2015

Building a Scalable Global Data Processing Pipeline for Large Astronomical Photometric Datasets

Paul F. Doyle
Technological University Dublin

Building a scalable global data processing pipeline for large astronomical photometric datasets

by

Paul F. Doyle MSc. BSc.

Supervised by: Dr. Fredrick Mtenzi

Dr. Niall Smith

Professor Brendan O'Shea

School of Computing

Dublin Institute of Technology

A thesis submitted for the degree of

Doctor of Philosophy

January 2015

Declaration

I certify that this thesis which I now submit for examination for the award of Doctor of Philosophy, is entirely my own work and has not been taken from the work of others save, and to the extent that such work has been cited and acknowledged within the text of my work.

This thesis was prepared according to the regulations for postgraduate study by research of the Dublin Institute of Technology and has not been submitted in whole or in part for an award in any other third level institution.

The work reported on in this thesis conforms to the principles and requirements of the DIT's guidelines for ethics in research.

DIT has permission to keep, lend or copy this thesis in whole or in part, on condition that any such use of the material of the thesis be duly acknowledged.

Signature _____ Date _____

Acknowledgements

The completion of this thesis has been possible through the support of my family, friends and colleagues who have helped and encouraged me over the last four years. I would like to acknowledge the support and guidance of my supervisors who remained ever confident that this journey would be enlightening and fulfilling; my children Connor, Oisín and Cillian, who shared their hugs of encouragement, and my wife Orla who gave me understanding, space and time to finish, which was no small sacrifice. To my colleagues and friends who generously gave their time and opinions I can assure you that this thesis was very much better for your contribution. Special thanks to my friends and colleagues at HEAnet, the Blackrock Castle Observatory, the Cork Institute of Technology, and the Institute of Technology Tallaght who provided services and support which were fundamental to the research performed within this thesis. This research was also supported in part by Amazon Web Services Ireland who provided extensive online virtual infrastructure resources. Finally I would like to express my deepest thanks to my parents who first set me on the path to a higher education. I will endeavour to pass the same values on to my own children.

Abstract

Astronomical photometry is the science of measuring the flux of a celestial object. Since its introduction in the 1970s the CCD has been the principle method of measuring flux to calculate the apparent magnitude of an object. Each CCD image taken must go through a process of cleaning and calibration prior to its use. As the number of research telescopes increases the overall computing resources required for image processing also increases. As data archives increase in size to Petabytes, the data processing challenge requires image processing approaches to evolve to continue to exceed the growing data capture rate.

Existing processing techniques are primarily sequential in nature, requiring increasingly powerful servers, faster disks and faster networks to process data. Existing High Performance Computing solutions involving high capacity data centres are both complex in design and expensive to maintain, while providing resources primarily to high profile science projects.

This research describes three distributed pipeline architectures, a virtualised cloud based IRAF, the Astronomical Compute Node (ACN), a private cloud based pipeline, and NIMBUS, a globally distributed system. The ACN pipeline processed data at a rate of 4 Terabytes per day demonstrating data compression and upload to a central cloud storage service at a rate faster than data generation. The primary contribution of this research however is NIMBUS, which is rapidly scalable, resilient to failure and capable of processing CCD image data at a rate of hundreds of Terabytes per day. This pipeline is implemented using a decentralised web queue to control the compression of data, uploading of data to distributed web servers, and creating web messages to identify the location of the data. Using distributed web queue messages, images are downloaded by computing resources distributed around the globe. Rigorous experimental evidence is presented verifying the horizontal scalability of the system which has demonstrated a processing rate of 192 Terabytes per day with clear indications that higher processing rates are possible.

Contents

List of Figures

List of Tables

Listings

Associated Publications

The publications that are related to this thesis are listed below:

1. Doyle, P; Mtenzi, F; Smith, N; Collins, A and O'Shea, B "Significantly reducing the processing times of high-speed photometry datasets using a distributed computing model", Proc. SPIE 8451, Software and Cyber infrastructure for Astronomy II, 84510C (September 24, 2012); doi: 10.1117/12.924863;

2. Doyle, P.; Deegan, M.; Markey, D.; Tinabo, R.; Masamila, B.; Tracey, D. Case Studies In Thin Client Acceptance. Ubiquitous Computing and Communication Journal. Special Issue on ICIT 2009 conference - Applied Computing, 2009.

3. Doyle, P.; Deegan, M.; O'Driscoll, C.; Gleeson, M.; Gillespie, B.; , "Ubiquitous desktops with multi-factor authentication," Digital Information Management, 2008. ICDIM 2008. Third International Conference, vol., no., pp.198-203, 13-16 Nov. 2008 doi: 10.1109/ICDIM.2008.4746797

Chapter 1

Introduction

Photometry is defined as the branch of science that deals with measuring the intensity of the electromagnetic radiation, or flux of a celestial object [1]. The word photometry is derived from the Greek word photos, meaning light, and métron, meaning measure [2]. This science can be traced as far back as 130 BC to Hipparchus [3], who devised the first measurement system categorising objects' apparent brightness from brightest to faintest. While the initial Hellenistic method provided only six classifications, the sophistication and sensitivity of the tools used in measurements evolved dramatically throughout the ages, with the current system of measurement of apparent brightness allowing for fractional measurements, both positive and negative, which have no specific upper or lower limits.

Modern photometry has evolved and been revolutionised by the use of Charge-Coupled Devices (CCD). CCDs are light sensitive integrated circuits that are used in imaging and signal processing. Information is represented as packets of electric charge which are stored in an array of closely space capacitors which can be moved from one capacitor to another. Charges can be moved systematically to a location on the device where the charge is converted to a digital value representing the light intensity for each pixel to form an image. Since their first introduction to astronomy, CCDs have received considerable attention from the astronomical community [4], and revolutionised this field of science providing levels of sensitivity beyond the capability of photographic plates, extending the detection range into the infrared spectrum, providing immediate results with a linear response, and allowing for software to compensate for CCD defects. When a CCD digital image is recorded it contains a digital count of the electrical charge of each of the pixels on the CCD array. The electrical charge per cell is converted to a digital pixel value by first transferring the charge to a corner of the array and then using an analog to digital conversion to record

its value. This digital image contains a number of different artefacts introduced by the process of recording and reading, which must be removed. These and other sources of noise require a computation operation to be performed across the image in order to quantify the signal to noise ratio. So for each image taken there is a computational overhead required before scientific analysis can be performed. As the number of images increases, so does this computation cost.

Over the last few decades the number of ground based astronomical research observatories has continued to increase and currently stands at approximately 400 sites globally [5] with each site using some form of CCD device. While CCD devices, which have improved both in terms of resolution and capture rates are still the primary capture device in use, Complementary Metal Oxide Semiconductor (CMOS) devices which offer the potential for faster imaging capture rates with lower power utilisation and higher resolutions [6] are increasing in popularity. It is the combination of these developments, faster image capture rates, higher resolutions and more telescope observatories that drives the increase in image processing based computing requirements. Even this requirement pales in comparison to the potential of highly distributed robotic telescope projects being developed [7] which could, over time, increase the number of CCD or CMOS images being produced to billions per second, raising the issue of data processing to terabytes and petabytes per day. Within the last few years astronomers have voiced serious concerns about the growing data tsunami with Berriman [8] predicting 60PB of data being available as an archive by the year 2020. The International Digital Coporation's 7^{th} annual Digital Universe Study echoes this concern with the issues of data growth and the subsequent data generation, storage, search and processing highlighted, identifying exponential growth in digital data expanding from 4.4 to 44 zettabytes[1] by the year 2020 [9]. As with astronomical data, much of the cause for this growth is identified as falling costs of capture devices and the increase in digital versus analogue data collection technology.

Many of the existing approaches to CCD image data processing are still primarily sequential in nature with software tools written for single Central Processing Unit (CPU) cores using single threads. This approach was reasonably successful given that for many years Moore's Law continued to hold true, and faster machines could be purchased to speed up the overall system performance every one to two years without dramatically altering the underlying software applications used to process raw CCD images. However, given that

[1] A zettabyte is one billion terabytes, or 10^{21} bytes

Moore's law stated that there would be a doubling of the number of transistors on an affordable CPU every two years, and that this has led to multi-core CPU chip designs, the law, while still technically true will no longer provide significant performance enhancements for single threaded sequential processing applications. Programs must become multi-threaded and software must evolve to take advantage of multiple CPU cores.

Other limitations also appear once the volume of data becomes sufficiently large. Network bandwidth becomes a limiting factor when large volumes of data are centralised, and even single server multi-core CPU systems may not provide enough of a performance enhancement to processing rates. As it becomes cheaper to generate data, more sophisticated processing techniques are required which can easily utilise large arrays of computing resources. A more sensible redesign of data processing software should consider exploiting additional processing and networking resources beyond a single infrastructure and embrace a distributed processing approach.

In order to address the issue of cleaning and preparing terabytes or even petabytes of CCD based astronomical photometry images per day, a distributed elastic cloud based computing model to perform standard image data processing is required. A processing pipeline has been designed which demonstrates a working CCD image reduction pipeline, which incorporates an elastic data processing model where resources can join or leave a swarm of distributed computing workers which communicate via a distributed web based messaging queue. Furthermore, taking advantage of the fact that CCD images can be cleaned in isolation from each other, image data is distributed for parallel processing to eliminate sequential image processing bottlenecks.

To ensure that access is provided to all data sources, experimental results and source code used within this thesis, the following URL is provided as an access point to all material `http://www.dit.ie/computing/staff/pauldoyle/research/`.

1.1 Background

1.1.1 Definitions

To ensure clarity of the terms used within the context of astronomical photometry, the following definitions are provided for reference.

Apparent Magnitude

The apparent magnitude of a source is based on its apparent brightness as seen on earth, adjusted for atmosphere. The brighter the source appears, the lower the apparent magnitude value. The apparent magnitude of an object is measured using a standard reference as shown in this equation where m_2 and F_2 are reference magnitude and reference flux values and F_1 is the observed flux. $m_1 - m_2 = 2.5log\frac{F_1}{F_2}$

Absolute Magnitude

The absolute magnitude is a measure of a stars brightness as seen from a distance of 10 parsecs (32.6 light years) from the observer. The absolute magnitude M of an object can be calculated given the apparent magnitude m and luminosity distance D which is measured in parsecs. $M = m - 5((log_{10}D_L) - 1)$

Instrumental Magnitude

The instrumental magnitude is an uncalibrated measure of the apparent magnitude of an object which is only used for comparison with other magnitude values on the same image. This is calculated using the following formula where f is the measure of the intensity value of the source image. $m = 2.5log_{10}(f)$

Luminosity

The Luminosity of an object is a measure of the total energy emitted by a star or other celestial body per unit of time and is independent of distance and is measured in watts. The luminosity of a star L is related to temperature T and the radius R of the star and is defined as follows where σ is the Stefan-Boltzmann constant. $L = 4\pi R^2\sigma T^4$

Flux

The flux is a measure of the apparent brightness of a source which is inversely proportional to the square of the distance and is measured in watts per square meter W/m^2. How bright a source appears is based on the distance from the object and the Luminosity of the object which can be defined as follows where L is the Luminosity and d is the distance to the source. $F = \frac{L}{4\pi d^2}$

1.1.2 Photometry

There have been many methods devised to estimate apparent magnitude values within astronomical photometry with each new system or advancement aimed to reduce the error margin and increase repeatability for each measurement, but until relatively recently, it was the skill of the observer that ultimately determined the accuracy of the recorded magnitude value [10]. A brief look at the origins of photometry and the evolution of magnitude measurements will help to explain the magnitude scale.

The earliest measurement of a star's magnitude is credited to Hipparchus (190 –120 BC), a Greek astronomer and mathematician who it is believed made his observations in the 2nd century, BCE. While his original work did not survive, it was referenced by Ptolemy in the Almagest (which is dated at approximately 147 AD, see Figure 1.1), which contains a star catalogue of just over 1000 stars referenced by positions within a constellation and their apparent brightness or magnitude. The original magnitude scale used by Hipparchus was a six point system where the brightest stars were designated as $m = 1$ (First Magnitude) and the faintest stars are designated $m = 6$ (Sixth Magnitude). An increase from magnitude 1 to 2 for example represents more than a halving of the light visible from an object. While Ptolemy claimed to have observed all of the stars himself, it has been argued that the data was based at least partially on the Hipparchus observations (almost 300 year earlier) [3]. The reason for the increasing number for less brilliant stars is most likely based on the division of the twilight into 6 equal parts and stars which became visible within each segment were assigned a magnitude value, hence more faint stars were visible later and received a high magnitude numerical number. This system of magnitude calculation stood for almost 2000 years and it is only relatively recently that the precision of magnitude calculations has increased significantly from these ancient times.

Hearnshaw [12] provides an excellent account of this progression pointing out that while many systems existed, the ability to combine and standardise star catalogues was a consistent concern. When telescopes were introduced to astronomy it became possible to see even fainter stars than those observed with the naked eye, and the magnitude scale moved beyond the value of 6. With current state of the art instrumentation we are entering an era of observing objects as faint as magnitude 30 [13] and possibly beyond. To determine where a star is on the magnitude scale, a reference star is chosen and allocated a standard number. At one point Polaris was assigned the magnitude value of 2.0, but this star is

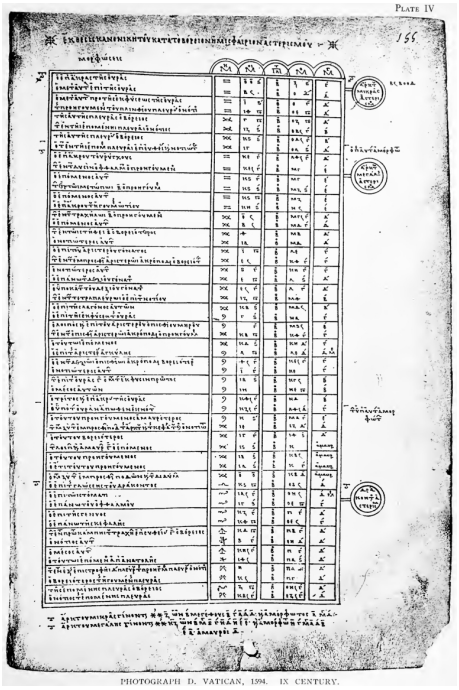


Figure 1.1: Claudius Ptolemy Star Catalogue. Alexandria, 2nd century. [11]

a variable star (apparent magnitude changes over time) so this was not an appropriate reference star. Vega was finally selected and assigned the value of 0. Using the star Vega as a reference point for magnitude 0, the table of magnitude values for the Moon, Planets and the Sun requires the magnitude scale to enter negative values as shown in Figure 1.2.

What is being measured during the photometric process is the apparent brightness (or apparent magnitude) of an object and not its actual magnitude. To illustrate the difference in actual versus apparent magnitude consider the apparent brightness of a 40 watt bulb as seen from 10 meters versus 10 kilometres. In both cases the light bulb retains the same luminosity, but the apparent brightness is dramatically different due to the distance between the observer and the light bulb.

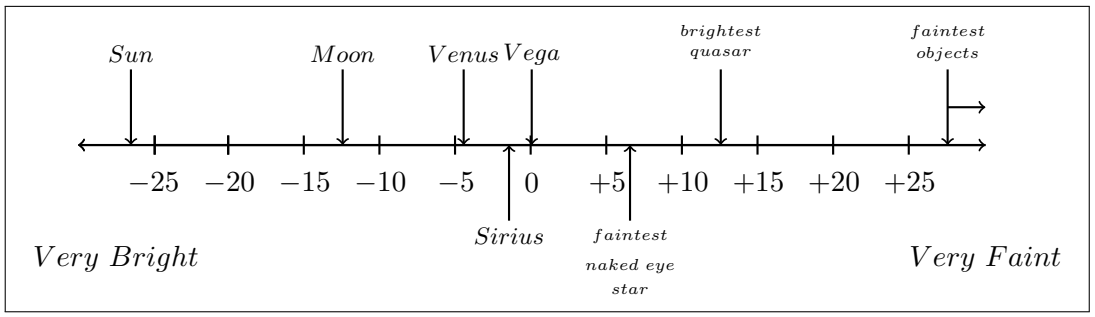


Figure 1.2: Apparent brightness of a selection of objects using the magnitude system

Photometry measurements from the time of Ptolemy remained relatively unchanged for about fifteen centuries with photometry estimations not improving until William Herschel (1738-1822) produced the first reliable naked eye star catalog using a telescope. Herschel used a system of estimating the difference between objects, which was later formalised by Friedrich Argelander (1799-1875) who established the step method. John Herschel continued his father's work achieving an estimated error of ± 0.12 magnitude which is close to the practical limit of visual photometry, ± 0.1 mag. The visual photometer, which appeared in the mid 19th century used prisms to project two objects into the field of view of the observer who would then equalise the apparent brightness of each object through a series of adjustments. The relative difference in the calibration process contributed to the calculation of the magnitude difference between the objects.

The next major advancement in photometry was the introduction of the photographic plate in 1871. By comparing existing stars within reference catalogues using visual inspection, large photographic surveys were undertaken. By 1930 it was estimated that measurements could now be made which were at a precision of ± 0.02 mag.

William Henry Stanley Monck (1839-1915), made the first electrical measurement of light 1892 but it was not until 1907 however, that Joel Stebbins (1878-1966) used the photoconductive cell in conjunction with the photocell achieving a photometric accuracy of ± 0.023 mag, surpassing existing accuracy levels of photographic photometry. Frequent technical issues ensured that photographic plates remained in place for the first half of the 20th century.

The glass photomultiplier tube (*pmt*) became the primary photometry measurement system in the mid 20th century due to to its quantum efficiency of about 10%-30% compared to just 1% for photographic systems. These systems would make way for the CCD by the early 1970's which provided two dimensional array detectors, and ultimately higher levels of precision. A good summary of the history of the photometric evolution is given by Richard Miles, 2007[14].

1.1.3 Charge Couple Devices

It is only quite recently that our ability to measure and record highly precise magnitude values for faint objects has developed, and this ability can be attributed to the introduction and use of the CCD in the 1970s [15]. With highly accurate photometry measurements based on CCD technology and careful observation it is possible to detect fluctuations

in apparent magnitude, which may be used, among other things, in the identification of extra-solar planets orbiting distant stars [16].

The CCD was invented in 1969 at Bell Labs [12] by William Boyle and George Smith and was initially conceived as a memory module, but it was only four years later that a program was initiated within the NASA Jet Propulsion Laboratory (JPL) to work on an imaging device of greater than 100x100 pixel resolution for space based probes using this technology. By 1974 Texas Instruments, under contract from NASA, produced a report outlining the feasibility of such a device [17], which would later be used in the Galileo mission to Jupiter (1989) and the Hubble telescope (1990), among others. By the mid 1970s it was decided that in order to engage the scientific community in this new CCD technology (the primary technology in use at the time was photographic film and vidicons) was to create a mobile CCD device, which was brought to a number of ground based telescopes. According to Janesick, new scientific discoveries were made each time the camera system visited a new site [18], and it was these trips and a high quantum efficiency rating (photon to electron conversion rate) which quickly led to a dramatic increase in the demand for CCD devices. Figure 1.3 shows the first ever image of Uranus [19] taken using a 400x400pixel CCD sensor on Mount Lemmon's 154cm telescope. Since then the CCD has evolved and become the dominant device used by professional astronomers.

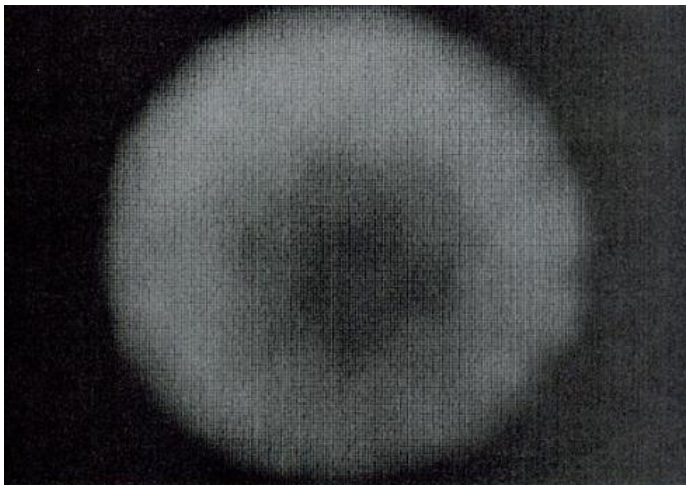


Figure 1.3: First CCD image of Uranus taken in 1976 at Mount Lemmon [19].

There are many reasons for requiring accurate magnitude calculations and an interesting and topical example of their use is in the hunt for extra-solar planets [16]. One such method is referred to as the photometric transit method [20] where stars with planetary bodies that rotate around the star in the same plane cause a reduction in the apparent

magnitude of the star. The transit is detected using a photometric light curve as shown in Figure 1.4 for star HD 209458, which was the first planetary transit of a star identified using this method.

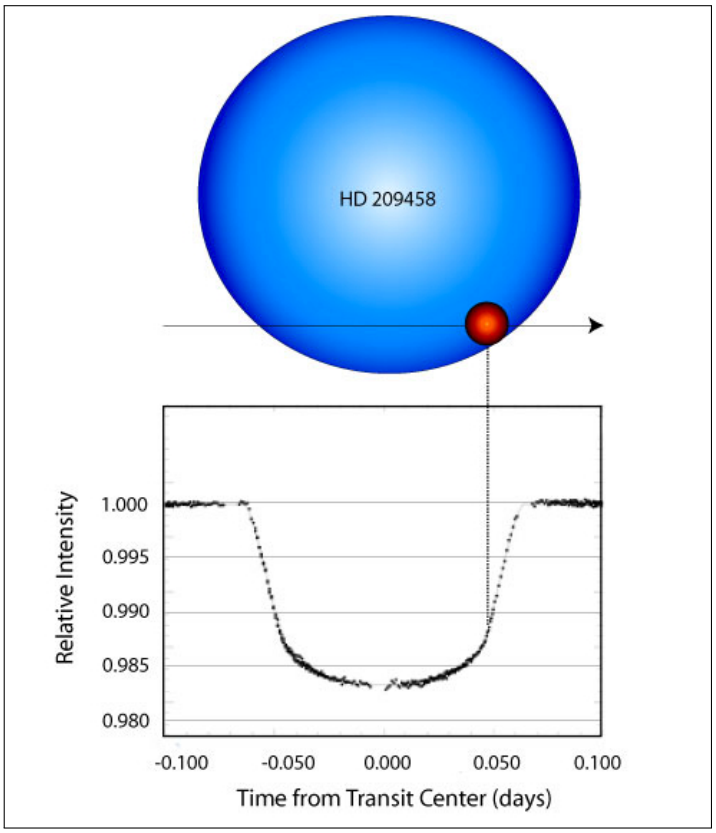


Figure 1.4: Detection of the first planetary transit of the star HD 209458. [21].

The most well known example of this method in use, is the recent Kepler Mission, which had the specific aim of detecting Earth-class extra-solar planets around stars with a detection method determined to be viable based on two factors, "that the stellar variability on the time scale of a transit be less than the decrease in brightness caused by the transit and that the photometric instrumentation has sufficient precision to detect the transit". [22]. The instrumentation referred to is the CCD of which there are 42 devices. Each 50x25 mm CCD has 2200x1024 pixels. The level of change to the brightness of the object is as small as 1/10,000 (100 parts per million, ppm) for a terrestrial sized planet, which provides some indication of the need for high precision photometry.

The CCD has contributed dramatically to photometry within the astronomical community. It is the primary instrument of most, if not all, of the large-scale optical astronomical survey systems currently in existence. As stated, the Kepler spacecraft uses 42 CCD devices, the Sloan Digital Survey comprises of 30 CCD devices of approximately 2048x2048

pixels in size, and the Hubble telescope initially used 8 low resolution CCD chips. A CCD chip is built on a single wafer and is made up of a two dimensional array of pixels. As a photon of light hits the silicon surface of a light sensitive CCD pixel the energy is absorbed raising some electrons to a higher energy state and releasing them, allowing them to flow towards the n-type silicon layer as shown in Figure 1.5 where an electrical charge accumulates which is directly related to the level of incident light. This potential well exists for each pixel. After a period of time (the exposure time) the accumulated charge for the pixel, is moved towards the readout point by transferring the charge across the device. Once the parallel shift of pixel charges is performed, the pixels at the edge of the device are transferred using a serial shift to transfer the charge to the measurement electronics. Using an analog to digital converter, the charge is converted to a digital numerical value for the specific pixel charge. This is used as the raw digital image value from the exposure and is transferred to the computer.

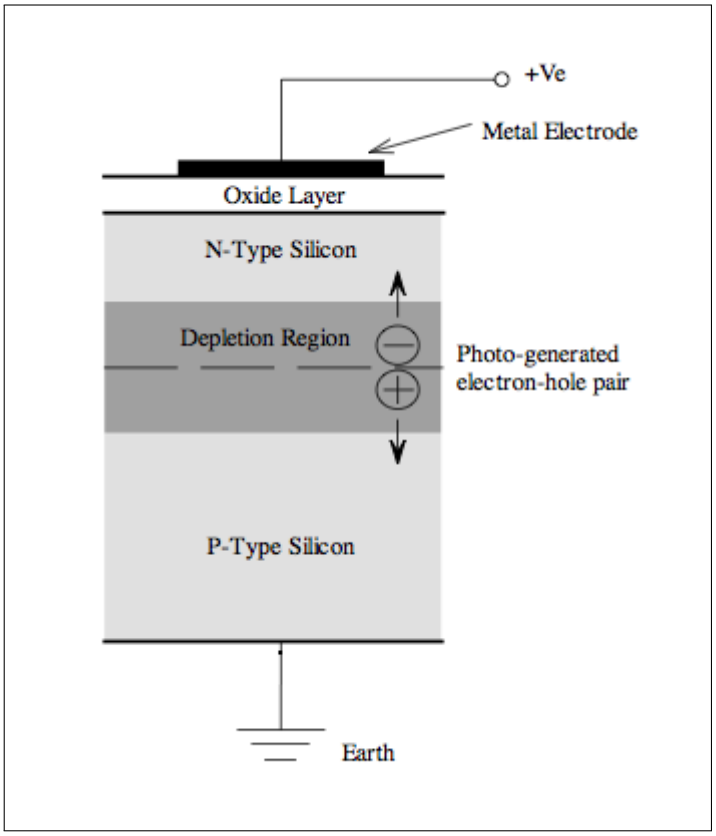


Figure 1.5: Cross section through a CCD pixel [23].

1.1.4 The Data Processing Challenge

When a single CCD detector records an image, the size of the digital image is usually dependant on the number of pixels on the device and the number of bytes used to store the value for the pixel. The size of the data-set generated by an array of CCDs is dependant on the size of each digital image, the image capture rate (ranging from milliseconds to minutes), the time period over which images are taken, and the number of CCDs in the array. While a small telescope may use a single CCD, larger telescopes may employ an array of CCDs, and robotic telescope farms may use an array of telescopes each with its own CCD array. With megapixel CCD arrays already in use, and with frame rates per second increasing, the tsunami of data production is already beginning. Indeed, Graham [24] refers to the *data avalanche, tsunami and explosion* of data, predicting that by 2020 the scale of the problem will be apparent as not just optical, but radio telescopes generate petabytes of data on a nightly basis. Ferguson et al [25], looking to the next decade of data reduction and analysis sees the three major challenges as follows:

1. Data rates growing rapidly as CPU processing rates level off.
2. Industry trends in computing hardware leading to major changes in astronomical algorithms and software.
3. Computationally demanding analysis techniques becoming more essential with increasing pressure on computing resource.

