision and Recall are given below.

Recall = (True Positive / (True Positive + False Negative))
Precision = (True Positive / (True Positive + False Positive))

Results of validation

Table 21 shows the results of the validation, the total number of features detected by MaxQuant and the parallel algorithm appear in the first two columns. True positives equal the number of features matching between MaxQuant and the parallel algorithm. False negatives equal the number of features found by MaxQuant but not by the parallel algorithm. False positives are the features found by the parallel algorithm and not by MaxQuant.

TABLE 23 VALIDATION OF PARALLEL ALGORITHM OUTPUT

File	MaxQuant	Parallel Algorithm	True Positive	False Negative	False Positive	Precision	Recall
371	17777	40887	16621	1156	23110	0.41	0.93
561L1AIL00.RAW	159201	330727	145191	14010	171526	0.44	0.91
561L1AIL01.RAW	159295	366379	148144	11151	207084	0.40	0.93
561L1AIL02.RAW	161481	339110	143718	17763	177629	0.42	0.89
561L1AIL03.RAW	158820	336698	146114	12706	177878	0.43	0.92
561L1AIL04.RAW	159201	342282	147738	11463	183081	0.43	0.93
561L1AIL05.RAW	158411	351672	148272	10139	193261	0.42	0.94
561L1AIL06.RAW	158054	319269	147306	10748	161215	0.46	0.93
561L1AIL07.RAW	154335	339537	143222	11113	185202	0.42	0.93
PT2441S1FP1A1.RAW	123016	270086	116250	6766	147070	0.43	0.94

In all cases the recall shows that the parallel algorithm is picking over 90% of the features found by MaxQuant. The MaxQuant supplement paper that describes the algorithm forming the basis of the parallel algorithm tested here does not provide sufficient detail to replicate the exact implementation of the algorithm as used by MaxQuant. The results, however are within the limits accepted by other feature detection software as defined by Chawade et. al [74]. In their review of proteomics data processing packages Chawade et. al also noted that each software solution has advantages and disadvantages, for example in their ability to detect features, distinguish between charge states and effectively filter out noise in the signal. As such the parallel algorithm is an approximation of the algorithm implemented by MaxQuant and certain complications will not be handled in the same way. An example of this is peaks with ambiguous charge states as described by Xiao et. al [154]. This situation occurs where isotopic envelopes overlap leaving a state where there could be, for example two peptides with a charge of +2 or a single peptide with a charge of +4.

The precision shows that the parallel algorithm is finding more features than MaxQuant; these are classed as false positives. In a complete system many of the false positives would be filtered out during the next step of the process [138]. This further filtering occurs because the detected features are compared to a known list of peptide molecules in the protein identification stage, those not matching are discarded at this stage but this has not been implemented as part of this research.

6.6 DISCUSSION

6.6.1 PARTITIONING

As stated in Section 6.4.6, the maximum time taken to process the reduce tasks is the major constraint on the overall processing time. One way to reduce this maximum time is a better partitioning strategy to minimize the processing skew on the cluster. Currently, the partitions are pre-calculated and hard-coded for each of the data files, the size and number of partitions have been calculated by sampling the data output from the map step and using a function to produce the minimum and maximum mass of the peaks in the partition. This process could be automated and carried out just after the map step or as a separate MapReduce process using a sample of scans from the input file. An automated process would also allow greater flexibility in adapting the number of reduce steps to suit the cluster setup. The sampling and calculating of max and min weights would be carried out in the same way as an algorithm called "Total Order Sort" which is used by Hadoop to sort data in a parallel fashion on a cluster. Hadoop uses the Total Order Sort algorithm to produce a system-wide sorted list as opposed to a standard sort that returns a list sorted at the node level. As the Hadoop is a shared-nothing system a system-wide sorting operation is complex and involves sampling the data on the nodes, reviewing the distribution of data by the required sort key and re-distributing the data by the sort key to provide approximately equal partitions of data on the nodes, which when sorted can be joined together to provide a correctly sorted dataset.

A further enhancement could be to calculate the partitions based on the level of processing required as opposed to the current method of basing the size on the number of peaks within them. The distribution of the 3D peaks within the sample is very uneven and unique to the sample that is being processed. Examples of these feature distributions can be seen in the figures in Section 6.3.4. When the algorithm detects a chain of 2D peaks that form a 3D peak it needs to iterate over the adjacent 2D peaks to check for others that may also be part of the 3D peak. However, if no chain is detected within two scans then no further action is needed, and the algorithm moves on without iteration. Therefore, it is not just the total number of 2D peaks in a partition that is significant but also the processing required to detect the 3D peaks that they constitute. This means that sampling the output of the reduce step to discover where the 3D peaks are forming can produce a more efficient use of cluster resources.

6.6.2 CHOOSING THE CORRECT NUMBER OF REDUCERS

As the number of reduce steps needs to be predetermined in the Hadoop MapReduce framework, several calculations need to be taken into consideration. As shown in Section 6.4.9 it is possible to calculate the minimum width of a partition plus the overlap required using the theoretical maximum width of a 2D isotopic envelope. This calculation is required to ensure that boundary peaks are fully accounted for. In an ideal situation, each reduce step would require the same amount of time to complete, and the number of reducers would equal some multiplication of the number of processing slots on the cluster. For example, if the cluster has three worker nodes, each with sixteen processing slots, then forty-eight slots are available, and the number of reduce tasks would equal forty-eight or ninety-six or one hundred and forty-four etc. If each reduce task completed in the same time then the cluster would be fully utilised during the entire process. However, the results show there is processing skew, which can be seen by the spread between minimum and maximum time to completion. The YARN framework does mitigate this to some extent by allocating tasks to the next available processing slot, meaning that some processing slots can process several faster tasks serially in the time taken to process the slower task.

More work is required to even out the processing done by each reduce task but for this research, the number of reduce tasks was fixed at eighty-four which provided two tasks per slot on the three node clusters and one task per slot on the seven node cluster.

6.6.3 ISO PEAK THRESHOLDS VS TIME TO COMPLETE AND ACCURACY

A simple way to reduce the time taken to process the data is to reduce the amount of data that needs to be processed. The parallel algorithm takes a large volume of input data points and in various stages reduces them to a much smaller data set.

During the process, various thresholds are used that affect the amount of data that is processed. The recall and precision metrics introduced in Chapter 5, Section 5.10 are a way of assessing how changing these thresholds affects the accuracy of the final output.

One threshold is the size of the ISO peaks used when creating 3D peaks. For each scan the maximum peak intensity is calculated, and peaks under a threshold percentage of this peak intensity are discarded. It was discovered during experimentation that this particular threshold value had a significant impact on the number of peaks detected and therefore the total processing time and also the accuracy of the algorithm.

More work is necessary to understand the exact value that this threshold should be set but for this research, it was set to 66% which equated to the value where the accuracy of the algorithm was not significantly affected. Values below this produced much faster run times but with a corresponding significant drop in accuracy compared to the MaxQuant output.

6.6.4 ENHANCEMENTS

The benchmark results show that the parallel algorithm does indeed speed up the process of feature detection as compared to a single-threaded pc-based process. It also proves to be accurate in relation to the results obtained by MaxQuant. Following on the process could be enhanced to produce a more complete protein identification pipeline. As referenced in previous chapters, other research has produced code such as that used in the Hydra [79] project, which comprises the protein lookup or database search. It would be a relatively simple matter to connect the output of detected features (peptides) to the input of the Hydra project resulting in a complete solution. A more thorough investigation into the slower parts of the process could be made namely 3D peak identification and 3D isotopic envelopes and performance enhancements made to the code.

7 STREAM PROCESSING

7.1 CHAPTER SUMMARY

This Chapter presents the experimentation and the results from running the parallel feature detection process in a streaming fashion. The stream process simulates data being made available while the mass spectrometer is in operation.

Section 1 describes the meaning of a stream processing approach, how it works in practice and contrasts this approach with processing data in a batch-mode.

Section 2 details the implementation of the parallel feature detection algorithm and describes the changes that were made to enable the batch algorithm described in Chapter 6 to operate on streaming data. This section also describes the differences in the processing approaches between Apache Spark and Flink.

Section 3 describes the benchmark experiments carried out and the method for obtaining and validating the results.

Section 4 details the results from the benchmarks.

Section 5 Describes the results obtained and indicates potential future research.

7.2 INTRODUCTION

7.2.1 STREAM PROCESSING

The previous chapter defines batch processing as processing a fixed amount of data (the batch) in a certain amount of time (the batch window). In contrast to the batch model where data is first stored and then subsequently processed, stream processing takes place as the data is created and pushed out from the producer to be subsequently captured by a consumer. The consumer will perform any necessary processing either on single data points as they arrive or, more usually, on windows of data. Different metrics are available to determine the size of the processing windows in a streaming system. These metrics include the number of incoming data points or the elapsed time between windows. The windows can be distinct, where each data point belongs to a unique window and is processed only once or sliding, where data points belong to multiple-windows that overlap meaning that the data is processed multiple times.

The stream of data is conceptually unbounded, the data flows continually from the source, and the consumer processes it as it arrives with no specific start or end point. In this way, streaming data is processed as it is generated and computation is done in memory before it is saved to disk. In the case of proteomics, the start and end of the mass spectrometer run are the actual boundaries of the stream, which is the same as with the batch process. However, when processing in a streaming fashion, feature detection is carried out while the mass spectrometer is still running, which can typically last between two and four hours.

7.2.2 STREAM PROCESSING PLATFORMS

Many open source and proprietary streaming solutions exist. To provide a comparison with the batch implementation, detailed in the previous chapter, this research concentrates on Apache Spark and Flink. Kafka is also used in the experimentation; Kafka’s role is to simulate the data streaming directly from the mass spectrometer. Kafka reads in a data file and streams it out as a topic (described in

Chapter 5 section 5.8.8). The data in the topic is then consumed by the Spark or Flink engines. Spark and Flink also have the capability of reading data directly from files on disk to simulate streaming, but the addition of Kafka provides a more realistic simulation. In a real-life situation, a framework such as Kafka would be used to ensure a robust environment that protects against any data loss in the event of system failure.

Other streaming solutions of note include infosphere streams from IBM and Tibco Streambase both of which are proprietary commercial systems. Open source distributions which allow stream processing to various degrees include Rabbit MQ, Storm, Samza, Esper, and Beam, although none of these has been evaluated during this research.

7.2.3 DATA MANAGEMENT

One potential benefit of stream processing is in the area of data management. As mentioned previously, a standard workflow for proteomics experiments involves moving data around between different environments and manual processing steps. If data can be streamed directly from a mass spectrometer into a central cluster, then some of these manual steps are not required. In an ideal system, the data would flow from the mass spectrometer into the processing cluster via a streaming framework such as Kafka; here it would be processed in parallel and the results stored in a central location where they can be accessed by the life scientists. This would all happen in close to real-time with the scientists being able to access the results of their experiments almost immediately after they finish and without having to move or copy any data.

The streaming architecture also allows partial results to be viewed before the mass spectrometer has completed its work. The in-stream results could be used for validating an experiment before it finishes, which could result in an experiment being halted early if issues were found. Stopping an experiment before it completes will free up the mass spectrometer for further work and save wasting time on processing results that will only be discarded.

7.2.4 STREAM SPECIFIC ISSUES

Logging, back pressure, and resilience need to be considered when using stream processing.

Streaming systems implement various forms of logging to ensure that all the data is made available to the target systems. The data will be tagged so that, for instance, the system will know whether data has arrived in a different order from which it was created or if it hasn't been processed yet.

Back pressure describes the balance between the velocity that data arrives compared to the speed at which it can be processed. Ideally, all the data will be processed as soon as it arrives at the consumer or as soon as the relevant size process window is available. If the consumer cannot process the incoming data fast enough, then the incoming data will queue up in memory and eventually spill to disk causing delays. In a typical streaming use case such as real-time fraud detection, this can mean that the data is not processed within the specified service level agreement (SLA). For proteomics data, back pressure and delays in the process would not have such a detrimental effect as say a missed fraudulent banking transaction but it will still affect the overall runtime of the processing and is therefore not desirable. One way of mitigating back pressure is providing more compute nodes on a parallel cluster. Another way is using the logging mechanism in a stream framework mentioned above. All data is logged as it arrives in the stream, meaning that it is not lost if it cannot be processed at the time of arrival but will be made available to the data consumer as and when it is able to process it.

A streaming framework should provide resilience in the event of failure of either the system handling the streams or the target system that is processing the data. Kafka provides resilience using its logging system and its cluster-based architecture.

7.2.5 STREAMING DATA IN PRACTICE

A previous section mentions the fact that Kafka is being used to simulate the mass spectrometer streaming out data in real-time. For the mass spectrometer to stream the data in practice requires collaboration with the mass spectrometer manufacturers. In the case of the University of Dundee, the manufacturer Thermo Scientific would need to provide a mechanism by which the data could be streamed as well as simultaneously writing the RAW file to disk. As mentioned in Chapter 3, Section 3.4.3 it is highly desirable to keep the original RAW file as a source in case the data needs reprocessing at some point in the future. Previous work by Graumann et al. [112] has used a beta set of API libraries, provided by Thermo Scientific, that enabled them to test a version of MaxQuant using a streamed data set on the PC that is attached to the mass spectrometer itself. This work could be extended in a future research project to enable a system such as Kafka to ingest the data and provide it as a stream for cluster-based processing in real-time.

7.3 IMPLEMENTATION DETAILS

7.3.1 CODE CHANGES FROM BATCH

Both the Flink and Spark batch processing code required significant changes to run in a streaming fashion. Firstly, the input interfaces needed changing to access the data from the Kafka topic as opposed to a file in HDFS. Flink provides a Kafka consumer that is simple to use and requires the name of the Kafka topic, the server address and port numbers of the Kafka and Zookeeper servers to access the stream data.

In addition to the changes required to connect to Kafka, the code needs to be restructured to make use of Flink operators. The main function used is the "flatmap" function, this coupled with the window operator replaces the MapReduce structure of the batch process and provides the means to split the incoming stream into windows of a certain period. To ensure that all the 3D peaks were detected the processing window was set to collect thirty seconds of input data. A window of thirty seconds, in this case, will mean that thirty seconds worth of mass spectrometer scans will be processed as one unit. Thirty seconds has been validated in other research as the maximum time taken for a peptide to elute in normal circumstances [76]. Note that in some rare cases it is possible for the 3D peak elution time to be longer than this, in this case a post-process is needed to join the pieces of the 3D peak together to form a complete feature. Some examples of where this situation occurs are with contaminants in the sample or with "manufactured" samples such as the files used to check the accuracy of a mass spectrometer. Note that the test file 371.RAW used in this research is such a manufactured file.

With the window of data defined, the core map and reduce code can be used to process it, this part of the code is unchanged apart from the input data format changing from key-value pairs to tuples. A tuple is the same construct as a row in a relational database, each of its fields is defined in advance and can be accessed directly without the need for additional parsing. The first part of the flatmap task equates exactly to the map task used in the batch processing testing. This means that each thirty second processing window receives all of the scan data to process and consequently the 2D peak processing is carried out in duplicate for every processing window that receives a particular scan. In practice this is not a constraint because the 2D peak picking algorithm is so fast that it does not affect the overall timings by a significant amount. If this were to be an issue then the stream processing could be set up in such a way as to receive the individual scans, process them and emit a second data stream consisting of detected 2D peaks. This second stream would then be consumed in thirty second windows as described in this section. Processing the data in time-bounded windows represents a change in the partitioning strategy discussed in Chapter 6. Instead of partitioning vertically and providing all the scans in a narrow band of mz, the windowed streaming approach partitions the data horizontally in small bands of retention time (scans) and includes the complete mz range of the scans. The implications and mechanics of this are discussed in more detail in the following section.

The Spark code required similar changes, namely the interface to read from the Kafka stream, the data types from key-value pairs to tuples and the partitioned window logic. In version 1.6 of Apache Spark, the streaming architecture is quite different to Flink's. Spark processes streams in "micro-batches"; each window of data needs to be complete before processing will start, this means that the map code cannot start processing the scan data row by row until a full thirty seconds of scans (approximately 150 records of data to process) are available. The Spark streaming code works differently to the Flink code in that the reduce step is actually carried by a flatmap function instead of a window function. This is because instead of taking in a partition of a number of rows and outputting a partition of a different number of rows, the Spark process takes in a key-value pair and outputs a key-value pair. In this instance, the key for input and output is the window identifier, the value for the input is an iterable type containing the 2D features, and the value for the output is an iterable type containing the 3D features.

7.3.2 BATCH PARTITIONS VS STREAMING WINDOWS

As noted above the move from a batch process to processing the data in a streaming fashion allowed a change in partitioning strategy. The need for partitioning still exists because it is the partitions that allow the process to run in parallel with each partition of data being processed independently. For the batch process, the partitions were defined along the mz dimension, each partition had to be above a minimum size to allow a complete isotopic envelope to fit inside one half of the partition. Then overlapping partitions were used to avoid counting incomplete peaks or envelopes occurring near to a boundary.

Overlapping time windows were chosen as the optimum solution for stream processing during this research. As noted, previous research shows that a window of thirty seconds is sufficient to reliably find 3D peaks in the dataset [76], and an overlap of one second was used to create the parallelism. Features that do not entirely fit inside the time window, that is they start before the first scan in the window or are not complete when the last scan in the window is reached need to be identified. Figure 29 shows an example of this, the arrowed line in the figure represents a 3D feature and the numbered bars represent sliding data processing windows. The 3D feature fits entirely into windows one, two and three but only partially into windows four and five. In this case it will be discarded from windows four and five. Note that in the 3d peak batch processing an overlap strategy was used; that is, processing windows that were double the minimum size and overlap the previous and following windows by half. Doing so in this case would result in sixty-second windows that would potentially take an inordinate time to complete. It is faster to use the thirty-second minimum size windows and to exploit the resources of a larger cluster.

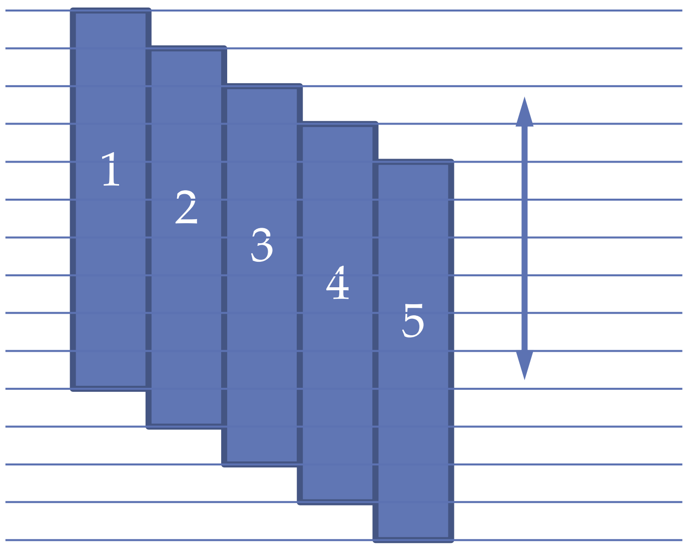


FIGURE 29 SLIDING PROCESSING WINDOWS AND A 3D FEATURE

A process similar to that used to find the peaks that occurred at the boundaries of partitions in the batch process (Chapter 6, Section 6.4.8) is used for the stream process, the rule being that features must start and complete with intensity levels at zero. Note that the same data is processed multiple times because of the overlapping windows. This reprocessing adds extra time to the total time to process the data, but it is possible because there is more time to complete the task than with the batch processing method. As the stream processing method is working to detect features while the mass spectrometer is in operation, there is a significant window of two to four hours in which to complete the feature detection task. Once the mass spectrometer finishes processing the cell sample, the stream processing method will only need to complete the final few windows of data. Whereas, the batch processing method must process the entire data set following the completion of the mass spectrometer run.

Note that the change to using partitions based on retention time as opposed to mz (which was used in the batch implementation) has solved some of the issues related to deciding where to create the partition boundaries. In the batch implementation it was necessary to tailor the partitions to a specific data file in order to reduce the amount of processing skew and use the cluster resources efficiently. When using retention time, the partition size of thirty seconds can be applied across the entire data file to all files. This does still result in processing skew (this can be seen in Table

24 presented later in this chapter) as some scans contain many more 2D features than others. However, this skew is mitigated by several factors:

- The amount of time available for processing while the mass spectrometer is in operation
- The thirty second partitions with the highest number of 2D peaks were processed in an acceptable time on the test cluster

7.3.2 DE-DUPLICATION

The overlapping window approach to stream processing naturally creates duplicate results. Duplicates need to be handled by a separate de-duplication step once the stream processing has completed. This de-duplication could be carried out in-stream, but in fact the number of output peaks, including duplicates was less than a million for even the largest of the test files. At this order of magnitude, a relational database, or equivalent such as Apache Hive, can efficiently de-duplicate the output very rapidly within several seconds. In a complete system where the protein identification part of the process is also completed in-stream then this process would be even faster because the step that identifies the proteins acts as another filter on the dataset resulting in even fewer results to de-duplicate.

7.4 EXPERIMENTATION

7.4.1 OVERVIEW

In contrast to the experimentation in batch processing where various processing and storage platforms were benchmarked, the stream processing experiments were only concerned with the Spark and Flink processing engines. All the data was consumed from a Kafka stream and written to files stored in HDFS. Separate tests were run on the output files to investigate the effect of de-duplication, but this was a distinct step outside of the timings given for stream-processing. The streaming benchmarks were run as a continuous block over a short period of several weeks and no re-runs were needed due to system or code changes. This process was possible as the work on creating and validating the parallel algorithm had already been completed and it was only necessary to engineer the streaming code for Spark and Flink and set up the environment with Kafka before running the benchmarks on the existing Hadoop environment. Most of the benchmark work for this chapter was completed using Flink; the Spark code was validated to ensure that it worked and produced correct results. However, as mentioned the release of a significant update to Spark during the final stages of this research means that it is no longer relevant to benchmark the code written using the previous release. Following version 2 of Spark the streaming architecture is now very similar to the native stream architecture that Flink uses. Given more time it would be far more beneficial to re-factor the Spark code to version 2.2 and benchmark this against Flink.