New sky survey systems in development, such as the Large Synoptic Survey Telescope (LSST), will produce up to 20 terabytes of data per day in the very near future. Supercomputers/high performance computing is proposed as a primary requirement with parallelism being the natural development to address challenge one above. With this level of data production, issues beyond processing must also be considered. Concerns about storage, input/output and processing have been in the published literature for many years. Shaw et al in 1995, in a short paper [26] described the growing issue of large databases of data, the possibility of moving to lossless compression, and stating that vast data arrays will *"tax networks, I/O systems, and storage capabilities, so compression techniques will be crucial"*. These concerns remain the same today. Murtagh et al [27] summarised the issue as follows going on to discuss the requirement for some form of image compression strategy for data movement. *"The quantity of astronomical data is rapidly increasing. This*

is partly owing to large digitized sky surveys in the optical and near infrared ranges. These surveys, in turn, are due to the development of digital imaging arrays such as charge coupled devices (CCDs). The size of digital arrays is also increasing, pushed by astronomical research demands for more data in less time."

It is only when considering the combination of these challenges that the extent of the problem of large dataset production and processing can be fully appreciated. The factors which contribute to large dataset generation are summarised as follows.

- Resolution: Number of pixels captured per image.
- Capture Rate: Number of images taken per second.
- Capture Period: The length of time over which images can be taken.
- Device Count: The number of capture devices operating at one time.
- Capacity: Ability to read and store data generated.

When a CCD image is created, the pixel value is a combination of both signal and noise. The process of performing image reduction and preparing the image for use in photometry is an essential step in all CCD image based pipelines and is often referred to as CCD reduction. There are typically three calibration frames used in the reduction of raw images, which are bias, dark and flat field frames [28]. In addition to these basic reductions, further image processing is required to complete the calculation of an accurate magnitude value for a series of reference objects within each image and all of these steps are precursors to the production of light curves from the CCD image. As the number of images produced increases, so does the processing requirement as this reduction process is applied to each image.

1.1.5 Research Scope

The data processing of CCD images is restricted in this research to pixel level calibration and basic photometric analysis. Figure 1.6 provides a summary of the operations performed by the NIMBUS pipeline which stops short of performing any actual science on extracted magnitude values from images. To ensure that the ability to analyse magnitude values can be done in real-time, PCAL and PHOT should process data at the same rate as data is being generated and supplied to the pipeline. Just-in-time processing must be completed

within a 24 hour period which would mean data processing must be no less than 3 times slower than data acquisition before a bottleneck is created, assuming an 8 hour image capture period per day.

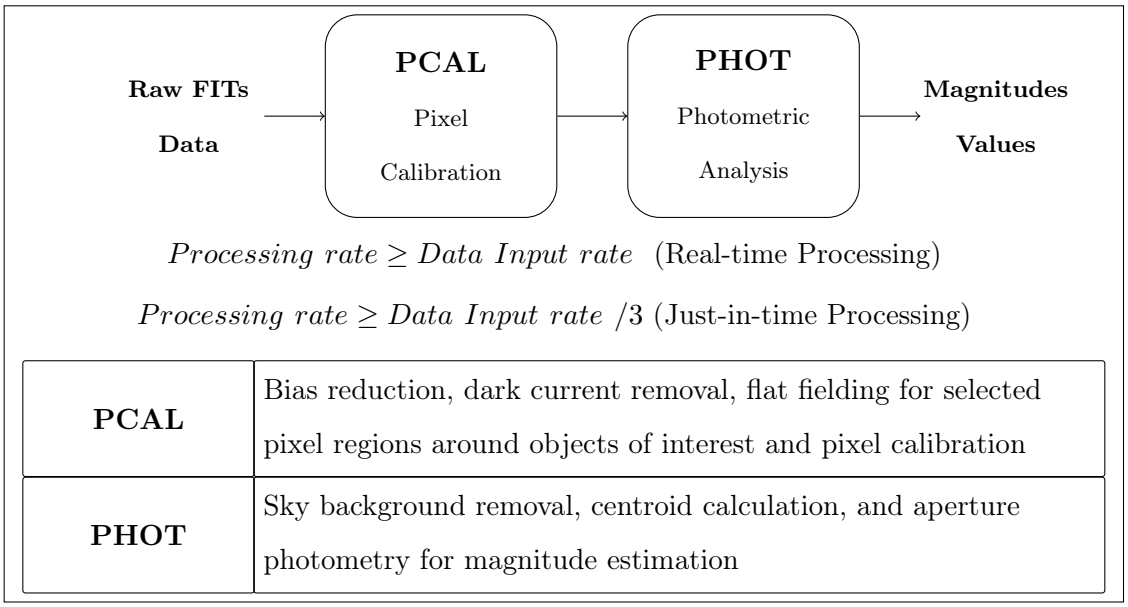


Figure 1.6: An overview of calibration and photometric analysis performed on raw CCD images within the NIMBUS pipeline.

1.2 Research Hypothesis

The research hypothesis proposed within this thesis was to determine if a globally distributed astronomical CCD image reduction pipeline can process data at a rate which exceeds existing data processing requirements and is scalable such that it can meet future data processing challenges.

To support scalable growth, a pipeline would be required to allow horizontal scaling of all components relying on parallel processing of data in a robust reliable manner. Work orchestration requires the communication of thousands of computing processes allowing for nodes to be added or removed without interfering with the running of the system. Image processing must be suitable to parallelised processing. The pipeline should also not be restricted to the specifics of CCD photometry, but be flexible enough to process other data products, which can also be processed in parallel.

1.3 Thesis Contributions

A distributed pipeline was conceived which was validated to determine how processing rates upwards of 200 terabytes of data per day could be achieved. The NIMBUS pipeline, developed as the primary contribution of this thesis, accomplishes this by enabling the use of global computing resources to easily and seamlessly contribute to image processing. The key enabling features of the architecture are distributed web queuing for message based communications, self configuring workers which allow for multiple science payloads to be processed, system resilience which allows running workers to join or leave the pipeline seamlessly, parallel processing of images, and decentralised storage and processing.

The main contributions of this thesis can be identified as follows:

1. NIMBUS, a globally distributed data processing architecture that can process hundreds of terabytes of data per day which is also scalable beyond this point.

2. A self configuring, balancing system that is scalable and resilient to failure.

3. A dynamically reconfigurable pipeline that has wider applications than astronomical image processing.

4. A real-time pipeline, which in this context may involve a small processing latency in the order of one to two minutes, but this latency would be small enough to enable a feedback response to the telescope, or observatory, to allow them to react to recently captured and processed scientific data.

1.4 Structure of this Thesis

This thesis is structured as follows.

Chapter 1

This chapter provides context for the thesis and introduces the challenge posed by the growing volume of image data within astronomy. An overview of how this thesis proposes to address this problem is presented through the globally distributed NIMBUS pipeline, and the contributions are clearly identified.

Chapter 2

This chapter reviews the literature on the processing of astronomical photometry, describing the primary noise sources within a CCD and the processes of reducing them. Data generation sources within the astronomical community are identified in addition to reviewing existing data processing techniques, processing rates and the data volume challenges facing the astronomical community. Distributed computing techniques and how these are being used within the scientific community at present within astronomy and other scientific disciplines are also reviewed.

Chapter 3

Chapter 3 gives an overview of the experimental methodology used in all experiments and discusses a series of experimental designs based on distributed computing techniques and how they can be used. The experimental setup for Chapters 5 and 6 are also presented with a review of the technology used in these experiments.

Chapter 4

Chapter 4 presents the design, implementation, results and evaluation of the Astronomical Compute Node (ACN) experimental model. This model uses a distributed hybrid cloud, which relies on a private Network File System (NFS) locking mechanism to communicate with workers available data for processing. The results and findings from these experiments are presented within the chapter.

Chapter 5

In Chapter 5 the design, implementation, results and evaluation of the NIMBUS experimental module are described. NIMBUS uses a global processing cloud controlled incorporating Amazon Web Services for web based message queuing, computing instances and data storage. The results and findings from these experiments are presented within the chapter.

Chapter 6

Chapter 6 provides an analysis of the findings of this thesis, reviews and summarises the work presented, drawing conclusions and identifying possible future work.

Appendicies

Appendix A through D provide additional data and material referenced within the main chapters of this thesis.

Chapter 2

Astronomical Photometry Data Processing

2.1 Introduction

In this chapter the literature for standard reduction techniques, basic photometry, the source of data and existing data processing practices is reviewed. These four sections are essential to demonstrate that the data processing techniques implemented within the NIMBUS pipeline are consistent with standard data processing operations performed on astronomical CCD images. It is also required to provide context for the NIMBUS pipeline against existing technologies and current state of the art practices.

CCD and CMOS imaging systems have well understood reduction processing steps designed to calibrate a raw image. The accuracy of photometric measurement is based on these well defined cleaning techniques which are discussed in more detail in this chapter.

Aperture photometry techniques provide a clear process for the estimation of apparent magnitude of objects, using a standard reference scale. Finding the centre of objects, estimating the sky background and calculating the flux of an object for a range of aperture sizes are all well defined procedures.

The sources of CCD data production are also considered, including existing and future data processing techniques and challenges. The most commonly referenced projects, given as examples of the growing data challenge within astronomy, are the Large Synoptic Survey Telescope (LSST) [29] which is primarily optical in nature, and the Square Kilometre Array (SKA) [30] which is radio based. The LSST (expected to come on-line in 2019) is predicting data acquisition sizes in the region of 20-30 terabytes per night, while the SKA (2024) has a variety of predicted data capture rates depending on the implementation size of the array, although most agree this has the potential for generating one terabytes per second [31].

Data processing requirements vary extensively and the following categories are reviewed in terms of their data generation capabilities.

Observatories large and small, space based and ground based all employ some form of software processing on images. Well established file formats, a range of data reduction pipelines and an extensive set of software packages and technologies ensures there are a mix of approaches in existence for performing data reduction.

2.2 Standard Image Reduction Techniques

When a CCD instrument is used, the recorded file output stored on the computer contains a measure of the source signal in addition to unwanted random contraptions for various sources. The noise introduces an error into the measurement. In this section the sources of noise in CCD image reading are described along with the techniques used to deal with them. These techniques are incorporated into the NIMBUS system.

2.2.1 Noise Sources

If a CCD recorded a single electron for every photon striking a pixel and this was the readout value obtained from the CCD then it could be considered to be a *perfect CCD*. In reality this is not the case, and a number of factors contribute to the introduction of noise to the process. Noise is the introduction of unwanted variations to the image, distorting the readings in some way. If a CCD pixel has a well depth of 100,000 electrons (the total amount of charge that can be stored in a pixel) and the average noise can be determined to be approximately 40 electrons per pixel then the SNR (Signal to Noise Ratio) is 100,000/40 or 2,500. If the amount of noise can be reduced then the SNR is increased. The process of reducing the level of noise in an image is critical to performing high precision photometry. The standard equation for SNR is given in equation 2.1 and is often unofficially referred to as the CCD Equation [32].

$$SNR = \frac{N_*}{\sqrt{N_* + n_{pix}(N_S + N_D + N_R^2)}} \tag{2.1}$$

- N_* represents the total number of photons collected from an object of interest which can be either 1 or more pixels.
- n_{pix} represents the total number of pixels considered.

- N_S represents the total number of photons per pixel from the sky background.
- N_D represents the total number of dark current electrons per pixel.
- ${N_R}^2$ represents the total number of electrons per pixel from read noise.

The main contributions to noise within a CCD are described here and the method for reducing them is expanded upon in the following subsections :

- Dark current is a thermal noise source build-up within a pixel. Longer exposures exhibit cumulative effects as it is exposure time dependant. Cooling the CCD is an effective strategy to reduce this noise in addition to the use of shorter exposures or through the use of *Dark Frames*.
- Pixel non-uniformity refers to the variation in pixel sensitivity when exposed to the same levels of light. While standards are exacting, differences in pixel sensitivity exist. This noise is eliminated to some degree by *Flat Fielding* which also eliminates other optical variations or dust.
- Read noise is an additive noise source introduced during the reading of values from the CCD during the conversion from an analog to a digital number. It is primarily removed using a *Bias Frame* as the amount of noise is independent of the exposure time. There have been dramatic improvements in this noise reduction over time. Initially this value was as high as 500 electrons per readout in early astronomical CCD imagers, but this value has been reduced to as low as 3 electrons per readout in CCDs and as low as 1 in CMOS devices.
- Charge transfer efficiency. As pixel charge values are moved across the CCD towards the readout point some electrons may get dropped or lost during the transfers. A charge may be transferred thousands of times (for example in a 1024x1024 pixel array the maximum a charge will be transferred is 2048 time). The CTE is not usually an issue with a typical efficiency rating of 99.99997% not being uncommon. This relates to approx. 2.5 electrons out of every 85,0000 being left behind or lost.
- Cosmic rays are particles which travel at high velocity and may dramatically increase the electron count for a pixel. The energy released by the particle releases many electrons which are then recorded as bright spots on the image. If this value is used,

it has the ability to distort calculations such as star magnitudes, bias pixel values or flat field pixel values.

The noise within the CCD can be characterised using equation 2.2 [33]. When a CCD image is taken, a two dimensional digital representation of the image is created, which we can reference using x for the column position and y for the row position. The values recorded in a file are the digital counts of the charge generated by the electrons detected by each pixel (the signal) and additional charges relating to unwanted sources (noise).

$$s(x,y) = B(x,y) + tD(x,y) + tG(x,y)I(x,y) + randomnoise \tag{2.2}$$

where

- (x, y) represents the pixel row and column position on the image.
- $s(x, y)$ is a raw pixel digital count recorded on the CCD for an integration time of t.
- t is the integration time of the exposure in seconds.
- $B(x, y)$ is the *bias* digital count of each pixel for a 0 length exposure.
- $D(x, y)$ is the *dark current* digital count of each pixel for an exposure length of t seconds.
- $G(x, y)$ is the *sensitivity* of each pixels.
- $I(x, y)$ is the digital count of the light flux received by the pixel.

2.2.2 Bias Frames

A bias frame has a dark frame with an exposure time of zero and is a measure a pixel's read-noise. This value is usually caused by a low level spatial variation caused by the on-chip CCD amplifiers. Read-noise from a CCD is an additive noise source that is introduced during the pixel read process which does not vary with exposure time. This is a systematic noise source which must be removed. A master bias frame is created through the combination of multiple bias frames using the average pixel values seen across each frames as shown in Figure 2.1. An average value is considered acceptable given that the CCD should not be exposed to cosmic rays since there was no exposure of the CCD sensors. The master bias frame can then be used in cleaning data images by subtracting the master bias value for each pixel.

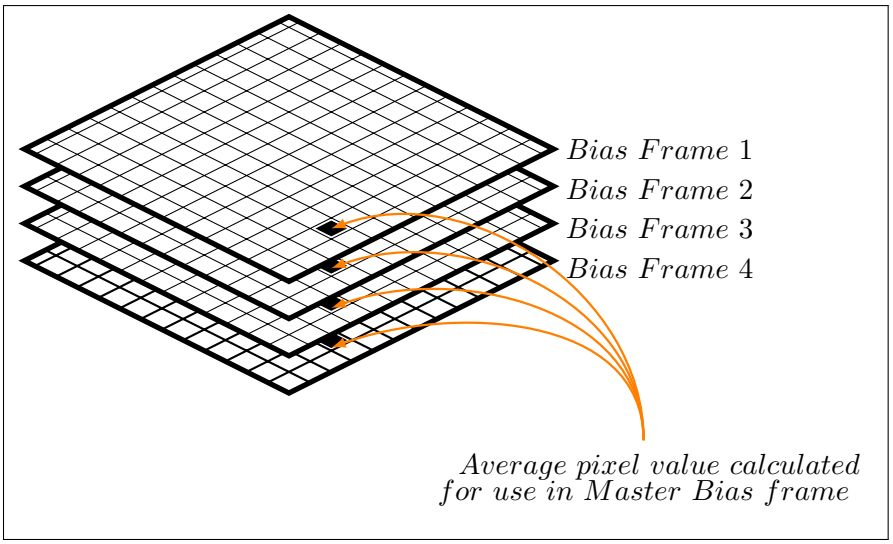


Figure 2.1: Master bias frame created using multiple bias frames.

Given that a bias frame is taken with a time interval of $t = 0$ equation 2.2 can be reduced to equation 2.3, where $b(x, y)$ is the bias value recorded in the pixel $s(x, y)$.

$$s(x, y) = b(x, y) = B(x, y) + noise \tag{2.3}$$

While a simple estimate of the bias value B of a pixel can be obtained by simply using the value $b(x, y)$ a better estimate of B can be obtained using the average value of the pixel across multiple bias images where N is the number of bias images used as shown in equation 2.4.

$$\widehat{B}(x, y) = \frac{1}{N} \sum_{i=1}^{N} b_i(x, y) \tag{2.4}$$

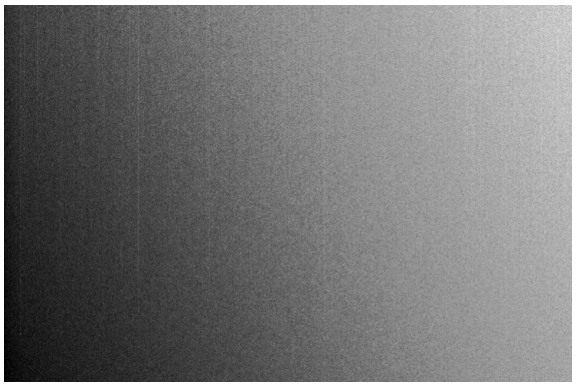


Figure 2.2: Example of a master bias frame.

2.2.3 Dark Current

A dark frame is used to determine the level of noise introduced by thermal events in the CCD device. Most devices are now cooled and thus reduce the amount of noise however it is still useful to understand this process. While a bias frame is taken with the shutter closed and the minimum exposure time, this dark frame is taken with the shutter closed, but for a specific period of time. The noise from thermal interference is time dependant, so the length of time of the exposure is important as the noise level is related to it.

The formula for a dark current frame with an exposure time t_{dark} is given in equation 2.5 when the closed shutter sets $I = 0$.

$$s(x, y) = d(x, y) = B(x, y) + t_{dark}D(x, y) + noise \tag{2.5}$$

An estimate of dark current for a pixel is found by subtracting the bias and dividing by the exposure time.

$$\widehat{D}(x, y) = \frac{d(x, y) - \widehat{B}(x, y)}{t_{dark}} \tag{2.6}$$

A more accurate estimate can be obtained by averaging a number of dark current frames M and eliminating the bias from each value. This gives a bias reduced master dark current frame for time interval t_{dark}

$$\widehat{D}(x, y) = \frac{1}{t_{dark}} \frac{1}{M} \sum_{i=1}^{M} d_i(x, y) - \widehat{B}(x, y) \tag{2.7}$$

A Master Dark Image can then be expressed using equation 2.8 for specific time intervals such as t_{data} to suit different levels of image exposures.

$$\widehat{D}'_M(x, y) = \widehat{D}(x, y)t_{data} \tag{2.8}$$

2.2.4 Flat Fielding

A flat field image is taken when the CCD has been evenly illuminated by a light source. Flat fielding is used to compensate for differences in pixel to pixel variations of the CCD response to illumination when the same amount and spectrum of light is illuminated across each pixel on the CCD. This technique also helps remove the effects of dust which can cause dark spots on an image and uneven illumination caused by vignetting in the optical system.

The flat field value is used to modify image pixel values to account for these variations. There are varying opinions on the best method to create a good flat field image, such as the use of an illuminated painted screen inside the telescope dome [34]. The difficulty with any technique is finding a means to illuminate the CCD with a flat distribution of light which is representative of the wavelength of the light expected during actual image recording of objects of interest. Techniques range from closing the telescope dome and using a specially treated surface for illumination, to using an image of the evening sky prior to data capture where there is still enough light available to illuminate the entire CCD sufficiently. Howell provides an excellent overview of many of these approaches [15]. A typical flat field is shown in Figure 2.3.

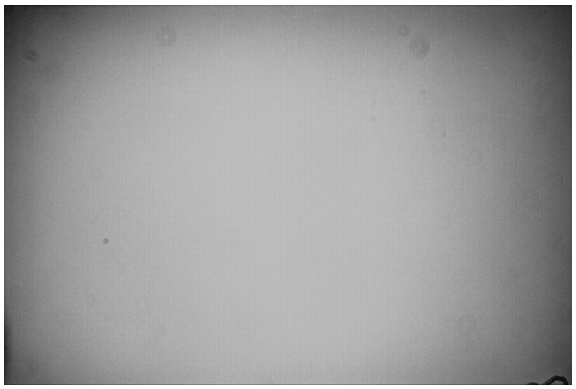


Figure 2.3: Example of a flat field Image.

Typically there are a number of flat field images taken and combined into a Master Flat Field Frame. There are similarities to the generation of a Master Bias, however due to the exposure of the CCD to a light source the possibility of encountering cosmic rays is increased, so median values, as opposed to average values, are used.

This can be expressed as the flat field value for a pixel with an exposure time of t_{flat} from Equation 2.2 using Equation 2.9 where L is the flux illuminated across all pixels equally.

$$s(x,y) = f(x,y) = B(x,y) + t_{flat}D(x,y) + t_{flat}G(x,y)L + noise \tag{2.9}$$

As already described in equation 2.8 a master dark can be created for the flat field image $\widehat{D}_M^F(x,y) = \widehat{D}(x,y)t_{flat}$ by using the same time integration as the flat field.

The noise of the flat field value can be further reduced by obtaining a *median* value for each pixel position using a number of flat field images as per equation 2.10.

$$\widetilde{f}(x,y) = Medianf(x,y) \tag{2.10}$$

To find a median value, the values for a pixel position are read into an array and sorted with the median value found using the following formulas.

Medians with odd number of data points

Sort data points and pick the middle data point and use its value. Where n is the number of data points you must read the value at the following index point in the sorted list $(n+1)2$[1].

Medians with even number of data points

Sort data points and pick the two middle data point and use the average of the two values. Where n is the number of data points[2] $value1 = \frac{n}{2}$ and $value2 = \frac{n}{2} + 1$ giving $Median = (value1 + value2)/2$

By subtracting the master dark frame which used the same time interval as the flat frames and subtracting the master bias a corrected flat field value is obtained for each pixel which gives a bias reduced master flat field frame as per equation 2.11.

$$f^{'}(x,y) = \widetilde{f}(x,y) - D_M^F(x,y) = t_{flat}G(x,y)L \tag{2.11}$$

The final step is to create a normalised flat field which has an average value of 1. First calculate $\overline{F}$ the average value of all values in the flat field where n is the number of elements in the flat field image in equation 2.12.

$$\overline{F} = \frac{\sum_{i=1}^{n} f_i^{'}(x,y)}{n} \tag{2.12}$$

Normalising each of the pixels in the master flat field can be performed by dividing each pixel by the average value $\overline{F}$ in equation 2.13.

$$G(x,y) = \frac{f^{'}(x,y)}{\overline{F}} \tag{2.13}$$

[1] for C where the array starts at 0, adjust this index as follows $(n-1)/2$

[2] for C where the array starts at 0, adjust this index as follows $value1 = (n/2) - 1$ and $value2 = n/2$

2.2.5 Image Reduction

The process of characterising the level of noise within a CCD pixel has been presented in equation 2.2. Using the estimation techniques identified, a basic image calibration process designed to reduce noise from the CCD raw images, a necessary process in preparing the CCD images for analysis, can be summarised. To simplify the process it can be assumed that the dark current frames were taken with the same time integration as the flat field frames and the data images. All images are stored using the Flexible Image Transport Systems (FITS) [35] format unless otherwise stated.

- Multiple CCD bias frames are captured with time integration of 0.
- Multiple CCD flat Field frames are captured for integration time t_{data}.
- Multiple CCD dark current frames for integration time t_{data}.
- Generate a master bias frame using equation 2.4.
- Generate a bias reduced master dark current frame for integration time t_{data} using equation 2.8.
- Generate a normalised bias reduced flat field master using equation 2.13.
- Capture raw CCD images frames.

Following the capture of raw CCD images, equation 2.2 can be used to estimate the value of flux for a pixel for a time interval t_{data} using equation 2.14.

$$\frac{s(x,y) - \widehat{B}(x,y) - \widehat{D}_M(x,y)}{G(x,y)} = \widehat{I}(x,y)t_{data} \tag{2.14}$$

A pixel value on a CCD frame has the bias and dark current removed and is then adjusted for the calculated responsiveness of the pixel relative to all other pixels. This calculation must be performed on all pixels which are ultimately used in the calculation of magnitude values. A new version of the image can then be created containing the calibrated pixel values. The creation of the master bias, flat field or dark frames is often done once for each night of observation and are then used in the calibration of pixels for that night.

2.3 Photometry using CCD images

The general steps in classical photometry using a cleaned digital image are usually identified as follows.

1. Image centring, the process of finding the centre of an object.
2. Estimation of the sky background for the purpose of removing it from the flux intensity value.
3. Flux value intensity calculation for an object for a specific aperture size.
4. Magnitude calculation for an object for a specific aperture size taking into account the sky background.

Multiple magnitudes can be generated based on variations in the software aperture size used in the calculation of the flux intensity. Each of these steps are described below, identifying the basic techniques and formulas as appropriate.

2.3.1 Centroid Algorithm

Once an image has been reduced, the first step is the calculation of the geometric centre of an object of interest, which must be precisely determined. There are a number of different algorithms available [36]. A gradient based technique is presented in Figure 2.4 along with the corresponding algorithm, Algorithm 6, given in section 4.2.3.4 of Chapter 4. The first step using this method is to clip a region of the image where the point source is located and apply a binary mask where all pixels above a chosen threshold are set to the value 1 and all pixels below the threshold are set to 0. The X position of the centre is found by using a column gradient where each pixel within the mask has its value set to the column number that it is in. The Y position of the centre is found by using a row gradient where each pixel within the mask has its value set to the row number that it is in. Where $N = number\ of\ pixels\ in\ the\ mask$, $M_C(x, y)$ representing a pixel within the column gradient image, and $M_R(x, y)$ representing a pixel in the row gradient image, the centroid X and centroid Yvalues using Equation 2.15 can be calculated.

$$CentroidX = \frac{\sum M_C(x, y)}{N} \quad CentroidY = \frac{\sum M_R(x, y)}{N} \tag{2.15}$$

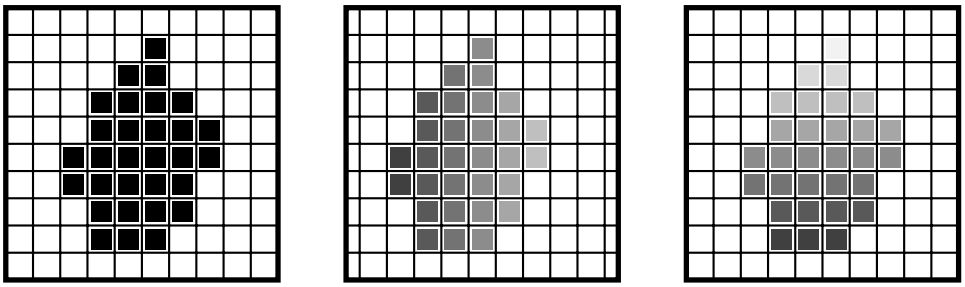


Figure 2.4: Centroid detection using a gradient technique

2.3.2 Sky Background Estimation

Using a calibrated image and the centre point of an object, the next step is magnitude calculation for specific objects on an image. Figure 2.5 shows the software aperture (solid line), around the star and the sky annulus (two dashed line circles). A simple estimate of the background sky level is to calculate the per pixel average value of the pixels within the sky annulus. These pixels also contain noise in addition to photons from the background sky which needs to be removed. More accurate estimates use the median value B_M and exclude values which are plus or minus 3 standard deviations form the median which will exclude cosmic rays and possibly other light sources. A buffer exists between the aperture and the sky annulus to ensure that the background is far enough away from the object as to be representative of the background.