A reduced set of data files used for the batch-processing benchmarks was used for the streaming experiments. The following four data files were tested 371.RAW, PT2441S1FP1A1.RAW, 100312_EXP229_GFPIP_5.RAW and 561L1AIL00.RAW These four were chosen to be representative of the range of types of files produced by the Lamond Laboratory in the University of Dundee. The complete set of data benchmarked in the previous chapter included eight files forming a full set of experimental results, these all had a very similar distribution of features and produced very similar timings in the batch-mode benchmarks. Therefore, only one of these files was used in the streaming benchmarks detailed in this chapter.

The same methodology for collecting timings for the batch-mode benchmarks was used for the streaming-mode. That is, each file was run through the system three times. No significant outliers were recorded during this process, and the complete set of data is available online from the GitHub repository for this research. The results are reported as the minimum, maximum and average times taken to process each window of data. These timings indicate how long it will take to complete the feature detection process after the mass spectrometer has finished processing a cell sample. The distribution of the number of 2D peaks detected in each scan is also detailed as this is the primary driver behind the amount of time each window of data takes to process. The more 2D peaks exist, the more time is required to iterate through them to detect any 3D peaks and isotopic envelopes.

The data window processing times and the distribution of the 2D peaks can then be used to estimate the required number of processing slots needed on the cluster to ensure the back pressure is kept to a minimum and the process completes as soon as possible after the mass spectrometer run completes. Post-processing steps such as de-duplication and writing the finished files to a central database are not included as these are trivial compared with the data processing steps.

7.4.2 VALIDATION

Validation of the output from the streaming benchmarks was an essential step in this research. As there are few examples of feature detection in a streaming fashion and little evidence that it is a credible way to process mass spectrometer data it was critical to show that the results matched those from the single-threaded and parallel batch processes. The results were written to text files stored in HDFS during the benchmarks, and these were then loaded into a relational database where the de-duplication step was tested and timed. Using SQL queries the de-duplicated results were compared to the output from the batch process described in chapter 4. The results were found to be an exact match with those from the batch process. This is to be expected because the core algorithm is the same code as long as the batch partition and stream window strategies are correct.

The fact that the detected 3D peaks are the same between batch and stream processes is a strong validation that the streaming approach is valid for feature detection, as the batch process output has already been validated against the output from MaxQuant in Chapter 6.

7.5 RESULTS

The same cluster hardware was used for stream processing as was used to run the benchmarks for the batch-mode processing, namely a three-node cluster with Hadoop and Flink installed. Note that the Kafka instance was installed on an "edge node". The term edge node is used to describe a compute node that is separate from the cluster and attached via a network. An edge node does not participate in any of the processing tasks assigned to the cluster. Edge nodes are commonly used as interfaces to clusters and are often used for loading data and for making interactive connections to the cluster. In this case, the edge node was physically located in the same cabinet as the cluster itself and connected via an internal network. For this research, the edge node was used so that the Kafka operations did not affect the CPU utilisation of the processing cluster. Kafka was set up to read the SCMI files generated from the test data and then stream it out at a rate of five scans per second to simulate what would happen if a mass spectrometer were performing this task.

At the rate of five scans per second, each of the Flink stream processing windows contained one hundred and fifty scans. The figures below (Figure 30, Figure 31, Figure 32 and Figure 33) show the distribution of 2D peaks within the test files. These show that in three of the test files, the 2D peaks are more densely concentrated in certain scans. Note that for the manufactured test file 371.RAW (Figure 32) this is not the case, and the 2D peaks are approximately evenly spread throughout the scans compared to the other files.

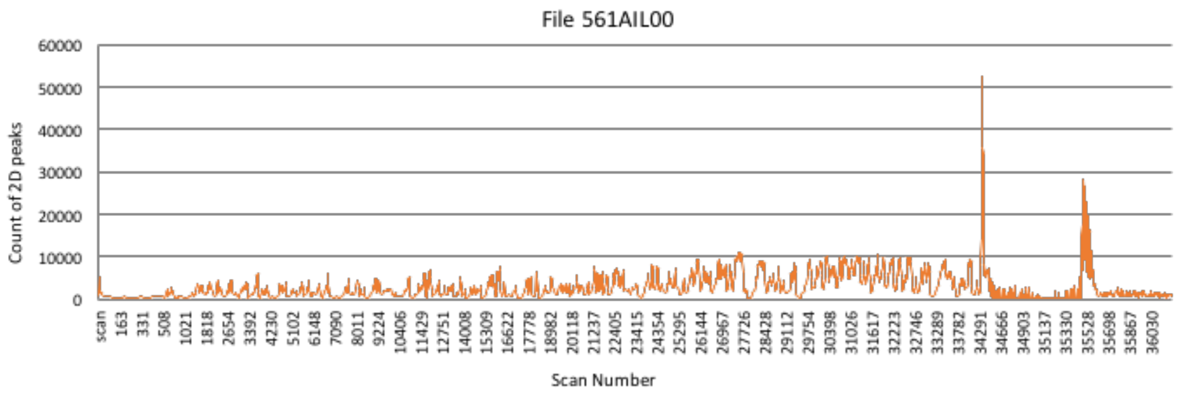


FIGURE 30 COUNT OF 2D PEAKS BY SCAN FOR FILE 561AIL00

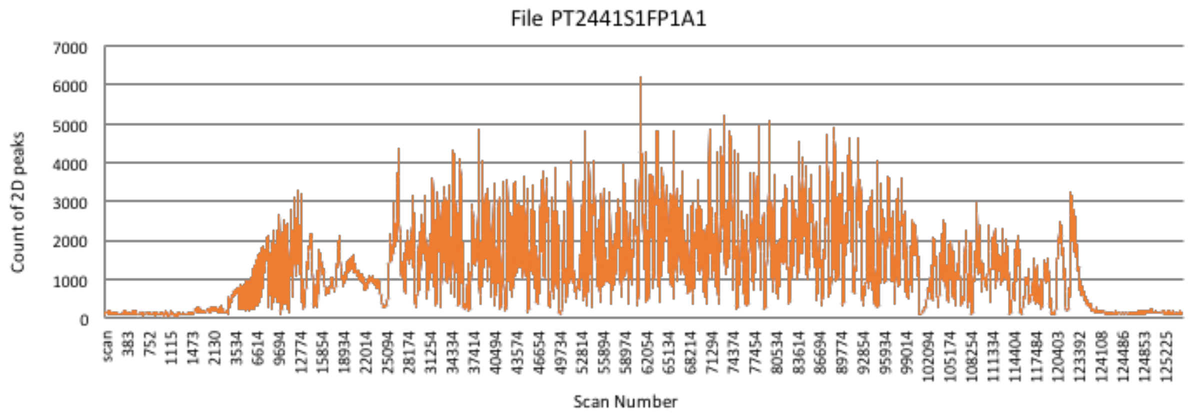


FIGURE 31 COUNT OF 2D PEAKS FOR FILE PT2441S1FP1A1

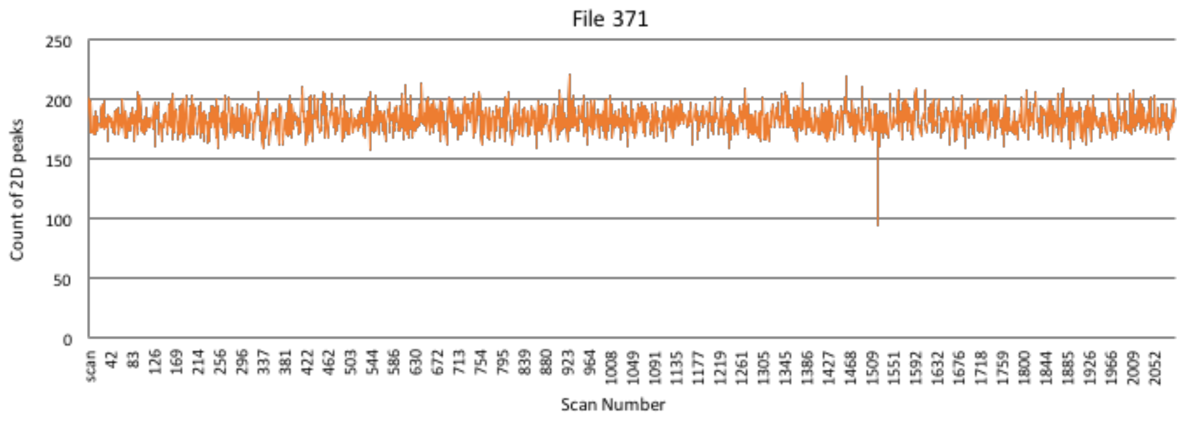


FIGURE 32 COUNT OF 2D PEAKS FOR FILE 371

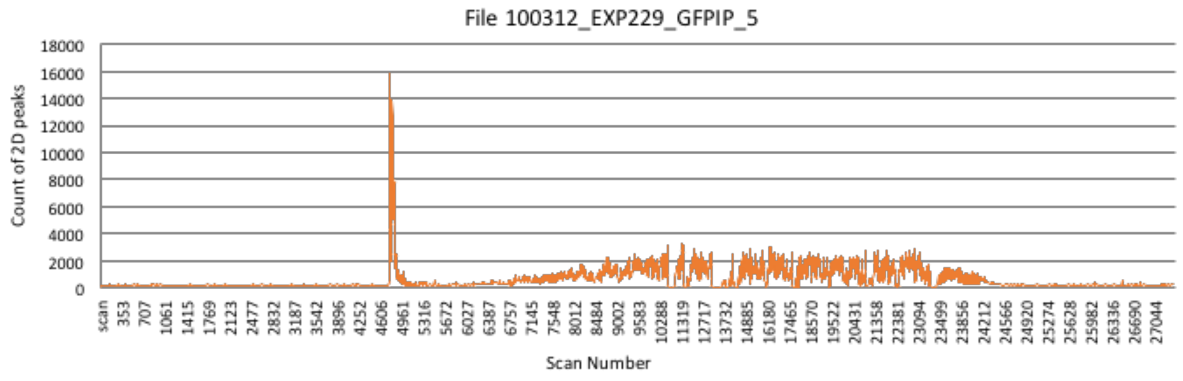


FIGURE 33 COUNT OF 2D PEAKS FOR FILE 100312_EXP229_GFPIP_5

To test the performance of the streaming process, four different samples were taken from each of the test files:

1) near to the beginning of the file where the early scans with a low retention time will have few peaks

2) in the middle of the file where there are relatively more peaks

3) towards the end of the file where the scans with high retention times can have more or fewer peaks dependent on the peak distribution in that particular file.

4) A sample taken around the scan which has the maximum number of peaks.

The results in Table 24 show the timings from the benchmark runs processing individual thirty second windows using Kafka to serve up the stream data and Flink to consume and process the one hundred and fifty scans in each of the windows that have been sampled from the test files.

TABLE 24 TIMINGS FOR STREAM WINDOW PROCESSING WITH APACHE FLINK

File	Window Position in File	2D Feature Detection (HH:MM:SS)	3D Feature Detection (HH:MM:SS)	Number of 2D Features	Number of 3D Features
100312	Beginning	00:02.8	00:01.9	27770	120
100312	Middle	00:06.3	00:20.9	214732	4130
100312	End	00:02.5	00:02.6	28288	102
100312	Max 2D Features	00:10.8	00:31.1	611544	3010
371	Beginning	00:03.7	00:03.1	27233	414
371	Middle	00:03.7	00:03.9	27890	609
371	End	00:03.8	00:03.8	29136	695
371	Max 2D Features	00:03.8	00:04.0	29989	697
PT2441S1FP1A1	Beginning	00:03.0	00:03.0	17937	356
PT2441S1FP1A1	Middle	00:05.8	00:22.0	174930	3705
PT2441S1FP1A1	End	00:04.6	00:10.7	103045	2093
PT2441S1FP1A1	Max 2D Features	00:05.9	00:25.2	189572	5313
561AIL000	Beginning	00:05.9	00:04.4	49198	831
561AIL000	Middle	00:03.6	00:21.0	212579	3012
561AIL000	End	00:13.2	01:14.0	434971	6408
561AIL000	Max 2D Features	00:48.0	03:52.0	1035220	35265

The timings to note are the ones that take longer than thirty seconds to process the 3D peaks. If the cluster is configured with thirty processing slots, then tasks taking longer than thirty seconds will cause back pressure on the system. The back pressure occurs as stream processing windows queue up behind a slow running window waiting for it to finish. Where there are scans with a high density of 2D peaks that require more processing time, the overlapping nature of the processing windows means that each of these high-density scans will appear in thirty consecutive windows. This has the potential to cause a high degree of back pressure in the system. As an example, the window taken around the scan with the highest density for the file 561AIL00 required approximately four and half minutes of processing time (adding the 2D and 3D processing times to get a total processing time). If each of the thirty processing windows containing this scan required a similar amount of time, then each window would cause nine subsequent processing windows to queue up in the system. One way to mitigate this is to add more processing slots; the cluster used in the benchmark tests here was configured with forty-two processing slots. The extra parallelism allows the processing of subsequent windows to continue without waiting for the slower ones. It should also be noted where the high-density scans occur in the data. For the case of file 100312_EXP229_GFPIP_5.RAW the high-density scans are near to the start of the file, see Figure 33. Therefore, windows will begin to queue up for processing causing back presseure. Using a framework such as Kafka, which manages the system back pressure automatically, the system could have time to "catch up" on processing the queuing windows for this file once the high-density ones are complete. Table 25 shows the minimum, maximum and average number of 2D features for each file; Figure 34, Figure 35, Figure 36 and Figure 37 are histograms showing the distribution of 2D feature counts across the scans for each of the files.

TABLE 25 MINIMUM, MAXIMUM AND AVERAGE NUMBER OF 2D FEATURES PER SCAN

File	Minimum Number of 2D Features per scan	Maximum Number of 2D Features per scan	Average Number of 2D Features per scan
561AIL000	30	52350	2624
PT2441S1FP1A1	38	6178	1225
371	4	6196	199
100312	12	15936	609

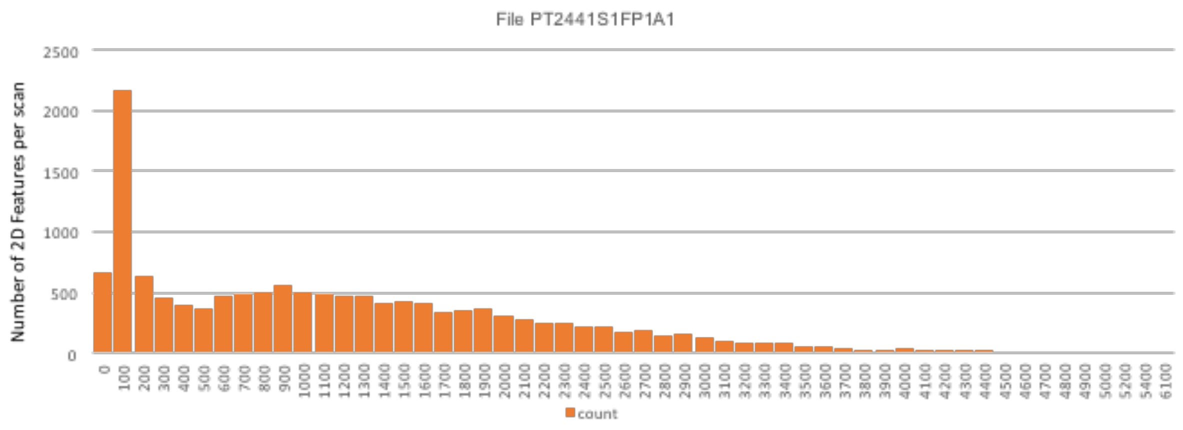


FIGURE 34 HISTOGRAM OF COUNT OF 2D FEATURES FOR FILE PT2441S1FP1A1

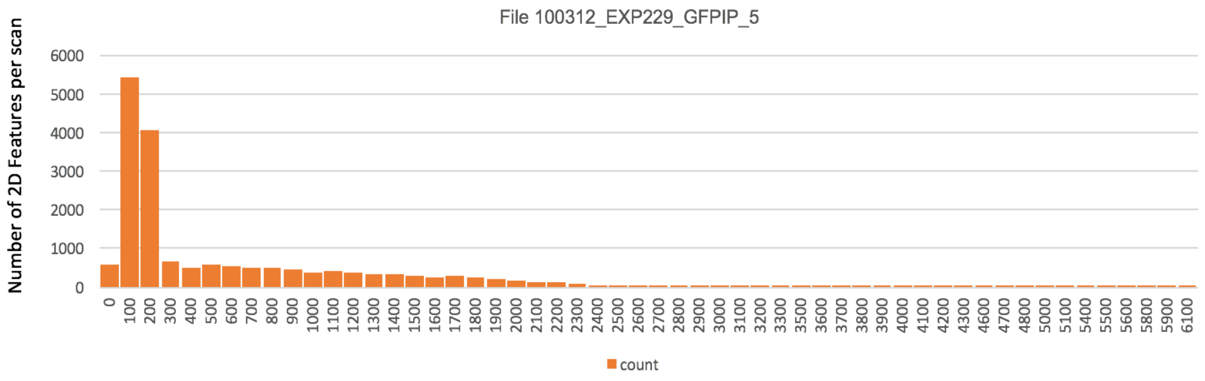


FIGURE 35 HISTOGRAM OF COUNT OF 2D FEATURES FOR FILE 100312_EXP229_GFPIP_5

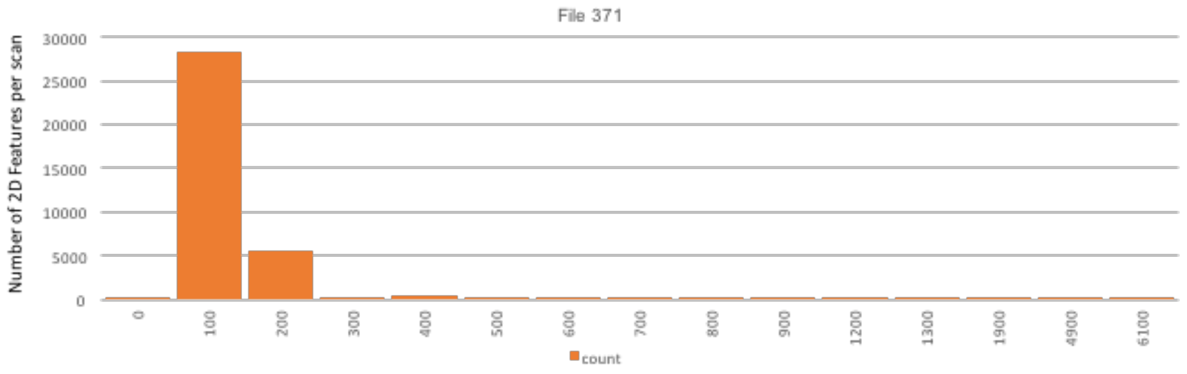


FIGURE 36 HISTOGRAM OF COUNT OF 2D FEATURES FOR FILE 371

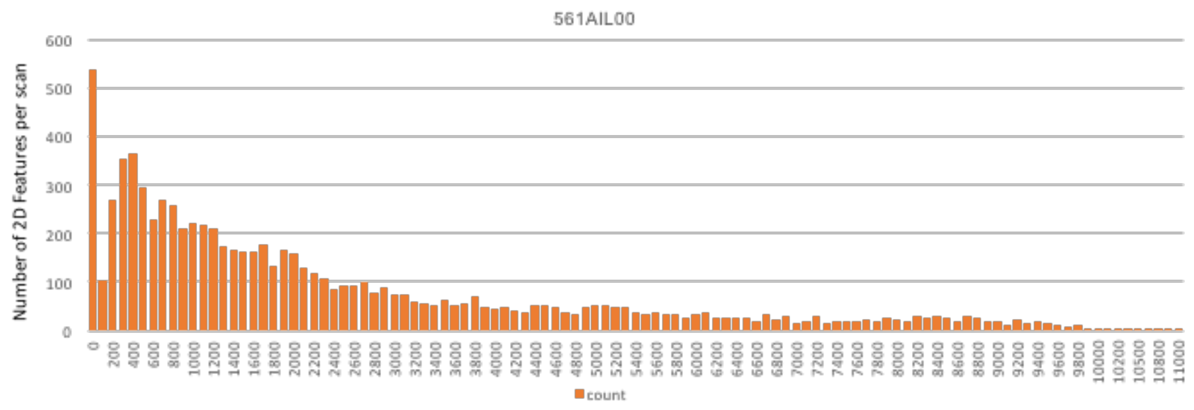


FIGURE 37 HISTOGRAM OF COUNT OF 2D FEATURES FOR FILE 561AIL00

Note the x-axis scale change in Figure 33 from 6100 to 11100 compared to Figures 30,31 and 32

The Histograms above show that while each of the test files has a different distribution for the count of 2D features, all the distributions are skewed towards the lower values. This skew means that for the files included in the benchmark testing most scans have counts of 2D features below the averages shown in Table 25. As noted in Chapter 5, the files used in the benchmarks were chosen to be representative of the range of experiments carried out in the Lamond Laboratory, University of Dundee. Therefore, the maximum counts shown in Table 24 can be classified as outliers and not common occurrences. Table 26 shows for each file, how long the feature detection process continues running once Kafka finishes streaming all the data. This measurement represents the time that a life scientist would have to wait for the feature detection results after the mass spectrometer completes an experiment. Due to the high number of 2D features present in the scans at the end of file 561AIL00, back pressure on the system resulted in a completion time of approximately twenty-eight seconds. However, this is a significant improvement over the batch process, which took approximately fourteen minutes to complete using Apache Flink. The number of 3D features detected match exactly with the results from the batch process.