The sky background B is an estimate of the amount of light which should be removed from the final flux value of a star and can be considered to be a photon based noise level. Typically the background calculation only includes pixels which are fully within the sky annulus and excludes partial pixels as shown in 2.6. By sorting these pixel values and obtaining a median value, the sky background $\widehat{B}$ per pixel can be estimated. The Euclidian distance is used to determine if a pixel is within the sky annulus. If R_1 is the distance between the centroid and the inner part of the sky annulus and R_2 is the distance to the outer part of the sky annulus, the distance from the centroid to a pixel r must be between these two values. The distance between the centroid Cx, Cy and the pixel position x, y can be calculated using equation 2.16. A pixel is considered within the sky annulus if r is greater than $R_1 + 0.5$ and less than $R_2 - 0.5$.

$$r = \sqrt{(Cx - x)^2 + (Cy - y)^2} \tag{2.16}$$

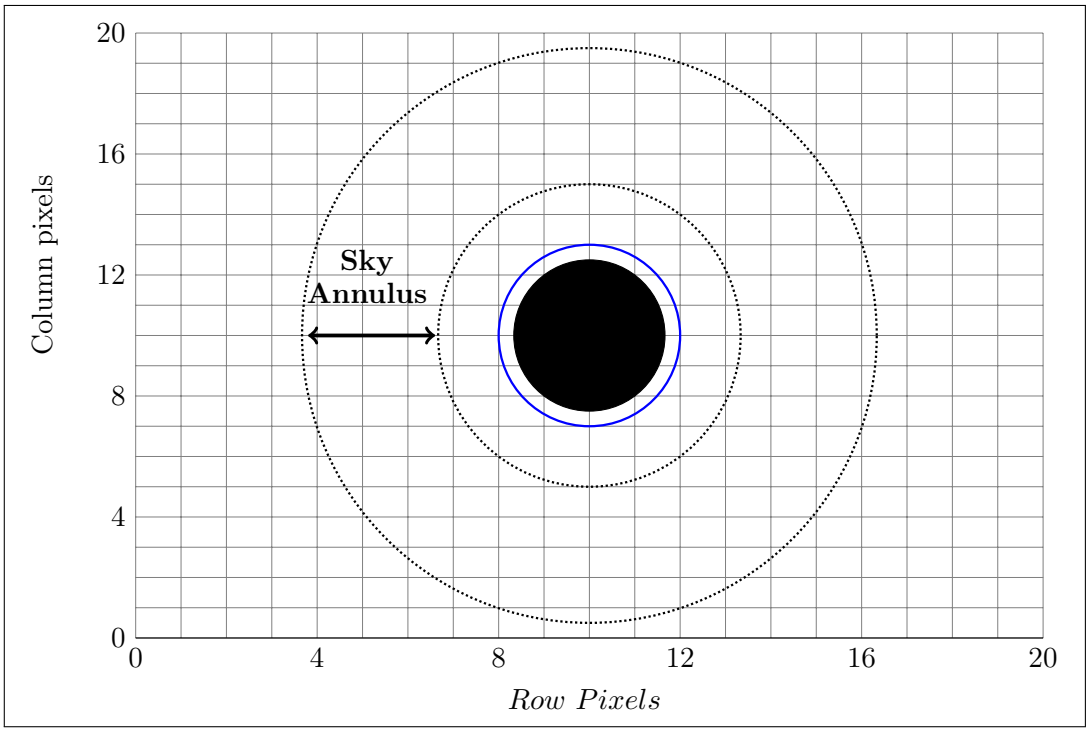


Figure 2.5: Representation of a star, with the aperture around the star shown in blue and the sky annulus shown between the dashed lines.[15]

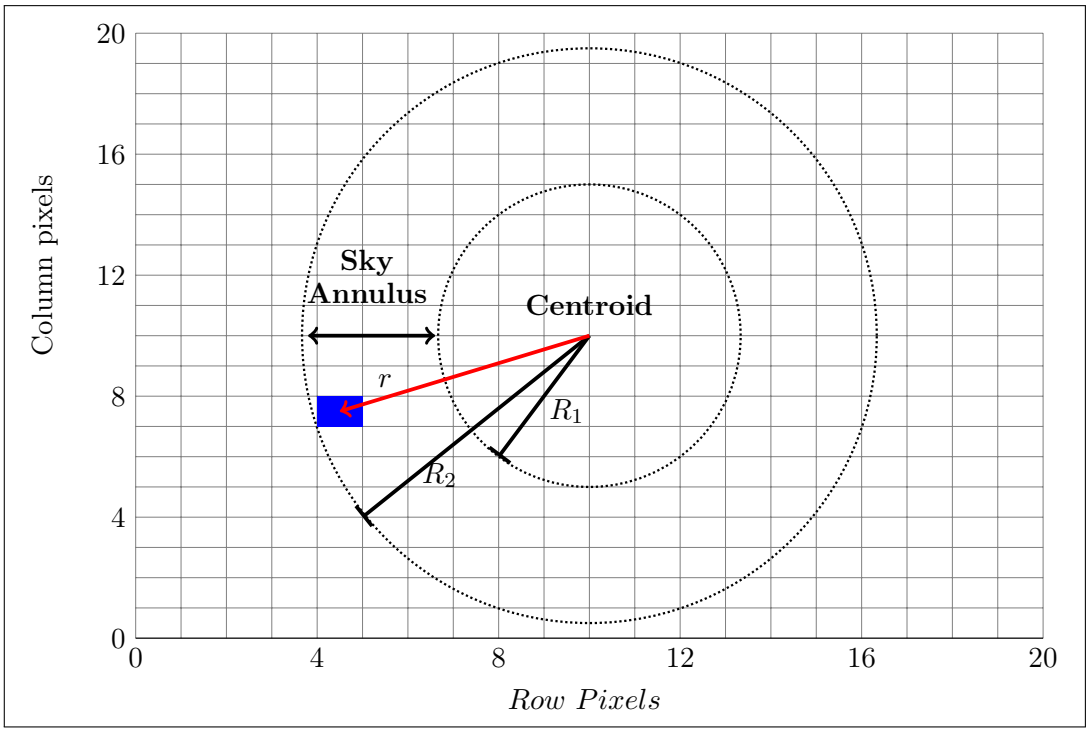


Figure 2.6: Determining if a pixel is within the sky annulus by ensuring the distance to the centre of the pixel is between R_1 and R_2 .

2.3.3 Calculating Flux Intensity values

The next step is to calculate the total flux value recorded for an object by summing all of the record pixel electron counts I within the software aperture. This is a total count of the unit values stored within each pixel within a specific aperture range. A more accurate flux total F_T is then obtained by including all pixels within the aperture and partial pixel counts. Using a similar method to the sky background calculation, pixels can be determined to be either fully inside the aperture, on the line, partially inside the aperture or outside of the aperture. If N represents the total number of pixels within the aperture (counting partial and full pixels) then the total flux can be estimated and the sky background subtracted using equation 2.17.

$$\widehat{F}_T = \sum_{i=1}^{N} I_i - N\widehat{B} \tag{2.17}$$

2.3.4 Calculating Instrumental Magnitude

Given an estimate of the total flux for the object the instrumental magnitude m calculation using a standard equation [37] can be completed.

$$m = -2.5\ log_{10}(\widehat{F}_T) \tag{2.18}$$

2.4 Data Sources

With an understanding of CCD calibration and magnitude calculations it is important to consider the context within which these operate. For any world-class scale project (space or ground based) significant investment is required in IT. Data products are produced, pre-processed to a pre-defined level, and made available to a Principle Investigator, supporting institutes or potentially to the public, either directly via download servers or via the Virtual Observatory (VO) [38]. For large projects, data capture, transfer, calibration and reduction, basic processing, archiving and access are considered as part of the observatory capabilities and bespoke solutions are often implemented. Smaller institutes often capture less data due to the capabilities of their instruments but investment in IT is still required, although more modest computing resources may be sufficient. Researchers and institutes will have varying requirements and capabilities either in data processing and/or data capture and it is the ability to match computing resources to large and potentially

varying data acquisition rates that is of interest. As smaller research groups have the capacity to generate larger volumes of data, a gap in processing capabilities emerges. As the pressure for data generation rates goes up, there should be pressure on bringing the IT costs in line so that smaller institutes take advantage of instrument improvements. Data processing costs cannot be allowed to grow linearly with data acquisition. Projects such as the ALMA (Atacama Large Millimeter/submillimeter Array) Correlator[3], Figure 2.7, which locate computing resources physically close to the capture devices due to remoteness of location or bandwidth restrictions, fail to avail of global resources and cannot be replicated for all observatories. For the purpose of categorising and reviewing existing sources of CCD image data the following general classifications are used.

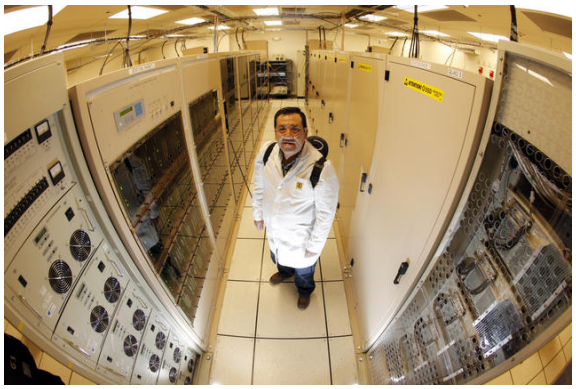


Figure 2.7: Technician breathing oxygen at the Alma Correlator, the world's newest and highest high performance computing system with over 134 million processors [39].

Space-based telescopes

Space-based telescopes operate with the significant advantage of being free of atmospheric conditions but have been limited in terms of data processing and data transfer. Bandwidth for sending data to Earth has been a bottleneck with transmission rates generally below 1Mbps [40], although recent tests of the Lunar Laser Communication Demonstration potentially paves the way for significant increases in bandwidth in future missions [41]. Bespoke and evolving data processing pipelines are often used per mission to process data, although reuse is becoming evident by the Operational Pipeline Unified System (OPUS) pipeline operated by the Space Telescope Science Institute (STScI) [42].

[3] Capable of 17 quadrillion operations per second. At an altitude of 5,000 meters on the Chajnantor Plateau, oxygen levels are only half of what they are at sea level.

Ground based telescopes - Large

Large ground based observatories involving often large consortiums or national funding providing survey and project based data. Until relatively recently many of these observatories have been required to physically move some or all of the science data to data centres for processing. Projects such as Enabling Virtual Access to Latin-American Southern Observatories (EVALSO) have significantly enhanced network connectivity and bandwidth and in 2010 EVALSO provided a 1Gbps connection between South America and Europe, dramatically reducing the bandwidth bottleneck. Due to these bandwidth restrictions, often due to the remoteness of the observatory location, High Performance Computing (HPC) centres are typically paired with these observatories and often use bespoke data processing solutions. Smaller observatories or institutes tend to have less computational resources available, lacking significant investment in IT and either build bespoke HPC solutions or use shared HPC resources when available.

Survey telescopes

Observatories both large and small designed to continually survey the sky have the ability to generate large volumes of data. Data management and processing is a key factor in these systems. Sample rates, image resolution and number of devices capturing data have the potential to exceed data bandwidth capabilities requiring local resources to have the capacity to constantly store and potentially process data.

2.4.1 Optical Space Telescopes

Due to the restrictions on data transfer all space-based telescopes will attempt some form of optimisation at the capture point to reduce the amount of data sent to Earth. Kepler for example performs pixel selection on captured images using a number of specific criteria [43]. Ultimately however all data needs to be sent to Earth for processing. The communication mechanism used by US based space crafts since the early 1960's [44] is the NASA Deep Space Network (DSN), a collection of Earth based antenna in three primary locations, Goldstone, Canberra and Madrid all of which connect directly with the Deep Space Operations Centre, Pasadena, California. The radio link to space crafts is a point to point system using different frequencies and ultimately different data transfer rates. For telescopes in Earth's orbit the Tracking and Data Relay Satellite System (TDRSS) offers

higher bandwidth communications with recent generations ranging between 300Mbits/s and 800Mbits/s depending on the microwave band used. Communications from more distant crafts tend to require higher power which typically results in reduced bandwidth. Further details are provided based on some of the high profile optical telescopes in space.

2.4.1.1 Hubble Space Telescope and the James Webb Space Telescope

The Hubble Space Telescope (HST) was launched in 1990, produces approximately 120 gigabytes of data per week and communicates from low Earth orbit with the TDRSS [45], a data relay and service designed to facilitate communications between Earth and orbiting space crafts. The TDRSS downloads its data to the ground station at White Sands in New Mexico where it is transferred to the Space Telescope Science Institute (STScl) in Balimore for processing. The data pipeline used is the Operational Pipeline Unified Systems (OPUS) [46] which was designed specifically for the HST, but is now used for other programs. OPUS does not actually perform image calibration but stages data for processing and then takes the processed data and stages it for inclusion into the Mikulski Archive for Space Telescopes (MAST) archive. OPUS uses a blackboard architecture of communication using file names providing a communication layer between processes which allows for distributed and parallel processing. Once data is staged for processing by OPUS, the calibration is performed by the Space Telescope Science Data Analysis System (STSDAS) [47], which is an Image Reduction and Analysis Facility (IRAF) based system.

The successor to the HST is the James Webb Space Telescope (JWST) due to be parked at L2 (Lagrange point 2) approximately 1.5 million kilometers from Earth and is expected to launch within the next 5 years. This spacecraft will also communicate with Earth via the DSN and transfer data to the STScl data centre in Baltimore Maryland for processing. The JWST and Kepler will be subject to the limits of the DSN bandwidth (the Ka-band is used for scientific data download, operating at 26 gigahertz) and while Kepler downloads approximately 23 GB of data per month, JWST may generate in the region of 30 GB per day based on an upgrade of the DSN infrastructure [48]. The limits of data transmission using the DSNs Ka-band effectively constrain the problem of data processing to a manageable level at a maximum of 20-30 GB per day.

2.4.1.2 Kepler Mission

The Kepler spacecraft (designed to search for Earth sized planets in other solar systems) sits in a heliocentric orbit (centred on the Sun) and is trailing the Earth by over 10 million kilometres staying within communications range via NASA's DSN using X-band for twice weekly command and status, and monthly Ka-band contact for data download at a speed of 4.33 Mbps [49] which takes 6 hours. Launched in 2009, Kepler monitored approximately 170,000 stars and downloaded approximately 23 GB of pixel data per month [50], which is approximately 6 million pixels captured every 30 minutes of operation. Due to bandwidth limitations, 5% of pixels are downloaded each month [51]. The detector is made up of 42 CCD arrays, which operate on 30-minute capture cycles [52], see Figure 2.8. Each CCD is 2.8 by 3.0 cm with 1024 by 1100 pixels. The entire focal plane contains 95 mega pixels.

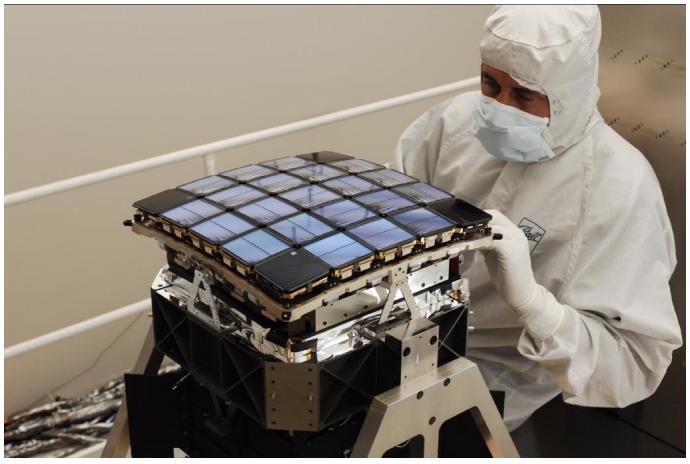


Figure 2.8: Kepler array consists of 42 charge coupled devices (CCDs). Credit: NASA and Ball Aerospace.

Data from the DSN is eventually routed to the STScl and packaged into FITS files before being sent to the Space Operations Center (SOC) at the AMES research centre, Moffet Field, California where data calibration and photometric analysis is performed. The Kepler pipeline, which uses a Java framework to create units of work, can process some data in parallel by allowing some modules to be run in a customised sequences per pipeline. An instance of the framework can run for each of the 42 CCD detectors, which is parallel processing on the dataset at a very high level. Within each instance of the pipeline, operations are run more sequentially for the CAL module [53] which puts images through a series of science algorithms as shown in the Figure 2.9. Each module is composed of a number of procedures written in Matlab.

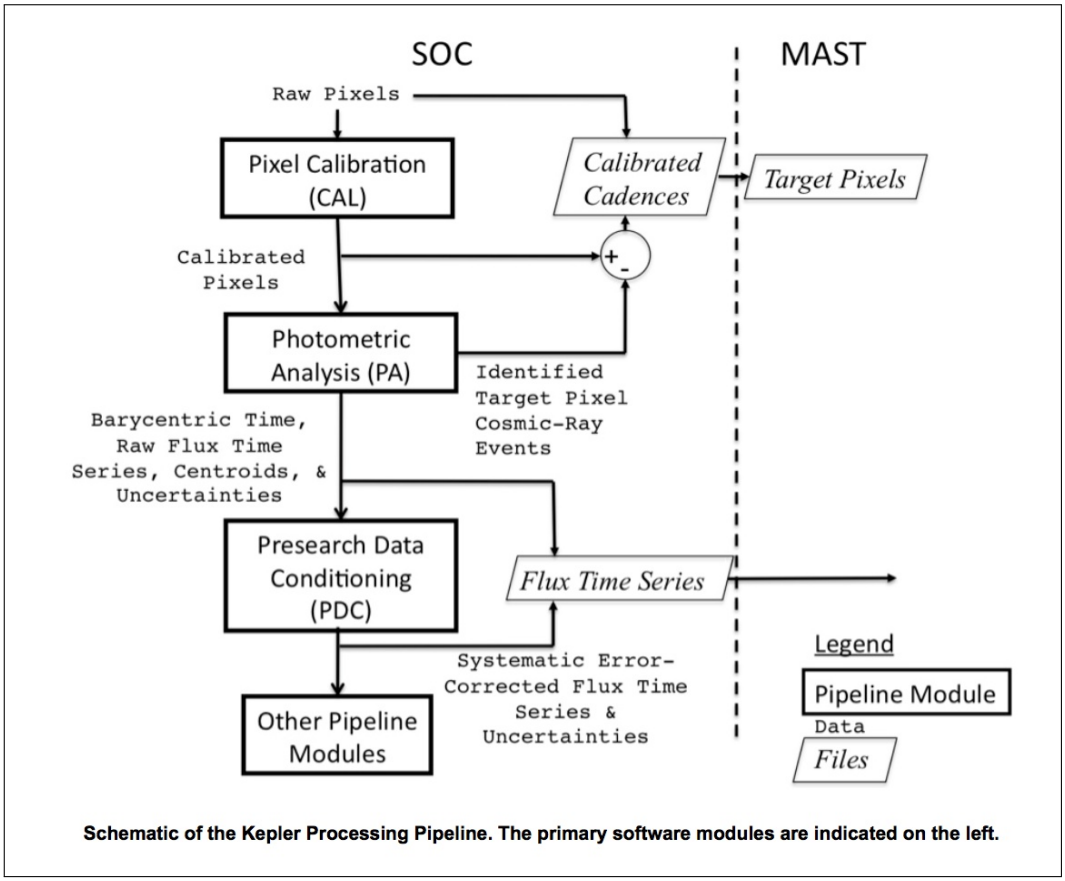


Figure 2.9: Kepler science data flow in the AMES Science Operations Centre (SOC). Credit: NASA

In 2011, the volume of data being generated by the Kepler mission and the associated processing requirements was sufficiently large for the pipeline to require porting to the Pleiades cluster at the NASA advanced Supercomputing Division. So while the data pipeline for Kepler was described as highly parallel, the architecture's ability to scale in the face of a growing data set was evidently limited. This need to fully reprocess all of the raw data as part of the constant reviewing of analysis techniques and parameters was a major driving factor [54]. Porting to the new pipeline required significant investment demonstrating the lack of expandability by the existing pipeline. At the time a comparative analysis was performed against the Amazon Elastic Compute Cloud (EC2) service which was ultimately rejected not on expandability, but on raw network performance [55].

The NASA Pleiades supercomputer is an example of a HPC used for computing intensive workloads. Data must be transferred into the system and jobs must be written and submitted to the central control system for processing. The Pleiades HPC is operated by the Ames research centre in California USA, was brought online in 2008, and is a collaborative effort with SGI. This computer cluster ranks in the top 20 supercomputers and is constantly being expanded. It supports a number of processing environments and is designed to assist NASA with a variety of processing requirements such as simulation and modelling. The Kepler mission migrated to this platform to complete processing of light curves for the hundreds of thousands of stars monitored by the Kepler space craft. The programming environment is Linux with jobs being batched for execution. The development environment is C, C++ or Fortran and the parallel processing components are supported via the SGI Message Passing Toolkit (MPT) . Open MultiProcessing (OpenMP), is an application programming interface (API) designed to allow control and execution of code that takes advantage of multiprocessing, shared memory which is also supported by Pleiades. Like many systems, OpenMP uses a fork-join approach to parallelism as shown in Figure 2.10.

2.4.1.3 Global Astrometric Interferometer for Astrophysics

The Global Astrometric Interferometer for Astrophysics (GAIA) is a European Space Agency survey mission launched December 19th 2013 which aims to provide a three dimensional map of the Milky Way using measures of star position and movement. The spacecraft is positioned at the L2 Lagrange point approximately 1.5 million kilometres from the Earth, in the same location planned for the James Webb Space Telescope. The

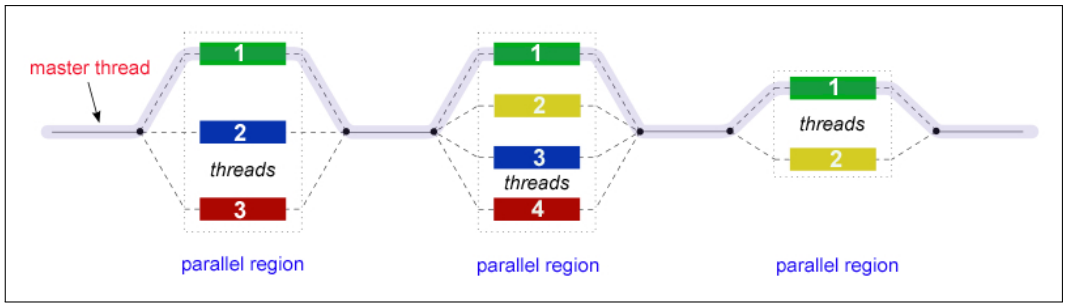


Figure 2.10: Fork-Join processing

mission is designed to survey appoximately 1% of the 100 billion stars in our Galaxy. The spacecraft contains an array of 106 CCDs and the data transfer from the spacecraft will be at a rate of approximately 5 Mbps [56] for eight hours per day, which will delivery over the lifetime of the project approximately 100 TB of raw uncompressed data. The total final storage requirements are expected to expand to 1 PB including data backup and provisional data processing steps. The final data set available to standard researchers when processed will consist of approximately 20 TB.

Due to the format used to capture the data, significant data processing is required to reconstruct the images. Two reasons for the complexity of the data are identified by Mignard et al [57] due to the large number of computations required to process the data, and the level of interconnections between different subsets of the data. The basic data flow for processing the data is shown in Figure 2.11.

Data processing will be performed by a European consortium, the Data Processing and Analysis Consortium (DPAC) which is comprised of contributors from 24 European countries.

2.4.2 Large Ground-Based Telescopes

Large ground based telescopes overcome atmospheric distortion, to some extent, by being located in the highest, and driest locations on Earth. As the number of instruments continues to increase, their data processing requirements also increase. There are a number of other larger ground based telescopes under construction or in planning at present, such as the Giant Magellan Telescope which is a 25 meter telescope expected to generate terabytes of data daily, The European Extremely Large Telescope is planned to have a 40 meter mirror, again expected to have a large volume of data daily for processing [58].

The Gran Telescopio Canarias (GTC) is currently the largest operational optical tele-

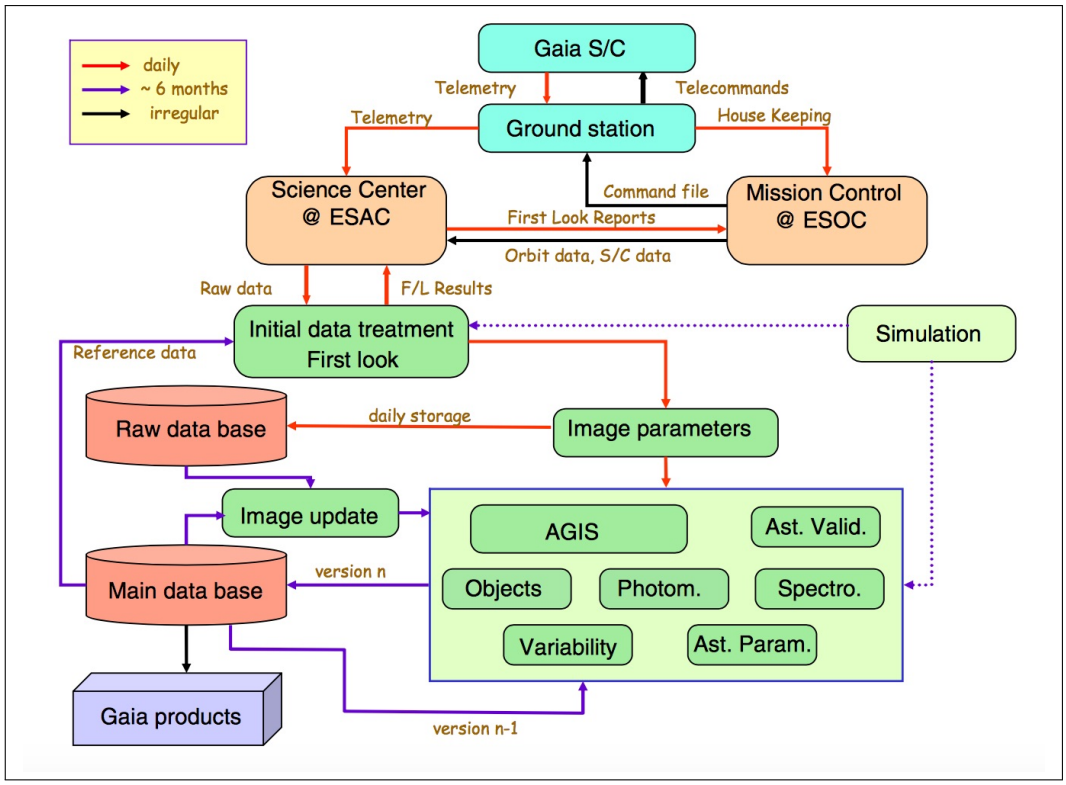


Figure 2.11: Main structure of the data flow in the processing of the Gaia raw data [57]

scope in the world with a 10.4 meter aperture [59]. Observations are scheduled according to a predetermined schedule with raw data provided to participating principle investigators for data processing via FTP. Data is collected on two 2k by 4k red optimised CCD devices giving a total of 4k by 4k pixels [60]. Flat field and bias files are provided along with the raw data. Full resolution image readout can run at a variety of rates, with single image readouts taking between 7.8 and 21 seconds depending on the readout mode. Data generation rates are in the region of 20 GB per hour assuming 32bit pixel storage while operating at 7.8 second capture time.

The Keck Observatory operates two 10m telescopes each containing 36 hexagonal mirrors which operate as a single reflective unit. The observatory is based on the summit of the Mauna Kea volcano in Hawaii and uses adaptive optics to overcome the effects of the atmosphere. Observation time is allocated to partner institutes with raw data made available for download over FTP, SCP or other similar transfer protocols. The LRIS red and blue detectors are comprised of 2 x 2k x 4k CCDs with a minimum readout time of 42 seconds. Raw data capture rates are roughly equivalent to the GTC. Image processing is performed by the participating principle investigator, files are usually available in the

FITS format.

The Large Binocular Telescope (LBT) (Figure 2.12) has two wide field cameras which can operate in tandem, one of which is blue optimised and the other red optimised. The LBT instrument consists of 4 x 2k x 4k CCDs per mirror [61]. Exposure times are given as 0.3 seconds minimum for accurate photometry, with 30 minutes being the maximum observing times before the instrument focus is rechecked. Data is made available for download using a mounted NFS directory from which the observer can copy the data and proceed to process it themselves. While the telescope has the potential to generate 42 Terabytes of data per day, this is not a constant data rate, and is unlikely to consistently hit this maximum data generating rate.

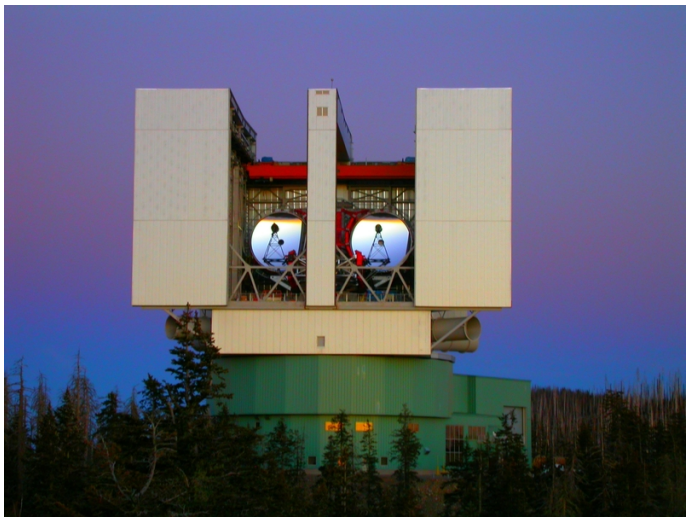


Figure 2.12: LBT Large Binocular Telescope Credit: Aaron Ceranski/LBTO

2.4.3 Survey Projects

In addition to telescopes which have large apertures there are projects which focus not on specific targets but rather sweep large areas of the sky to provide survey data. Surveys are expensive as data constantly streams into the instrument and must be captured, stored and processed. Surveys have the capacity to generate terabytes or petabytes of data. The LSST for example is a multi-million dollar consortium expecting to generate terabytes of data per day.

The Sloan Digital Survey Systems (SDSS) [62], which is currently in operation using a 2.5 metre telescope, reported a maximum data capture rate in the region of 160GB per day in 2004 [63]. This is probably closer to 200GB per day at present. The photometric

data processing pipeline was written at Princeton University, and the data processing is performed at Fermilab using their HPC data centre to process data. The telescope uses a 30 x 2k x 2k CCD array, with approximately 54 seconds between exposures [64]. The imaging pipeline ultimately produces files in the FITS format, but internal formats are used within the Serial Stamp Collecting pipeline (SSC) with a custom written C implementation which processes images in sequence.

Building on the success of the SDSS is the LSST [65] which has an 8.4 single mirror system which will be online by about 2019 and is expected to generate in the region of 20 terabytes of data per night with a resolution of 3.2 billion pixels per image sustaining a data capture rate of 330Mbytes per second [66]. The LSST plans to move tens of terabytes of data per day over high-speed fibre network from the Chilean site to the U.S. Processing is focused on high-speed cores close to the datasets and the belief is that on going advances in server technology will allow a traditional data centre HPC approach to data processing [29]. The data reduction pipeline is still in development and is most likely a bespoke solution. Interestingly they rejected the European Southern Observatory (ESO) Common Pipeline as an option, given the volume of data to be processed and due to concerns over its stability and extensibility.

Extensive archives for astronomical data are being produced and made available to researchers from multiple sources. Diverse data archives exist which are consolidations of optical data from multiple observations or projects such as the ESO Science Archive Facility which contains approximately 1/3 of a petabyte of data and is growing by terabytes per month [67]. Similarly the National Optical Astronomy Observatory (NOAO) provides access to data from its primary facilities such as Kitt Peak, and limited access to other data products from partnerships such as the Keck Observatory with plans to include future data products from the LSST and the Giant Magellan Telescope. The NASA National Space Science Data centre provides approximately 230TB of digital data covering 5,500 distinct data collections, while MAST provides data from the HST and other space based telescopes. There has been a concerted effort to create a central repository of astronomical data by the International Virtual Observatory Alliance (IVOA) which was formed in 2002 [68]. The concept is for astronomical data to be made available from multiple sources/instruments to scientists around the world in a centralised, standardised way. In 2003 it was reported that ESO would have greater than 1 petabyte of compressed data available by 2012 [38]. A distributed architecture to support multiple repositories of data was discussed by Hanisch

in the year 2000 where it is conceded that a single data archive cannot hold all of the data products being produced. [69]. A distributed data archive would seem to logically require a distributed data processing solution using a similar argument. No single data centre could be big enough to process all of the data available.

2.4.4 Radio Astronomy

The issues associated with large data set generation is not limited to optical telescopes or CCD photometry. Additional projects exists which offer the potential to generate Petabytes of data, such as the ASTRON initiated Low Frequency Array for radio astronomy (LOFAR) project [70] and the Square Kilometre Array (SKA) [30]. The LOFAR project requires raw data to be transferred from each of the distributed array nodes to a Central Processing System (CEP) which recombines the data for distribution to offline user based processing. Data recombination was initially performed using an IBM Blue Gene/P supercomputer which provided at its peak, 34 TFlops of processing power before it is transported to the Long Term Archive (LTA) were 20 PB of data is expected to be stored over the next 5 years [71]. Processing is currently performed by a GPU cluster. For researchers to use the data products produced by such projects, large scale computing resources are required to access, download and process the data. Such resources are often costly to construct and difficult to maintain. Challenges for data processing and management also exist within an Irish context as the I-LOFAR `http://www.lofar.ie` consortium aims to join the European wide LOFAR initiative, helping to extend the east-west baseline of the array to just over 1400 kilometres.

The SKA will provide even further data processing challenges with data capture rates measured in the 100's of Gbps. Construction is expected to begin in 2018 with a completion date of 2024. The Science Data Processor consortium will be responsible for focusing on the software, hardware and algorithms required to process the raw data into a series of data products. These challenges are expected to be in excess of anything existing within the field of science at present. Using the LOFAR project as a reference example, the IBM/ASTON Dome project will also review state of the art computing technologies in an effort to build an exascale computing system to process and store data. Further technical challenges will also be presented to researchers seeking to use these data products in terms of data management and data processing and the technology accessible by researchers will require significant advancement in line with that of the SKA itself.

2.5 Data Reduction Software

The astronomical community does not suffer from a poor selection of data processing tools. The variety of low-level tools emphasizes the lack of standards in this area. In the 2007 ESO Instrument Calibration Workshop [72] the session on Data Flow and Data Reduction Software reviewed the history of pipelines, and looked at some of the pipelines proposed across various sites. A brief summary of these technologies is provided in Section 2.5.1.

2.5.1 Fits Formats and APIs

The FITS format (the Flexible Image Transport System) is a standard astronomical image format endorsed by the International Astronomical Union (IAU) and NASA, which was approved in 1981. FITSIO and CFITSIO are C and FORTRAN libraries supported by NASA to access and manipulate the FITS file format. There are many other languages with interfaces to these libraries allowing reduction software to be written in a multitude of languages including Python, Java, Perl, MATLAB and C++. Most data produced for researcher to consume are provided using this format.

2.5.2 IRAF

Image Reduction and Analysis Facility is a standard application used throughout the astronomical community which has been in development since the mid to early 1980s [73], and is a Linux based software package. It provides the closest thing to a standard for data reduction and analysis within much of the astronomy community. Many of the tools referenced by data archives are IRAF based. It is well documented and available on a variety of platforms. Much of the code is implemented in FORTRAN and data processing of files is sequential using batch type processing. Researchers interested in processing data from data archives will often use IRAF as the basic reduction software package.

2.5.3 NHPPS

The NOAO High-Performance Pipeline System (NHPPS) is a python based pipeline which can operate with local processing clusters of software nodes. Connections to nodes are NFS based with SSH access required between each node. Each node must be password-less and have the the NHPPS software installed. Originally based on the OPUS system [74] the NHPPS uses the blackboard architecture for communication across a multi-node

distributed environment which is a multi-queue based system which contains queues of work which can be assigned to pipelines and queues which contain available datasets. Parallelisation within the pipeline is referred to as coarse grain parallelisation, which splits the data into chunks which can be processed independently and runs multiple processes on single servers to reduce CPU idle time. The distributed features of the pipeline are based on a directory server requiring the CPU nodes to be connected via sockets as shown in Figure 2.13. Limitations identified with the pipeline include the requirement for controlled shutdown of the computing nodes in case of data loss due to partial completion of data processing by a node. [75].

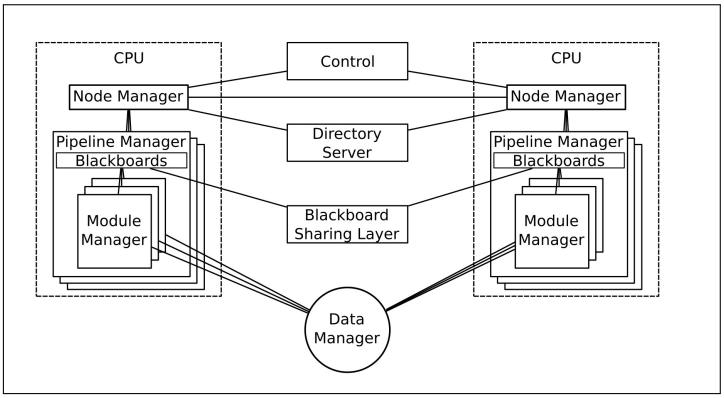


Figure 2.13: NOAO High Performance Pipeline System Architecture [75].

2.5.4 ESO: Common Pipeline Library (CPL)

The VLT instrument pipelines are based on the ESO Common Pipeline Library (CPL), a C based technology closely coupled to the FITS format, which is used as the basis of a number of pipelines, the first of which was the GIRAFFE pipeline [76]. CPL was designed to consolidate different software implementations that existed within various ESO instruments to provide a common technology to assist with rapid data reduction development. Currently all VLT pipelines are either using CPL or are being converted to it. CPL based applications will require faster processors to run pipelines faster and does not provide multi-threaded support although work is on-going to enable multi-threading applications using a thread safe version of CPL [77]. CPL runs on Linux based operating systems, primarily Scientific Linux, and relies on NASAs CFITSIO libraries.

2.5.5 OPUS

In a review of NASAs experience with Data Reduction pipelines over the last 30 years given by Don Lindler [72] the initial Hubble pipeline implementation was considered a step backwards from previous pipelines and the Space Telescope Science Institute, responsible for data processing quickly moved to development of the OPUS pipeline, which was later used in Chandra X-Ray Observatory, the Spitzer Space Telescope and the Far Ultraviolet Spectroscopic Explore. The new pipeline quickly proving to be quite versatile. Initially a VAX/VMS solution, it was moved to Sun/Solaris platform based on an IRAF implementation. Its legacy will continue into the JWST although Mac OSX is a more likely platform with Python/PyRAF as the implementation software.

2.5.6 IUE

Prior to OPUS, one of the first and most successful reduction pipelines was developed to support the International Ultraviolet Explorer launched in 1978 and designed to run for 3 years. It was 19 years later however that the IUE pipeline was eventually shutdown (despite the satellite being in sound working order). To maintain the accessibility of the data, the pipeline was ported to IRAF in 1998 [78]. The HST system, based on Sun workstations is having similar longevity issues with hardware being difficult to replace and repair.

2.5.7 Other pipelines

In addition to the STScI use of OPUS and the ESO focus on CPL there are some primary technologies, which should be briefly mentioned. There are many other examples of pipeline software development within the astronomical community. In 2005 Ó Tuairisg [79] described a distributed computing model using a GRID system allowing images to be processed in parallel. Having data on a shared file system helped reduce data transfer cost and facilitates processing times. This approach builds a large cluster around the data to be processed.

The Apsis package on the other hand is a more traditional system developed in Python to process the early release images from the Advanced Camera for Surveys in 2002. Images and tables were processed via Pyraf and Pyfits [80].

2.6 Distributed Computing

The concept of distributing computing tasks to multiple machines is a well-established discipline within computer science dating back as early as 1969 to ARPNET. Instead of a single processor processing data in sequence, work is processed on distributed computing nodes at the same time. There are many issues with this approach and much research into optimising the process. Broadly speaking there are two areas to consider, the first is how work can be partitioned into smaller jobs, and the second is the practical management of those jobs across a distributed system. A good summary of the pitfalls of working within a distributed system were initially formalised in 1994 by Peter Deutsch, a fellow at Sun Microsystems, when identifying the *seven* fallacies of distributed systems. James Gosling, also of Sun extended these into the *eight* fallacies (shown below) which were later explained succinctly in a white paper by Rotem-Gal-Oz [81].

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

The Cloud Computing model provides an example of a form of distributed computing, enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. A definition of cloud computing was put forward by the National Institute of Standards and Technology (NIST) [82] followed by a special publication in 2012 with a series of recommendations [83]. The report identified five essential characteristics, three service models and four deployment models as shown in Table 2.1.

The cloud model is composed of five essential characteristics, three service models, and four deployment models.	
Essential Characteristics	On-demand self-service
	Broad network access
	Resource pooling
	Rapid Elasticity
	Measured Service
Deployment Models	Private Cloud
	Community Cloud
	Public Cloud
	Hybrid Cloud
Service Models	Software as a Service (SAAS)
	Platform as a Service (PAAS
	Infrastructure as a Service (IAAS)

Table 2.1: NIST definition of Cloud Computing.

For infrastructure as a service (IAAS), the distributed nature of a solution is based purely on the architecture used to build the system. Resources are provided, such as the Simple Queue Service, the Elastic Compute Cloud, with architectural decisions on their implementation left to the user. Platform as a service (PAAS) provides implementations of Map-Reduce, a programming model designed to allow a cluster of computer nodes to perform highly parallel operations on large datasets. Based largely on the LISP functions of similar names, the technique allows for robust parallel operations to be performed inside a cluster of machines. The system aims at being fault-tolerant, providing automatic parallelization and distribution to worker nodes while offering status monitoring. Large volumes of data are processed in parallel by distributed CPU nodes using a distributed file system. The open-source standard for this implementation is Hadoop running on the Hadoop Distributed File System (HDFS) [84]. The technique should allow scaling to thousands of CPU nodes with the right type of problem. While initial reference examples were text search based, additional material has been published demonstrating successful implementation of the Map-Reduce technique in scientific data processing environments. This system provides a programming paradigm that builds private HPC style solutions within

a cloud infrastructure.

Each of the service models outlined by NIST exist as commercial services such as Amazon's AWS [85] an IAAS solution. Clouds can also be constructed using open source technology such as OpenStack [86]. Lenk provides a useful list of vendors and technologies used in cloud construction [87].

Distributed computing is currently in use by the scientific community and a considerable amount of literature exists in reviewing the suitability, cost and performance of clouds and other techniques used for scientific data processing [88], [89] [90] [91] [92] [93] [94]. In December 2011 the Natural Sciences and Engineering Research Council of Canada (NSERC) published a detailed report on the potential role of cloud computing in science based mid-range computational and data intensive workloads [95]. A summary of these findings is provided below.

1. The elastic nature of the cloud is a significant advantage, allowing for elastic provisioning primarily through the use of virtualisation technologies.

2. There is a potential issue in the level of work required for porting existing approaches to the cloud model, including considerable levels of skills required to do so. This upfront cost should be considered as part of the economic analysis when deciding to potentially move this model.

3. Significant gaps exist in managing data within the cloud environment and the process is neither simple nor easily accessible. Scientific workflows are not specifically catered for, and there is an inherent difficulty in exploiting the features of technologies such as Map-Reduce. Other problems include the lack of bootstrap starting points and complex management of cloud environments.

4. Economic benefits come from consolidating resources to improve system utilisation (which it was felt exist in the US Department of Energy). Incorporating aspects of the cloud model into existing data centres is a worthy objective. Private clouds should be considered first before the use of commercial clouds avoiding issues of security, data management and performance of public clouds.

5. Scientific applications have specific requirements that require cloud solutions tailored to their needs.

Various reactions to this report [96] would suggest that the DOE struggle with the adoption of public clouds on the basis of a paradigm shift from HPC to Cloud being non trivial, and the business model of pay-as-you-go not being fully compatible with the scientific requirement of open-ended need for resources. Below is a brief overview of some high profile projects, which are actively engaged in processing datasets using distributed or cloud computing.

2.6.1 Scientific Projects Overiew

The Kepler Project (not to be confused with the Kepler spacecraft mission) uses a Map-Reduce [97] programming model and demonstrates early results in processing biometric scientific data for large HADOOP clusters [98]. One of the concerns being addressed within this paper is that the Map-Reduce model still offers a layer of complexity, which they believe excludes many from gaining access to its potential benefits.

Wiley [99] demonstrated the use of Map-Reduce for image co-addition using the Sloan Digital Sky Survey imaging database to produce a single image from multiple image files with improved signal to noise ratios (SNR). The basic Map function processed each file as a single job, determining if it should be included in the co-added image, and if so, a bitmap was passed on for the Reduce function to take and include in the new image. Of interest is that 100,000 FITS files were processed using this technique and while many files were not required for the final image, the research demonstrates the feasibility of FITs file processing in some form.

The Kepler CCD data is downloaded from the spacecraft and processed on Linux based clusters running 64 nodes with 512 CPU cores. Data is chopped into parallel jobs for processing on worker nodes and a Java Message Service (JMS) used to distribute jobs across 4 clusters [100] [50]. With worker nodes capable of being added, job definition is flexible and the examples provided are single image in and calibrated image out the other end of the pipeline, or multiple images in, and light curve out. Much of the processing is done using MATLAB libraries and Oracle. Similar in many ways to formal cluster computing, distributed elements and pipelines are controlled within a local cluster.

2.6.2 SETI@home

Public resource computing uses the spare CPU cycles of computers to perform data processing. Early research in the use of unused processor capacity included *The worm programs*

as early as 1978 at Xerox PARC using a small set of 100 machines to measure Ethernet performance [101]. But it is the SETI@home project [102] which stands out as the best example of a public resource project utilising spare CPU cycles from millions of online users. Radio data is recorded at source and physically shipped every few days to its headquarters in Berkeley (approximate 2 terabytes of data every few days) where the data is split into work-units, which are accessed by clients across the world. Notable within this approach is that this is a computationally intensive problem, each 350k of data requiring multiple hours of processing. Low bandwidth requirements ensure that even users with modest connections to the Internet can contribute to the overall project. Challenges identified by the Berkeley team have centred on the requirement to issue duplicated work-units (3 sets) to combat malicious users, and the challenges in running and maintaining the server infrastructure to support their data distribution model.

The SETI@home initiative demonstrated a distributed solution for astronomical data processing as shown in Figure 2.14. The SETI pipeline splits work into parallel jobs, which are processed by clients around the world in a distributed manner. A key factor, as already mentioned, in the approach taken by the SETI project is that the I/O rates are low and the computational requirements were high. For kilobytes of data, multiple hours of computation are potentially required [102]. In 2001 the estimated processing completed by the distributed system was approximately 437,000 years of CPU, leading to the claim of being the largest supercomputer in the world (at the time). [103]

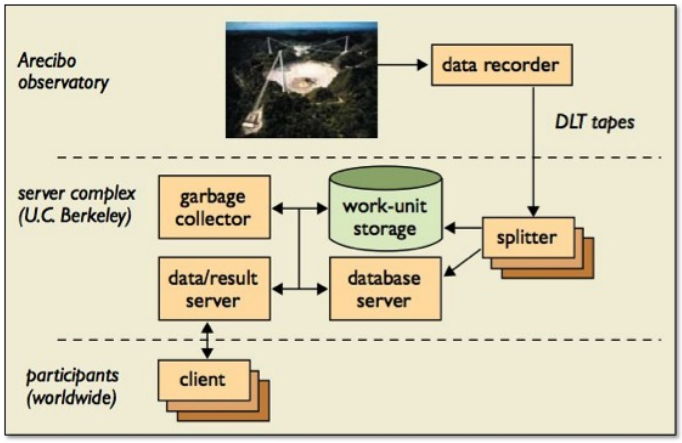


Figure 2.14: Distribution of radio data using SETI@Home (2002)

With most examples presented from the literature so far, existing models for large astronomical data, processing is usually performed on centralised data centres. SETI@home offers an alternative distributed approach but it stresses that the computation to data ratio

should be high.

This research seeks to address the key question of whether a distributed model can be created when the computation to data ratio is low while allowing for tens of terabytes of data to be processed. The distributed model potentially offers a cost advantage to the smaller institute/facility while providing a powerful processing network.

2.7 The data challenge

A dataset was provided by the Blackrock Castle Observatory (BCO), a research facility engaged in high-speed photometry research. The reference dataset contained 3262 cubed FITS[4] files, each containing 10 images and each approximately 512x512 pixels in resolution (0.7MB per image) and the total size of the dataset was 26GB. This data was replicated to simulate a multi-terabyte data. The dataset was generated on September 22nd 2003 at Calar Alto, targeting S5 0716+71 as part of an engineering equipment test of a new hardware/software stack using an Andor CCD device.

A reference image processing speed from BCO was in the order of 1 image processed per second, which is 0.7MB of data processed per second, or roughly 60GB of data per day. This processing pipeline was sequential in nature and used a fixed number of computing devices, which could not be expanded during image processing.

Researchers at the Blackrock Castle Observatory in 2003, using a high-speed Electron Multiplying Charge Coupled Device (EMCCD) detector, operating at 10 images per second, with a detector resolution of 0.2 megapixels, generated approximately 7 megabytes of data per second, equivalent to 200 gigabytes in an 8-hour period. Recent CMOS detectors (Andor Zyla sCMOS 5.5) with a resolution of 5.5 megapixels are under test at that facility and are capable of capturing 100 frames per second. With each pixel value stored as a 32 bit number (8 bytes), the size of a dataset can be calculated as per equation 2.19, where N_{pix} is the number of pixels on the detector, p is the numeric precision used to store the pixel value (typically 8 bytes per pixel), t_{sec} is the time in seconds for the data capture period, and f_{ps} is the number of frames recorded per second.

$$Total_{bytes} = N_{pix} * p * f_{ps} * t_{sec} \tag{2.19}$$

Using the formula in Figure 2.15 the data production rate in terabytes per 8 hours can

[4] FITS is the Flexible Image Transport System digital format used for storing and processing scientific images

be plotted for varying CCD or CMOS pixel resolutions and frame rates. While it may not be feasible to operate at 100 frames per second continuously, it can be seen that at lower capture rates the dataset generated within 8 hours is in the order of terabytes. With capture rates in the order of terabytes per night, large dataset generation is clearly well within the capability of smaller observatories. With the development of robotic farms, even at the lower capture rates, these rates are clearly in the realm of big data.

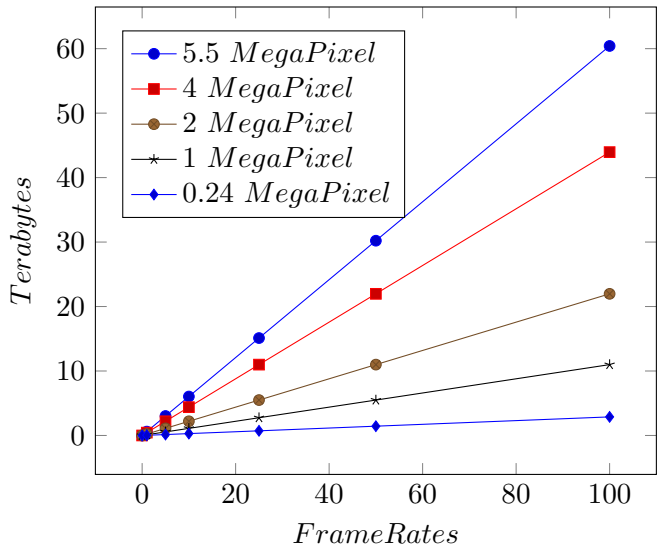


Figure 2.15: Data generation rates per 8 hours for varying camera resolutions running at various frame rates. See Table A.1

2.7.1 Sequential versus Distributed Data Processing

The approaches to processing large datasets are largely dependent on the performance requirement of the task and the volume of data. It is perfectly reasonable to use a brute force approach to solving a problem when the problem is sufficiently small, or computing resources are sufficiently powerful. In these cases results can be produced within a reasonable amount of time so there is no need to process data using any specific method other than sequential. As the volume of data increases, and traditional approaches start to incur unreasonable delays in processing time, further thought is required to address the problem of performance and processing efficiency. Within a typical pipeline, the cleaning and reduction process is a two-step sequential pipeline (Figure 2.16). The first step is reading in a raw image, calibrating all the pixels in the image and writing a cleaned image file. This is performed on all image files. The second step is reading the cleaned image file and calculating a series of magnitude values for each star (or light source) and writing

out a file containing magnitude values. In this pipeline, work is typically performed on a single but powerful server. A sequential processing pipeline must have the capability of processing data at the capture rate to ensure that the pipeline does not back up. If run over 24 hours with a capture period of 8 hours then the slowest speed of the pipeline must be $3x$ the capture rate. As resolutions or frame rates increase there is a race for processing rates to keep pace. With the slowing down of processor rate improvements and the end of Moore's law in sight, alternative processing approaches are required.

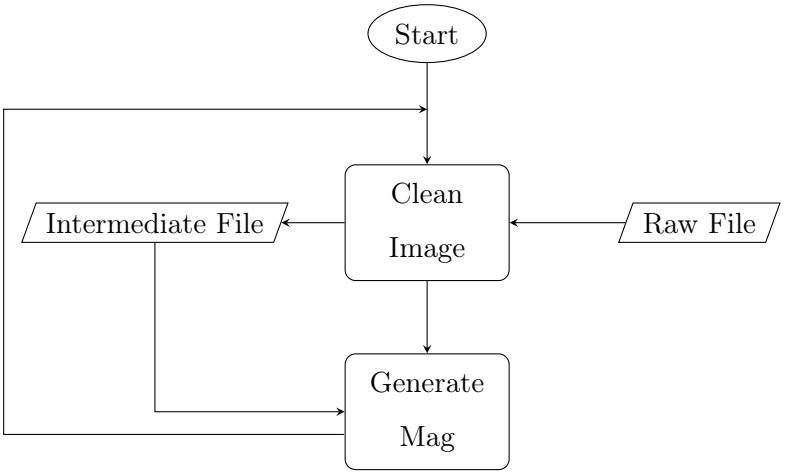


Figure 2.16: A sequential processing pipeline. A raw file is read and has bias, flat field and dark current master frames applied, creating an intermediate file from which instrument magnitude values are calculated before the next file is read. Files are processed in a sequential order.

A distributed processing approach has the advantage of potentially employing large numbers of resources. To distribute the processing of data in a meaningful way, the data must be parallelised to some extent. If the data must be processed in a sequence then distributed computing may not be very relevant. Astronomical CCD data however can be reduced in parallel once the calibration frames are provided with each image. As in Figure 2.17 a distributed pipeline would take in blocks of raw data which can be processed independently and have them queued waiting for a distributed CPU node to process them. Ideally limits would not be imposed by the communications protocols between a work queue and the CPU process. The NHPPS, Kepler and OPUS pipelines use a form of message exchange to allow for multiple CPU nodes to communicate and receive notifications of work to be performed.