TABLE 26 STREAM PROCESSING RESULTS

File	Time for processing to complete following the end of the data stream (HH:MM:SS)	Number of 3D Features Detected
561AIL00	00:00:27.6	159201
PT2441S1FP1A1	00:00:08.2	123016
371	00:00:03.8	17777
100312	00:00:03.2	45613

7.6 DISCUSSION

7.6.1 POSSIBLE IMPROVEMENTS

As previously noted there is the possibility of making improvements to the general efficiency of the parallel algorithm. These improvements would reduce the overall execution time for the batch process, whereas for the stream process it would reduce back pressure and enable the process to run on fewer nodes as each node could finish its tasks faster and therefore be available to process other data windows.

To provide a complete proteomics data processing pipeline requires several more steps to be included. These include:

- A process to "stitch together" 3D peaks that are larger than the stream processing windows.
- Deduplication of the complete list of 3D peaks detected during the stream process.
- Integrating the protein identification step into the stream process.

  One possible scenario is that the data is processed in-stream using the method described in this chapter and then the output is passed onto another application such as MaxQuant for the protein identification step. Another scenario is to pipeline the data to another cluster-based solution such as that described by Lewis et al. [79], which is designed to run on a Hadoop cluster. An ideal situation would be for the protein identification to be completed in-stream as well, this would mean that the complete results could be made available within a short time after the experiment ends. To adapt an algorithm such as that developed by Lewis et al. would require a great deal validation and research to ensure correctness. The protein identification step would need to be adapted to work with a set of results from a number of the stream processing windows as they complete.

- A console to monitor jobs and perform administrative tasks.

  The primary concern of this research is the feature detection process. However, it is acknowledged that a non-technical user of such a system would need a graphical user interface (GUI). This interface is not within the scope of this research but a system as described in this chapter would require such an interface for life scientists to monitor their experiments and see the results as they are processed in real-time. The interface should perform several functions; it must include reports that make it simple to see when processing has been completed and to download and view results without getting involved with any technical details of the cluster or data management. It should also be possible to start a job to re-process data with different parameters and to abort processing for an experiment that isn’t producing the expected results.

- Data Validation

  There is the possibility to include some data validation steps into the process. These could be used to flag up experiments that should be aborted before completion. It would also be possible to perform validation of replicates in the streaming process; this would involve processing a cell as a reference sample and then comparing the results from the replicate to the reference during the stream processing. Validating replicates in this way could result in significant time savings for the life science researchers as currently. This validation is often a manual process performed once the proteins in each experiment have been identified.

Overall the experiments detailed in this chapter using a three cluster show that in-stream processing of mass spectrometer is possible and can be completed in near real-time. The results have been validated against those obtained from MaxQuant and the batch-process described in the previous chapter and found to be accurate.

8 DISCUSSION

8.1 CHAPTER SUMMARY

This Chapter presents a summary of the research conducted, the results presented and discusses potential future research stemming from the findings.

Section 2 details the main research question and how it was answered

Section 3 presents the novel contributions to knowledge made during the research

Section 4 describes and compares the results from the batch and stream mode experimentation.

Section 5 discusses the architectural design considerations for batch and real-time feature detection using a horizontally scalable cluster.

Section 6 details potential future work

Section 7 concludes the thesis

8.2 RESEARCH QUESTION

This research addresses the following research question.

Can feature detection in proteomics data be performed in near real-time using a parallel algorithm on a horizontally scalable compute cluster?"

The first step is to define the term "real-time". Real-time in practice can mean many things, from a sub-second response to fraudulent activity in a banking transaction scenario at one end of the spectrum to processing multi-second windows of data in a streaming fashion where response times are not so critical. The processing of mass spectrometer data fits into the latter example. Real-time here meaning that the results of experiments are available to the life scientist with a perceivable delay measured in seconds or even several minutes.

The results given in Chapter 6 show that when using a batch mode of processing and the parallel algorithm developed as part of this research, the aim of reaching a benchmark time for feature detection of less than five percent of the time taken by MaxQuant is sometimes possible but not in all cases. The experiments used a feature detection algorithm written in Java, designed to run in parallel on a cluster in a shared-nothing environment. The algorithm was benchmarked using several different processing frameworks in conjunction with several different distributed storage layers. Some of the larger, more complex files took longer than required by the benchmark and the degree of parallelism possible was constrained by the nature of the features that need to be detected in the data. This constraint limits the number of tasks that can be employed in the 3D peak detection part of the feature-detection process and therefore restricts the speed-up possible from using larger clusters. This constraint means that just adding more nodes to a cluster and increasing the level of parallelism does not speed up the process beyond a certain point.

During the research, it was also shown that intra-file processing of mass spectrometer data is possible. Intra-file processing splits the data file into pieces and processes each part in parallel, as opposed to processing a complete file on each node in a cluster. This knowledge led to the development of a streaming processing pipeline which used Apache Kafka to simulate a mass spectrometer streaming out data as it is produced. Apache Flink consumed the stream and detected the features using a sliding window processing technique and the same parallel algorithm used in the batch processing benchmarks. The results from the streaming experiments show that near real-time processing is possible using this streaming technique. This discovery is significant because receiving results from proteomics experiments in a matter of seconds instead of hours or days means that life scientists can iterate much faster over a problem and act on insights immediately and therefore increase the number of experiments and amount of research that they can carry out.

8.3 ORIGINAL CONTRIBUTIONS

As outlined in the introduction chapter this thesis makes several original contributions, which are closely related to the research question addressed in the previous section. It is expected that these contributions will have a positive impact on the proteomics research community by allowing faster access to results and by reducing the data management burden.

1) A feature detection algorithm designed for use on a parallel cluster environment has been implemented and tested using several processing platforms and data storage layers. The algorithm is written in Java using a MapReduce-style where the 2D peak picking is carried out using a Map step and the 3D peak picking using a Reduce step. The research and experimentation have highlighted the benefits and constraints of using such an algorithm and indicated the speed improvement which is possible. The complete source code, including variants designed to work with Hadoop, Spark, Flink, Aster, HDFS, HBase and Cassandra is available on Github for further research and development. The algorithm has also been successfully adapted to execute in a streaming fashion using either Spark or Flink and a stream of data which, for experimentation was simulated using Kafka.

2) This research also showed that intra-file processing of mass spectrometer data can produce accurate results. This is vital if stream processing is to become mainstream. Intra-file processing contrasts with other research into the parallel processing of mass spectrometer data which propose processing a complete file on each node of a cluster to achieve parallelism.

3) A new file format, specifically intended for parallel processing, has been designed and tested. The format (called SCMI abbreviated from Scan, Mass and Intensity) is row-based containing one mass spectrometer scan per row and is a limited subset of the metadata provided in the complete mzML or RAW file. Only the data required for feature-detection and the later protein identification step are included and stored as tab-separated values. The experiments carried out show that the new format is emminently suitable for parallel processing and is ideal for distribution around the nodes of a compute cluster. As well as the text-based format used in this research,

other formats were investigated. During this investigation, the binary Avro format was also found to be a suitable candidate for the SCMI files and provided a data compression advantage over plain text. Note that the metadata included in the mzML file that is not included in the SCMI format is still required and a different mechanism should be used to store and retrieve that data. The metadata does not require processing and can be stored in a database where it can be referenced alongside the results from the feature detection process.

4) Investigating the use of a central processing cluster, whether in-house or cloud-based, has shown that it can significantly reduce the data management overhead and manual steps that proteomics researchers are involved in when using a PC-based architecture. The automation of data management tasks reduces delays and the chance of errors and allows life scientists to spend more time on their research. A central processing cluster and the introduction of technologies used in an internet of things architecture brings many other benefits including scalability, better governance, simplifying backup and restore activities and security.

8.4 BATCH VS STREAMING RESULTS

Chapter 6 details the implementation of the feature detection algorithm and its use in batch-mode on a parallel cluster. The results of the benchmarks from the systems tested show that a substantial speed-up in processing times over a PC-based process is possible. Some notable discoveries are listed below:

Of all the systems used in this research Apache Flink was the fastest with Apache Spark a close second. It is possible that this order could change if the newest distribution of Spark had been used. However, this was not tested due to version 2.2 of Spark being made available in the summer of 2017, which was too late to be included in this research.

The Aster system requires more nodes to reach the same level of performance as the Hadoop system because the degree of parallelism is fixed at six virtual workers per Aster node, whereas on Hadoop this is configurable using YARN. Given the same degree of parallelism then the Aster system was faster than Flink for the larger, more complex files.

Using HDFS as the storage layer was faster than either Cassandra or HBase. The feature detection algorithm reads the entire data set record by record in a sequential manner without requiring any random access. More complex data storage layers, with higher processing overheads, such as Cassandra and HBase were not needed. For a sequential full file scan, the distributed file structure of HDFS was all that was needed to serve the data up to the processing frameworks. If random access to the data becomes a requirement in the future, then data stores such as Cassandra and HBase will need further investigation.

The timings of individual reduce tasks show that there is still a significant amount of processing skew taking place in the cluster. For example, when processing file 561L1AIL00.RAW, the fastest reduce tasks completed in approximately 30 seconds whereas the slowest took around 12 mins. When distributing data on a cluster for processing, the usual method is to ensure that each node receives an equal amount of data. A skewed distribution of data around the nodes of a cluster means that some nodes will have to do more work than others when performing simple operations such as aggregation. However, in a highly complex task such as feature detection in proteomics data, the skew can occur in the amount of processing required for equal size sets of data. Mass spectrometer data consists of a significant amount of noise with relatively few features. As these features (peaks) are not distributed evenly, some sections of data will be processed very quickly as there is no work for the algorithm to do. In contrast, sections of data that contain many peaks will be processed more slowly as the algorithm must iterate over the section several times while detecting the peaks. To produce an optimum solution for batch-processing, further work is required to sample the 2D feature output before it is distributed for 3D feature detection.

Chapter 7 describes the research into adapting the algorithm developed for use in a batch-mode so that it can process data in a streaming fashion. A system was designed using Apache Kafka to simulate the mass spectrometer streaming its results out in real-time as it produces data. In practice, this would require collaboration with the mass spectrometer manufacturer. However, some work has already been done in this area with Thermo Scientific using an API provided to Graumann et al. [112].

The major change required for stream processing was to implement time-based window processing to replace the map and reduce steps of the batch-mode process. The individual mass spectrometer scans are still processed in the same way as in the map tasks to produce 2D peaks. However, the reduce tasks are replaced with overlapping windows that collects thirty seconds of scans at a time (approximately one hundred and fifty scans). To avoid missing features that occur at window boundaries the thirty-second windows overlap, with a new window being created every second. This overlap means that data is processed multiple times and necessitates a de-duplication step on completion. De-duplication is required as many of the 3D peaks will be detected in multiple windows. Landing the results of the stream processing into a database means that the final de-duplication step is fast and straightforward.

The core of the feature detection algorithm is unchanged between the batch and stream mode processing, the implementation of the 3D peak detection is different in that during the batch process, a reduce step takes a partition of 2D peaks falling in a narrow partition of mass values for all retention times. This partitioning strategy also requires an overlap of data to handle peaks that occur at the boundaries of the partition. Whereas, the stream processing takes a limited band of thirty seconds of retention times for the complete scan containing all mass values in the band. Of the two approaches, the stream processing window method is much more straightforward and does not require any pre-calculation or sampling of the data to produce the window. For the batch-mode processing, the distribution of data and the definition of the boundaries of the partition is complicated, requiring either manual pre-calculation or in-process sampling of the output from the 2D map tasks to produce optimum partitions.

The stream processing method also provides a better chance of achieving near real-time processing given that a cluster can be made large enough to process the data without back pressure. Back pressure occurs when the compute nodes cannot process the incoming data quickly enough, and the data from the stream builds up in a queue waiting for processing. Finally, the stream processing method allows specific checks and validations to be carried out during the processing rather than waiting until the process has completed.

8.5 DESIGN OF AN IDEAL ENVIRONMENT

8.5.1 BATCH PROCESSING

The batch processing method of feature detection has been thoroughly tested using a variety of software frameworks both for data processing and data storage. It is possible to design a reference architecture that could be built into a production environment running in a life sciences laboratory by considering the results and experience gained during this research. The data processing challenges start once the mass spectrometer has completed the experiment and a binary RAW output file has been deposited on the PC attached to it. An automated process needs to be running that can detect the completed file creation. This process will then need to copy the RAW file on to the central cluster. An edge node should be used as a gateway to the cluster to avoid security and other governance issues. An edge node is a name given to a compute node that is connected to a cluster but does not have any of the cluster software running on it and is not used as part of the processing system. The RAW file can be transformed into a format designed for parallel processing during the copy process using the method developed during this research and described in Chapter 5 Section 5.3.4. Once copied to the cluster, the data file in either plain text or Avro format, is then ready to be processed. A file containing the information necessary to start the batch can be transferred to the cluster immediately after the data file to act as a flag for the processing system to start and to show that the data file transformation and the copy have completed successfully. A process running on the cluster will scan for the flag files and start the batch process running. Once the batch feature detection process completes, the output can be saved to the cluster while a second process is initiated to complete other parts of the proteomics processing pipeline (such as the previously mentioned protein lookup process, implemented by the Hydra project). The final results are then stored on a network share where the life sciences researcher can access them from a personal computer. In this way, the process will be completely automated; the researcher need only start the experiment at the mass spectrometer and then wait for notification that results are available. If any reprocessing is required, perhaps due to different processing parameters, then a simple web-based graphical user interface could be used to restart jobs and also used to monitor the progress of the job on the cluster.

8.5.2 STREAM PROCESSING

Using a streaming approach to data processing results in a more straightforward architecture than using a batch-mode of operation. Using the API provided by the mass spectrometer manufacturer, a stream of data is created that flows out continuously while the mass spectrometer is in operation. A streaming tool such as Kafka can be used to make the stream available to consumers in a robust way. Kafka logs streams as they are produced and provides the ability to reprocess data in the event of failure; it also ensures that data is not lost in the event of back pressure on the system. Back pressure occurs when the compute nodes can not process data from the stream fast enough and a backlog of data to process builds up. Apache Spark and Apache Flink have been benchmarked processing the stream data during this research. Both of these tools provide a stream processing capability, in the versions tested here they work in different ways with Spark employing a micro-batch mode and Flink employing a true streaming capability where each piece of data is handled as it arrives. As mentioned in Chapter 7, following the release of Spark version 2.2 in the summer of 2017 Spark now works in a streaming fashion in the same way as Flink. Because of this, most of the research and benchmarking has been carried out using Flink. The code for processing the data using Spark has been produced, but this is less relevant now since Spark streaming has undergone such a significant change in underlying architecture.

In a large laboratory with many mass spectrometers operating simultaneously, using Kafka and a processing framework such as Flink in conjunction with a large central cluster would provide a robust, efficient environment. The stream processing would be carried out in parallel on the cluster, with each window of data being processed independently of the others. Also, a cloud-based architecture could be employed to allow the environment to be scaled up or down in response to demand. The mass spectrometers do produce a large volume of data which would need to be transferred via an internet connection to a cloud system provider. However, the streaming model means that the data volume would be four or five mass spectrometer scans per second for each machine, which amounts to approximately 70Kb/sec of data. The results from the streaming process need to be passed to a post-process. This post-process will “stitch together” any 3D peaks which have occurred across more than one processing window, in a process described in Chapter 7. Finally, the data needs to be de-duplicated as the overlapping window style of processing means that the same data is processed several times. Both of these steps could be run efficiently in a relational database; this would require the streaming process to land the data into a database and then trigger the post-processing once the stream is complete.

8.6 BENEFITS AND FUTURE WORK

The research described in this thesis and the cluster-based processing solution that stems from it provide many benefits to the proteomics researcher. These include:

- Faster access to the results of experiments, the algorithm developed here is capable of running in parallel on a cluster. Therefore, data can be processed in near real-time and made available to the researchers within minutes of the completion of an experiment.
- In-stream monitoring of experiments. Using the streaming method of processing described in Chapter 7, the in-stream processing means that some of the data validation and checks that are currently carried out manually after the experiment has completed are possible while the mass spectrometer is working. It is usual to run the experiment multiple times and create what is known as biological and technical replicates of the data. Biological replicates are created by processing the same cell sample multiple times, and technical replicates are created by using precisely the same machine setup and parameters on a different cell sample. The results from these replicates are then compared to check that they are consistent. Stream processing would allow this validation to be done in real-time while the mass spectrometer is running.
- Less involvement in data management tasks. The central processing cluster architecture using stream processing provides a mechanism that allows data to flow from the mass spectrometer to the cluster for processing and then for the results to be deposited in a database ready for analysis. This mechanism frees the researcher from the tasks of copying files between systems and running a processing pipeline manually.
- Access to a fully managed solution. The central processing cluster means that standard IT services such as backup, restore, disaster recovery, security and support can be provided in a far more comprehensive and straightforward manner than using a PC-based solution.
- Robustness and efficiency. Many of the software and systems that currently exist for processing proteomics data have been produced by life science

researcher who is also skilled in the use of computers and software development. One of the aims of this research has been to contrast such a system with one that has been designed by a computer scientist who has proteomics domain knowledge.

- The quantity of data produced by a life science laboratory varies over time so the processing power required also values. A central cloud-based cluster can be expanded quickly and efficiently for as long as extra processing power is needed and then scaled back during less busy periods.

The research into parallel feature detection leads to the possibility of further research:

- Parallel Feature detection provides only part of the solution for an architected proteomics pipeline, and further work will be required to integrate other research such as the protein detection work done by Lewis et al. for the Hydra project [79]. Providing a complete pipeline could be a relatively simple task using a batch-mode of execution; the results would only need to feed into the Hydra system in the correct format. However, to enable this in a streaming mode would be far more complex and require much more work in adapting and validating an algorithm. The protein identification step would need to be able to work with partial results sets from either single or groups of outputs from the processing windows rather than the complete set of features as is the case with the batch-mode output. However, this does offer the potential of an actual real-time system where the proteins in a sample are fully identified within a short time of the mass spectrometer completing the processing of a cell sample.
- Updating the processing frameworks to the latest versions to take advantage of new features. This is particularly relevant to Spark where the changes made in version 2.2 released in the summer of 2017 mean that the stream processing in Spark behaves similarly now to the stream processing in Flink.
- Further performance enhancements of the algorithm, either using Java or by investigating the use of Scala on Spark or compiling C libraries.
- The method for creating the reduce task partitions in the batch-mode processing described in Chapter 6 is manual and requires a new calculation

for every new file. This manual effort has been acceptable to benchmark the algorithm during this research, however, in a production process, this manual process would not be suitable. Work is required to produce code to sample the map task output and produce the partitions for the reduce task automatically.

- An investigation into the use of containers for virtualisation and reproducibility. Recent developments in software such as docker mentioned in chapter 5 allow complex systems to be built in a way that enables efficient scalability. The containers themselves are a lightweight virtual machine containing program code and application libraries. A major benefit of this technology is that it allows different versions of applications to run simultaneously. The co-existence of different versions of software can be very important when using open source frameworks as often there are frequent updates to test and install. The use of containers means that code that has been proven to work in older versions can continue to run using that version without needing to be changed and validated on a newer version.
- Further research into different formats for the SCMI file could find an optimum solution. During this research, the Avro file format produced promising results. The Avro binary format results in smaller files than plain text, Avro also has the benefit of being self-describing thanks to its in-built schema mechanism. Another line of research could be to investigate the parquet file format which like Avro is binary and self-describing. The developers of the Spark processing framework have built Parquet read and write optimisation code into the Spark Framework itself.

8.7 CONCLUSIONS

Proteomics is an essential area of research within life sciences and systems biology. The large-scale study of proteins and their interactions is one of the major components of producing personalised medicine, which is at the forefront of medical care. Personalised medicine means that drugs and other treatments are designed specifically for an individual patient's unique biology, significantly reducing adverse reactions and side-effects and increasing the efficacy of treatment. The advance of technology in mass spectrometers has led to an increase in the volume and complexity of data that needs to be processed before insights can be gained from it. The output data is, in many cases, currently processed using PC-based technology and requires the proteomics researcher to be involved in the details of data management. By employing a centralised processing cluster with connected instruments streaming data in real-time, researchers will benefit from the removal of much of the data management burden and faster access to results. A more robust and efficient process internet of things style architecture will also bring benefits for the life sciences laboratory.

The research has shown that it is possible to detect features in proteomics data using a parallel cluster in less time than using a PC-based process. The previous section details these benefits, along with some ideas for future related research. Proteomics is still an evolving field of study, and as the technology used to process the cell, samples improve the amount of data will only continue to grow, meaning that a processing solution involving parallelism may become a necessity. The streaming study carried out here also points to a future where samples are processed and analysed in near real-time creating many new possibilities for proteomics research, such as immediate diagnosis of conditions and personalised medicine.

REFERENCES

[1] S. Orchard, P. Binz, A.R. Jones, J.A. Vizcaino, E.W. Deutsch and H. Hermjakob, “Preparing to work with Big Data in Protemomics - A report on the HUPO-PSI Spring workshop,” Proteomics, vol. 13, no. 20, pp. 2931–2937, 2013.

[2] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” Comput. Networks, vol. 54, no. 15, pp. 2787–2805, 2010.

[3] J. Cocke, “The search for performance in scientific processors,” Commun. ACM, vol. 31, no. 3, pp. 250–253, 1988.

[4] L. G. Valiant, “A bridging model for parallel computation,” Commun. ACM, vol. 33, no. 8, pp. 103–111, 1990.

[5] J. Dean and S. Ghemawat, “MapReduce Simplified Data Processing on Large Clusters 2004.pdf,” in Sixth Symp. Oper. Syst. Des. Implement, 2004, pp. 137–149.

[6] S. Miller, “Collaborative Approaches Needed to Close the Big Data Skills Gap,” J. Organ. Des., vol. 3, no. 1, pp. 26–30, 2014.

[7] V. Özdemir, E. S. Dove, U. K. Gürsoy, S. Şardaş, A. Yıldırım, S. G. Yılmaz, B. Ömer, K. Güngör and A. S. Mete, “Personalized medicine beyond genomics: alternative futures in big data—proteomics, environtome and the social proteome,” J. Neural Transm., pp. 1–8, 2015.

[8] A. Bairoch and R. Apweiler, “The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000,” Nucleic Acids Res., vol. 28, no. 1, pp. 45–48, 2000.

[9] L. E. Hood, G. S. Omenn, R. L. Moritz, R. Aebersold, K. R. Yamamoto, M. Amos, J. Hunter-Cevera and L. Locascio, “New and improved proteomics technologies for understanding complex biological systems: Addressing a grand challenge in the life sciences,” Proteomics, vol. 12, no. 18, pp. 2773–2783, 2012.

[10] M. W. Senko, P. M. Remes, J. D. Canterbury, R. Mathur, Q. Song, M. Eliuk, C. Mullen, L. Earley, M. Hardman, J. D. Blethrow, H. Bui, A. Specht, O. Lange, E. Denisov, A. A. Makarov and S. Horning, “Novel Parallelized Quadrupole / Linear Ion Trap / Orbitrap Tribrid Mass Spectrometer Improves Proteome Coverage and Peptide Identification Rates,” Anal. Chem., vol. 85, no. 24, pp. 11710–11714, 2013.

[11] Q. Peter He, J. Wang, J. A. Mobley, J. Richman and W. E. Grizzle “Self-calibrated warping for mass spectra alignment,” cancer inform., vol. 10, pp. 65–82, 2011.

[12] J. Cox and M. Mann, “MaxQuant enables high peptide identification rates, individualized p.p.b.-range mass accuracies and proteome-wide protein quantification.,” Nat. Biotechnol., vol. 26, no. 12, pp. 1367–72, 2008.

[13] S. Nahnsen, C. Bielow, K. Reinert, and O. Kohlbacher, “Tools for label-free peptide quantification.,” Mol. Cell. Proteomics, vol. 12, no. 3, pp. 549–556, 2013.

[14] A. D. Weston and L. Hood, “Systems Biology, Proteomics, and the Future of Health Care : Toward Predictive, Preventative, and Personalized Medicine Introduction : Paradigm Changes in Health Care,” J. Proteome Res., vol. 3, no. 2, pp. 179–96, 2004.

[15] L. Martens, M. Chambers, M. Sturm, D. Kessner, F. Levander, J. Shofstahl, W. H. Tang, A. Römpp, S. Neumann, A. D. Pizarro, L. Montecchi-Palazzi, N. Tasman, M.

Coleman, F. Reisinger, P. Souda, H. Hermjakob, P. Binz and E. W. Deutsch, “mzML--a community standard for mass spectrometry data.,” Mol. Cell. Proteomics, vol. 10, no. 1, p. R110.000133, Jan. 2011.

[16] N. Neuhauser, N. Nagaraj, P. McHardy, A. Zanivan, R. Scheltema, J. Cox and M. Mann, “High Performance Computational Analysis of Large-scale Proteome Data Sets to Assess Incremental Contribution to Coverage of the Human Genome,” J. Proteome Res., vol. 12, no. 4, pp. 2858–2868, 2013.

[17] J. Von Neumann, “First Draft of a Report on the EDVAC,” IEEE Ann. Hist. Comput., vol. 15, no. 1, 1993.

[18] J. Backus, “Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs,” Commun. ACM, vol. 21, no. 8, pp. 613–641, Aug. 1978.

[19] G. Moore, “Cramming more components onto integrated circuits,” Electronics, vol. 38, no. 8, pp. 33–35, 1965.

[20] L. J. Flynn, “Intel halts development of 2 new microprocessors,” New York Times, vol. 8, 2004.

[21] M. J. Quinn, Parallel Computing (2Nd Ed.): Theory and Practice. New York, NY, USA: McGraw-Hill, Inc., p 6, 1994.

[22] W. J. Bouknight, A. Stewart, D. Denenberg, D. E. McIntyre, J. M. Randall, A. H. Sameh, D. Slotnick, “The illiac iv system,” in IEEE, 1972, vol. 60, no. 4, pp. 369–388.

[23] J. S. Dietrich, “Cosmic Cubism,” Eng. Sci., vol. 47, no. 4, pp. 17–20, 1984.

[24] M. J. Flynn, “Some Computer Organizations and Their Effectiveness,” IEEE Trans. Comput., vol. C-21, no. 9, pp. 948–960, Sep. 1972.

[25] M. J. Flynn, “Very high-speed computing systems,” in Proceedings of the IEEE, 1966, vol. 54, no. 12, pp. 1901–1909.

[26] D. J. Paterson, Computer Architecture: A Quantitative Approach. Waltham, MA: Morgan Kaufmann, pp. 35-76, 1996.

[27] V. S. Sunderam, “PVM: A framework for parallel distributed computing,” Concurr. Pract. Exp., vol. 2, no. 4, pp. 315–339, Dec. 1990.

[28] D. Becker and T. Sterling, “BEOWULF: A parallel workstation for scientific computation,” in International Conference on Parallel Processing, 1995, pp. 2–5.

[29] K. Hokamp, D. C. Shields, K. H. Wolfe, and D. R. Caffrey, “Wrapping up BLAST and other applications for use on Unix clusters,” Bioinformatics, vol. 19, no. 3, pp. 441–442, Feb. 2003.

[30] B. E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky, “SETI @ HOME — Massively distributed computing for SETI,” Sci. Program., no. 1, pp. 78–83, 2001.

[31] G. M. Amdahl, “Validity of the single processor approach to achieving large scale computing capabilities,” in Proceedings of the April 18-20, 1967, spring joint computer conference on - AFIPS ’67 (Spring), 1967, pp. 483–485.

[32] J. L. Gustafson, “Reevaluating Amdahl’s law,” in Communications of the ACM, 1988, vol. 31, no. 5, pp. 532–533.

[33] A. Kalyanaraman, W. Cannon, B. Latt, and D. Baxter, “MapReduce implementation of a hybrid spectral library-database search method for large-scale peptide identification,” Bioinformatics, vol. 27, no. 21, pp. 3072–3073, 2011.

[34] E. Brewer, “CAP twelve years later: How the ‘rules’ have changed,” Computer (Long. Beach. Calif)., vol. 45, no. 2, pp. 23–29, 2012.

[35] R. Lämmel, “Google’s MapReduce programming model — Revisited,” Sci. Comput. Program., vol. 70, no. 1, pp. 1–30, Jan. 2008.

[36] D. Srivastava and X. Dong, “Big data integration,” Int. Conf. Data Eng. (ICDE), 2013 IEEE 29th, pp. 1245–1248, 2013.

[37] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares and X. Qin, “Improving MapReduce performance through data placement in heterogeneous Hadoop clusters,” Parallel Distrib. Process. Work. Phd Forum (IPDPSW), 2010 IEEE Int. Symp., vol. 9, pp. 29–42, 2010.

[38] S. Chaudhuri, U. Dayal, and V. Narasayya, “An overview of business intelligence technology,” in Communications of the ACM, 2011, vol. 54, no. 8, pp. 88–98.

[39] R. P. Weicker, “An overview of common benchmarks,” Computer (Long. Beach. Calif)., vol. 23, no. 12, pp. 65–75, Dec. 1990.

[40] M. Poess, B. Smith, L. Kollar, and P. Larson, “TPC-DS, taking decision support benchmarking to the next level,” in SIGMOD’02, 2002, pp. 582–587.

[41] M. Poess, R. Nambiar, and D. Walrath, “Why you should run TPC-DS: a workload analysis,” in VLDB 2007, 2007, pp. 1138–1149.

[42] TPC, “TPC-HS Specification,” http://www.tpc.org/tpcx-hs (2014), 2014. [Online]. Available: http://www.tpc.org/tpcx-hs/.(last accessed 08/02/2018)

[43] C. Baru, M. Bhandarkar, R. Nambiar, M. Poess, and T. Rabl, “Benchmarking Big Data Systems and the BigData Top100 List,” Big Data, vol. 1, no. 3, pp. 60–64, Mar. 2013.

[44] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan and S. Weeratunga, “THE NAS PARALLEL BENCHMARKS,” Int. J. High Perform. Comput. Appl., vol. 5, no. 3, pp. 63–73, 1991.

[45] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. Lee and K. Skadron, “Rodinia: A benchmark suite for heterogeneous computing,” in 2009 IEEE International Symposium on Workload Characterization (IISWC), 2009, pp. 44–54.

[46] N. Raghunath, “Towards an Industry Standard for Big Data Systems,” in Advancing Big Data Benchmarks, vol. 8585, T. Rabl, N. Raghunath, M. Poess, M. Bhandarkar, H.-A. Jacobsen, and C. Baru, Eds. Cham: Springer International Publishing, 2014, pp. 193–201.

[47] C. Baru, M. Bhandarkar, R. Nambiar, M. Poess, and T. Rabl, “Setting the direction for big data benchmark standards,” in Workshop on Big Data Benchmarking, 2011, pp. 1–13.

[48] D. Laney, D. Management, and C. D. Volume, “Controlling Data Volume, Velocity and Variety,” META Gr. Res. Note 6, no. February 2001, 2001.

[49] Z. Ren, W. Shi, and J. Wan, “Towards Realistic Benchmarking for Cloud File Systems: Early Experiences,” in IEEE International Symposium on Workload Characterization, 2014.

[50] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte and H. Jacobsen, “Bigbench: Towards an industry standard benchmark for big data analytics,” in SIGMOD’13, 2013, pp. 1197–1208.

[51] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, “Benchmarking cloud serving systems with YCSB,” in Proceedings of the 1st ACM symposium on Cloud computing - SoCC ’10, 2010, pp. 143–154.

[52] S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. López, G. Gibson, A. Fuchs and B. Rinaldi, “YCSB ++ : Benchmarking and Performance Debugging Advanced Features in Scalable Table Stores,” in Proceedings of the 2nd ACM Symposium on Cloud Computing. ACM, 2011.

[53] O. O’Malley and A. C. Murthy, “Winning a 60 Second Dash with a Yellow Elephant Hadoop implementation,” in Proceedings of sort benchmark, 2009, pp. 1–9.

[54] J. Gray, “The cost of messages,” Proc. seventh Annu. ACM Symp. Princ. Distrib. Comput., pp. 1–7, 1988.

[55] G. D. Bader, A. Heilbut, B. Andrews, M. Tyers, T. Hughes, and C. Boone, “Functional genomics and proteomics: charting a multidimensional map of the yeast cell,” Trends Cell Biol., vol. 13, no. 7, pp. 344–356, Jul. 2003.

[56] W. C. S. Cho, “Proteomics Technologies and Challenges From Genomics to Proteomics,” Genomics. Proteomics Bioinformatics, vol. 5, no. 2, pp. 77–85, 2007.

[57] S. D. Patterson and R. Aebersold, “Proteomics: the first decade and beyond.,” Nat. Genet., vol. 33 Suppl, no. 3, pp. 311–323, Mar. 2003.

[58] S. T. Kosak and M. Groudine, “Gene order and dynamic domains.,” Science, vol. 306, no. 5696, pp. 644–7, Oct. 2004.

[59] V. Dhingra, M. Gupta, T. Andacht, and Z. F. Fu, “New frontiers in proteomics research: a perspective.,” Int. J. Pharm., vol. 299, no. 1–2, pp. 1–18, Aug. 2005.

[60] P. Legrain, R. Aebersold, A. Archakov, A. Bairoch, K. Bala, L. Beretta, J. Bergeron, C. Borchers, G. Corthals, C. Costello, E. Deutsch, B. Domon, W. Hancock, F. He, D. Hochstrasser, G. Salekdeh, S. Sechi, M. Snyder, S. Srivastava, M. Uhle, C. Wu, T. Yamamoto and Y. Paik, “The Human Proteome Project : Current State and Future Direction,” Mol. Cell. Proteomics, vol. 10, pp. 1–5, 2011.

[61] P. Mallick and B. Kuster, “perspective Proteomics : a pragmatic perspective,” Nat. Biotechnol., vol. 28, no. 7, pp. 695–709, 2010.

[62] B. Futcher, G. I. Latter, P. Monardo, C. S. Mclaughlin, I. Garrels, and C. S. M. C. Laughlin, “A Sampling of the Yeast Proteome A Sampling of the Yeast Proteome,” Mol. Cell. Proteomics, vol. 19, no. 11, pp. 7357–7368, 1999.

[63] J. J. Thomson, “Cathode Rays,” Philos. Mag., vol. 44, no. 269, pp. 25–29, Feb. 1897.

[64] G. Squires, “Francis Aston and the mass spectrograph Early life,” Dalt. Trans., no. 23, pp. 3893–3899, 1998.

[65] A. F. M. Altelaar, J. Munoz, and A. J. R. Heck, “Next-generation proteomics : towards

an integrative view of proteome dynamics," Nat. Rev. Genet., vol. 14, no. 1, pp. 35–48, 2012.

[66] R. Aebersold and M. Mann, "Mass spectrometry-based proteomics," Nature, vol. 422, no. March, pp. 198–207, 2003.

[67] J. B. Fenn, M. Mann, C. K. A. I. Meng, S. F. Wong, and C. M. Whitehouse, "Electrospray ionization for mass spectrometry of large biomolecules," Science (80-.)., vol. 246, no. 6, pp. 64–71, 1989.

[68] A. Michalski, E. Damoc, J. Hauschild, O. Lange, A. Wieghaus, A. Makarov, N. Nagaraj, J. Cox, M. Mann and S. Horning, "Mass Spectrometry-based Proteomics Using Q Exactive, a High-performance Benchtop Quadrupole Orbitrap Mass Spectrometer," Mol. Cell. Proteomics, vol. 10, pp. 1–12, 2011.

[69] S.-E. Ong and M. Mann, "A practical recipe for stable isotope labeling by amino acids in cell culture (SILAC).," Nat. Protoc., vol. 1, no. 6, pp. 2650–2660, Jan. 2006.

[70] J. Rappsilber and M. Mann, "What does it mean to identify a protein in proteomics?," Trends Biochem. Sci., vol. 27, no. 2, pp. 74–78, Feb. 2002.

[71] C.-H. W. Chen, "Review of a current role of mass spectrometry for proteome research.," Anal. Chim. Acta, vol. 624, no. 1, pp. 16–36, Aug. 2008.

[72] M. Katajamaa and M. Orešič, "Data processing for mass spectrometry-based metabolomics," J. Chromatogr. A, vol. 1158, no. 1–2, pp. 318–328, 2007.

[73] J. Dudley and A. Butte, "A quick guide for developing effective bioinformatics programming skills," PLoS Comput. Biol., vol. 5, no. 12, pp. 1–7, 2009.

[74] A. Chawade, M. Sandin, J. Teleman, J. Malmstrom, and F. Levander, "Data processing has major impact on the outcome of quantitative label-free LC-MS analysis," J. Proteome Res., vol. 14, no. 2, pp. 676–687, 2015.

[75] L. Martens, A. I. Nesvizhskii, H. Hermjakob, M. Adamski, G. S. Omenn, J. Vandekerckhove and K. Gevaert, "Do we want our data raw? Including binary mass spectrometry data in public proteomics data repositories," Proteomics, vol. 5, no. 13, pp. 3501–3505, 2005.

[76] W. X. Schulze and B. Usadel, "Quantitation in Mass Spectrometry based Proteomics," Annu. Rev. Plant Biol., vol. 61, pp. 491–516, 2010.

[77] M. Sandin, J. Teleman, J. Malmström, and F. Levander, "Data processing methods and quality control strategies for label-free LC-MS protein quantification.," Biochim. Biophys. Acta, vol. 1844, no. 1 Pt A, pp. 29–41, Jan. 2014.

[78] R. Craig and R. C. Beavis, "TANDEM : matching proteins with tandem mass spectra," Bioinformatics, vol. 20, no. 9, pp. 1466–1467, 2004.

[79] S. Lewis, A. Csordas, S. Killcoyne, H. Hermjakob, M. R. Hoopmann, R. L. Moritz, E. W. Deutsch and J. Boyle, "Hydra: a scalable proteomic search engine which utilizes the Hadoop distributed computing framework," BMC Bioinformatics, vol. 13, no. 1, p. 324, 2012.

[80] A. Csordas, D. Ovelleiro, R. Wang, J. M. Foster, D. Ríos, J. A. Vizcaíno and H. Hermjakob, "PRIDE: quality control in a proteomics data repository.," Database (Oxford)., vol. 2012, p. bas004, Jan. 2012.

[81] K. R. Coombes, S. Tsavachidis, J. S. Morris, K. A. Baggerly, M. C. Hung, and H. M. Kuerer, “Improved peak detection and quantification of mass spectrometry data acquired from surface-enhanced laser desorption and ionization by denoising spectra with the undecimated discrete wavelet transform,” Proteomics, vol. 5, no. 16, pp. 4107–4117, 2005.

[82] E. Lange, C. Gröpl, K. Reinert, O. Kohlbacher, and A. Hildebrandt, “High-accuracy peak picking of proteomics data using wavelet techniques.,” Pac. Symp. Biocomput., vol. 254, pp. 243–254, 2006.

[83] O. Kohlbacher, K. Reinert, C. Gropl, E. Lange, N. Pfeifer, O. Schulz-trieglaff and M. Sturm, “TOPP — the OpenMS proteomics pipeline,” Bioinformatics, vol. 23, pp. 191–197, 2006.

[84] M. Bellew, M. Coram, M. Fitzgibbon, M. Igra, T. Randolph, P. Wang, D. May, J. Eng, R. Fang, C. Lin, J. Chen, D. Goodlett, J. Whiteaker, A. Paulovich and M. McIntosh, “A suite of algorithms for the comprehensive analysis of complex protein mixtures using high-resolution LC-MS,” Bioinformatics, vol. 22, no. 15, pp. 1902–1909, 2006.

[85] F.F. Gonzalez-Galarza, C. Lawless, S.J. Hubbard, J. Fan, C. Bessant, H. Hermjakob, and A.R. Jones, “A critical appraisal of techniques, software packages, and standards for quantitative proteomic analysis.,” OMICS, vol. 16, no. 9, pp. 431–442, Sep. 2012.

[86] R. Kouzes, G. Anderson, and S. Elbert, “The changing paradigm of data-intensive computing,” Computer, pp. 26–34, 2009.

[87] Y. Mohammed, E. Mostovenko, A. Henneman, R. J. Marissen, A. M. Deelder, and M. Plamblad, “Cloud Parallel Processing of Tandem Mass Spectrometry-based Proteomics Data,” J. Proteome …, vol. 11, no. 8, pp. 5101–5108, 2012.

[88] P. Kasson, “Computational biology in the cloud: methods and new insights from computing at scale,” in Pacicfic Symposium on Biocomputing., 2013, pp. 451–453.

[89] D. Trudgian and H. Mirzaei, “Cloud CPFP: a shotgun proteomics data analysis pipeline using cloud and high performance computing,” J. Proteome Res., vol. 11, no. 10, pp. 6282–6290, 2012.

[90] T. Gunarathne, T.-L. Wu, J. Y. Choi, S.-H. Bae, and J. Qiu, “Cloud computing paradigms for pleasingly parallel biomedical applications,” Concurr. Comput. Pract. Exp., vol. 23, pp. 2338–2354, 2011.

[91] S. Bressan, A. Cuzzocrea, P. Karras, X. Lu, and S. H. Nobari, “An effective and efficient parallel approach for random graphs generation over GPUs,” J. Parallel Distrib. Comput., vol. 73, no. 3, pp. 303–316, 2012.

[92] R. Taylor, “An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics,” in 11th Annual Bioinformatics Open Source Conference, 2010, pp. 1–6.

[93] A. Matsunaga, M. Tsugawa, and J. Fortes, “Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications,” in Fourth IEEE International Conference on eScience, 2008, pp. 222–229.

[94] M. Koehler, Y. Kaniovskyi, S. Benkner, V. Egelhofer, and W. Weckworth, “A cloud framework for high throughput biological data processing,” in The International Symposium on Grids and Clouds and the Open Grid Forum, 2011.

[95] K. Verheggen, H. Barsnes, and L. Martens, “Distributed computing and data storage in proteomics : Many hands make light work, and a stronger memory,” Proteomics, vol. 14, pp. 367–377, 2014.

[96] D. T. Duncan, R. Craig, and A. J. Link, “Parallel tandem: a program for parallel processing of tandem mass spectra using PVM or MPI and X!Tandem.,” J. Proteome Res., vol. 4, no. 5, pp. 1842–7, 2005.

[97] L. Wang, W. Wang, H. Chi, and Y. Wu, “An efficient parallelization of phosphorylated peptide and protein identification,” Rapid Commun. Mass Spectrom., vol. 24, pp. 1791–1798, 2010.

[98] C. Hillman, Y. Ahmad, M. Whitehorn, and A. Cobley, “Near Real-Time Processing of Proteomics Data Using Hadoop,” Big Data, vol. 2, no. 1, pp. 44–49, 2014.

[99] S. Wandelt, A. Rheinländer, and M. Bux, “Data Management Challenges in Next Generation Sequencing,” Datenbank Spektrum, vol. 12, pp. 161–171, 2012.