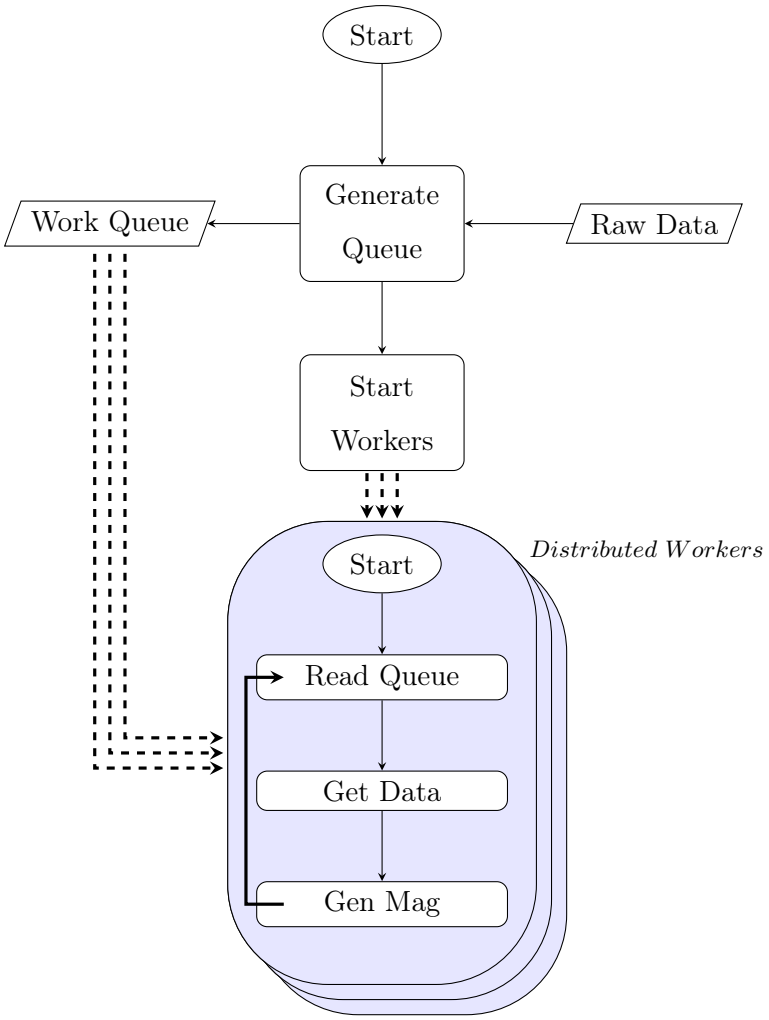


Figure 2.17: Distributed processing pipeline. A queue of work is created of available raw files. Once distributed worker nodes are activated, they use the queue to get work in parallel.

2.8 Conclusions

In this chapter a summary of the history of photometry has been presented along with an overview of the current process involving the use of CCDs for taking images. Since its introduction in the mid 1970's, CCD technology has provided astronomers with the tools to measure the flux from stars in an increasingly precise manner facilitating a significant increase in the precision of magnitude calculation since the early work of Hipparchus.

To improve the accuracy of these measurements an understanding of the sources of noise within the CCD instrument is required and the techniques required to minimise them. The sources of noise and the steps required to reduce their impact on the final

magnitude calculation have been outlined. As the number of images increases and the resolution of the images goes up, the processing required to generate magnitude values is also increasing.

While space based telescopes and large survey systems employ large high performance computing solutions for initial data processing, this data, both raw and reduced is presented to researchers for analysis. The online virtual observatory now provides access to almost petabytes of data world-wide. Without software tools which easily facilitate large scale distributed computing for image processing, the volume of data will become a barrier to performing science. As the number of sources for data increase, possibly even moving to robotic farms, the tsunami of data will overwhelm most researchers and institutes.

NIMBUS, a globally distributed pipeline is described in this thesis as an alternative approach to the data processing techniques reviewed. This requires that the images be processed in parallel with as little work performed as necessary without compromising the quality of the data. Using the analysis of magnitude calculations, it can be shown that data can be safely processed in parallel with the same outcome as a sequential pipeline as is done in some existing pipelines. The methods used to allow the NIMBUS pipeline to scale should ensure that the distribution of computing nodes can truly reach global levels and not be restricted to local network domains. The following chapter discusses the approach taken within the NIMBUS pipeline demonstrating through experimentation the capability of a globally distributed system.

Chapter 3

Research Methodology

The purpose of this research was to determine if a globally distributed network can process terabytes of astronomical CCD image data per day and this chapter reviews the methodology used to make that determination.

The research performed is quantitative, iterative, and experimental based. With the use of distributed non-homogenous resources, operating on shared network environments it was felt that a theoretical analysis would not identify real world system performance limitations. A pipeline was developed in the initial stages of this research as a pilot system against which experiments were performed to ensure the accuracy of the core processing software against a reference BCO pipeline [104]. The name given to this pipeline was *FEBRUUS*, so named after the Roman god of purification. This pilot calibrated the cleaning software and provided an image cleaning rate baseline against which the remaining experiments would be compared. A list of the architectural designs which form the basis of this thesis is shown in Table 3.1.

The initial pilot was performed within the Dublin Institute of Technology but later experiments were performed across multiple locations in Ireland. The final experiment was run using globally distributed resources.

A number of architectures were conceived for the purpose of testing the hypothesis that a global distributed network can perform high speed data processing for large astronomical CCD images. Existing pipelines and technologies have been reviewed and it has been shown that the emphasis on parallel or distributed processing has been primarily confined to HPC systems typically within large data centres. Kepler provides a clear example of a message based distributed system which lacked the capability to scale as the data sets increased, requiring a full system port to the Pleiades supercomputer. SETI@Home while a globally

Experiment Name	Description
FEBRUUS Pilot	Pilot system designed to demonstrate the basic principles of data image cleaning and to validate the accuracy of the reduction against the existing BCO Matlab pipeline
IRAF Cloud	System designed to consider the possible implementation of a series of IRAF virtual instances distributed within a cloud infrastructure
ACN Pipeline	Pipeline designed to test the effectiveness of data compression and distribution in a private cloud using an NFS queuing model.
NIMBUS Pipeline	Pipeline designed to test a global processing pipeline which is dynamically reconfigurable and which can deal with processing nodes joining and leaving without impacting the integrity of the pipeline.

Table 3.1: Experimental designs and pipelines discussed within this chapter

distributed architecture, has an underlying principle that there is a high CPU to I/O ratio. In this chapter multiple architectures which were central to the iterative process used within this thesis are presented and discussed.

In this chapter, the data used within this thesis is described and the two principle distributed designs which form the basis of the research performed, the ACN pipeline and the NIMBUS pipeline are introduced. The initial pilot study, FEBRUUS, is also presented showing the core algorithms used for pixel calibration.

3.1 Dataset

Initial contact with the Blackrock Castle Observatory, Cork, in September of 2009 led to a series of discussions which explored an existing reduction pipeline system in use by the BCO research team to process raw CCD image data. BCO is engaged in high-speed photometry research [105] and generates datasets which contain multiple images per second. The BCO team had implemented a MATLAB based pipeline which had its science data output compared and verified against an IRAF implementation of the same algorithms. The MATLAB system provided, among other things, faster processing rates than the IRAF

solution. The processing rate of this pipeline was approximately 1 image per second, while the capture rate of the CCD device was about 10 images per second. Limitations of the pipeline included the inability to take advantage of additional computing resources, and difficulties in transferring data to alternative systems. A faster data processing pipeline was required. The BCO are also developing a new science instrument, which has the potential of generating terabytes of data. The $TO\phi CAM$ [58] (Two-Channel Optical Photometric Imaging Camera, pronounced toffee-cam, see figure 3.1) uses two CCD97 EMCCDs from Andor Technologies each capable of generating 34 frames/s, approximately 68 Mbytes per second or nearly two terabytes per 8 hours observing. Existing processing rates using the MATLAB pipeline of 1 image/s would require almost 23 days of processing for a full night of observation. BCO are involved in High time-resolution astrophysics (HTRA) and the science objective of generating high-speed photometry images was part of a research project to perform Point Spread Function fitting (PSF) photometry to an estimate accuracy over a timeframe of about one hour [105].

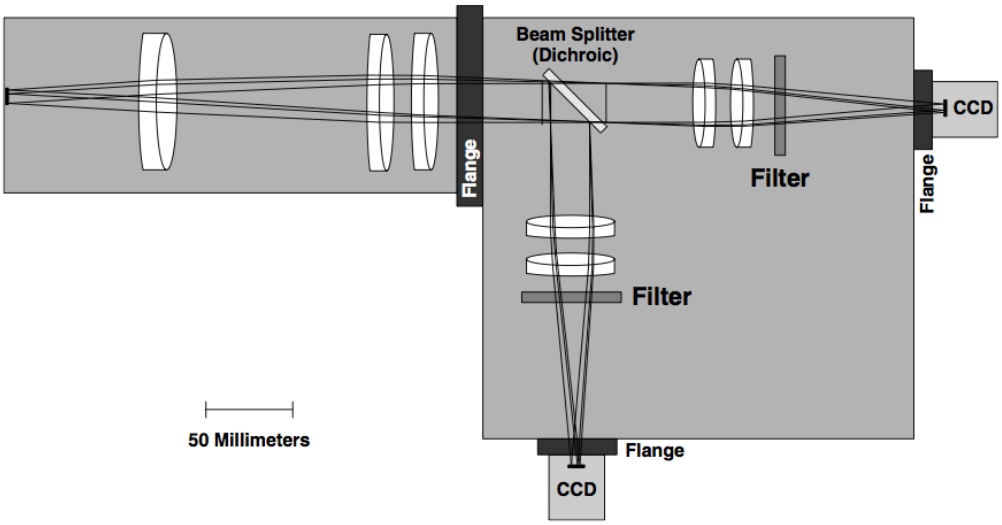


Figure 3.1: The optical layout of the $TO\phi CAM$

A 26 Gigabyte dataset was supplied by BCO consisting of 36,820 images stored in data cubes of 10 data images per file, with data frame integration times of 0.08 seconds per image. This data set was the primary source of CCD images used in all experiments. The dataset was generated on September 22nd 2003 at the Calar Alto Observatory in southern Spain, targeting S5 0716+71 as part of an engineering equipment test of a new hardware/software stack using an Andor CCD device. This dataset acts as a clear reference when discussing existing processing techniques ensuring that future systems deliver the

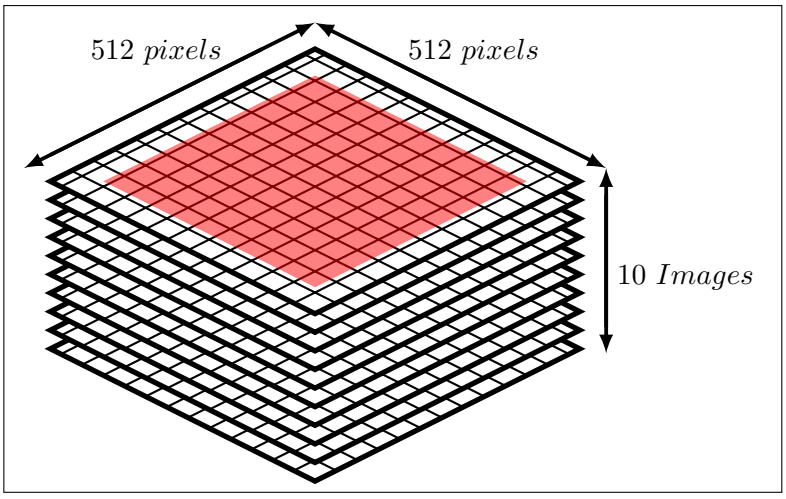


Figure 3.2: Visualisation of the raw BCO data set with SUBRECT region shown in red. The unique data set contains 3682 data cubes, each containing 10 raw images.

same science output. This also provided a point of qualitative and quantitative comparison for new architectures.

The raw data images are stored in an uncompressed FITS (Flexible Image Transport System) file format and are approximately 7MB in size using 32bit integer values. The image borders of approximately 20 pixels have been removed from each raw file reducing them in size. Details of the raw image FITS header file is shown in Table 3.2 giving the precise FITS SUBRECT values, which define a clipped region of the CCD image. In addition to the raw data frames, 200 bias frames stored in 20 data cubes of 10 images each, and 111 single image flat field files were provided. No dark current frames were taken. The flats and bias frames were taken without any SUBRECT which required the correct alignment of master files against raw data image file as shown in Figure 3.2. The total number of raw data pixels within this data set is approximately 6.6 billion pixels. This dataset has already been processed [106] and it is for this reason that it was considered a good reference data set allowing for calibration of the master bias and flat images and comparison of reduced images against the BCO pipeline. Given the short integration times of the data, and the fact that the data processing was focused on differential photometry, the dark current master was not deemed a requirement by the BCO team. While the dark current process has already been described in Chapter 2, dark current removal has not been performed on the data.

Keyword	Value
SIMPLE	= T / file does conform to FITS standard
BITPIX	= -32 / number of bits per data pixel
NAXIS	= 3 / number of data axes
NAXIS1	= 428 / length of data axis 1
NAXIS2	= 426 / length of data axis 2
NAXIS3	= 10 / length of data axis 3
EXTEND	= T / FITS dataset may contain extensions
COMMENT	FITS (Flexible Image Transport System) format defined in Astronomy and Astrophysics Supplement Series v44/p363, v44/p371, v73/p359, v73/p365
HEAD	='DV887 ' / Head model
ACQMODE	= 'Kinetics' / Acquisition mode
ACT	= 1.304200E-01 / Integration cycle time
KCT	= 1.304200E-01 / Kinetic cycle time
NUMACC	= 1 / Number of integrations
NUMKIN	= 10 / Series length
READMODE	= 'Image ' / Readout mode
IMGRECT	= '1, 512, 512, 1' / Image format
HBIN	= 1 / Horizontal binning
VBIN	= 1 / Vertical binning
SUBRECT	= '30, 457, 453, 28' / Subimage format
DATATYPE	= 'Counts ' / Data type
XTYPE	= 'Pixel number' / Calibration type
XUNIT	= 0 / Type of system
TRIGGER	= 'Internal' / Trigger mode
CALIB	= '0,1,0,0 ' / Calibration
EXPOSURE	= 8.000000E-02 / Total Exposure Time
TEMP	= -6.500000E+01 / Temperature
READTIME	= 1.000000E-06 / Pixel readout time
OPERATN	= 4 / Type of system
DATE	= '2003-09-22T02:04:34' / file creation date (YYYY-MM-DDThh:mm:ss UTC)

Table 3.2: Raw Data Fits Header

The 26 GB dataset was sufficient for the initial pilot and the ACN processing pipeline, however for the NIMBUS pipeline, a much larger data set was required. To accomplish this the BCO data was replicated a number of times and stored on multiple storage devices. This was considered acceptable due to the fact the focus of the experimentation was to review processing speed and did not require terabytes of unique data frames. The amount of processing for duplicated image data was identical to processing of unique data. To ensure this, data files when duplicated required unique names to eliminate the possibility of web servers or worker nodes caching the image data and artificially reducing the processing time of the system.

3.1.1 Performance Analysis

The calibration time of an individual CCD frame in most modern computing environments is typically measured in seconds or fractions of a second. Given that images are often taken over multiple seconds, many existing applications process astronomical images using software not specifically optimised for performance. MATLAB-based custom applications and of course the ubiquitous IRAF application offer reasonably easy access to image reduction and processing for scientists, while the CPL and other frameworks are in use by larger centres where specialised calibration workflows are required. However an issue arises, as the number of CCD images increases and the processing time becomes a function of the number of images to process. Data transfer, storage/backup, and retrieval also require careful consideration due to the number of times these operations are performed. Apparently trivial decisions relating to the use of intermediate files when cleaning an image, or the use of compression can have a dramatic influence on overall system performance and resource utilisation. For example, using a two step reduction process where the first step uses Master Bias and Master Flat to clean the pixels, and the second step calculates magnitudes, doubles storage requirements through the use of intermediate files.

In many cases much of the data captured within an image is not used in photometric measurements, and generic workflows that calibrate this data result in work being performed which does not contribute to the accuracy of a calculated magnitude. (E.g. Pixel cleaning is performed on all pixels within the image, including pixels not used in reference object magnitude calculations). It should be possible to only clean a subset of the image and only calibrate pixels required for magnitude calculations, eliminating a high percentage of work from the workflow as shown in Figure 3.3.

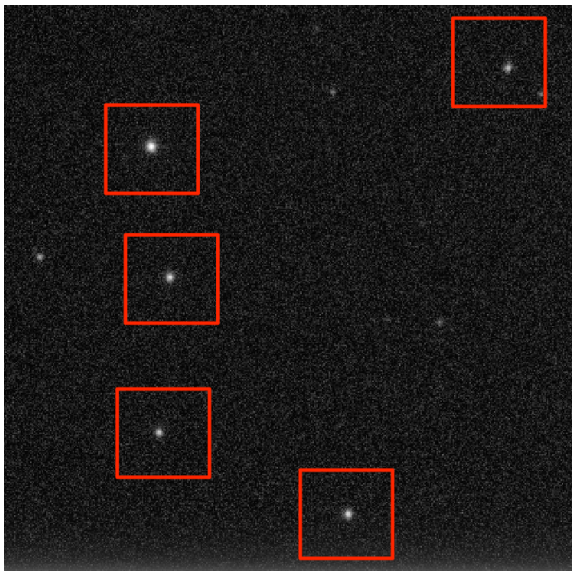


Figure 3.3: Clip regions on a CCD frame Credit: BCO.

One reason why many infrastructures are not easily expanded is that most data reduction tools such as IRAF and CPL are sequential in nature, processing files interactively or in a batch sequence relying on high performance hardware devices to ensure that the data reduction process is kept within a reasonable timeframe.

As more data is captured and uploaded to archives and made available through the Virtual Observatory the amount of data available for research is also increasing. When processing archive raw data, calibration is often performed, and researchers need to find resources for this data processing. Depending on the size of the dataset this could require significant computing resources.

3.1.2 Parallel Data Processing

The first step in working out how data should be logically grouped it is necessary to understand what data is relevant to the operations being performed. If pixels can be determined as non-contributory to the generation of magnitude values then by not including them in the data pipeline both the file IO and the CPU requirements may be reduced.

CCD image calibration is often performed as a series of steps within a pipeline. In many cases this is done for flexibility within the pipeline framework allowing for a modular approach to software development or image processing as shown in the Kepler Science dataflow pipeline in Figure 2.9. It is worth reiterating that this research is focused on the equivalent processing performed within Kepler's CAL and the PA modules. As is evident

from the Kepler pipeline there is a clear sequence that must be followed and which cannot be performed in parallel. This is true for some logical portion of data however and not necessarily true for the whole dataset, or for portions of that dataset. In the case of Kepler, each CCD has its data processed within a parallel pipeline.

This research considers what work can be run in parallel and how this impacts performance. As an example, looking at Figure 3.4, if the output goal of a pipeline is to calibrate a pixel then all pixels can be treated as independent pieces of data which can be processed in parallel without any issue. If the output goal was the magnitude calculation of a star, a larger logical grouping of pixel data into a clipped region of the image is required. Multiple images can be grouped into data cubes, but once the timestamp is preserved then these images can be processed independently and then reassembled into light curves using the time sequence. There are other considerations such as the grouping of images in data cubes. Existing raw data from BCO has a FITS data cube containing 10 images but there are other configurations possible. In many cases the reality of the file I/O costs of the systems will have an impact on data processing.

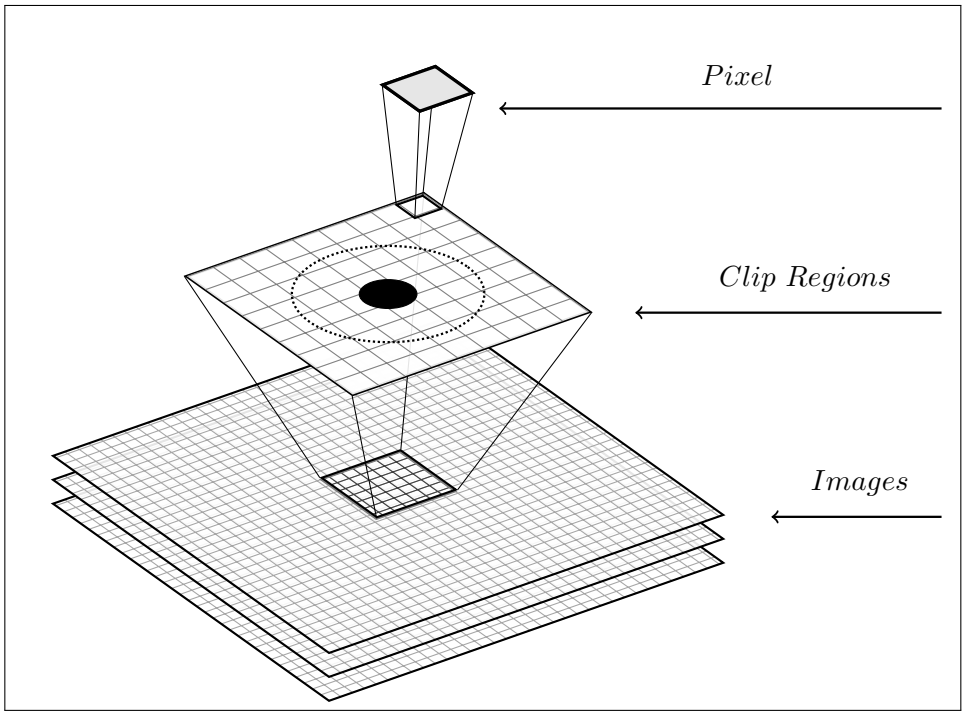


Figure 3.4: Identifying parallel processing opportunities

3.2 System Designs

In addition to the FEBRUUS pilot study, three primary designs have been considered and reviewed with two of them fully implemented. Experiments were devised for the implemented designs to test their limits and capabilities, with lessons learned helping improve the next evolution. The purpose of these experiments was to test the performance, elasticity and flexibility of the architectures. Three primary designs were considered, and the final design incorporated the key components of the other three.

- IRAF Virtualisation. Use of standard IRAF installations, virtualised and deployed into a cloud environment, using torrents as a distribution and replication technique for CCD image data. A torrent is a file sent via the BitTorrent protocol which is initially incomplete. The file continues to download form multiple computers using a torrent client which locates additional copies of the file on different computers.

- The ACN Pipeline: A distributed private cloud using commodity servers, with a lightweight data processing appliance and a private centralised queue to advertise work.

- The NIMBUS Pipeline: A distributed public cloud based on virtualised and physical servers using a distributed web queue to advertise work.

The key reason for using a distributed model was to allow resources from multiple locations to participate in the pipeline. This requires that data is accessible to processing nodes, which obtain work from a central queue, download data, process it, and upload results. The ACN and the NIMBUS pipelines were built to facilitate experimentation.

3.2.1 Pixel Calibration - FEBRUUS Pilot

The aim of the pilot system was to implement a series of programs designed to clean raw CCD image data using the identified formulas presented in Chapter 2. The output of this system was a series of calibrated images, which used Master Bias and Master Flat images. A summary of the key objectives of this pipeline are presented below.

- Write a series of CFITSIO based programs to generate Master BIAS, and Master FLAT frames

- Write a lightweight program to perform pixel calibration of raw CCD images using the Master Flat and Master Bias frames
- Develop a series for tools for comparing FITS files to calibrate the results with results from BCO
- Determine the performance of a CFITSIO program compared to the MATLAB pipeline from BCO
- Provide a benchmark for future processing pipelines
- Learn the process of image pixel calibration.

By implementing the standard pixel calibration algorithms it was possible to become more familiar with the process to ensure decisions regarding what could be performed in parallel and what could be excluded from magnitude calculations. By focusing on basic calibration it was possible to obtain a basic benchmark for image cleaning. By using a CFITSIO library within a C program, the speed of imaging cleaning within a sequential environment could be estimated as a basis for comparison within a distributed environment.

In total seven CFITSIO based tools were built to support the pilot. Details of these tools are provided below in Table 3.3

Program	Function
gmb.c	Generate the master bias frame through the combination of 200 bias frames using an average pixel value per pixel coordinate
gmf.c	Generate master flat through the combination of 111 flat frames using median pixel values per pixel coordinate
bmf.c	Bias reduce the master flat by subtracting the master bias values for each pixel coordinate
nmf.c	Normalise the values within the bias reduced master flat frame
rrf.c	Reduce raw file by performing pixel calibration using the master bias and normalised, master flat frames
cfd.c	Compare two fits files by checking all pixel values to see if they are identical
lde.c	List all of the pixel values or a subset of pixel values within a FITs file

Table 3.3: Basic tools developed to calibrate against the BCO MATLAB pipeline

3.2.1.1 Generate Master Bias

The purpose of this program is to create a single two-dimensional image, which contains an accurate representation of the systematic wide additive values, present in all pixels. The program takes in as input a directory location, which contains a number of BIAS files and the name of the required output master bias file. Debugging is offered at different levels to have various amounts of data generated to the screen, which can be redirected and reviewed. The output of this code is validated against a file supplied by BCO. While initial differences were found when comparing the output of the *gmb.c* program and the BCO supplied file, this was found to be related to a difference in precision for the stored values during the average calculation. When the pipeline system performs calculations is uses DOUBLE_IMG when setting the precision settings on the output MasterBIAS file.

Unless the corresponding file from BCO uses similar precision, the following differences in values can occur. BCO Data: 233 compared to Pilot Data: 232.8800000000. Since this is an additive source for noise it may well make more sense to store this in an integer value within the Master Bias, however the primary aim is to retain as much high precision processing as possible.

Algorithm 1 describes the operation of the gmb.c program and how a master bias frame is created from a stack of bias images. The master bias is created one row at a time by summing pixels from the same position across multiple images and then dividing them by the number of images. This process is repeated for all pixels on a row by row basis. The flowchart for this program is listed in Appendix B.

3.2.1.2 Generate Master Dark

The data provided by BCO did not require dark frame cleaning due to the specification of the CCD device used, the image exposure time and the fact that differential photometry was being performed. For completeness Algorithm 2 is presented which describes the process of creating a bias reduced dark current image master.

3.2.1.3 Generate Master Flat

This process is similar to *gmb.c* in that there are multiple images which combine to form a single image. There is an important difference however in that the flat files are created by exposing the CCD to light which introduces the possibility of cosmic rays striking a pixel

Algorithm 1 Generate Master Bias

```
1: procedure GMB(row, col, img)          ▷ row and columns width and no. of images
2:     ImageRow[col]
3:     PixVal[col]
4:     BiasVal[col]
5:     for i = 1 to row do
6:         for j = 1 to img do
7:             ImageRow ← readframerow[i, j]
8:             for k = 1 to col do
9:                 PixVal[k] ← PixVal[k] + imageRow[k]   ▷ sum all values for each pixel
10:            end for
11:        end for
12:        for k = 1 to col do          ▷ calculate avg value for each pixel in the row
13:            BiasVal[k] ← PixVal[k]/img
14:        end for
15:        writerow(BiasVal)            ▷ write out 1 row of master bias values
16:    end for                                                  ▷ go to the next row
17: end procedure
```

causing its value to dramatically increase. The use of an average value across images which includes these values could affect the overall result. Instead pixel values for a specific x,y coordinate across all images are sorted to obtain the median value.

The median value for a pixel coordinate is not the final step in creating a master flat file as it must be cleaned by eliminating the bias (using the master bias file) and then have the pixel values normalised. For the purpose of calibration with the BCO system these step were broken down into distinct steps.

The output from the *gmf.c* program was compared to the reference file provided by BCO to ensure that resulting values were identical. Due to the fact that a median is required and that the source data is a whole number, there is no need for higher precision than INT, although a LONG data type was used. The *bmf.c* program uses the output of the gmf.c and subtracts the bias value associated with the specific pixel coordinate from the master bias frame. Sample output from this process is shown in Appendix B.