[100] V. A. Fusaro, P. Patil, E. Gafni, D. P. Wall, and P. J. Tonellato, “Biomedical cloud computing with Amazon web services,” PLoS Comput. Biol., vol. 7, no. 8, pp. 1–6, 2011.

[101] W. Wu, H. Zhang, Z. Li, and Y. Mao, “Creating a Cloud-based Life Science Gateway,” in Seventh IEEE International Conference on eScience, 2011, pp. 55–61.

[102] L. Hodgkinson, J. Rosa, and E. Brewer, “Parallel software architecture for experimental workflows in computational biology on clouds,” in International Conference on Parallel Processing and Applied Mathematics, 2012, pp. 281–291.

[103] A. O’Driscoll, J. Daugelaite, and R. D. Sleator, “‘Big data’, Hadoop and cloud computing in genomics.,” J. Biomed. Inform., vol. 46, no. 5, pp. 774–81, Oct. 2013.

[104] Y. Mao, W. Wu, H. Zhang, and L. Luo, “GreenPipe: A Hadoop Based Workflow System on Energy-efficient Clouds,” in IEEE 26th International Parallel and Distributed Processing Symposium, 2012, pp. 2211–2219.

[105] H. A. Prahalad, A. Talukder, S. Pardeshi, S. Tamsekar, R. Hari Krishna, M. A. Chandrashekar, B. Niket, and S. Gandham, “Phoenix: System for implementing private and hybrid cloud for OMIC sciences applications,” in 2010 Seventh International Conference on Wireless and Optical Communications Networks - (WOCN), 2010, pp. 1–5.

[106] W. Wruck, “Data management strategies for multinational large-scale systems biology projects,” Brief. Bioinform., 2012.

[107] M. Niemenmaa, A. Kallio, A. Schumacher, P. Klemelä, E. Korpelainen, and K. Heljanko, “Hadoop-BAM: directly manipulating next generation sequencing data in the cloud.,” Bioinformatics, vol. 28, no. 6, pp. 876–7, Mar. 2012.

[108] P. Mell and T. Grance, “The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology,” 2011.

[109] B. D. Halligan, J. F. Geiger, A. K. Vallejos, A. S. Greene, and S. N. Twigger, “Low cost, scalable proteomics data analysis using Amazon’s cloud computing services and open source search algorithms.,” J. Proteome Res., vol. 8, no. 6, pp. 3148–3153, Jun. 2009.

[110] T. Muth, J. Peters, J. Blackburn, E. Rapp, and L. Martens, “ProteoCloud: a full-featured

open source proteomics cloud computing pipeline.," J. Proteomics, vol. 88, pp. 104–8, Aug. 2013.

[111] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in SIGCOMM'08, 2008, pp. 63–74.

[112] J. Graumann, R. Scheltema, Y. Zhang, J. Cox, and M. Mann, "A framework for intelligent data acquisition and real-time database searching for shotgun proteomics.," Mol. Cell. Proteomics, vol. 11, no. 3, p. M111.013185, Mar. 2012.

[113] C. Hillman, K. Petrie, A. Cobley, and M. Whitehorn, "Real-Time processing of proteomics data. The internet of things and the connected laboratory," in IEEE Big Data, 2016.

[114] L. Gatto, K. D. Hansen, M. R. Hoopmann, H. Hermjakob, O. Kohlbacher, and A. Beyer, "Testing and Validation of Computational Methods for Mass Spectrometry," J. Proteome Res., no. 15, pp. 809–814, 2016.

[115] M. Chambers, B. Maclean, R. Burke, D. Amide, D. Ruderman, S. Neumann, "A cross-platform toolkit for mass spectrometry and proteomics," Nat. Biotechnol., vol. 30, no. 10, pp. 918–920, 2012.

[116] ECMA-404, "The JSON Data Interchange Format," ECMA Int., vol. 1st Editio, no. October, p. 8, 2013.

[117] H. Rost, T. Sachsenberg, S. Aiche, C. Bielow, H. Weisser, F. Aicheler, S. Andreotti, H. Ehrlich, P. Gutenbrunner, E. Kenar, X. Liang, S. Nahnsen,L. Nilse, J. Pfeuffer, G. Rosenberger, M. Rubrik,U. Schmitt, J.Veit, M.Walzer,D. Wojnar, W. Wolski, O. Schilling, J. Choudhary, L. Malmstrom, R. Aebersold, K. Reinert, O. Kohlbacher, "OpenMS: a flexible open-source software platform for mass spectrometry data analysis," Nat Meth, vol. 13, no. 9, pp. 741–748, 2016.

[118] H. L. Rost, U. Schmitt, R. Aebersold, and L. Malmström, "pyOpenMS: A Python-based interface to the OpenMS mass-spectrometry algorithm library," Proteomics, vol. 14, no. 1, pp. 74–77, 2014.

[119] M. Sturm, A. Bertsch, C. Gröpl, A. Hildebrandt, R. Hussong, E. Lange,N.Pfeifer, O. Schulz-Trieglaff, A, Zerck, K, Reinert, O. Kohlbach, "OpenMS - an open-source software framework for mass spectrometry.," BMC Bioinformatics, vol. 9, no. 1, p. 163, 2008.

[120] L. Gatto and A. Christoforou, "Using R and Bioconductor for proteomics data analysis.," Biochim. Biophys. Acta, vol. 1844, no. 1 Pt A, pp. 42–51, Jan. 2014.

[121] J. Teleman, A. Chawade, M. Sandin, F. Levander, and J. Malmstrom, "Dinosaur: A Refined Open-Source Peptide MS Feature Detector," J. Proteome Res., vol. 15, no. 7, pp. 2143–2151, 2016.

[122] C. Hillman, "Investigation of the Extraction, Transformation and Loading of Mass Spectrometer RAW files.," University of Dundee, 2012.

[123] N. Kratzke, "Lightweight Virtualization Cluster How to Overcome Cloud Vendor Lock-In," J. Comput. Commun., vol. 2, no. 12, pp. 1–7, 2014.

[124] C. Boettiger, "An introduction to Docker for reproducible research," ACM SIGOPS Oper. Syst. Rev., vol. 49, no. 1, pp. 71–79, 2015.

[125] J. Schmidhuber, “Deep Learning in neural networks: An overview,” Neural Networks, vol. 61, pp. 85–117, 2015.

[126] P. Soni and N. S. Yadav, “Quantitative Analysis of Document Stored Databases,” Int. J. Comput. Appl., vol. 118, no. 20, pp. 37–41, 2015.

[127] D. M. German, “A Study of the Contributors of PostgreSQL,” Proc. 2006 Int. Work. Min. Softw. Repos., pp. 163–164, 2006.

[128] E. Friedman, P. Pawlowski, and J. Cieslewicz, “SQL/MapReduce: A Practical Approach to Self-Describing, Polymorphic, and Parallelizable User-Defined Functions,” Proc. VLDB Endow., vol. 2, no. 2, pp. 1402–1413, 2009.

[129] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable,” ACM Trans. Comput. Syst., vol. 26, no. 2, pp. 1–26, 2008.

[130] O. Marcu, A. Costan, G. Antoniu, M. S. Perez. "Spark versus Flink: Understanding Performance in Big Data Analytics Frameworks." Cluster 2016 - The IEEE 2016 International Conference on Cluster Computing, Sep 2016

[131] T. Ly, Y. Ahmad, A. Shlien, D. Soroka, A. Mills, M. J. Emanuele, M. R. Stratton and A. I. Lamond, “A proteomic chronology of gene expression through the cell cycle in human myeloid leukemia cells,” Elife, vol. 2014, no. 3, pp. 1–36, 2014.

[132] N. J. Gunther, “A New Interpretation of Amdahl’s Law and Geometric Scalability,” Performance Dynamics Company eprint arXiv:cs/0210017, pp 1-19, 2002.

[133] D. Fenyö and R. C. Beavis, “Informatics and data management in proteomics.,” Trends Biotechnol., vol. 20, no. 12, pp. S35–S38, 2002.

[134] Narwal, Abhikriti; Dhingra, Sunita. "A Systematic Review of Scheduling in Cloud Computing Framework" International Journal of Advanced Studies in Computers, Science and Engineering, Gothenburg Vol. 5, Iss. 7, pp 1-9, 2016

[135] A. Fallis, “MaxQuant paper supplement,” Nature, vol. 53, no. 9, pp. 1689–1699, 2013.

[136] M. R. Dale and C. Izurieta, “Impacts of design pattern decay on system quality,” Proc. 8th ACM/IEEE Int. Symp. Empir. Softw. Eng. Meas. - ESEM ’14, pp. 1–4, 2014.

[137] F. Long and Oracle, The CERT Oracle Secure Coding Standard for Java. Addison-Wesley, 2012.

[138] H. Choi and A. I. Nesvizhskii, “False Discovery Rates and Related Statistical Concepts in Mass Spectrometry-Based Proteomics,” J. Proteome Res., vol. 7, pp. 47–50, 2008.

[139] Thermo Scientific, http://planetorbitrap.com/q-exactive#tab:schematic, last accessed 12th February 2018.

[140] M. R.Head and M. Govindaraju "Parallel Processing of Large-Scale XMLBased Application Documents on Multi-Core Architectures with PiXiMaL." in IEEE Fourth International Conference on eScience, pp. 261–268, 2008.

[141] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale

Woodford. "Spanner: Google's Globally-Distributed Database". In Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation (OSDI'12). USENIX Association, Berkeley, CA, USA, 251-264, 2012.

[142] C. Zheng and L. Q. Zhou, "Principle Study and Usage of Hadoop Benchmark", Advanced Materials Research, Vols. 926-930, pp. 1988-1992, 2014.

[143] A. O'Driscoll, V. Belogrudov, J. Carroll, K. Kropp, P. Walsh, P. Ghazal, R. D. Sleator "HBLAST: Parallelised sequence similarity – A Hadoop MapReducable basic local alignment search tool", Journal of Biomedical Informatics, Vol. 54, pp 58-64, 2015.

[144] T. Jach, E. Magiera, W. Froelich, "Application of HADOOP to Store and Process Big Data Gathered from an Urban Water Distribution System", Procedia Engineering, Vol. 119, pp 1375-1380, 2015.

[145] K. Wiley, A. Connolly, S. Krughoff, J. Gardner, M. Balazinska, B. Howe, B, "Astronomical Image Processing with Hadoop", Astronomical Data Analysis Software and Systems XX. ASP Conference Proceedings, Vol. 442, p.93, 2011.

[146] J. M. C. Loaiza, G. Giuliani and G. Fiameni, "Big-Data in Climate Change Models — A Novel Approach with Hadoop MapReduce," 2017 International Conference on High Performance Computing & Simulation (HPCS), Genoa, pp. 45-50, 2017

[147] C. Yang, Z. He, W. Weichuan, "Comparison of public peak detection algorithms for MALDI mass spectrometry data analysis." *BMC Bioinformatics*, Vol 10, issue 1, pp. 1471-2105, 2009

[148] Röst HL, Schmitt U, Aebersold R, Malmström L (2015) Fast and Efficient XML Data Access for Next-Generation Mass Spectrometry. PLOS ONE 10(4): e0125108

[149] Compute Unified Device Architecture (CUDA) Programming Guide. http://developer.nvidia.com/object/cuda.html: NVIDIA, 2007

[150] A. Munshi, "The OpenCL specification," 2009 IEEE Hot Chips 21 Symposium (HCS), Stanford, CA, pp. 1-314. 2009

[151] R. Raina, A Madhavan, Anand, A. Y. Ng, "Large-scale Deep Unsupervised Learning Using Graphics Processors", Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, pp 873–880, 2009

[152] D-H Kim, "Evaluation Of The Performance Of GPU Global Memory Coalescing", Journal of Multidisciplinary Engineering Science and Technology (JMEST) Vol. 4 Issue 4, pp 7009-7013, April - 2017

[153] N. M. Amato and L. K. Dale, "Probabilistic roadmap methods are embarrassingly parallel," Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C), Detroit, MI, pp. 688-694 vol.1, 1999

[154] K. Xiao, F. Yu, H. Fang, B. Xue, Y. Liu, Z. Tian, "Accurate and Efficient Resolution Overlapping Isotopic Envelopes in Protein Tandem Mass Spectra." *Scientific Reports 5* (2015), 2018

APPENDIX A – SYSTEMS USED IN THE RESEARCH

1 HADOOP

Hadoop consists of HDFS (Hadoop Distributed File System) and MapReduce. HDFS is a distributed fault tolerant file system based on Google's GFS (Google File System). MapReduce, first proposed by engineers working for Google [5], is a parallel processing framework used to distribute tasks across clusters of computers. Hadoop is designed to run on clusters of commodity hardware and has built-in features to cope with hardware failure. For example, by default all data is replicated across three nodes which ensures data consistency even if two nodes fail. Several commercial distributions of Hadoop exist but this research focused on the open source implementation, available from the Apache Foundation.

Early Hadoop releases up to version number 2 had an architecture as described above, namely the HDFS file system, and the MapReduce programming framework. This environment runs on top of a system of slave nodes also called data nodes. Each data node runs Java daemon processes called the task tracker and job tracker that control data distribution and job execution. The master node (commonly called the name node) coordinates the data nodes and communicates with the daemon processes as data is loaded and accessed or as MapReduce jobs are run.

As data is loaded into HDFS, it is split into blocks and distributed across the data nodes. The default block size is 64Mb, and the default replication factor is three, meaning that all data is stored three times in the cluster. These values are configurable, but for the purposes of this research, defaults were used except where indicated. Figure 34 shows the basic setup of HDFS across the name and data nodes of a cluster: the data blocks can be seen as numbered boxes with the three replicates of each block distributed across the cluster.

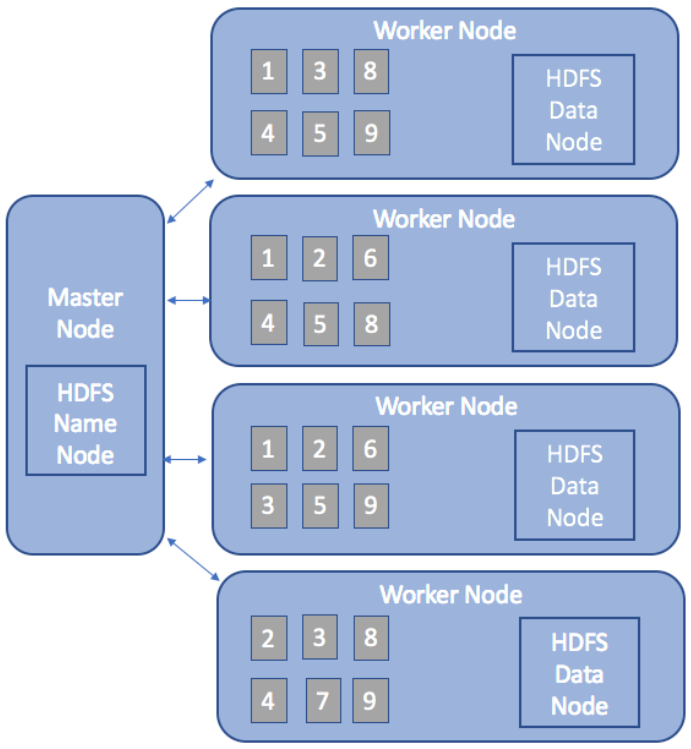


FIGURE 38 DISTRIBUTION OF REPLICATED DATA BLOCKS IN HDFS

The main method for loading files and submitting jobs to the cluster is a command line interface. A Java Application Programmers Interface (API) exists, which allows front-end tools to be built on top of Hadoop to abstract the command line away from end users. However, one objective of this research is to remove the need for users to explicitly run processing jobs and as such none of these front-end tools are used in this study. Being a file system HDFS is format-agnostic and allows data of any internal structure to be loaded and stored.

The earlier versions of Hadoop were dependent solely on the MapReduce framework for data processing. An eco-system has developed around the core Hadoop components to allow other programming languages to be used to process data. These include Hive, a SQL-like language developed by Facebook, and Pig, an abstracted high-level query language. These tools are interfaces to the MapReduce framework so when the abstracted language is used the code is converted into a MapReduce job which is submitted to the cluster in the background. This removes the complexity of writing MapReduce code in Java from the developer but can introduce a time delay into the process. With the release of Hadoop version 2, Hadoop was re-engineered to include the YARN (Yet Another Resource Negotiator) framework. YARN consists of a resource manager and node managers which replace the job tracker and task managers of version 1. An overview of the Hadoop cluster architecture can be seen in Figure 35.

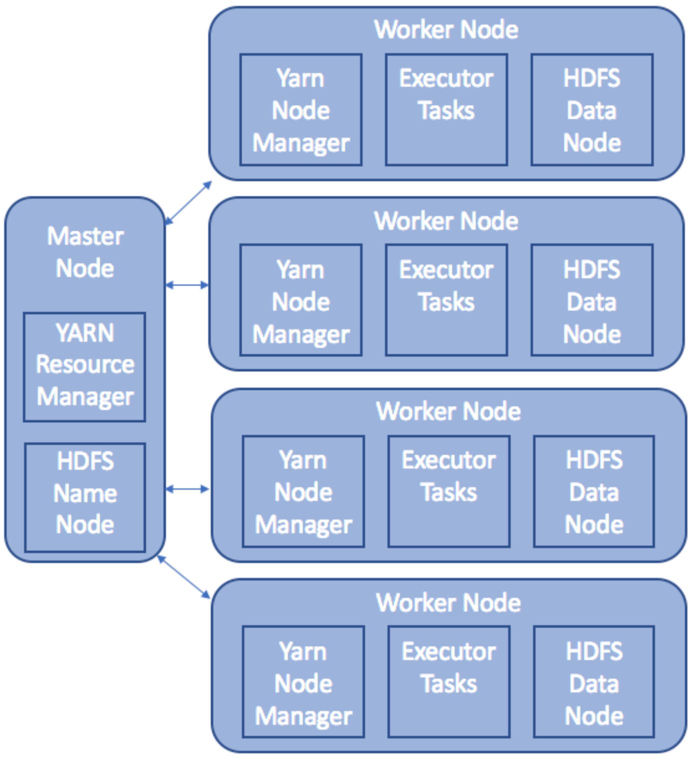


FIGURE 39 ARCHITECTURE OF A HADOOP CLUSTER

YARN is only concerned with resources such as CPU and memory and does not specify how tasks should be completed. The result of this is that MapReduce has become an application that can run on the YARN framework and Hadoop becomes a more general purpose parallel platform that can run other parallel processing jobs. For example, code written using older parallel frameworks such as MPI or BSP can now be run on an Hadoop cluster. The earlier versions of Hadoop that did not include the YARN framework are not used in practice now because they also lacked critical components necessary to recover the system in the event of a name node failure. Therefore, only version 2.4 of Hadoop is benchmarked here. Other changes were introduced in the version two release such as enhanced security and a backup name node, which mitigates the effect of having the name node as the single point of failure in the system but none of these enhancements is benchmarked or used in the testing.

2 CASSANDRA

Apache Cassandra belongs to a class of cluster-based systems known as “NoSQL”. The name stands for Not only SQL and refers to a class of databases that do rely on the standard SQL language for querying and managing data. Cassandra stores data in a table-like structure called a Column Family. The column family does contain rows and columns, but individual rows do not need to include the same number of columns as other rows in the same object. This column family structure means that a Cassandra database can be used to store data that contains a different number of attributes for each row. This capability contrasts with a relational database table where each row of a table contains the same number of columns. NoSQL databases commonly provide a query language for retrieving data that differs from the ANSI SQL used by relational databases. In the case of Cassandra, querying data is accomplished with the CQL language that is similar in structure and syntax to the SQL language but with a limited number of operators.

Cassandra provides high availability and partition tolerance in the event of a node and network failure. Blocks of data are written to nodes in the cluster using “eventual consistency”. Eventual consistency means that as data is written to nodes, it can be accessed before all the nodes report the successful update or write. This method can lead to a situation where queries return different results for the same data, depending on which node is queried. A level of tolerance for eventual consistency can be set in the configuration files to mitigate this behaviour.

The architecture of Cassandra is described as a ring where all nodes in the cluster are peers with no head or coordinator node, which makes Cassandra a very robust and resilient option for data storage and processing as there is no single point of failure. Data is distributed across the cluster either at random or by ordering the data in a certain way according to a user-provided key for each row in a column family.

Cassandra clusters are considered scalable with high performance [126] and are used in production environments at major organisations such as Netflix, CERN and Twitter. The Hadoop MapReduce framework can be configured to retrieve data from Cassandra column families, replacing HDFS as the means of data storage. The configuration that is usually deployed is to install Cassandra on the Hadoop data nodes. In this configuration, each Cassandra node will run a separate version of the Hadoop task tracker daemon, which is an efficient configuration as it minimises network traffic at the Map task level while utilising Cassandra's high-performance data store. See Figure 36 for how Cassandra fits into a Hadoop-based cluster. It is only necessary to change the input and output interfaces to allow MapReduce code to read from and write to Cassandra. Therefore, the actual feature detection algorithm code is unchanged from that used in Hadoop with HDFS for data storage.