The discussion on pixel to pixel variation in sensitivity to light earlier in Chapter 2

Algorithm 2 Generate Master Dark

```
1: procedure GMD(row, col, img)          ▷ row and columns width and no. of images
2:     ImageRow[col]
3:     PixVal[col]
4:     DarkVal[col]
5:     for i = 1 to row do
6:         for j = 1 to img do
7:             ImageRow ← readframerow[i, j]
8:             for k = 1 to col do                    ▷ sum all values for each pixel
9:                 PixVal[k] ←= PixVal[k] + imageRow[k]
10:            end for
11:        end for
12:        for k = 1 to col do
13:            DarkVal[k] ← PixVal[k]/img − BiasVal[i, k]
14:        end for
15:        writerow(DarkVal)
16:    end for
17: end procedure
```

looks at the quantum efficiency of a pixel for a particular wavelength of light. Exposing a CCD to a flat field of light should, in theory, produce the same value in each pixel after subtracting the bias value. In reality the same value is not seen, which may be due to a number of reasons such as the use of a non-uniform light source, the quantum efficiency is actually variable across the CCD or dust particles which create shadows or patterns across the CCD. If on average a pixel collected 1000 electrons during an exposure, then to normalise a specific pixel the value needs to be divided by the average (in this case 1000). The result of this calculation is a normalisation value which can then apply to actual readings in object files either reducing the recorded value (because this specific pixel collects higher than average electrons) or increasing the recorded value (because this specific pixel collect lower than average electrons). The flowcharts for these programs are listed in Appendix B.

Algorithm 3 is a combination of these programs that creates a final master flat frame from a stack of flat field images.

Algorithm 3 Generate Master Flat

1: **procedure** GMF(row, col, img) ▷ row and columns width and no. of images
2: $ImageRow[col]$
3: $PixVal[col][img]$
4: $Medians[col]$
5: $Normalise[col]$
6: ▷ Generate Median pixel values from multiple Flat Frames
7: **for** $i = 1$ **to** row **do**
8: **for** $j = 1$ **to** img **do**
9: $ImageRow \leftarrow readRow[i, j]$ ▷ read in a row from each image
10: **for** $k = 1$ **to** col **do**
11: $PixVal[j][k] \leftarrow PixVal[j][k] + imageRow[k]$
12: **end for**
13: **end for**
14: **for** `(k = 1 to col)` **do**
15: $Median[k] \leftarrow MEDIAN(QSORT(PixVal[k])) - BiasVal[i, k]$
16: $writeRow(Medians)$
17: **end for**
18: **end for** ▷ Go to next Row
19: **for** $i = 1$ **to** row **do** ▷ Generate AVG pixel value
20: **for** $k = 1$ **to** col **do**
21: $Total \leftarrow Total + ReadMedianRow[i][k]$ ▷ total all Flat Fids values
22: **end for**
23: **end for**
24: `AVG=Total/(row*col)` ▷ Calculate avg value for Master Frame
25: **for** $i = 1$ **to** row **do**
26: **for** $k = 1$ **to** col **do**
27: $NormaliseVal[i][k] \leftarrow ReadMedianRow[i][k]/AVG$
28: **end for**
29: `writeRow(Normalise[i])` ▷ write out 1 row of master flat values
30: **end for**
31: **end procedure**

The following three primary steps are incorporated within the algorithm.

- Calculate the median pixel value across multiple flat field frames
- Remove the bias value for that pixel using the master bias frame
- Normalise the pixel values across the bias reduced image

The master flat frame is created a row at a time by obtaining the pixels value from the same location across multiple images, sorting them, and identifying the median value. This process is repeated for all pixels on a row by row basis. The purpose of using a median is to eliminate from the calculation any effects from a cosmic ray which could significantly distort an average value. To obtain a median value, all values must be sorted, and odd and even numbers of images must be accounted for within the algorithm. Sample output from this process is shown in Appendix B.

3.2.1.4 Pixel Cleaning Image Files

The *rrf.c* program is the primary appliance in the pilot system as it is responsible for cleaning the raw data images using the Master Bias and the Master Flat files. The formula used by this program has already been described in Chapter 2. For all pixels in a raw object frame the corresponding pixel in the same location in the Master Bias frame is subtracted. This value is then divided by the corresponding pixel in the same location in the Master Flat frame, which adjusts the value up or down depending on the relative quantum efficiency for the pixel as compared to the average efficiency across the CCD. This final value is then written to the output file as the newly cleaned value.

Algorithm 4 demonstrate the simplicity of the actual calibration operation once master frames have been created and pixel alignment between the master frames and the raw images has been achieved. Where the raw images and the master frames are different in size, as with the BCO dataset, clipping is required on the master frames, or the use of an index into the master files to ensure pixel alignment with the raw images is correctly performed.

3.2.1.5 Supporting Tools

Pixel level comparisons were performed on the pilot master bias, master flat and reduced raw images agains the BCO equivalents to ensure that the basic processes were correctly

Algorithm 4 Calibrate raw image

```
1: procedure CALIBRATE(row, col, img)                    ▷ row and columns width
2:     file raw                                          ▷ file handle for raw image
3:     file cal                                          ▷ file handle for new calibrated image
4:     file mb                                           ▷ file handle for Master Bias
5:     file mf                                           ▷ file handle for Master Flat
6:     file md                                           ▷ file handle for Master Dark Current
7:     CalPix[col]                                       ▷ 1 dimensional array to store calibrated pixels
8:     for i = 1 to row do
9:         for k = 1 to col do
10:            biasVal ← ReadPixel(mb, i, k)
11:            flatVal ← ReadPixel(mf, i, k)
12:            darkVal ← ReadPixel(md, i, k)
13:            rawVal ← ReadPixel(raw, i, k)
14:            CalPix[k] ← (rawVal − biasVal − darkVal)/flatVal        ▷ clean pixel
15:        end for
16:        writerow(cal, CalPix)                         ▷ write out 1 row of new values
17:    end for
18: end procedure
```

implemented. The full data set provided by BCO was processed using the *rrf.c* program from a local NFS disk on a Linux system to determine the maximum processing rate using a sequential processing approach. The *rrf.c* program held the master frames in memory and opened and processed each image data cube in sequence until all were processed.

3.2.2 Virtual IRAF instances - Design 1

The initial design concept for cloud based distributed computing was for a virtualisation of existing reduction pipelines, which ran on instances in a cloud environment as shown in Figure 3.5. It was considered that an IRAF virtual machine appliance could be constructed and copies of the virtualisation instance could be run within a variety of cloud infrastructures. Using pre-fabricated images, IRAF based virtual machines could be created on any supported hypervisor.

Figure 3.6, shows a torrent [107] based solution for data movement and replication

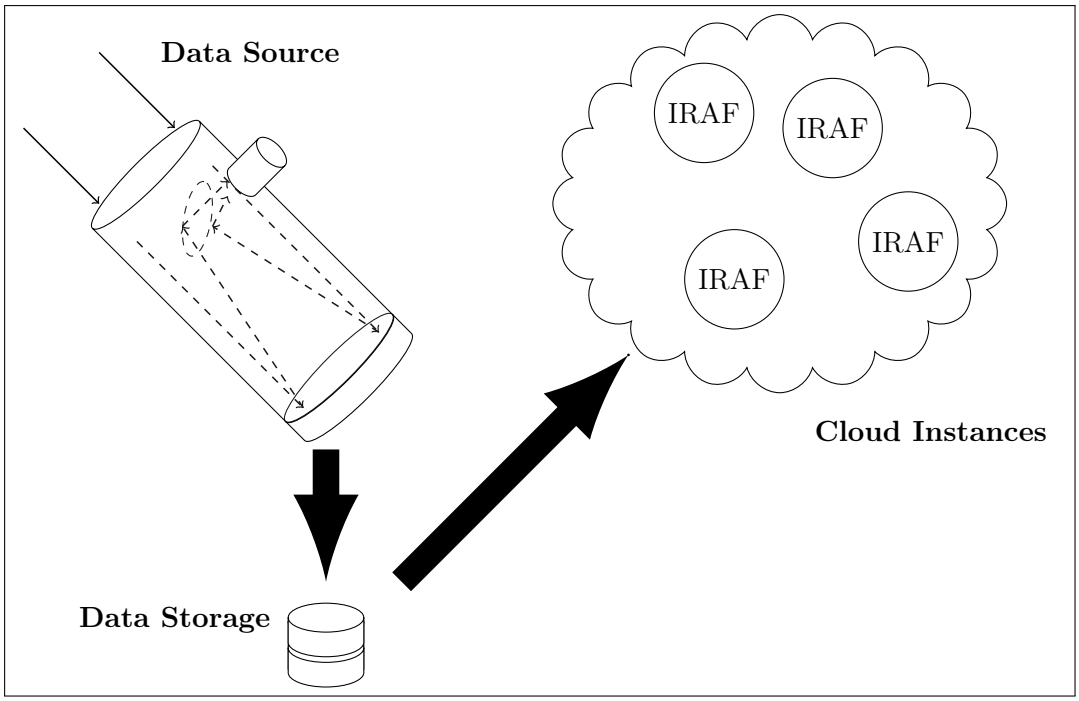


Figure 3.5: IRAF virtual instances in the cloud

where there were multiple copies of data distributed to various storage nodes. This design considered a message swapping system similar to the blackboard model within the OPUS pipeline. When data is uploaded to a remote storage location it is first replicated to other data centres using a torrent based infrastructure. All data uploaded to a data store would generate a message containing information about the data such as where it was, and what processing was required. Worker nodes running in virtual machines would read the message queue to consume a job which it would then download, process and upload to a location also specified in the message.

Issues arose with this design quite quickly however. The complexity of the IRAF solution and the size of a full virtual machine image imposed a large data copying, installation and management overhead for creating computing nodes. Instead of virtualizing an IRAF instance, a smaller appliance could be virtualized or installed natively on a number of platforms. A smaller appliance could be distributed more easily allowing for a greater number of computing nodes to participate using fewer resources on the computing node. Other reasons were also identified which ultimately led to a rejection of this design.

- IRAF contains components which are not relevant to the cleaning process.

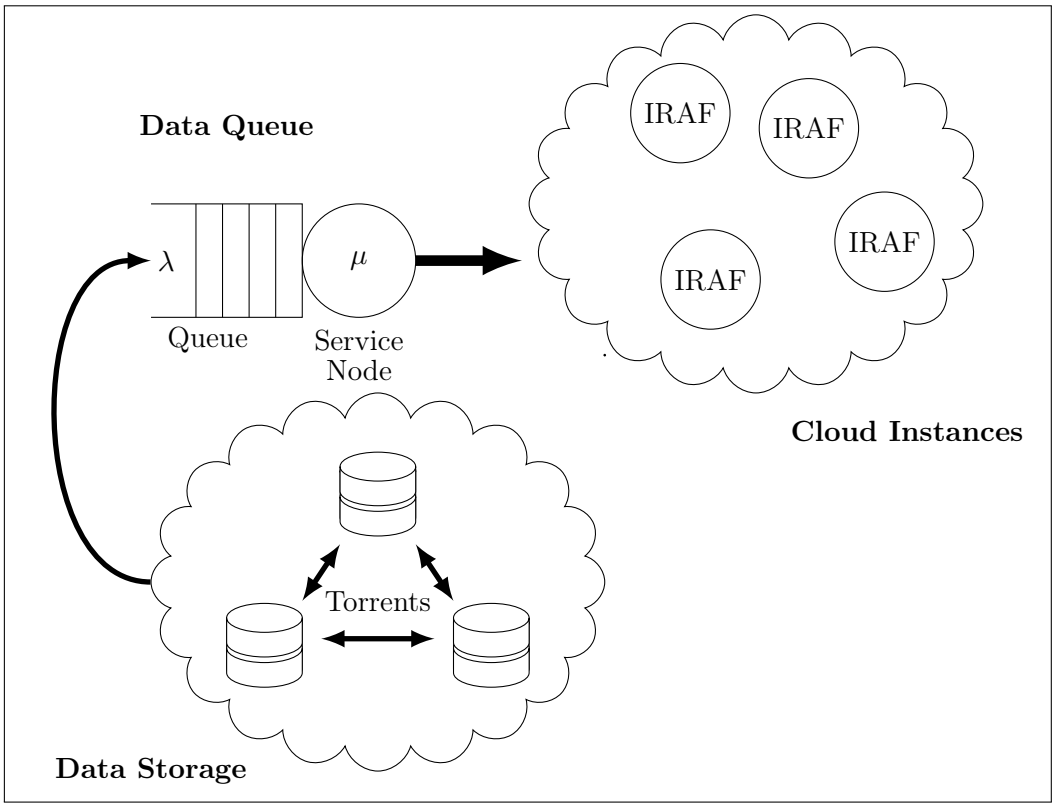


Figure 3.6: Torrents for data distribution using a central queue. A messages μ is downloaded by an IRAF instance containing a torrent file λ which is incomplete. The entire file is downloaded using a torrent client within the IRAF instance which connects to multiple torrent servers.

- IRAF performance is reasonably slow [108].
- The installation and configuration of IRAF was non-trivial.
- There are many manual steps in using IRAF although batch processing is possible.
- IRAF would reduce the requirement to fully appreciate the details of CCD image processing which was considered an essential skill to develop in order to potentially understand future optimisations.

The BCO group are no longer using IRAF for cleaning and analysis of CCD images instead using MFITSIO with MATLAB. MATLAB instead of IRAF was also considered and rejected due to the license requirements for MATLAB, which would limit the number of deployable instances when processing a large dataset in a distributed environment. OCTAVE, a free alternative to MATLAB was also considered but this is based on the libcfitsio library and it was considered more appropriate to implement CCD reduction with the CFITSIO library using standard processing techniques. A lightweight processing utility was considered preferable to facilitate a faster deployment of software to distributed computing nodes.

Another issue with this design was the use of torrents for data distribution. Many firewalls are configured to block traffic of this type. A private torrent server was built internally for testing but this would limit data transfer to internal nodes. The data store was still considered a generic storage facility and not specific enough at this point. This design also required the creation of torrent files for all files. Unless the data was to be downloaded by multiple sources there appeared also to be a considerable overhead in storing the data multiple times to facilitate downloading. Unless the same data was to be downloaded by multiple users then the swarm based download mechanism would not be used which meant that the benefits of torrents were not being utilised. For large archive storage sites however this might be a reasonably interesting protocol to support.

3.2.3 The ACN Pipeline - Design 2

Following the initial IRAF design, a proposal was made to HEAnet, (Ireland's National Educational and Research Network), to facilitate a peer-to-peer network between DIT, and BCO for the construction of an Astronomical Data Processing Cloud. This proposal was for a private network connection between the two institutes where data was moved to

the DIT data centre and replicated to other storage facilities. Linux computing nodes are distributed to locations within this cloud to process image data.

The Astronomical Data Process Cloud was expanded to include the Institute of Technology Tallaght in Dublin (ITTD) to allow a wider distribution of nodes as shown in Figure 3.7. The private layer-2, point-to-point network, was constructed in 2011 with the assistance of HEAnet.

This pipeline reduction software was developed in two phases. The first phase integrated the previously verified calibration software used in the FEBRUUS pilot and the second phase focused on increasing the functionality of the image processing by producing magnitude values for stars. This required the creation of a single appliance written in C which cleaned pixels and then performed aperture photometry, producing a range of magnitude values for each star using multiple aperture sizes. This program ran on a Linux ubuntu 12.04 server finding a filename using a shared NFS based queue before downloading the actual data file from an Amazon Simple Storage Service (S3) bucket. The S3 storage contained compressed versions of the BCO dataset, and the compressed and upload process was included as part of the running pipeline. Table 3.4 summarises the experiments used to test the ACN pipeline.

Reference	Measure	Experimental Objectives
Exp:ACN1	ACN-APHOT Performance	Determining the performance of this program by running in multiple modes using various storage devices. Two step versus one step cleaning is examined.
Exp:ACN2	Storage Performance	Determining the Impact of the location of the storage devices and their ability to support multiple queries.
Exp:ACN3	Data Compression	Compression of data reduces the size of data for both storage and transfer. Data compression techniques and approaches are considered.
Exp:ACN4	Data Transfer	Data stores are compared in terms of data transfer rates.
Exp:ACN5	Pipeline Limits	Determining how fast the pipeline can operate within the proposed architecture.

Table 3.4: ACN performance experimental set

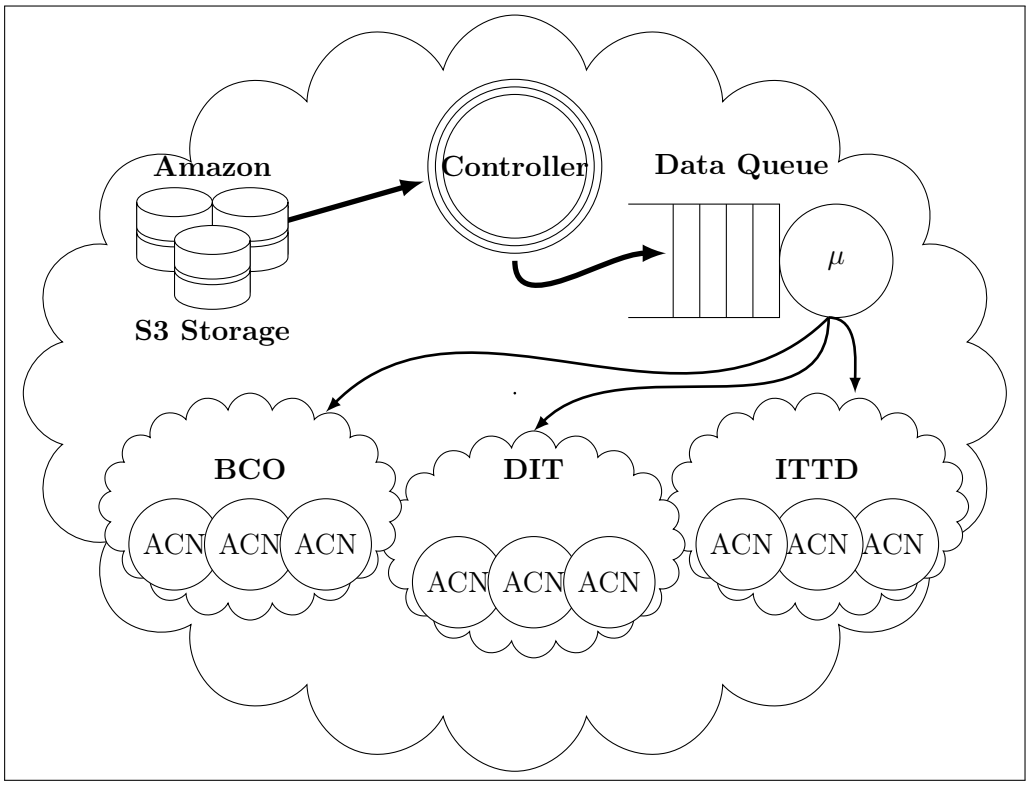


Figure 3.7: ACN Pipeline: Multi-institute private cloud using AWS S3 storage. Worker nodes download messages μ from an NFS based queue.

3.2.4 NIMBUS Pipeline - Design 3

This design moved the queuing system to the distributed Amazon AWS Simple Queue Service (SQS) [109] to allocate work to multiple computing nodes. This allowed global access to the queue potentially increasing the number of systems which could participate in image processing. The data set was replicated and pushed to a number of different NGINX webservers within the DIT and within the AWS cloud to simulate distributed data sources. In this pipeline, the data was already compressed and in position before data processing began. The AWS SQS message queue service was used to store the location of work, which a worker node could use to contribute to the overall experiment. The basic workflow is for a controller to instruct a storage node to publish the address of all files in its data store, and to then activate AWS EC2 nodes. Each node upgrades its software when activated, by downloading the latest version of the package software with instructions on how it should operate. The node then proceeds to take messages off the SQS system, download the file named within the message and processes the file. Once results are obtained they are written to an AWS S3 facility. Nodes can be added or removed at any time. Any work not completed is automatically reinserted onto the queue for another node to take. A node can run multiple threads, the number of files downloaded can be configured, the queue which is used can be updated and the software used for processing can be updated centrally. Multiple web servers containing data can all contribute to the worker queue, the instances can be of any size or configuration once they can run the software stack downloaded from the software distribution web server. The NIMBUS architecture is shown in Figure 3.8.

A summary of the variables controlled for within the architecture is shown below with Table 3.5 providing details of high level experiments run.

- Number of worker nodes
- Size/type of the worker nodes
- Number of webservers activated
- Location of webservers activated
- Number of worker instances per node
- Batch size for a worker instance to process
- Length of time for the experiment to run

The combinations of these variables causes a potentially large set of experiments to be required to develop a comprehensive view of the system behaviour. In total approximately 100 different experimental combinations were run to provide a reasonable view of the system capabilities.

Reference	Measure	Experimental Objectives
Exp:NIM1	SQS Performance	Testing the read and writing times of the web message queues
Exp:NIM2	Single-Instance Node Performance.	Determine the variables which affect the performance of the overall processing power of a single instance.
Exp:NIM3	Multi-Instances Node Performance.	Focus on scaling the number of instances up to 100 looking for factors which could affect the scalability of the system.
Exp:NIM4	System Limits.	Identify the full scalability of the pipeline and to identify strategies to continue improving the system performance
Exp:NIM5	System Scalability.	The scalability and flexibility of the system is tested taking into account any limits observed in previous experiments.

Table 3.5: NIMBUS: performance experimental set

There are six components central to this pipeline; (i) data capture and staging, (ii) serving archive data, (iii) distributed worker queues, (iv) distributed data processing, (v) results storage and (vi) monitoring. Each component is required to operate continuously and asynchronously, allowing for resource utilisation to be varied without interrupting the overall pipeline. While tested to a processing rate of 200 terabytes per day, the experiments were not at the limit of possible processing rates, with the primary restriction being a lack of additional resources available. Some of the larger experiments utilised over 10,000 processing worker threads across 100 distributed servers.

Data Capture Cloud

The data capture cloud consists of multiple distributed telescope sites containing CCD devices which record image data to a local storage device. Lossless data compression on images is performed to reduce the bandwidth required for data transfer.

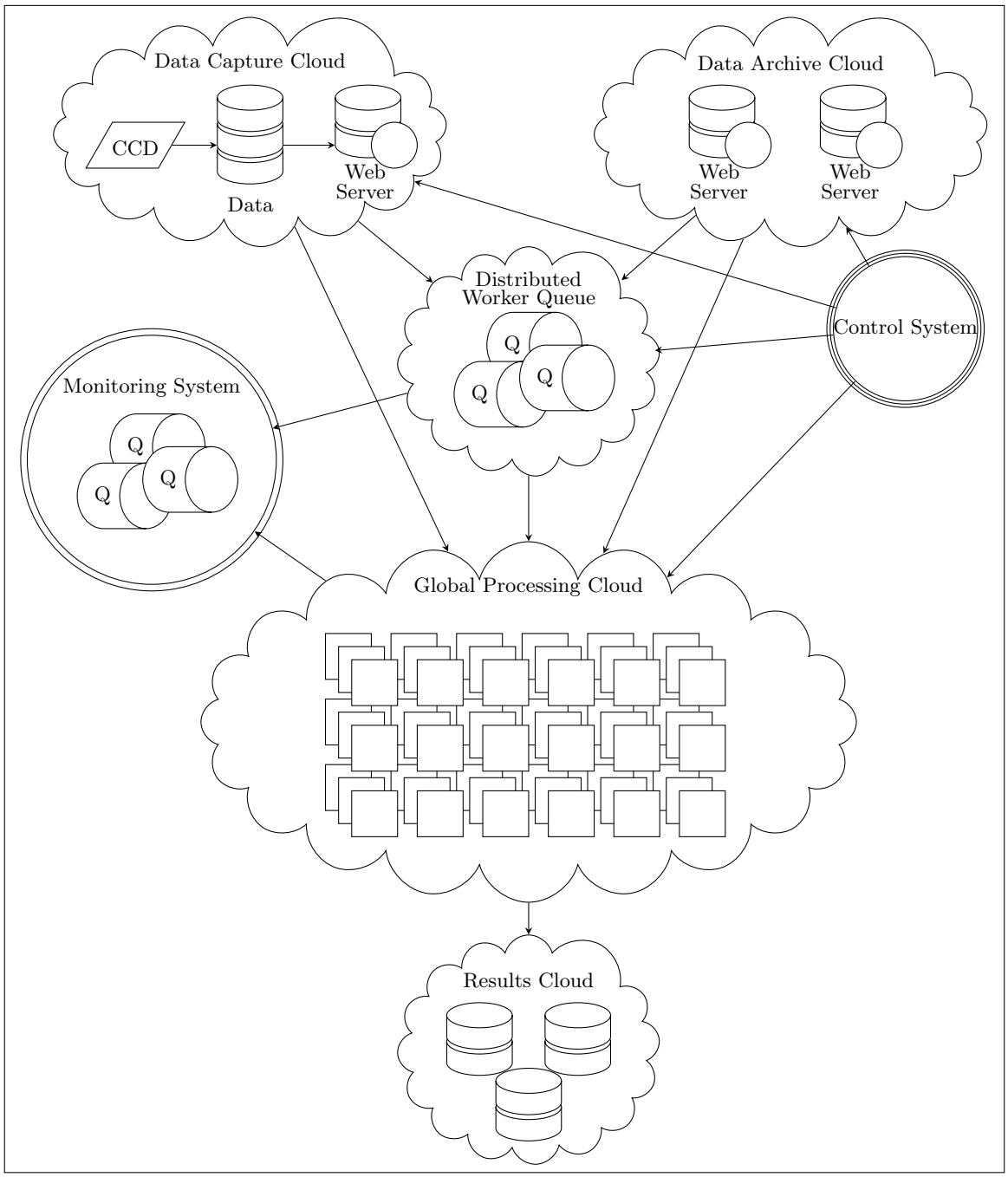


Figure 3.8: NIMBUS architecture.

Image clipping reduces images to only include the pixels used in the calculations of magnitude values. Compressed, clipped images are stored on fast storage disks attached to static web servers which serve http requests from the global processing cloud. The Webservers inform the distributed worker queue that there is work to be performed.

Data Archive Cloud

The data archive cloud consists of multiple distributed websites containing image datasets. Images will already be compressed and possibly clipped. The images are stored on fast storage disks attached to static web servers which serve http requests from the global data processing cloud. The Webservers advertise files to be processed via the distributed worker queue.

Distributed Worker Queues

When the worker web queue is informed of a file available for processing it stores the url of the file in a simple message which is available for worker nodes to read. The web queue ensures that only one copy of a message can be read from the queue at a time. When a worker completes its processing it permanently deletes the message. If a worker node fails to complete the processing of the image, the message will eventually reappear on the queue. This ensures that the overall system is resilient against compute node failures[1].

Global Processing Cloud

Worker nodes contain an initialisation boot script which installs worker sandboxes using tools downloaded from a predefined URL. These tools ensure that the work performed is configurable both in terms of the work to be performed and web queues to listen or write to. Worker nodes within the processing cloud can be located anywhere in the world. Workers can join or leave the processing cloud at any point, without impacting the overall processing pipeline.

Results Cloud

When a worker has completed its work, the resulting data file is uploaded to a distributed storage facility and a message is then written to the result queue that contains the URL for the location of the upload file. Using this queue, a processing

[1] A compute node failure is when a process terminates prior to the completion of the reduction process

cloud can be reconfigured to read the message queue to identify the URL of the result and to download results to a central location if required.

Monitoring Cloud

For both experimental and control reasons a series of message queues is constructed to observe the performance of the pipeline. Worker nodes use queues to log their progress, and all queues are monitored to determine their read and write rates.

3.2.5 Conclusion

The architectures considered within this chapter have each contributed in an iterative way to the ultimate development of the NIMBUS pipeline. The methods used for evaluation are quantitative, focusing on the overall processing speed of the pipeline architecture, while considering the expandability and flexibility of the system.