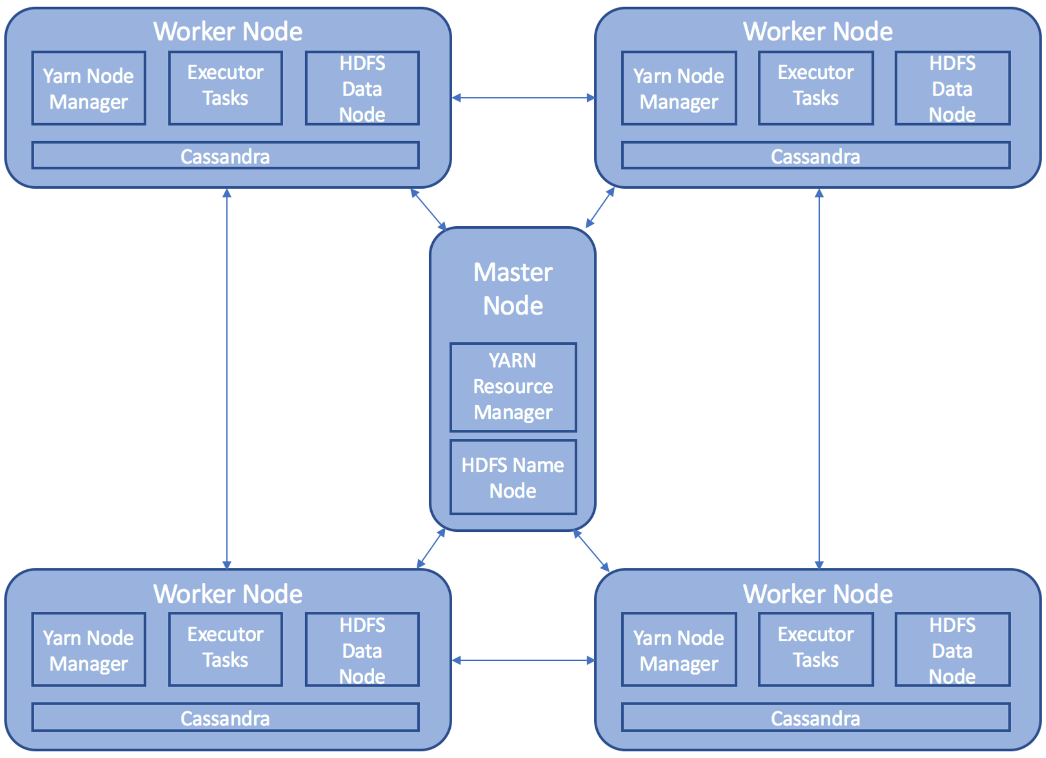


FIGURE 40 HADOOP ENVIRONMENT WITH CASSANDRA INSTALLED ON THE DATA NODES

Aster has been chosen as an example of a relational database. It is based on the widely used open source Postgres database [127] and shares a common SQL syntax with it. The Teradata Corporation acquired the technology in September 2011 and it is still known widely as Aster or Asterdata. Several aspects of the Aster database make it highly suitable for this research work. Firstly, it is an MPP system with an architecture comprising of worker nodes that carry out the processing and store data in a distributed fashion, plus a coordinator node. Conceptually the architecture is like the Hadoop architecture. As the coordinator node receives queries from clients it distributes the query out to the worker nodes which process the query where the data is located. Secondly, Aster contains the patented SQL-MapReduce framework that allows MapReduce code to execute in parallel using data stored in the relational database [128].

This combination of access methods allows a programmer to use the relevant parts of the SQL set-based language or Java procedural code as required. The Aster MapReduce framework operates in a similar fashion to the Hadoop MapReduce framework, with the major differences being in the input and output interfaces. Whereas Hadoop reads in and writes out through the passing of key-value pairs, Aster achieves these operations by reading data directly from relational tables and ensuring that the output from any query is also in the form of a relational table. The code written and tested for Hadoop can be compiled ready to run in the Aster SQL-MapReduce framework with minimal changes.

As Aster is based on an MPP version of the database software Postgres, it stores data in relational database tables. These tables can store data in standard formats such as, integers, strings and decimals. The tables are also able to store data in a Binary Large Object (BLOB) format; these BLOBs are stored as separate files outside the main database tables but are still accessible via standard SQL statements and the SQL-MR functions. To benchmark Aster, the mass spectrometer output was converted from Thermo Scientific RAW format to the SCMI format described in Section 5.3 of this chapter. The base64 strings holding the arrays of mass and intensity data could then be processed in a schema on read fashion using the

MapReduce algorithm. Figure 37 shows a basic view of the Aster Architecture, the queen node is the coordinator; it receives queries and directs them to the worker nodes, which are equivalent to Hadoop data nodes. Each worker node contains a mix of primary (P) and secondary (S) virtual workers. The virtual workers are a unit of CPU, memory and disk space. The secondary workers provide a backup in the event of failure, therefore by default all data is mirrored on an Aster system. Aster provides separate loader nodes dedicated for data loading tasks and backup nodes for use in the Backup and Recovery (BAR) process.

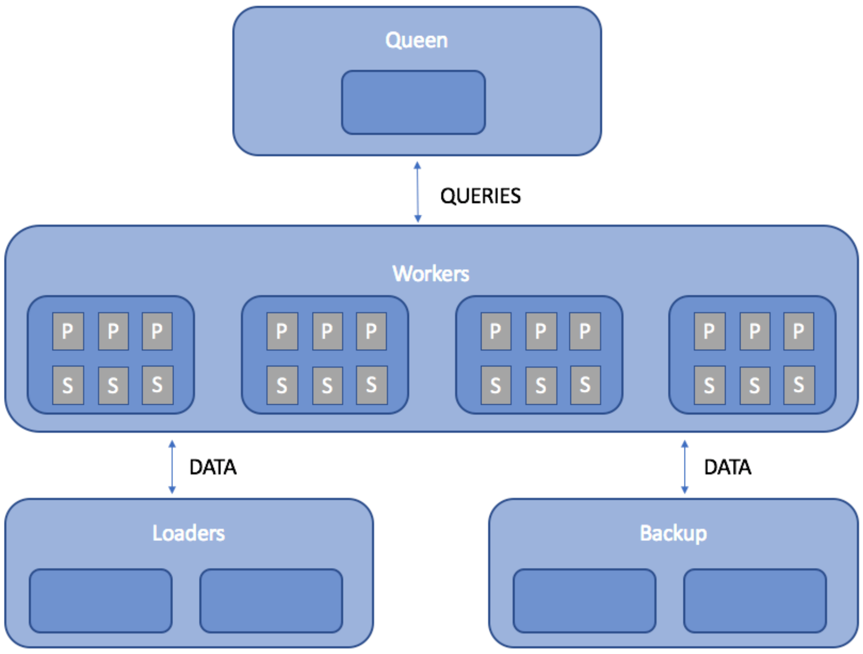


FIGURE 41 ASTER ARCHITECTURE

4 HBASE

HBase is part of the Apache Hadoop ecosystem and as such is usually found ready configured on most Hadoop clusters as it is part of the base install. Figure 38 shows the Hadoop cluster with HBase services and Zookeeper installed, Zookeeper is a cluster coordination tool used by HBase for synchronisation and to provide failover support. Google's Big Table system [129] first proposed in 2006 was the inspiration for developing HBase. It runs on top of Hadoop's HDFS file system and provides a column store in a non-relational format that is like the column families implemented by Cassandra. A fast, random access environment is achieved by the HBase implementation of table structures, as opposed to the simple file storage of HDFS. An HBase table has rows that are indexed by a row key and each row may contain a different number columns. Hadoop's MapReduce framework can read and write directly from and to HBase tables. To enable this, the Java MapReduce code must be changed to use the HBase interface instead of the HDFS text based input and output format.

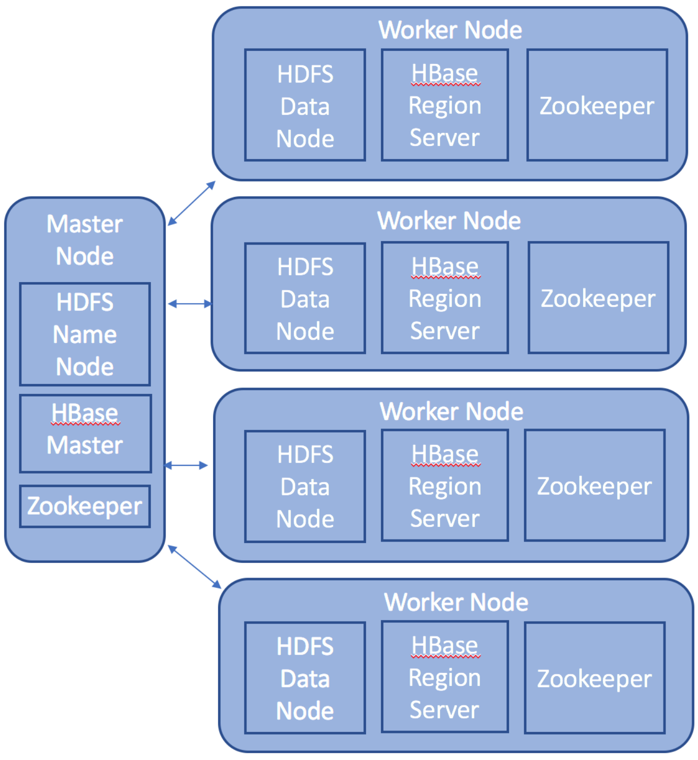


FIGURE 42 HADOOP CLUSTER INCLUDING THE HBASE SERVICES

HBase tables are automatically distributed across the cluster nodes (a process known as sharding) by grouping rows into clusters called regions. For this reason, in HBase terminology, the data nodes are called region servers. The default behaviour is to load all data into a single region on a single region server and then start to split it into multiple regions as the size of the table grows. It is possible to override this behaviour and supply information on how the data should be distributed. HBase provides a mechanism for this called pre-splitting using the RegionSplitter utility that takes the output from a plug-in algorithm. Several algorithms are available, and their effectiveness at minimising data skew is dependent on the distribution of the keys in a specific data set. A Java API is supplied for loading and extracting data which allow data to be stored either as text or as binary sequence files as with HDFS.

5 SPARK

Spark was initially developed at the University of Berkeley and is now an Apache open source project. Unlike other systems reviewed in this research project, Spark does not supply a native data repository. Instead, it reads data from sources held on Hadoop or Cassandra clusters into memory ready for processing. Resilient Distributed Datasets (RDD) that allow efficient data sharing in memory across computations form the basis of the Spark environment. Spark can run as a YARN application on Hadoop version 2 and can read from many formats used in clusters including HDFS text and sequence files, HBase and Cassandra. Spark employs a native processing framework that is similar in principle to the Hadoop MapReduce framework.

Spark jobs can be coded in a variety of programming languages including Java, Python and Scala. Spark also provides a SQL interface that allows a developer to query table-like data structures called data frames in the same program flow as Java and Scala. In this way, Spark is like the Aster platform allowing a mix of SQL and procedural code to be run on the data, although the implementation is very different. As the programming framework differs from MapReduce, there is some refactoring of code required before the mass spectrometer pre-processing code can run. The changes include altering the input and output interfaces and wrapping the map and reduce steps into Spark functions. More detail can be found in Chapter 6 where the batch-based process is described in detail. Spark also includes a stream processing interface; this allows data to be processed in real-time. Chapter 7 details the use of this stream processing interface and the results obtained from it. To run as a YARN application, the Spark slave services are installed on the data nodes of a Hadoop cluster with the master service installed on the name node. Figure 39 shows the Hadoop architecture updated to include the Spark services. This setup allows Spark to run in parallel on the cluster and access data in HDFS, Cassandra and HBase (if it is installed).

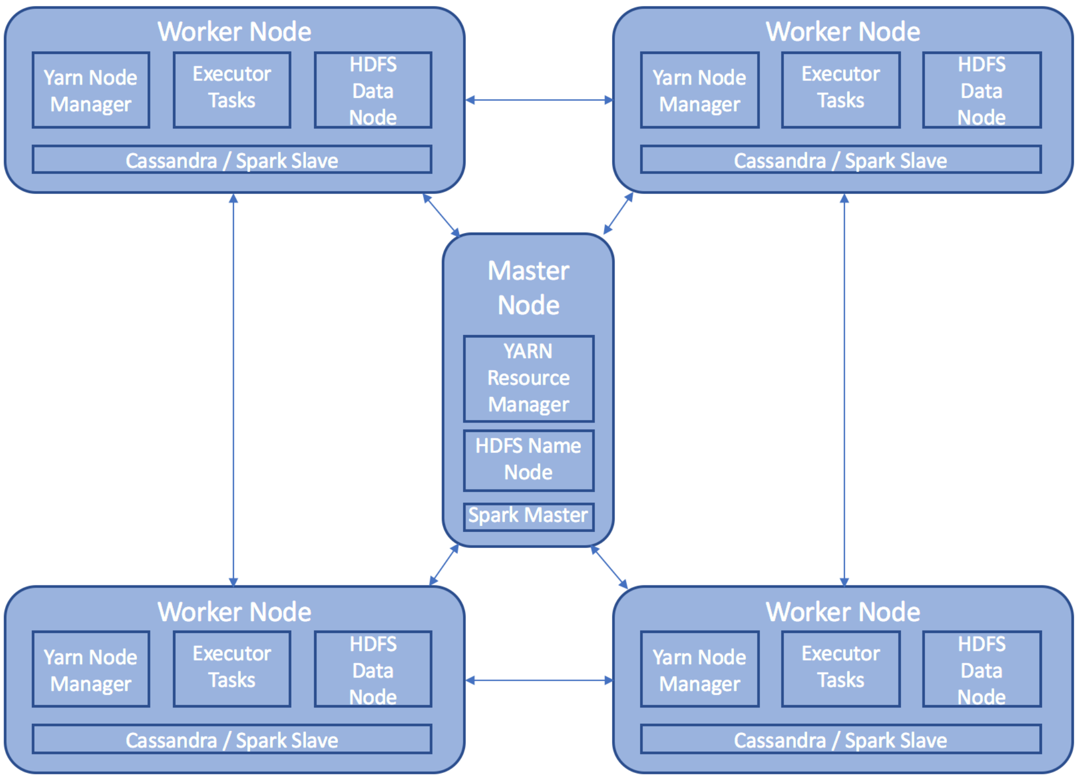


FIGURE 43 HADOOP ENVIRONMENT UPDATED TO INCLUDE APACHE SPARK

6 FLINK

Apache Flink is a relatively new software package, developed at the University of Berlin. Flink is gaining traction as an alternative to Spark and as a capable streaming data processing platform. Like Spark, Flink is an execution engine that can run as a YARN application on a Hadoop cluster. Although Flink can run as a standalone system, when operating on a Hadoop cluster YARN takes control of scheduling and resource allocation. The control from YARN is necessary if the cluster is a multi-tenanted environment with many users or systems accessing data and resources at the same time.

Superficially Flink and Spark are very similar; they are both capable of parallel processing, have Scala, Java and Python APIs and use in-memory processing for intermediate results. However, at the time of writing Flink has an advantage over Spark in stream processing. Whereas Spark processes streams of data in mini-batches, Flink uses a true streaming approach; this means that data is processed record by record as it arrives from the stream. This method of handling data streams is the reason for the inclusion of Flink as a platform to be researched during this work. The results of the streaming process testing and benchmarking can be found in Chapter 7 of this thesis. The differences in performance between Flink and Spark have been studied, and the results published [130]. A comprehensive set of results showed that in many cases for batch processing the elapsed times are similar between the two. However, some significant differences are noted in the method and efficiency of real-time stream processing, mainly due to Flink's pipelined execution, which allows tasks to be executed concurrently where possible. Figure 40 shows the Hadoop environment updated to include Flink; note the similarity with Figure 39 which shows the same environment but with Spark installed.

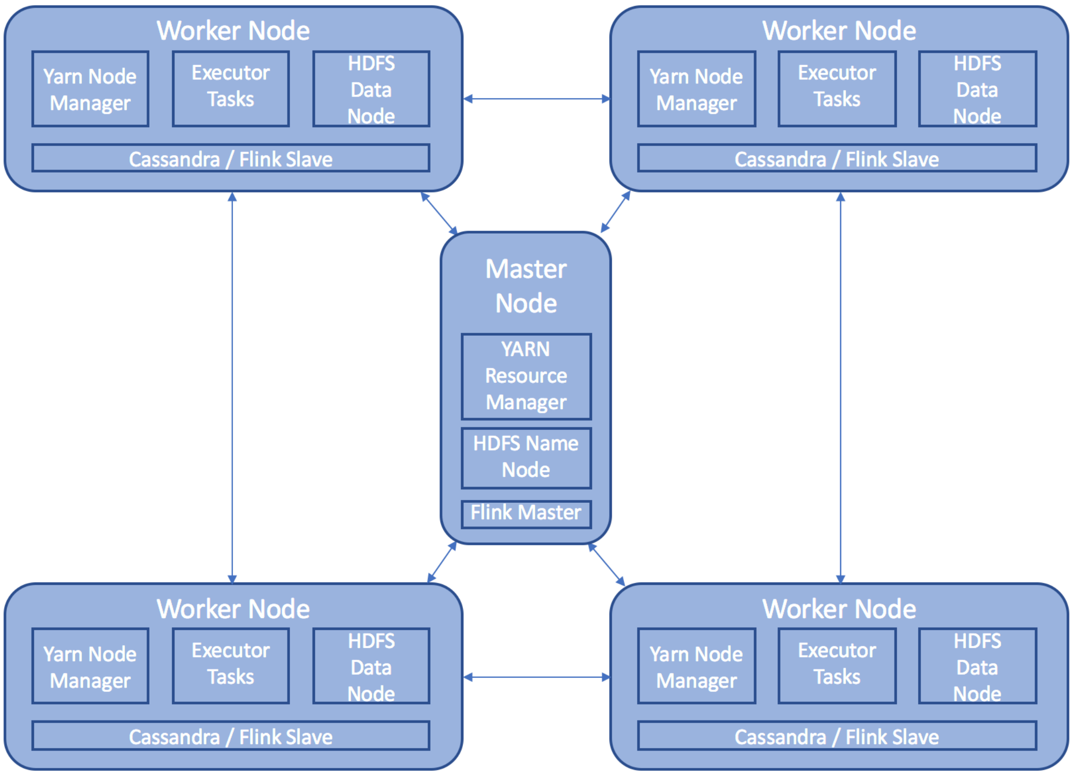


FIGURE 44 HADOOP ENVIRONMENT UPDATED TO INCLUDE APACHE FLINK

Unlike Flink and Spark, Kafka is not an execution engine. It does not process data and perform complex transformations on it. Instead, Kafka is a distributed publish-subscribe messaging system. During this research, Kafka is used to publish a stream of data to which the stream processing engines can subscribe. Once subscribed, data is received record by record in a constant flow. Using Kafka, it was possible to simulate the situation where mass spectrometers could stream out the results from an experiment as they were created instead of writing them to a file on disk that can only be accessed when the experiment is complete. Kafka was originally developed by the online social media company LinkedIn before being open-sourced and becoming an Apache Foundation project in 2011. In common with other publish-subscribe messaging systems, Kafka is organised by topics.

A topic is a collection of data that is published as a stream; for example, the data from a single proteomics experiment could be a topic, or a group of mass spectrometers could all publish their data to the same topic. In the Kafka environment records of data are called messages and are stored in a log for a set amount of time. This logging allows for recovery of data in a distributed environment in case of communication or server failure. The important point is that Kafka can be configured so that the streamed data is not lost and processing of the complete dataset can be continued later. Kafka can support many consumers so that in a complex environment many topics can be published and many subscribers can be consuming and processing the messages simultaneously. As part of this research, Kafka was used to simulate mass spectrometers streaming their data out directly, as opposed to the current situation where the output data is saved to disk. Apache Flink was then used to process the data in real-time. The cluster architecture including the Kafka component is show in Figure 41.

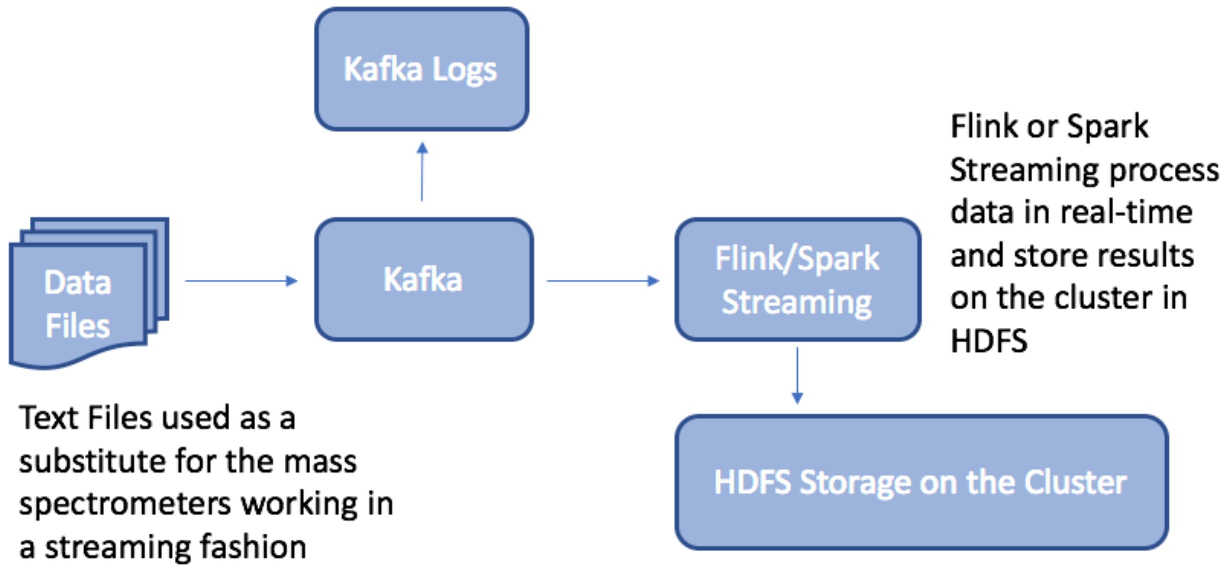

Kafka Logs
Data Files
Kafka
Flink/Spark Streaming
Flink or Spark Streaming process data in real-time and store results on the cluster in HDFS
Text Files used as a substitute for the mass spectrometers working in a streaming fashion
HDFS Storage on the Cluster

FIGURE 45 REAL-TIME ARCHITECTURE USING KAFKA

APPENDIX B – FEATURE DETECTION PROCESS

For the purposes of reproducibility, the setup and process for feature detection that was used in this research is presented in this appendix.

HARDWARE AND SOFTWARE

The cluster used for all the benchmarks was made up of one name node and three data nodes for all benchmarks except for Aster which used a similar cluster with one Queen and three worker nodes. Note that additional benchmarks were also run on a seven node Aster cluster. The specification for the types of node are as follows

Name/Queen Nodes
8-core Xeon E5-2670 @ 2.6GHz.
512Gb Ram

worker/Data Nodes
6-core Xeon E5-2670 @ 2GHz.
256Gb Ram

Software

Hadoop version	2.4
Cassandra Version	2.1.2
Aster Version	6.2
HBase Version	0.98.9
Spark Version	1.6.1
Flink Version	1.0
Kafka Version	2.10
Java Version	1_8.0_77SE

Code is available at the Github repository https://github.com/chillman99/phd
Data Is available online at this URL http://bit.ly/2BhqLdx

Loading and Parsing Files

From Local File System to Local File System:

```
     java -jar parsemzML.jar local <file>.mzml
<localdir>/<file>.scmi
```

Example:

```
     java -jar parsemzML.jar local phd/data/myfile.mzml
phd/data/myfile.scmi
```

From Local File System to Hadoop

```
     java -jar parsemzML.jar hdfs <file>.mzml
<remotedir>/<file>.scmi hdfs://<Hadoop master node IP> : <port>
```

Example:

```
     java -jar parsemzML.jar hdfs phd/data/myfile.mzml
/user/user1/scmifiles/myfile.scmi hdfs://192.168.100.10 9000
```

From Local File System to Cassandra

```
     java -jar parsemzML.jar Cassandra <file>.mzml <Cassandra
column family> <Cassandra IP address><Cassandra keyspace>
```

Example

```
     java -jar parsemzML.jar cassandra myfile.mzml scandata
192.168.100.10 phd
```

From Local File System to HBase

```
     java -jar parsemzML.jar hbase <file>.mzml <HBase table>
<HBase IP address> <port>
```

Example

```
     java -jar parsemzML.jar hbase myfile.mzml scandata
192.168.100.10 2181
```

Hadoop HDFS map tasks only (for testing)
```
hadoop jar ../code/PeakPick.jar hdfsmap phd/flat1000 phd/flatout
```

Hadoop HDFS
```
hadoop jar ../code/PeakPick.jar hdfs phd/flat1000 phd/flatout
```

Hadoop HBase
```
hadoop jar ../code/PeakPick.jar hbase node0
```

Hadoop HBase write results to HDFS (not used)
```
hadoop jar ../code/PeakPick.jar hbasehdfs node0 phd/flattestout13
```

Hadoop Cassandra
```
hadoop jar ../code/PeakPick.jar cassandra node0 phd/flattestout42
```

Flink
```
./flink run ../code/PeakPickFlink.jar phd/flat phd/flinkmap -c
PeakPickFlink -m node0 –p42
```

Spark
```
SPARK_JAR=/usr/local/spark/lib/spark-assembly-1.2.0-hadoop2.4.
0.jar HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
/usr/local/spark/bin/spark-submit --master yarn --deploy-mode
cluster --class peaks.PeakPickSpark ../code/PeakPickSpark.jar
"hdfs://node0:9000/user/user1/phd/flattest/100312_100.txt"
```

Aster
```
CREATE TABLE phd.peaks2d DISTRIBUTE BY HASH(pkey)
AS SELECT *
   FROM PeakPick2D(
      ON  (SELECT scan, mslvl, rettime, mzarray, intensityarray
           FROM scandata_100312
           WHERE mslvl = 1));


CREATE TABLE phd.peaks3D DISTRIBUTE BY HASH(charge) AS
SELECT charge, mass, intensity, rt
FROM PeakPick3D(
     ON PeakPick2D)
     Partition by newindex
     order by scan);
```

RUNNING THE STREAM CODE

Run Kafka to simulate the stream

```
java -cp  StreamScmiFile.jar StreamScmiFile 127.0.0.1:9092
127.0.0.1:2181 PhD/data/100312_100.scmi scmiStream 500
```

Run a console consumer to monitor output (optional for testing)

```
/Users/localadmin/Documents/working/kafka/kafka_2.10-0.8.2.1/b
in/kafka-console-consumer.sh  --zookeeper localhost:2181
--topic scmiStream
```

Run Flink code to process stream and create output (note the topic name scmiStream is hardcoded in the source code)

```
./flink run ../code/PeakPickFlink.jar -m node0 -p42
```

OUTPUT

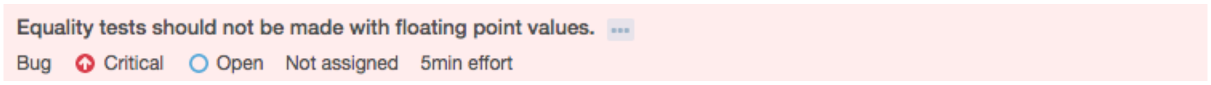
Depending on the mode of operation, the output will either be file in HDFS at a specified location, a file on the local system of hadoop, a table in HBase or a column family in Cassandra. The output is tab delimited if in a text format with each row describing a detected 3D peak and consisting of the following columns.

Column 1	charge
Column 2	mass
Column 3	intensity
Column 4	retention time

APPENDIX C – SONARQUBE CODE CHECKING OUTPUT EXAMPLE

The SonarQube software scans program code and produces a web-based output that lists the issues it finds in several categories:

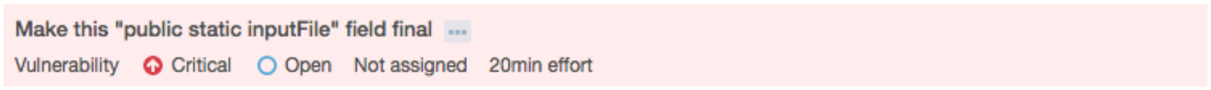
Bugs issues with the code that can prevent correct functioning



```
Equality tests should not be made with floating point values. ...
Bug   Critical   Open   Not assigned   5min effort
```

In the above example an attempt to directly compare floating point values has been detected, which can lead to an incorrect result.

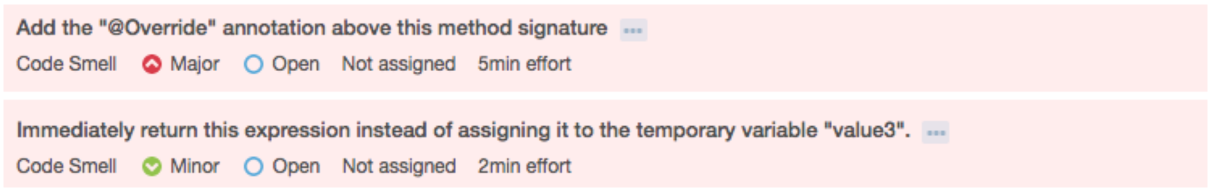
Vulnerabilities code that can cause a security issue



```
Make this "public static inputFile" field final ...
Vulnerability   Critical   Open   Not assigned   20min effort
```

This example shows a variable has been declared as public and static but not final, meaning that other code can modify its value.

Code Smells This refers to bad practice that does not cause incorrect results but could mean that code is difficult to understand or inefficient

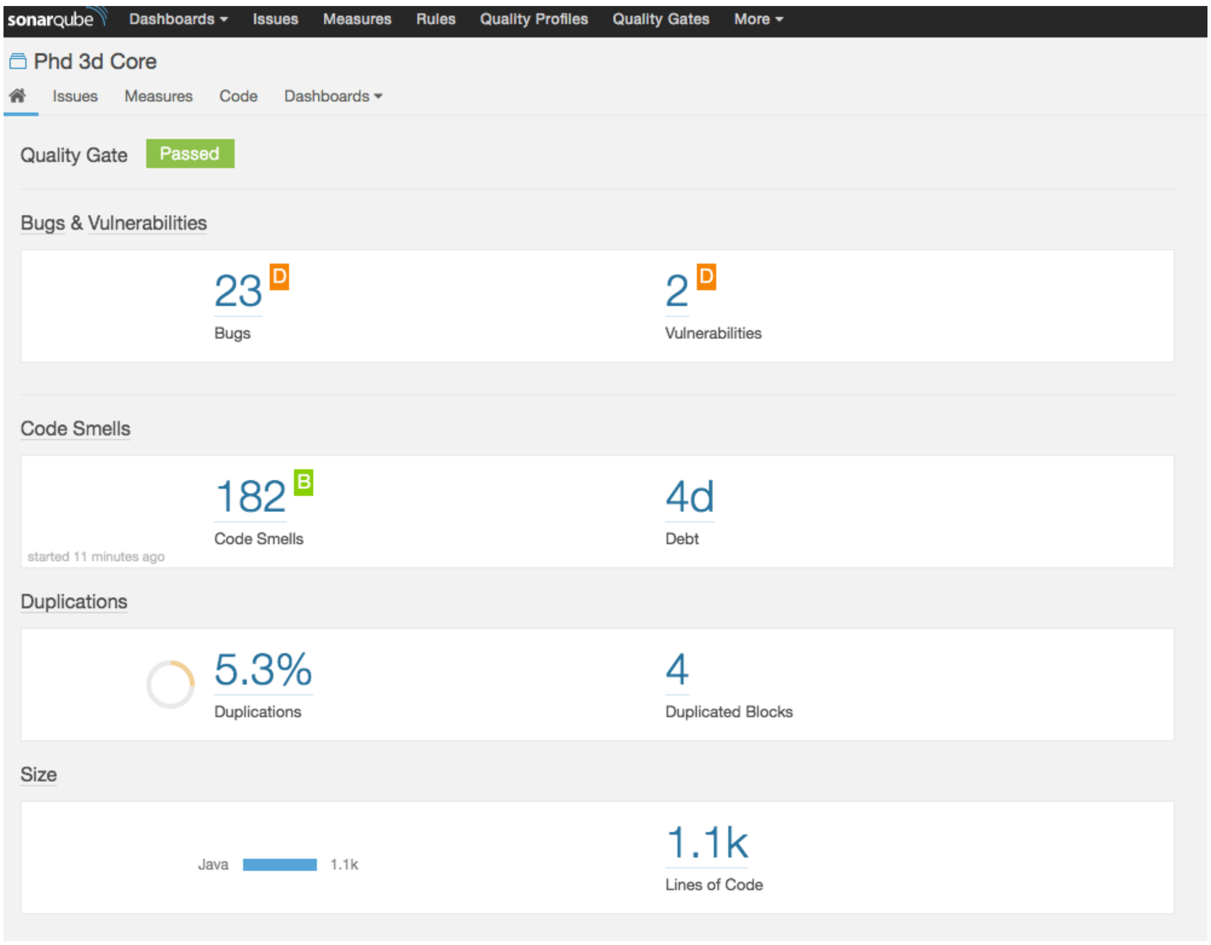


```
Add the "@Override" annotation above this method signature ...
Code Smell   Major   Open   Not assigned   5min effort

Immediately return this expression instead of assigning it to the temporary variable "value3". ...
Code Smell   Minor   Open   Not assigned   2min effort
```

Here, adding the @override annotation is standard practice and the 2nd example shows inefficient coding that introduces an unnecessary temporary variable.

The web console displays the complete analysis including the "Debt" measured in days. This is the estimated amount of time needed to fix all of the issues detected.



Quality Gate Passed

Bugs & Vulnerabilities

23 D Bugs

2 D Vulnerabilities

Code Smells

182 B Code Smells

4d Debt

started 11 minutes ago

Duplications

5.3% Duplications

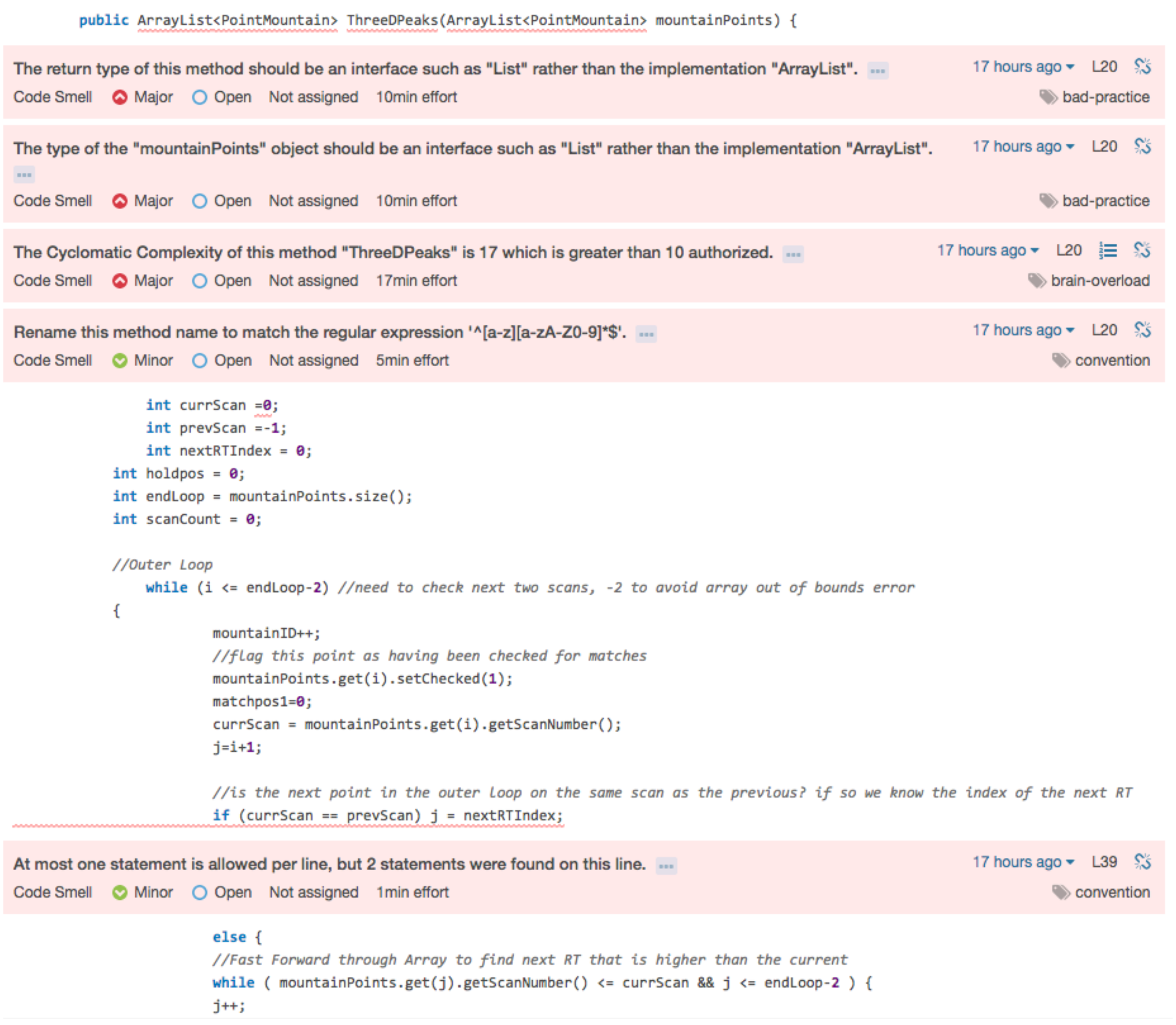
4 Duplicated Blocks

Size

Java 1.1k

1.1k Lines of Code

FIGURE 46 SONARQUBE DASHBOARD



```
        public ArrayList<PointMountain> ThreeDPeaks(ArrayList<PointMountain> mountainPoints) {
```

The return type of this method should be an interface such as "List" rather than the implementation "ArrayList". 17 hours ago L20
Code Smell Major Open Not assigned 10min effort bad-practice

The type of the "mountainPoints" object should be an interface such as "List" rather than the implementation "ArrayList". 17 hours ago L20
Code Smell Major Open Not assigned 10min effort bad-practice

The Cyclomatic Complexity of this method "ThreeDPeaks" is 17 which is greater than 10 authorized. 17 hours ago L20
Code Smell Major Open Not assigned 17min effort brain-overload

Rename this method name to match the regular expression '^[a-z][a-zA-Z0-9]*$'. 17 hours ago L20
Code Smell Minor Open Not assigned 5min effort convention

```
            int currScan =0;
            int prevScan =-1;
            int nextRTIndex = 0;
        int holdpos = 0;
        int endLoop = mountainPoints.size();
        int scanCount = 0;

        //Outer Loop
            while (i <= endLoop-2) //need to check next two scans, -2 to avoid array out of bounds error
        {
                    mountainID++;
                    //flag this point as having been checked for matches
                    mountainPoints.get(i).setChecked(1);
                    matchpos1=0;
                    currScan = mountainPoints.get(i).getScanNumber();
                    j=i+1;

                    //is the next point in the outer loop on the same scan as the previous? if so we know the index of the next RT
                    if (currScan == prevScan) j = nextRTIndex;
```

At most one statement is allowed per line, but 2 statements were found on this line. 17 hours ago L39
Code Smell Minor Open Not assigned 1min effort convention

```
                    else {
                    //Fast Forward through Array to find next RT that is higher than the current
                    while ( mountainPoints.get(j).getScanNumber() <= currScan && j <= endLoop-2 ) {
                    j++;
```

FIGURE 47 SONARQUBE CODE DETAIL

Drilling into the issues allows the users to see the actual code and the issues that need to be fixed. The interface also displays a categorisation of each issues such as “convention”, “bad practice”, “brain-overload” which is a synonym for complexity in the code.

APPENDIX D – DATA STORAGE SCHEMAS

This appendix details the Data Schemas used in the various storage systems that were used in this research

Aster

As Aster is a relational database based on the Postgres system, the Aster schema consisted of a relational table mirroring the format of the SCMI text file. The SQL Data Definition Language (DDL) is detailed below:

```
 CREATE TABLE scandata(
  fileName varchar,
  scan int,
  mslvl int,
  retTime double,
  precursorIonMz double,
  precursorIonIntensity double,
  precursorIonCharge int,
  mzArray varchar,
  intensityArray varchar)
  DISTRIBUTE BY HASH(scan);
```

The Aster DDL syntax is mostly ANSI standard SQL with the addition of the DISTRIBUTE BY clause. This clause specifies how data is distributed between the nodes of a cluster. In this case, the HASH(scan) clause specifies that a Java hash code generated from the scan number is used; in other words the scans are distributed evenly around the cluster.

Cassandra

The Cassandra syntax for creating a column family is similar to the SQL syntax.

```
CREATE KEYSPACE phd WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 3 };

USE phd;
```

```
CREATE TABLE scandata(
    fileName varchar,
    scan int,
    mslvl int,
    retTime double,
    precursorIonMz double,
    precursorIonIntensity double,
    precursorIonCharge int,
    mzArray varchar,
    intensityArray varchar,
    PRIMARY KEY (scan, fileName));
```

Here a Keyspace is like a Schema in a relational database in that it specifies an area of the system where related tables can be referenced. Cassandra handles the distribution of the data automatically as the data is loaded.

HBase

HBase uses a different syntax in that the column families are specified but not the individual columns:

```
  create 'scandata', 'meta', 'array', 'precursor'
```

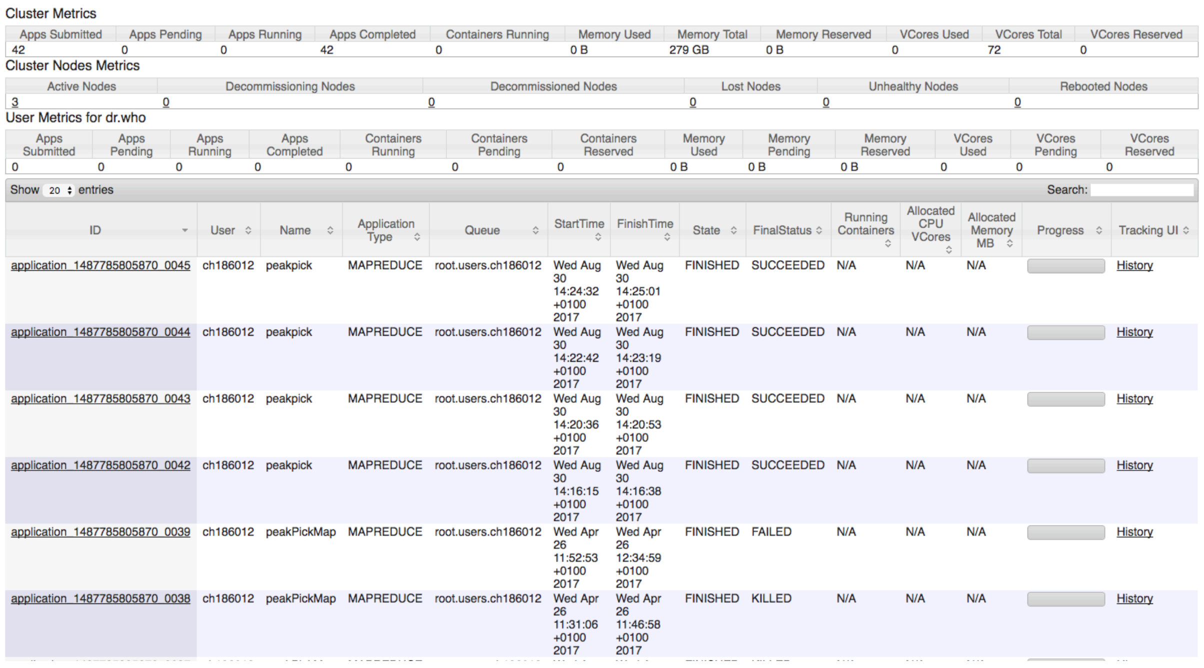
This statement means that a table called scandata is created with three sections or column families. The column names themselves are referenced in the load and query statements. The fields from the source file are arranged in the column families as follows:

```
- Meta
      Scan
      Mslvl
      rt
- Array
      Mzarray
      Intensityarray
- Precursor
      PrecursorIonMz
      PrecursorIonIntensity
      PrecursorIonCharge
```

APPENDIX E – SYSTEM CONSOLES

This appendix shows the browser-based consoles used to monitor and control the processing frameworks used in this research.