The pilot system established a baseline for performing accurate calibration ensuring that the tools required at the centre of the pipeline are representative of the work performed on CCD images by the existing BCO MATLAB pipeline.

The initial IRAF virtualisation design proposed a central queueing system designed to allow multiple virtual instances to obtain work. The method for data replication however was torrent based and practical limitations within secure network environments prohibited its use. The virtualisation image was large and would be an impediment to the dynamic provisioning of computing nodes within a pipeline. This design was not implemented.

Design 2 focuses on the data compression and distribution using a central queue and experiments are focused on identifying the limits of the system, seeking bottlenecks and limitations within data compression and staging. Design 3 focuses on global scaling of the processing nodes using a web queuing model to ensure worker nodes can be easily added to the pipeline. Experiments for the ACN and NIMBUS pipelines were designed to identify the limits and capabilities of each design and more detail regarding their architectures and performance are presented in Chapter 4 and 5.

Chapter 4

The Astronomical Compute Node (ACN) Pipeline

Abstract

The scientific community is in the midst of a data analysis crisis. The increasing capacity of scientific CCD instrumentation and their falling costs is contributing to an explosive generation of raw photometric data. This data must go through a process of cleaning and reduction before it can be used for high precision photometric analysis. Many existing data processing pipelines either assume a relatively small dataset or are batch processed by a High Performance Computing centre. A radical overhaul of these processing pipelines is required to allow reduction and cleaning rates to process terabyte sized datasets at near capture rates using an elastic processing architecture. The ability to access computing resources and to allow them to grow and shrink as demand fluctuates is essential, as is exploiting the parallel nature of the datasets. A distributed data processing pipeline is required. It should incorporate lossless data compression, allow for data segmentation and support processing of data segments in parallel. Academic institutes can collaborate and provide an elastic computing model without the requirement for large centralised high performance computing data centers. This paper demonstrates how a base 10 order of magnitude improvement in overall processing time has been achieved using the *ACN pipeline*, a distributed pipeline spanning multiple academic institutes.

Doyle, P; Mtenzi, F; Smith, N; Collins, A and O'Shea, B "Significantly reducing the processing times of high-speed photometry datasets using a distributed computing model", Proc. SPIE 8451, Software and Cyber infrastructure for Astronomy II, 84510C (September 24, 2012); doi: 10.1117/12.924863;

4.1 Overview

The Astronomical Compute Node pipeline (ACN) [110], was designed and implemented as part of this research to demonstrate the feasibility of performing distributed data processing using large data volumes hosted on a cloud based storage solution. The ACN also validated the effective use of a centralised queue to manage multiple workers, facilitating a load balancing data processing solution for disparate server types. This pipeline architecture uses a distributed private cloud to demonstrate the scaling nature of the system across multiple Institutes of Technology. The ACN pipeline builds upon the *rrf.c* program discussed in the previous chapter, which was a core component of the FEBRUUS pilot system. The *acn-aphot.c* program uses the NASA CFITSIO library, and extends the cleaning features of the *rrf.c* program to include magnitude calculations for each reference star within an image. This pipeline incorporates hardware capable of running a compiled instance of the *acn-aphot.c* program to contribute to the processing of FITS images. The pipeline distributed design is shown in Figure 4.1 and the BCO dataset was used. The following components are central to this pipeline and were designed and developed specifically for this experiment. These components are described in more detail within the experimental architecture section of this chapter. The code used for this pipeline is available on github in the following repository. `https://github.com/paulfdoyle/acn.git`

- Storage Control. Storage services including data compression, transport to central storage and downloading data to each of the ACN nodes.
- Queue Control. Management of the NFS locking system including queue creation and management.
- ACN Nodes. Individual workers and the infrastructure required to perform work.
- Node Control. Activation and control of connected node systems including defining work to be performed.

A private point to point IP network was constructed between three institutes[1] with 8 IBM eServer 326 machines connected to the network at each location with each server operating as an ACN node as shown in Figure 4.2. Multiple FreeNAS storage devices were added to provide the ACN nodes with common utilities, and a central NFS queue was implemented to allow nodes to obtain work. In addition to the physical IBM servers, VMware based ACN instances were added to the network running on 4 Sun Microsystems x4150 servers. The internet was accessible through a gateway router which provided network address translation of addresses thereby allowing all nodes to access the S3 storage service.

[1] Dublin Institute of Technology, Blackrock Castle Observatory (Cork Institute of Technology), and the Institute of Technology Tallaght, Dublin

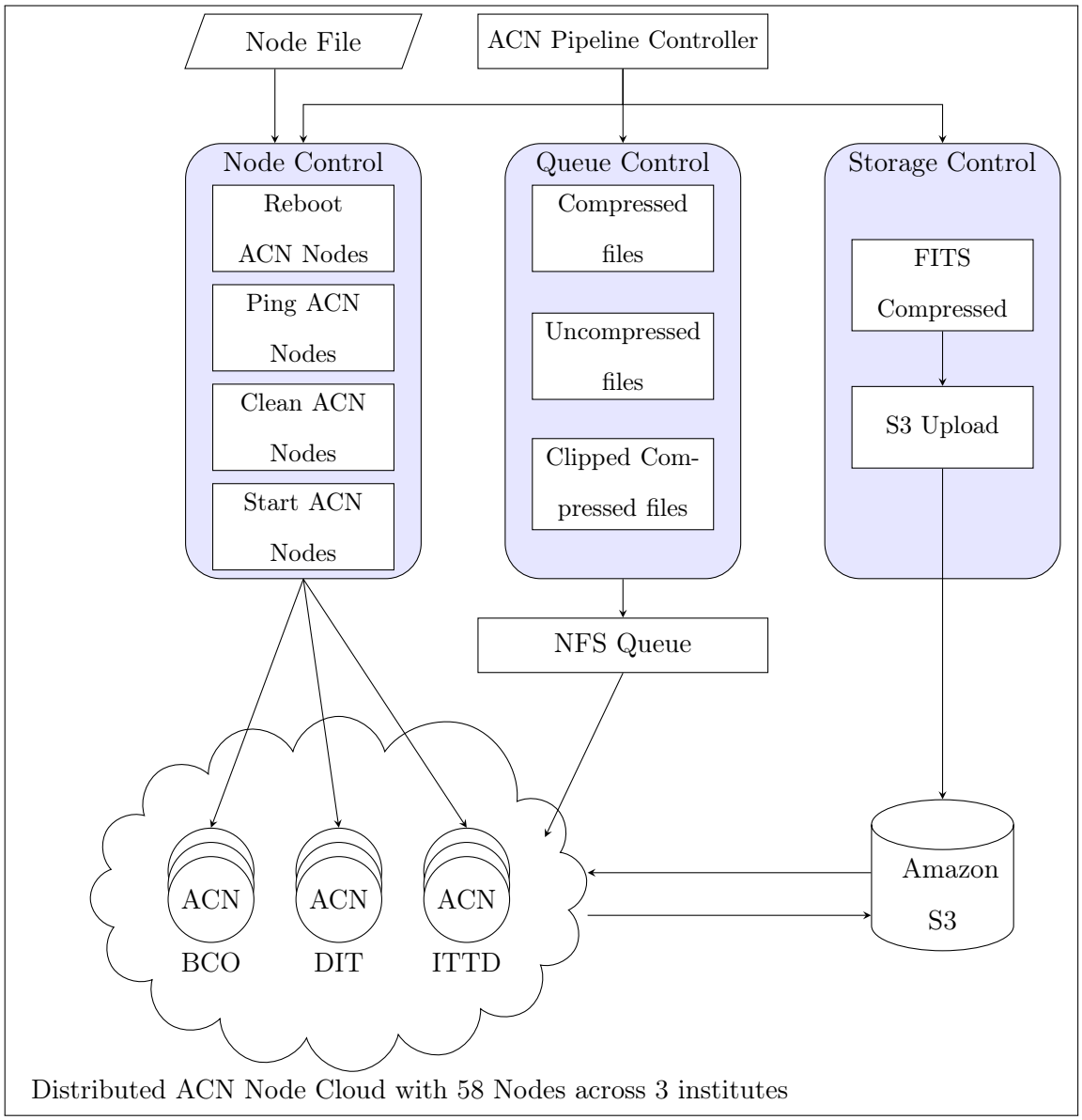


Figure 4.1: ACN distributed design.

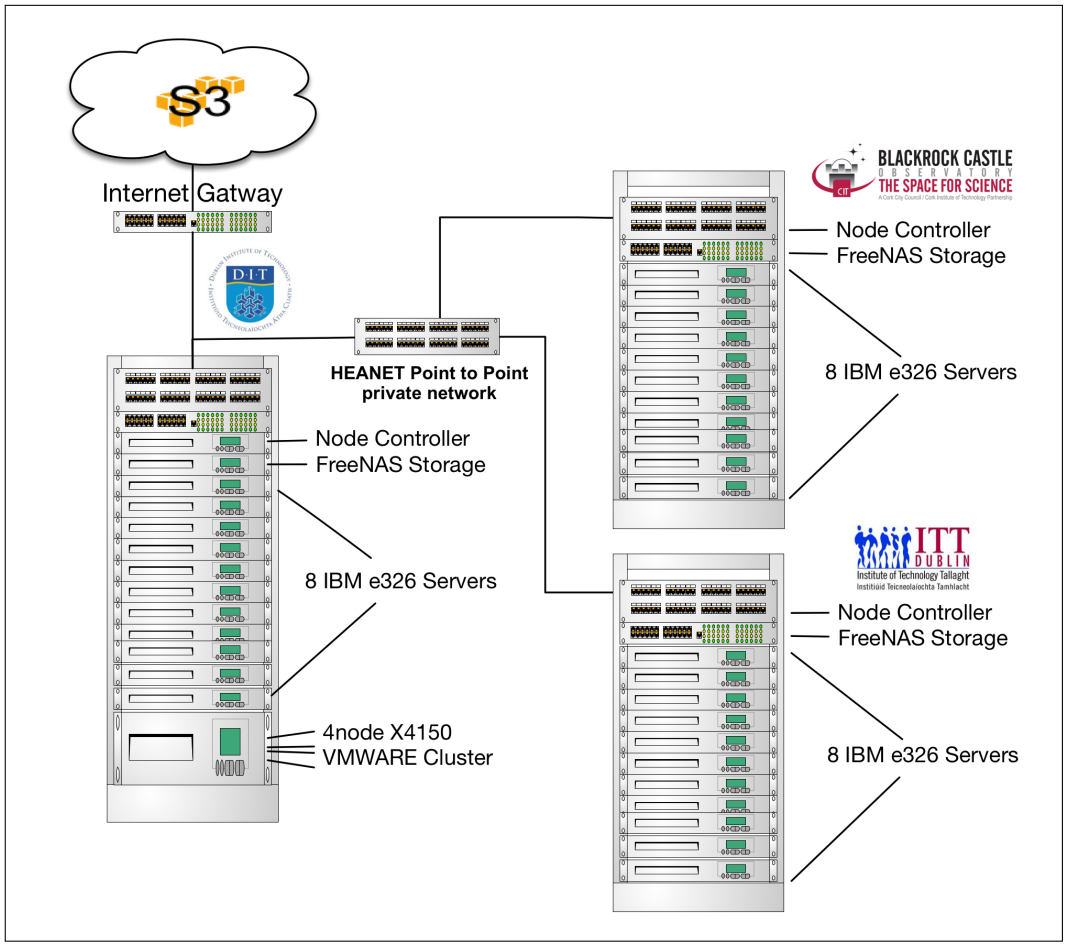


Figure 4.2: ACN network diagram connecting multiple institutes.

4.2 System Architecture

This pipeline has a number of distinct components, each of which collaborate to allow for a high processing rate of images. All of the systems are implemented within a Linux shell environment or are written using the *C* programming language using the *CFITSIO* library. The system spanned three institutes, each of which contained processing nodes for data processing but only one of which, the central control within the DIT, managed the queue of work to be performed. Data was stored in a publicly accessible location which all nodes had access to, and a private queue only accessible from within the Hybrid Cloud system. The four primary components of the design as described in this section.

4.2.1 Storage Control

A high performance data storage system is essential to this design, as all nodes must have fast access to raw data files to allow for scaling of nodes without blocking on the I/O performance of the disks. Initially a single central FreeNAS [111] storage node was used which contained a full copy of the BCO dataset, but bottlenecks in processing file requests were quickly evident due to the limited number of disks physically used. Data was then replicated across a number of FreeNAS nodes, however the equipment used clearly didn't offer a scaling solution and bottlenecks continued to be observed. The Amazon's Simple Storage Service (S3) was selected instead given that it is a scalable storage solution that is available to all processing nodes, simplifying any assumptions regarding file location. The S3 storage also provides access to files using the HTTP protocol on port 80, which is typically not blocked by institutes. The Linux utility s3cmd was used to create storage "buckets" on S3 and transfer data to the S3 system. Each file is downloadable using a command line browser *wget*, a Linux utility. The S3 storage supports concurrent file requests and while it has a significant delay in reading a single file, that delay is not cumulative and remains constant over multiple concurrent file requests [112]. The S3 storage provided publicly accessible files to all computing nodes and buckets were populated with compressed, clipped and uncompressed versions of the dataset for use in performance experiments.

File compression was used to determine the difference in transfer costs for different file sizes. Lossless compression was achieved on FITS files using *fpack* [113], a utility available within the CFITSIO library.

Compression was performed by a multi-core Dell 410 PowerEdge Server with 64 GBs of RAM. The assumption to test was that the cost of compressing a file and the transfer time for the compressed file would be less than the transfer time of an uncompressed file. Compression was seen as a potentially essential step to reduce transfer times across networks. A third version of the dataset was prepared using a clipped region [2] of each image. For the calculation of a magnitude

[2] A clipped region refers to a portion of an image which is identified for a boundary which is used to

value for an object, the pixels around the star which contribute to that calculation are selected to create a subset of the file. Each of these clipped regions is stored as a FITS file and can be compressed independently of each other. The more reference stars, the higher the number of files created. To simplify the process of image calibration, clipped master Bias, Dark, and Flat Field images can be created with identical dimensions. Figure 3.3 shows the actual pixels used in magnitude calculations, and in this dataset there are 5 clip regions. The processing of clipping this dataset involves an increase in the number of files in the dataset but a reduction in the overall number of pixels.

4.2.2 Queue Control

The process by which the ACN nodes determine what file is available for processing is the NFS file locking system which only allows one process to change a filename. The queue is a directory of empty filenames on an NFS mount point shared by all ACN nodes. Each filename corresponds to a data file to be processed on the S3 storage system. The queue is created using a utility which traverses the list of all available files and creates the set of empty files. The files conform to the following naming convention with the files being initialised with the *Queued* name state tag.

1. Queued-<filename.fits>
2. Locked-<filename.fits>

Each ACN worker will traverse the queue as a directory listing looking at all filenames in turn. When it finds a file with the *Queued* tag in the filename it will attempt to rename the file to add the *Locked* tag. If the file is renamed successfully then an NFS lock was successfully obtained and the ACN node will then download the file from S3, process it and upload the results file to an S3 storage location. If the NFS lock is not secured then the next file in the directory is checked.

This NFS locking system was both simple and reliable although there is a delay in obtaining a lock as new nodes are brought online and have to traverse the entire list of files from the start until it finds a file not yet processed (not marked as locked). The greater the number of files, the longer it will be before a newly active worker node will be in finding a file to process. Similar modifications to the process are possible, such as file removal once the uploading has occurred to S3. The principle algorithm used to secure a lock is given in Algorithm 5.

4.2.3 Worker Nodes

The core function of the ACN node is to find work in the queue, download image data, process it and upload the result, and to continue this process until there is no more work available. The *acn-aphot.c* program, which runs at the heart of each ACN node, is a C program compiled for

identify a subset of the image

Algorithm 5 File Locking using NFS

```
1: procedure OBTAINLOCK(QUEUEDIR)                    ▷ Directory containing queue
2:     nameparts[]
3:     for filename in $(ls QUEUEDIR ) do
4:         nameparts ← $splitfilename(-)    ▷ Split filename into strings separated by -
5:         if nameparts[0] ≠ "Queued" then                          ▷ Request NFS lock
6:             mv $QUEUEDIR/$filename to $QUEUEDIR/Locked − $nameparts[1]
7:             if $? = "0" then                                     ▷ NFS lock obtained
8:                 ProcessFile(nameparts[1])
9:             end if
10:        end if
11:    end for                                  ▷ go to the next file in the queue
12: end procedure
```

multiple Linux systems including the Mac OSX. Its primary function is to generate a valid magnitude estimate for stars in an image, ensuring that the pixels used in the calculation are correctly calibrated. To support this process there are a number of supporting scripts within the node which are activated when the node first starts. The basic process for each ACN node is given in Figure 4.3. The queue of work is made available on an NFS shared drive. Once the worker node is activated it cycles through the available files until the queue is empty.

The init scripts within the node will mount the central NFS directory containing the queue and worker utilities. Once the directory is mounted, all cleaning data and utilities within the node are removed. The latest version of the cleaning tools, including updated master frames, and any node control scripts will be downloaded and installed onto the node. This is required to ensure that with minimal effort, each node is always running the latest software versions. A script will traverse the NFS queue and when it succeeds in obtaining a lock on a file, it downloads that file from S3 to a working directory. The *acn-aphot.c* program is then called to perform the image cleaning. If the program completes without any errors, then the results file is uploaded to the S3 storage.

The ACN node can process compressed, uncompressed or clipped files. All that is required is that the latest cleaning scripts downloaded are configured to uncompress data as required. This process continues until there are no files left in the queue.

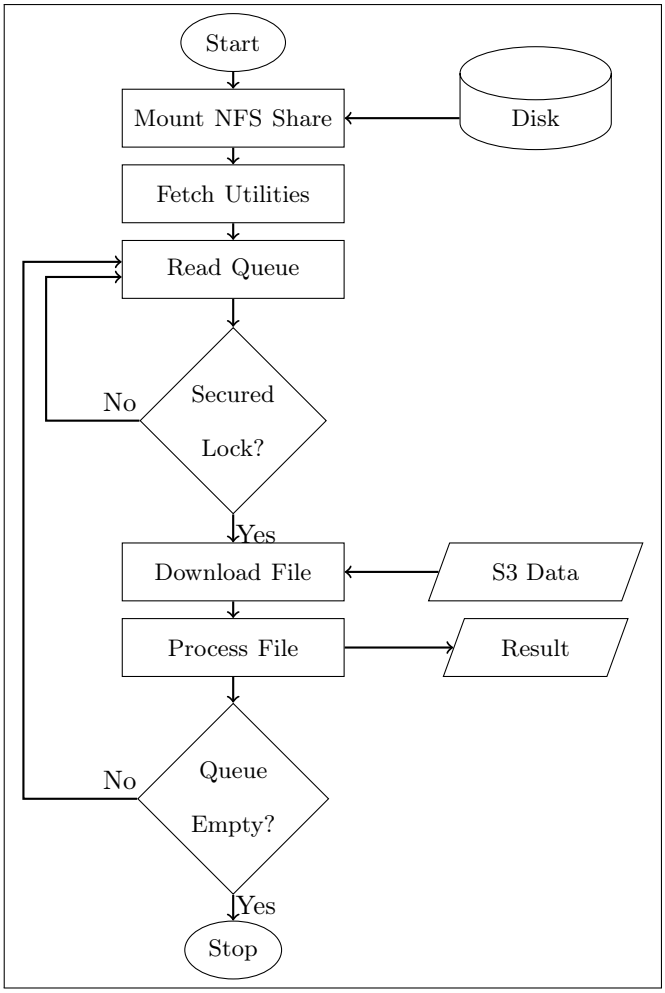


Figure 4.3: The ACN worker node processing flowchart.

4.2.3.1 Aperture Photometry in ACN-APHOT

The *acn-aphot.c* program is built upon the *rrf.c* program from the initial pilot system. These changes extend the functionality to include aperture photometry and the following new features were developed.

- Use an initial set of coordinates find the centre of an object on the image file.
- Calculate the sky background.
- Calculate the flux intensity of the object using a partial pixel algorithm for a given aperture size.
- Generate multiple magnitude values for an object using a range of aperture sizes.
- Calibrate pixels only as they are used.
- Calibrate all pixels without performing aperture photometry.

These features extended the core software within the pipeline to reduce the data to a range of magnitude values for each star on each image. The result files are measured in kilobytes and not megabytes allowing for faster uploading of results to improve the overall performance of the system. The option to have pixels cleaned before they were used allows for a comparison of the performance of a one phase reduction compared to a two phase reduction. In one phase reduction, the program loads the master frames into memory and only when a pixel value is used in a calculation such as sky background calculation or flux intensity is the pixel calibrated. If a pixel in the image is not used, then it will never be calibrated. In two phase reduction the images are first calibrated using the *rrf.c* program from the pilot system and these calibrated image files are passed to the *acn-aphot.c* program which this time does not clean the pixels. Figure 4.4 provides an overview of the *acn-aphot.c* work flow. The algorithms used to extend the program are briefly described below.

4.2.3.2 Centroid Algorithm

The gradient centroid approach described in Chapter 2 was used and Algorithm 6 was implemented. The initial x,y co-ordinate of the object is provided in this case using a preprepared co-ordinate file containing references to the objects in the image as shown in Table 4.1. Additional information is provided within this configuration file which will be discussed later. For the BCO dataset there are 5 objects of interest. For each of these objects the centroid algorithm was run to determine the centre of the object.

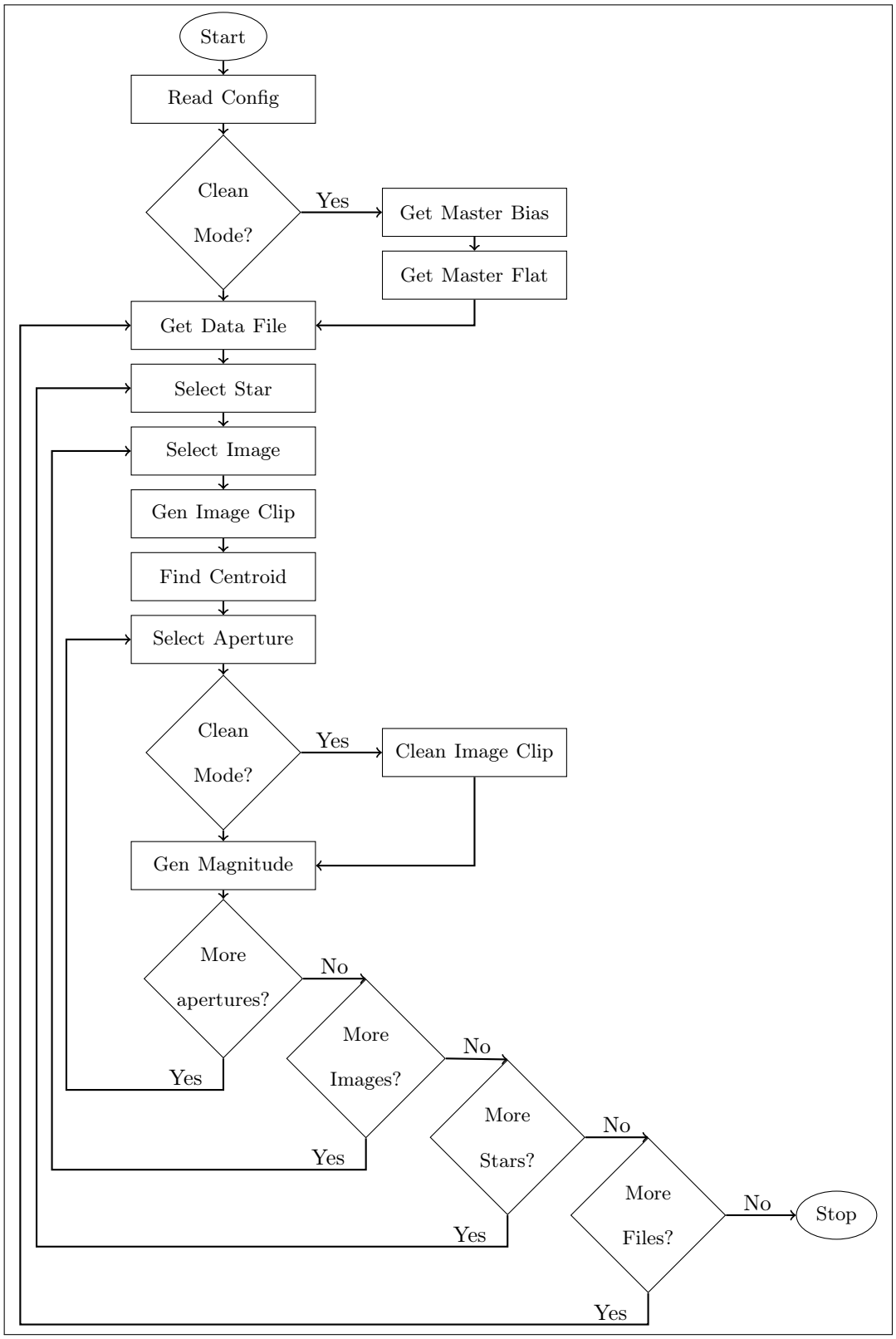


Figure 4.4: ACN-APHOT Image cleaning and reduction process work flow.

Xval	Yval	Radius	Annulus	Dannulus	Box	Threshold
112	320	15	10	15	80	660
127	221	15	10	15	80	660
119	104	15	10	15	80	660
260	43	15	10	15	80	660
380	378	15	10	15	80	660

Table 4.1: ACN Configuration File for the ACN-APHOT program

Algorithm 6 Gradient Centroid Algorithm

```
1:  procedure CENTROID(col, row, threshold, img)
2:      for i = 1 to row do
3:          for k = 1 to col do
4:              if img[i, k] ≤ threshold then      ▷ test threshold to create a binary mask
5:                  img[i, k] ← 0
6:              else
7:                  img[i, k] ← 1
8:                  TotalMasks ← TotalMasks + 1
9:              end if
10:         end for                                ▷ go to the next column
11:     end for                                    ▷ go to the next row
12:     for i = 1 to row do
13:         for k = 1 to col do
14:             if img[i, k] = 1 then              ▷ test threshold to create a binary mask
15:                 Tx ← Tx + i                    ▷ Sum all row gradients
16:                 Ty ← Ty + k                    ▷ Sum all col gradients
17:             end if
18:         end for
19:     end for                                    ▷ go to the next row
20:     Cx ← Tx/TotalMasks                         ▷ Find X center point
21:     Cy ← Ty/TotalMasks                         ▷ Find Y center point
          return Cx, Cy
22: end procedure
```

4.2.3.3 Sky Background Algorithm

The sky background is a measure of the average amount of light contained within each pixel which should be removed before calculating the intensity flux value. For a pixel value to be included in the sky background calculation it must be fully within the sky annulus. Algorithm 7 shows how to determine if a pixel is within the sky annulus. The sky background value per pixel is calculated as the average value of all of the pixels fully within the sky annulus. The inner radius of the annulus is the radius of the aperture used plus the annulus value given in the configuration file in Table 4.1. The dannulus is the width of the sky annulus, also given within Table 4.1.

Algorithm 7 Sky Background

1: **procedure** SKYBACKGROUND($x, y, centX, centY, radius, annulusval, dannulusval$)
▷ The x,y values are the pixel coordinates
▷ CentX,centY are the centroid values
2: $annulus = radius + annulusval$
3: $dannulus = annulusval + dannulusval$
4: $distance = euclid_distance(x, y, centX, centY)$
5: **if** $distance < dannulus - 0.5$ && $distance > annulus + 0.5$ **then**
return $TRUE$
6: **else**
return $FALSE$
7: **end if**
8: **end procedure**

4.2.3.4 Partial Pixel Algorithm

Unlike the sky background, the calculation of the flux intensity required for the magnitude calculation uses a partial pixel algorithm. This is an extension of the sky background algorithm where pixels can be partially within the radius value used for the aperture. Algorithm 8 was implemented within the *acn-aphot.c* program.