Figure 48 shows the Hadoop web console, this view shows basic information on the jobs that have run, the job names are hyperlinks that lead to detailed information child pages.



Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
42	0	0	42	0	0 B	279 GB	0 B	0	72	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
3	0	0	0	0	0

User Metrics for dr.who

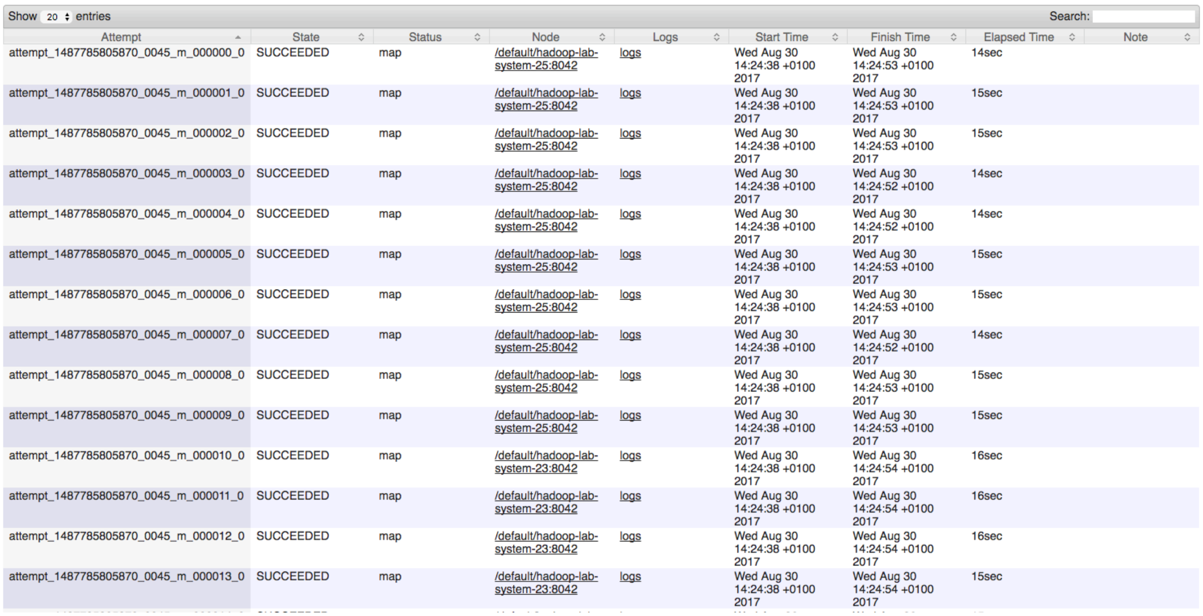
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Containers Pending	Containers Reserved	Memory Used	Memory Pending	Memory Reserved	VCores Used	VCores Pending	VCores Reserved
0	0	0	0	0	0	0	0 B	0 B	0 B	0	0	0

Show 20 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Progress	Tracking UI
application_1487785805870_0045	ch186012	peakpick	MAPREDUCE	root.users.ch186012	Wed Aug 30 14:24:32 +0100 2017	Wed Aug 30 14:25:01 +0100 2017	FINISHED	SUCCEEDED	N/A	N/A	N/A		History
application_1487785805870_0044	ch186012	peakpick	MAPREDUCE	root.users.ch186012	Wed Aug 30 14:22:42 +0100 2017	Wed Aug 30 14:23:19 +0100 2017	FINISHED	SUCCEEDED	N/A	N/A	N/A		History
application_1487785805870_0043	ch186012	peakpick	MAPREDUCE	root.users.ch186012	Wed Aug 30 14:20:36 +0100 2017	Wed Aug 30 14:20:53 +0100 2017	FINISHED	SUCCEEDED	N/A	N/A	N/A		History
application_1487785805870_0042	ch186012	peakpick	MAPREDUCE	root.users.ch186012	Wed Aug 30 14:16:15 +0100 2017	Wed Aug 30 14:16:38 +0100 2017	FINISHED	SUCCEEDED	N/A	N/A	N/A		History
application_1487785805870_0039	ch186012	peakPickMap	MAPREDUCE	root.users.ch186012	Wed Apr 26 11:52:53 +0100 2017	Wed Apr 26 12:34:59 +0100 2017	FINISHED	FAILED	N/A	N/A	N/A		History
application_1487785805870_0038	ch186012	peakPickMap	MAPREDUCE	root.users.ch186012	Wed Apr 26 11:31:06 +0100 2017	Wed Apr 26 11:46:58 +0100 2017	FINISHED	KILLED	N/A	N/A	N/A		History

FIGURE 48 HADOOP WEB CONSOLE

Figure 49 shows the detailed information of a job, listing the individual map tasks and their execution times, a similar view is available for the reduce tasks.

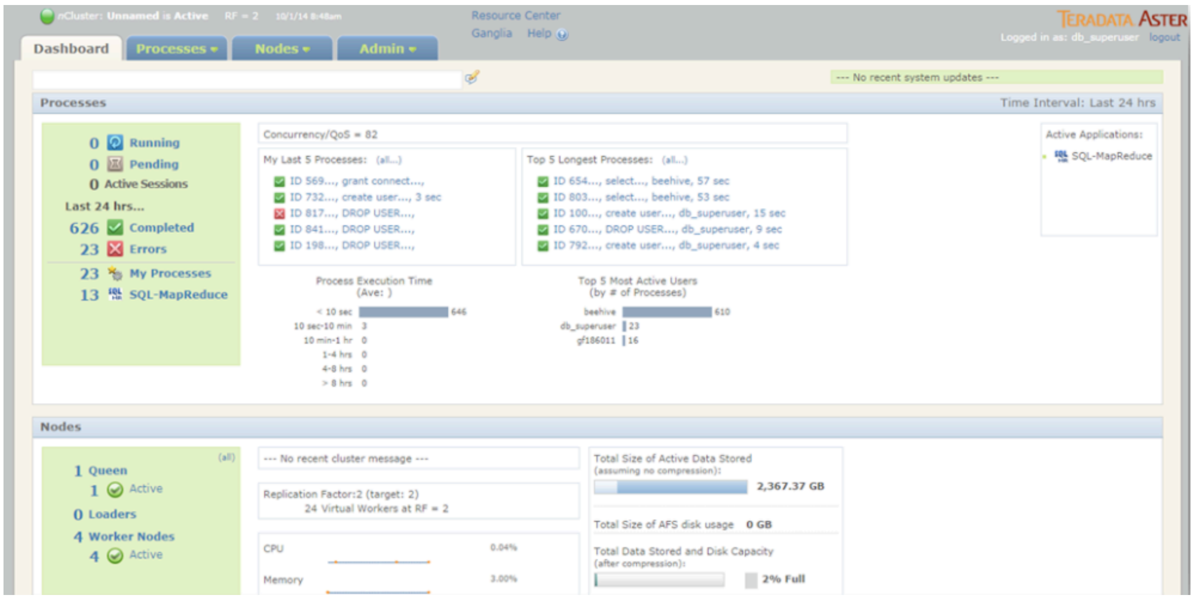


Show 20 entries Search:

Attempt	State	Status	Node	Logs	Start Time	Finish Time	Elapsed Time	Note
attempt_1487785805870_0045_m_000000_0	SUCCEEDED	map	/default/hadoop-lab-system-25:8042	logs	Wed Aug 30 14:24:38 +0100 2017	Wed Aug 30 14:24:53 +0100 2017	14sec	
attempt_1487785805870_0045_m_000001_0	SUCCEEDED	map	/default/hadoop-lab-system-25:8042	logs	Wed Aug 30 14:24:38 +0100 2017	Wed Aug 30 14:24:53 +0100 2017	15sec	
attempt_1487785805870_0045_m_000002_0	SUCCEEDED	map	/default/hadoop-lab-system-25:8042	logs	Wed Aug 30 14:24:38 +0100 2017	Wed Aug 30 14:24:53 +0100 2017	15sec	
attempt_1487785805870_0045_m_000003_0	SUCCEEDED	map	/default/hadoop-lab-system-25:8042	logs	Wed Aug 30 14:24:38 +0100 2017	Wed Aug 30 14:24:52 +0100 2017	14sec	
attempt_1487785805870_0045_m_000004_0	SUCCEEDED	map	/default/hadoop-lab-system-25:8042	logs	Wed Aug 30 14:24:38 +0100 2017	Wed Aug 30 14:24:52 +0100 2017	14sec	
attempt_1487785805870_0045_m_000005_0	SUCCEEDED	map	/default/hadoop-lab-system-25:8042	logs	Wed Aug 30 14:24:38 +0100 2017	Wed Aug 30 14:24:53 +0100 2017	15sec	
attempt_1487785805870_0045_m_000006_0	SUCCEEDED	map	/default/hadoop-lab-system-25:8042	logs	Wed Aug 30 14:24:38 +0100 2017	Wed Aug 30 14:24:53 +0100 2017	15sec	
attempt_1487785805870_0045_m_000007_0	SUCCEEDED	map	/default/hadoop-lab-system-25:8042	logs	Wed Aug 30 14:24:38 +0100 2017	Wed Aug 30 14:24:52 +0100 2017	14sec	
attempt_1487785805870_0045_m_000008_0	SUCCEEDED	map	/default/hadoop-lab-system-25:8042	logs	Wed Aug 30 14:24:38 +0100 2017	Wed Aug 30 14:24:53 +0100 2017	15sec	
attempt_1487785805870_0045_m_000009_0	SUCCEEDED	map	/default/hadoop-lab-system-25:8042	logs	Wed Aug 30 14:24:38 +0100 2017	Wed Aug 30 14:24:53 +0100 2017	15sec	
attempt_1487785805870_0045_m_000010_0	SUCCEEDED	map	/default/hadoop-lab-system-23:8042	logs	Wed Aug 30 14:24:38 +0100 2017	Wed Aug 30 14:24:54 +0100 2017	16sec	
attempt_1487785805870_0045_m_000011_0	SUCCEEDED	map	/default/hadoop-lab-system-23:8042	logs	Wed Aug 30 14:24:38 +0100 2017	Wed Aug 30 14:24:54 +0100 2017	16sec	
attempt_1487785805870_0045_m_000012_0	SUCCEEDED	map	/default/hadoop-lab-system-23:8042	logs	Wed Aug 30 14:24:38 +0100 2017	Wed Aug 30 14:24:54 +0100 2017	16sec	
attempt_1487785805870_0045_m_000013_0	SUCCEEDED	map	/default/hadoop-lab-system-23:8042	logs	Wed Aug 30 14:24:38 +0100 2017	Wed Aug 30 14:24:54 +0100 2017	15sec	

FIGURE 49 DETAILED INFORMATION ON A MAPREDUCE JOB

Figure 50 shows the management console for an Aster system, the landing page displays general system information such as current load, disk space usage and the last processes that ran. Figure 51 shows a Job detail page which displays the execution time and the SQL statement that was executed.



nCluster: Unnamed is Active RF = 2 10/1/14 8:48am

Resource Center
Ganglia Help

TERADATA ASTER
Logged in as: db_superuser logout

Dashboard | Processes | Nodes | Admin

--- No recent system updates ---

Processes — Time Interval: Last 24 hrs

0 Running
0 Pending
0 Active Sessions
Last 24 hrs...
626 Completed
23 Errors
23 My Processes
13 SQL-MapReduce

Concurrency/QoS = 82

My Last 5 Processes: (all...)
- ID 569..., grant connect...,
- ID 732..., create user..., 3 sec
- ID 817..., DROP USER...,
- ID 841..., DROP USER...,
- ID 198..., DROP USER...,

Top 5 Longest Processes: (all...)
- ID 654..., select..., beehive, 57 sec
- ID 803..., select..., beehive, 53 sec
- ID 100..., create user..., db_superuser, 15 sec
- ID 670..., DROP USER..., db_superuser, 9 sec
- ID 792..., create user..., db_superuser, 4 sec

Active Applications:
- SQL-MapReduce

Process Execution Time (Ave:)

< 10 sec	646
10 sec-10 min	3
10 min-1 hr	0
1-4 hrs	0
4-8 hrs	0
> 8 hrs	0

Top 5 Most Active Users (by # of Processes)

beehive	610
db_superuser	23
gf186011	16

Nodes

(all)
1 Queen
1 Active
0 Loaders
4 Worker Nodes
4 Active

--- No recent cluster message ---

Replication Factor:2 (target: 2)
24 Virtual Workers at RF = 2

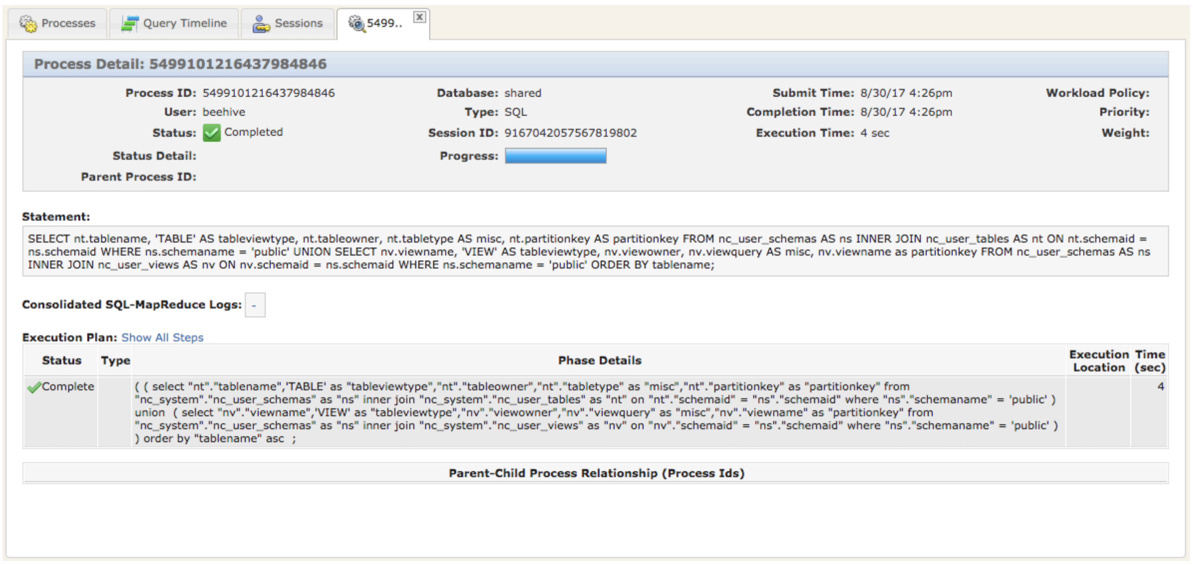
CPU 0.04%
Memory 3.00%

Total Size of Active Data Stored (assuming no compression): 2,367.37 GB

Total Size of AFS disk usage 0 GB

Total Data Stored and Disk Capacity (after compression): 2% Full

FIGURE 50 ASTER MANAGEMENT CONSOLE



Process Detail: 5499101216437984846

Process ID:	5499101216437984846	**Database:**	shared	**Submit Time:**	8/30/17 4:26pm	**Workload Policy:**	
User:	beehive	**Type:**	SQL	**Completion Time:**	8/30/17 4:26pm	**Priority:**	
Status:	✓ Completed	**Session ID:**	9167042057567819802	**Execution Time:**	4 sec	**Weight:**	
Status Detail:		**Progress:**					
Parent Process ID:							

Statement:

```
SELECT nt.tablename, 'TABLE' AS tableviewtype, nt.tableowner, nt.tabletype AS misc, nt.partitionkey AS partitionkey FROM nc_user_schemas AS ns INNER JOIN nc_user_tables AS nt ON nt.schemaid =
ns.schemaid WHERE ns.schemaname = 'public' UNION SELECT nv.viewname, 'VIEW' AS tableviewtype, nv.viewowner, nv.viewquery AS misc, nv.viewname as partitionkey FROM nc_user_schemas AS ns
INNER JOIN nc_user_views AS nv ON nv.schemaid = ns.schemaid WHERE ns.schemaname = 'public' ORDER BY tablename;
```

Consolidated SQL-MapReduce Logs: -

Execution Plan: Show All Steps

Status	Type	Phase Details	Execution Location	Time (sec)
✓Complete		((select "nt"."tablename",'TABLE' as "tableviewtype","nt"."tableowner","nt"."tabletype" as "misc","nt"."partitionkey" as "partitionkey" from "nc_system"."nc_user_schemas" as "ns" inner join "nc_system"."nc_user_tables" as "nt" on "nt"."schemaid" = "ns"."schemaid" where "ns"."schemaname" = 'public') union (select "nv"."viewname",'VIEW' as "tableviewtype","nv"."viewowner","nv"."viewquery" as "misc","nv"."viewname" as "partitionkey" from "nc_system"."nc_user_schemas" as "ns" inner join "nc_system"."nc_user_views" as "nv" on "nv"."schemaid" = "ns"."schemaid" where "ns"."schemaname" = 'public')) order by "tablename" asc ;		4

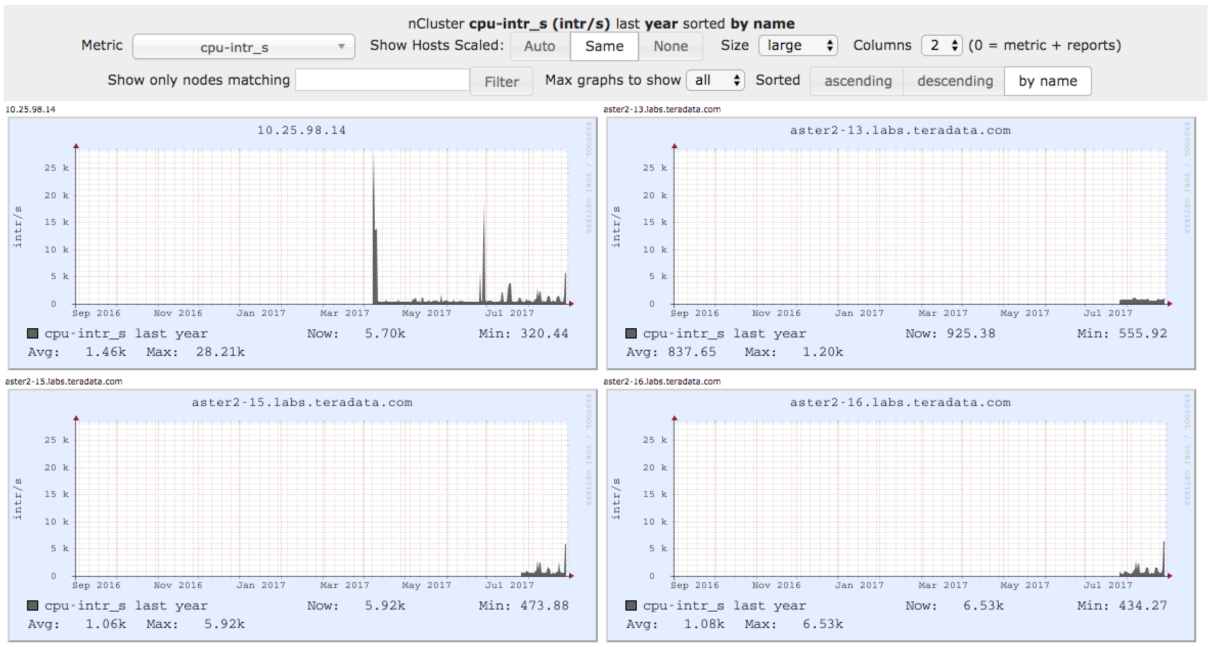
Parent-Child Process Relationship (Process Ids)

FIGURE 51 ASTER JOB PAGE

Both the Aster and the Hadoop systems include a tool called Ganglia which is an add-on system monitoring tool. Ganglia allows a finer degree of control than the total run time when investigating performance on a distributed system. Using Ganglia, it is possible to view individual nodes and CPUs within nodes to see task completion times, memory usage and disk access.

Figure 52 is an example report generated from Ganglia, it is possible to select from many different system metrics and display them per node in the cluster in this format. This allows a fine degree of system monitoring and troubleshooting.

Note that Spark and Flink also supply web based monitoring consoles, which show similar information to those of Hadoop and Aster: these are not shown here.



nCluster **cpu-intr_s (intr/s)** last **year** sorted **by name**

Metric: cpu-intr_s Show Hosts Scaled: Auto [✓ Same] None Size: large Columns: 2 (0 = metric + reports)

Show only nodes matching: Filter Max graphs to show: all Sorted: ascending descending [✓ by name]

10.25.98.14

```
■ cpu-intr_s last year       Now:    5.70k       Min: 320.44
Avg:    1.46k   Max:  28.21k
```

aster2-13.labs.teradata.com

```
■ cpu-intr_s last year       Now: 925.38         Min: 555.92
Avg: 837.65     Max:    1.20k
```

aster2-15.labs.teradata.com

```
■ cpu-intr_s last year       Now:    5.92k       Min: 473.88
Avg:    1.06k   Max:    5.92k
```

aster2-16.labs.teradata.com

```
■ cpu-intr_s last year       Now:    6.53k       Min: 434.27
Avg:    1.08k   Max:    6.53k
```

FIGURE 52 GANGLIA REPORT