4.2.4 Node Control

The pipeline is controlled via the ACN-Control script run on the Remote Control command console system which must be on the same logical network as the DIT. From this command console a number of functions are supported which control the running of the pipeline. These controls are primarily for experimental execution.

Algorithm 8 Partial Pixel

```
1:  procedure INTENSITYMEASURE(pixval, radius, distance)
2:      if distance = radius then                ▷ test if pixel is on the line of the aperture
3:          val ← (0.5 * pixval)
4:      else
5:          if distance < (radius − 0.5) then
6:              val ← pixval                     ▷ pixel is fully within the aperture
7:          else
8:              if distance > (radius + 0.5) then
9:                  val ← 0                      ▷ pixel is fully outside of the aperture
10:             else
11:                 val ← radius + 0.5 − distance   ▷ pixel is partially within the aperture
12:             end if
13:         end if
14:     end if
        return val
15: end procedure
```

All ACN nodes be registered in a nodes file allowing experiments to be run with different numbers of nodes. A nodes file contains the name, IP address and a storage location for data retrieval is specified. The following commands are supported and executable from the ACN Pipeline Controller via an SSH script which remotely connects to each node. A brief description of each command is provided under each component of the pipeline.

- *Activate-ACN -r NODEFILENAME* This command will activate all ACN Nodes in the nodes file so they start looking to a queue for image files to process. The ACN nodes will wait for a queue to be created, but this option provides better control for experimentation so it is clear which nodes are active. Once a node is running, it checks the queue for unlocked files and locks them as already described. Once locked, the full image file is downloaded from an S3 bucket and cleaned using the *acn-aphot.c* program. This program runs once for each file downloaded.

- *Activate-ACN -c NODEFILENAME* This command will reset all ACN Nodes in the nodes file so they stop running, remove all temporary data and download the latest utility files, configuration files and Master Bias/Master Flat images ready for the next round of processing.

- *Activate-ACN -p NODEFILENAME* Ping all of the nodes to ensure that they are accessible to the pipeline.

- *Activate-ACN -x NODEFILENAME* Reboot all of the nodes to ensure that they have flushed all caches. If a node processes the same data multiple times, it may operate faster on the subsequent executions due to caching of data in memory.

- *ACN-Control -q compressed | standard | clipped* This command will create a list of empty files in a directory, which can be used as a simple queue. Files are named with a prefix source and traverse a dataset creating an entry in the queue for all files found. A compressed source file of 0000001.fits.fz has a corresponding queue entry of Queued-0000001.fits.fz. When successfully locked for processing by an ACN-Node this is changed to LOCKED-0000001.fits.fz. The NFS file system ensures only one lock can be obtained.

- *fits-compress -p DATADIRECTORY* This script has the option of performing compression in parallel or in sequence. The parallel execution spawns off processes and requires a machine with good RAM and processing capabilities. A comparison of performance for this script running in both modes is given in this chapter.

- *s3-upload -p DATADIRECTORY* This script has the option of performing upload in parallel or in sequence. The parallel execution spawns off processes and requires a machine with sufficient RAM and processing capabilities. A comparison of performance for this script running in both modes is given in this chapter. Compressed or uncompressed data (depending

on what is in the data directory) is uploaded to an S3 bucket. A comparison of performance for this script running in both modes is given in this chapter.

4.3 Experimental Methodology

To ensure that all experiments are executed in a consistent manner, and that experiments were repeatable, a strict process of experimental setup and execution was followed. Where possible issues of caching were eliminated along with network contention issues. For example all experiments were run late at night because of the requirement to use the Institute network to route all traffic to the internet.

The network setup remained the same for all experiments. The storage servers used the same disk types, ran on the same hardware and used a central FreeNAS server for NFS locking. Each of the worker instances ran the same version of Ubuntu 12.04, and were fully cleaned before each experiment ran. The central control script logged into each of the workers before an experiment via SSH and removed all of the files in the user directory, then installed a new version of the software taken from the central NFS server. This ensured that all worker instances ran the same version of software. After each server was updated they were rebooted.

The queue was also constructed using a central script which created a list of empty files after first archiving previous versions if they exist. The order of files created was consistent in each case, as was the length of the queue.

The queue refers to files which have been uploaded to the S3 website prior to the experiment and the scripts which take the files from the NFS queue will then construct a URL which be used to access the file on the AWS service.

Once the queue has been created, the workers are reinstalled and rebooted and the files uploaded to S3, so the experiment is initiated. A central script forks off a process to remotely connect to each of the worker instances and starts the cleaning script which obtains NFS locks, downloads files, cleans them and uploads results to S3. All workers however remain in standby mode until a specific file is detected in the central NFS server. When that file is detected by the worker it begins the processing cycle. This process ensures that all workers are running before the experiment begins. Each worker contains a timer which is written to a central directory on the NFS server when they have completed all available work. Each worker records their name, the time spent cleaning, the number of files cleaned and the average cleaning rate achieved.

Each worker instance only runs a single process for downloading, processing and uploading results.

Multiple experiments were run to determine that the procedures were functional, then formal experiment ran using the same experimental software to ensure they could be compared. Each experiment focused on an increase in the number of servers processing data to record the incremental

performance of each worker.

The experimental run script used for the ACN experiments is shown in Figure 4.5.

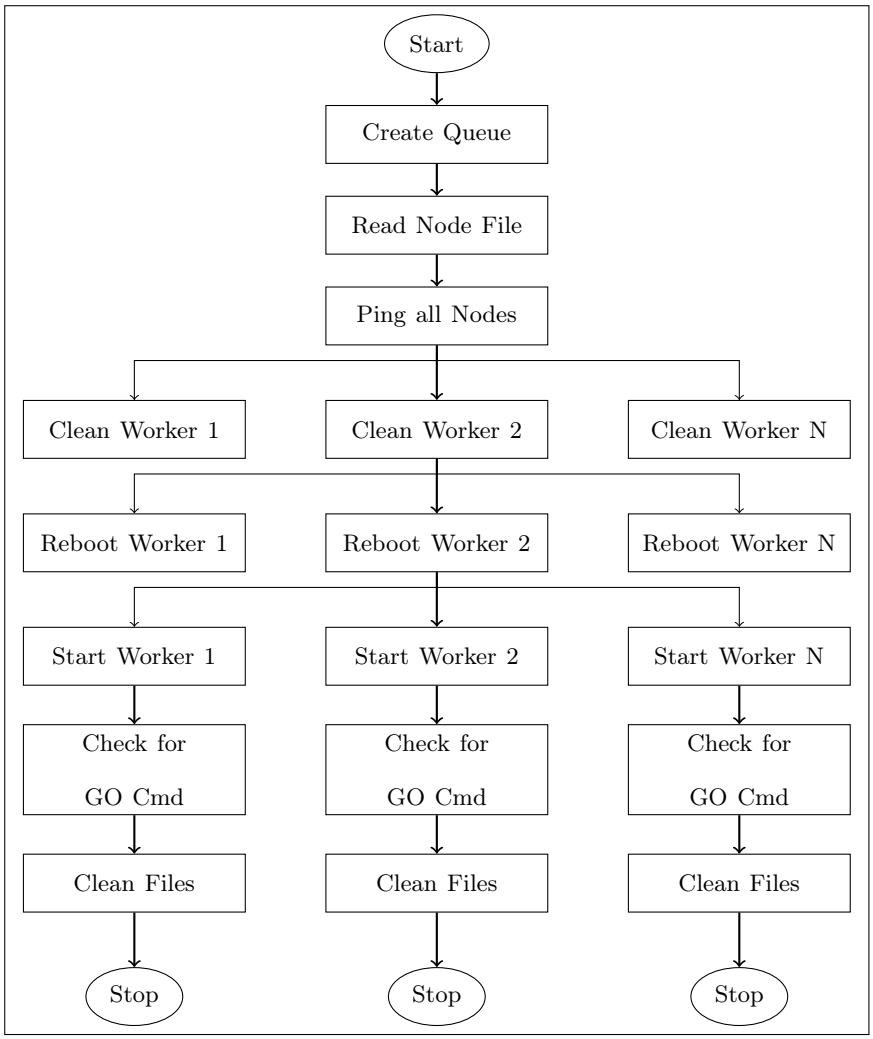


Figure 4.5: ACN experimental run script flow chart.

4.4 Results and Discussion

In this section, the performance of the ACN pipeline is reviewed seeking insights into the system design which can be later used within the NIMBUS pipeline. Each of the nodes central to the pipeline are performance tested. Table 4.2 provide a list of the primary experiments run. Each experiment is discussed and the results presented. A summary and conclusion for these experiments is provided at the end of the chapter. All data sources for all experiments and graphs are identified in Appendix Table D.2 which references an accompanying supplementary USB disk which contains raw and processed data relevant to these experiments.

Reference	Measure	Description
Exp:ACN1	ACN-APHOT Performance	Determining the performance of this program by running in multiple modes using various storage devices. Two step versus one step cleaning is examined.
Exp:ACN2	Storage Performance	Determining the Impact of the location of the storage devices and their ability to support multiple queries.
Exp:ACN3	Data Compression	Compression of data reduces the size of data for both storage and transfer. Data compression techniques and approaches are considered.
Exp:ACN4	Data Transfer	Data stores are compared in terms of data transfer rates.
Exp:ACN5	Pipeline Limits	Determining how fast the pipeline can operate within the propose architecture.

Table 4.2: ACN performance experimental set.

4.4.1 ACN1: ACN-APHOT Performance

The purpose of this experiment is to look at how long it take for the BCO dataset to be fully calibrated with magnitude values generated for all objects within all images using different hardware, different approaches to cleaning and different file storage systems. The two programs used are *rrf.c* and *acn-aphot.c*, and the IBMe326 and a Macbook Pro[3] were the hardware platforms. The dataset was either stored on the local disk of the hardware, or mounted as an NFS drive over the network. Table 4.3 shows the results of this experiment with the processing rate given as file per second. To compare with later experiments we convert the processing rate to gigabytes per second[4]. The *Step 1* time is the time it took for all raw image files to be pixel calibrated. In each case, all of the pixels in all of the images were calibrated. When *Step 1* has a time, the *Step 2* time refers to the time taken to read in calibrated images and to calculate the magnitude values. When *Step 1* has an N/A then pixels in *Step 2* were calibrated using the previously discussed just in time process.

With the exception of P1-1, the experiment results are reasonably consistent and clear. The time required to process images using two steps is almost double the time taken to use a single step as shown in Figure 4.6. This can be attributed to the number of file input and output events and using the CFITSIO library and the disk seek time. The amount of data read is only half of a 2 step approach, and the write operations are also dramatically reduced given the fact that result

[3] MacBook Pro 2.4 Ghz Intel Core i5 and 8 GB RAM, IBM eServer 326 Operon 2.8Ghz and 4 GB RAM

[4] Data was not compressed so with each file = 7.297920 MB, GB/s $= rate * 7.29792/1024$

Ref	Data Store	ACNs	Steps	Hardware	Step 1	Step 2	Rate	Total	GB/s
P1-1	Local	1	2	Macbook	11:21:45	00:54:48	0.083	12:16:33	0.0006
P1-2	Local	1	1	Macbook	N/A	00:55:53	1.098	00:55:53	0.0078
P1-3	Local	1	2	IBMe326	00:49:00	01:15:23	0.493	02:04:23	0.0035
P1-4	Local	1	1	IBMe326	N/A	01:22:30	0.744	01:22:30	0.0053
P1-5	NFS	1	2	IBMe326	00:49:53	01:15:08	0.490	02:05:01	0.0035
P1-6	NFS	1	1	IBMe326	N/A	01:17:25	0.793	01:17:25	0.0057
P1-7	NFS	2	1	IBMe326	N/A	01:07:43	0.906	01:07:43	0.0065
P1-8	NFS	4	1	IBMe326	N/A	00:43:15	1.413	00:43:26	0.0101

Table 4.3: P1: Calibrating the processing time for full BCO Dataset using one or two pass cleaning.

files are now measured in kilobytes rather than in megabytes. Raw data for this graph can be found in the Appendix C, Table C.4.

When *Step 1* is not used, the processing performed in *Step 2* is slightly higher than in *Step 1* but the processing times remain very similar. *Step 2* holds the Master Bias and the Master Flat file in memory and as each pixel value is used, it is first calibrated. The output result files from these two approaches were compared and the results were identical. The disk storage had some effect with the NFS disks operating slightly faster. This may have been due to the fact that they were faster disks, but overall the storage location for these experiments was not significant.

In addition to testing the two step versus one step cleaning, the impact of adding additional nodes was introduced. In P1-7 the two nodes accessed NFS storage, with each having access to exactly half of the dataset. In P1-8 there were four nodes running each having access to a quarter of the data. No queuing system was used, just a copy of the data made available in a private location for each node. The processing times were reduced but not linearly with the number of nodes. The NFS storage used contains a single disk and is most likely the bottleneck in processing. The data transfer rates as shown are small and the network was unlikely to be the issue. The next experiment, *ACN2: Disk Testing* explored this further.

4.4.2 ACN2: Disk Storage Testing

This experiment builds on the observations from experiment P1-4, where the file processing rate was 0.744 files for a single node. The aim was to see how additional nodes could run in parallel and still retain this processing rate by specifically looking at the disks used for serving data. In

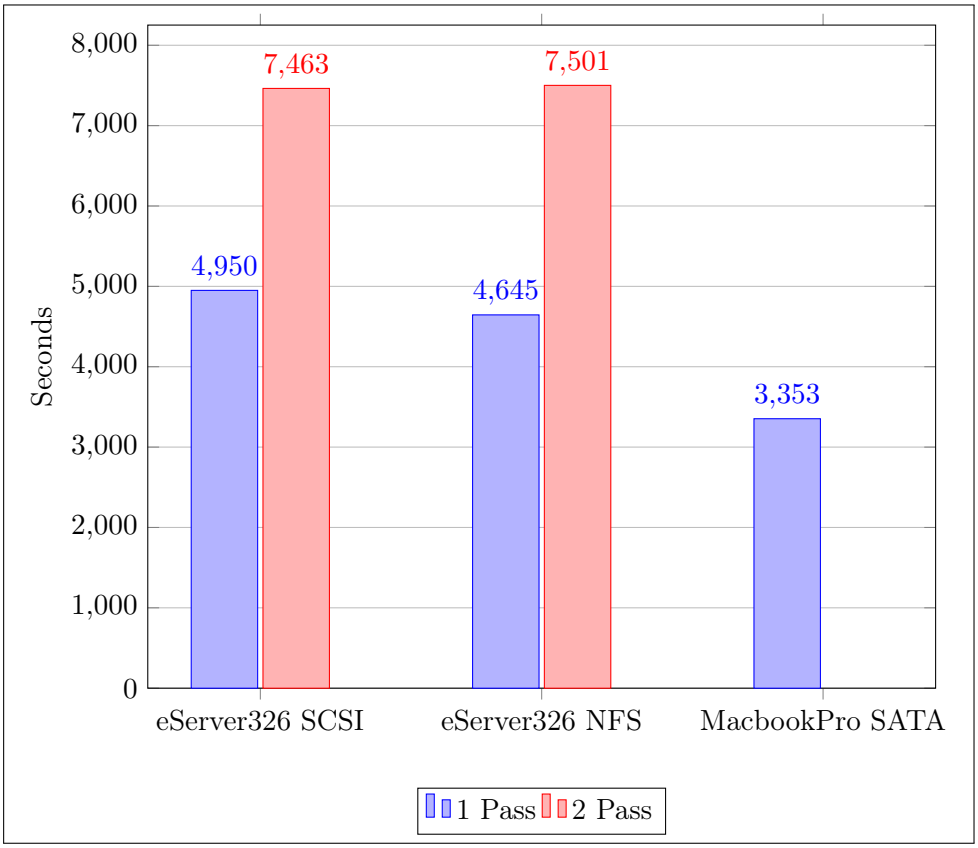


Figure 4.6: ACN1: One step versus two step cleaning using different storage media on full dataset.

these experiments the primary difference is the location of the data during processing. The data was split into eight equal parts and copied to the local storage of each of the 8 nodes. Three experiments were run.

- P2-1: Eight nodes processing $\frac{1}{8}th$ of the data from a local disk
- P2-2: Eight nodes processing $\frac{1}{8}th$ of the data from 3 NFS mounted disks
- P2-3: Comparing the NFS Storage Systems

The results of P2-1 are given in Table 4.4 showing a consistent processing rate for each of the nodes, although the rate is less than seen in P1-4. In these experiment the overall data processing time for the full dataset is the time at which the last node stopped processing. The overall processing rate of the system is therefore $3682files/928seconds = 3.968$ files per second, or $0.028GB/s$.

Ref	Data Store	Node Number	Hardware	Rate	Time
P2-1-1	Local	1	IBMe326	0.522	00:14:41
P2-1-2	Local	2	IBMe326	0.512	00:14:59
P2-1-3	Local	3	IBMe326	0.514	00:14:56
P2-1-4	Local	4	IBMe326	0.500	00:15:28
P2-1-5	Local	5	IBMe326	0.499	00:15:23
P2-1-6	Local	6	IBMe326	0.496	00:15:28
P2-1-7	Local	7	IBMe326	0.497	00:15:27
P2-1-8	Local	8	IBMe326	0.504	00:15:14

Table 4.4: P2-1: File processing rates on local disks using 8 ACN Nodes.

Because the NFS processing times were faster than the local disks, P2-2 was run using 3 different NFS storage servers on the local network to see if further improvements were possible. Since each of the nodes are identical, the differences observed in Table 4.5 are either related to the NFS system or the network. Additional testing was then performed using P2-3 which repeated eight single node tests against each of the NFS stores with no two tests running at the same time. Using a smaller dataset of only $\frac{1}{24}$ of the total, each of these experiments ran against each NFS storage device. The results for P2-3 are given in Table 4.6 and clearly show a consistent processing rate depending on the storage node used, with NFS1 providing the best rate. The reduction in processing rates as the number of nodes increased, and the variability of the nodes resulted in a move to the S3 storage system which offered more consistent download times

Ref	Data Store	Node Number	Hardware	Rate	Time
P2-2-1	NFS1	1	IBMe326	0.522	00:08:03
P2-2-2	NFS1	2	IBMe326	0.512	00:07:54
P2-2-3	NFS1	3	IBMe326	0.514	00:07:30
P2-2-4	NFS2	4	IBMe326	0.500	00:12:43
P2-2-5	NFS2	5	IBMe326	0.499	00:12:39
P2-2-6	NFS3	6	IBMe326	0.496	00:11:14
P2-2-7	NFS3	7	IBMe326	0.497	00:11:07
P2-2-8	NFS3	8	IBMe326	0.504	00:11:00

Table 4.5: P2-2: File processing rates using 3 different NFS servers and 8 ACN Nodes

4.4.3 ACN3: Data Compression

The compression used on the BCO data was the CFITSIO *fpack* utility. The raw dataset is 26.4GB, but when compressed this is reduced to 4.6GB. If images are first clipped to only include the regions around the objects for which magnitude values are required, then the dataset is further reduced to 1.7 GB as shown in Figure 4.7. By compressing the data using a powerful server, transfer times should be shorter. The *fpack* utility in this case reduced each image by over 80% making a significant difference to the size of the total dataset.

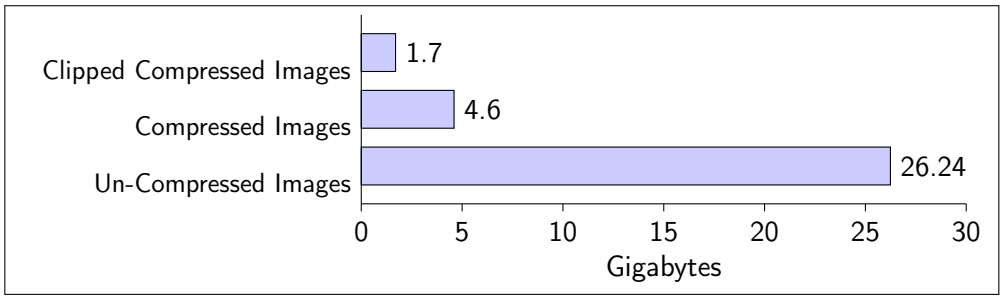


Figure 4.7: ACN3: BCO Dataset sizes in Gigabytes

To determine the optimal performance of the Dell 410 server, FITS lossless compression was run using two different methods. The first compressed files in sequential order where a file-list was given to a single instance of the utility. The second approach started thousands of processes at staggered intervals where each process was running concurrently utilising large portions of the available memory on the server. This approach is valid for a server with sufficiently large quantities of memory and CPU cores. If there are not enough resources then the system begins to thrash. The

Ref	Data Store	Node Number	Hardware	Rate	Time
P2-3-1	NFS1	1	IBMe326	0.546	00:04:41
P2-3-2	NFS1	1	IBMe326	0.562	00:04:33
P2-3-3	NFS1	1	IBMe326	0.542	00:04:43
P2-3-4	NFS1	1	IBMe326	0.544	00:04:42
P2-3-5	NFS1	1	IBMe326	0.540	00:04:44
P2-3-6	NFS1	1	IBMe326	0.542	00:04:43
P2-3-7	NFS1	1	IBMe326	0.540	00:04:44
P2-3-8	NFS1	1	IBMe326	0.546	00:04:41
P2-3-9	NFS2	1	IBMe326	0.248	00:10:20
P2-3-10	NFS2	1	IBMe326	0.195	00:13:07
P2-3-11	NFS2	1	IBMe326	0.214	00:11:56
P2-3-12	NFS2	1	IBMe326	0.222	00:10:30
P2-3-13	NFS2	1	IBMe326	0.223	00:10:29
P2-3-14	NFS2	1	IBMe326	0.194	00:13:11
P2-3-15	NFS2	1	IBMe326	0.243	00:10:31
P2-3-16	NFS2	1	IBMe326	0.245	00:10:27
P2-3-17	NFS3	1	IBMe326	0.376	00:06:48
P2-3-18	NFS3	1	IBMe326	0.384	00:06:48
P2-3-19	NFS3	1	IBMe326	0.372	00:06:48
P2-3-20	NFS3	1	IBMe326	0.369	00:06:48
P2-3-21	NFS3	1	IBMe326	0.369	00:06:48
P2-3-22	NFS3	1	IBMe326	0.367	00:06:48
P2-3-23	NFS3	1	IBMe326	0.369	00:06:48
P2-3-24	NFS3	1	IBMe326	0.373	00:06:48

Table 4.6: P2-3: Comparing File Processing rates against each of the NFS storage systems

compression time reading and writing files on the NFS server mount point are shown in Figure 4.8. The parallel performance of the Dell 410 was instrumental in the pipeline providing a compression time considerably faster than the sequential method. It was interesting to note that the older IBM systems provided comparable compression times to the Dell when run in sequential mode.

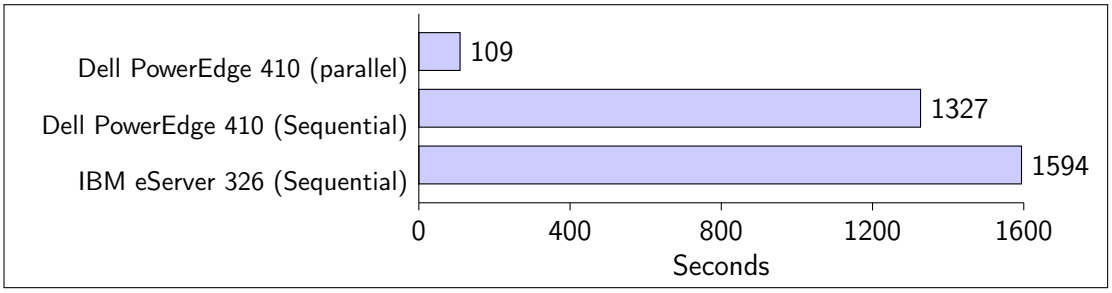


Figure 4.8: ACN3: Comparison of Data Compression times using different modes and hardware

4.4.4 ACN4: Data Transfer

The transfer times of the datasets between different types of storage is shown in Figure 4.9. The private network was connected via a gigabit switch to the DIT network, which has an external institute-wide Internet connection of 1 Gigabits per second. Approximately 20% of the bandwidth was used in this transfer although the use decreased to approximately 8% as the number of files was increased. The nature of the network is that it is variable, however the experiments were run late at night when the Institute's network was lightly loaded. The clipped and compressed data files were files where the FITS files were clipped into the smallest possible region around the star so pixels, which were not going to be cleaned, were not transported. The time saved in data transfer however was negligible.

4.4.5 ACN5: Pipeline Limits

Once all data was compressed and uploaded to the S3 storage system, the central queue was activated to allow for as many nodes as possible to come online. It took approximately 20 seconds to create the 3682 empty files in the NFS queue. Experiments were run using the compressed dataset on S3 and using a one pass approach where data was cleaned only as pixels were used. In each experiment the full dataset needed to be fully processed for the experiment to finish. Experiments were run for increments of 5 ACN Nodes up to a maximum of 58 Nodes. The overall processing time in seconds is given, and includes the data compression and data upload time to S3. The single node processing time is taken from experiment P1-4 which had a time of 1 hour, 22 minutes and 30 seconds. Figure 4.10 shows a graph of the time in seconds for the full dataset to be processed for varying numbers of ACN Nodes. Because of the way the compression and upload

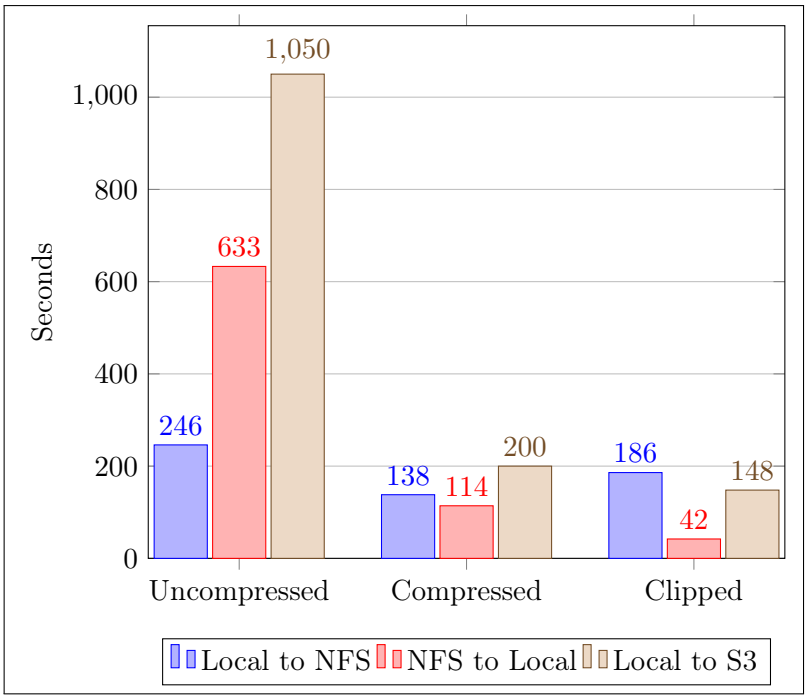


Figure 4.9: ACN4: Comparison of data transfer times between storage types.

process was performed, it is the limiting factor on the pipeline. As more nodes are introduced the processing time moves closer to the compression and upload times. Table 4.7 provides full details of the times and file processing rates for each experiment run. While these experimental results have the compression and upload time costs added there is no specific dependancy on these steps completing before the next step can start. A file could be compressed, uploaded and processed while other files are going through the same process.

Due to the mixture of server types used within the last experimental set it is worth taking a closer look at the breakdown of the nodes and their overall processing contribution to the pipeline. Specifically we will look at ACN5-11 which had 58 nodes running. In Figure 4.11 the number of nodes for each server type and server location is shown. Each of the institutes had 8 IBM e326 servers which were all configured exactly the same and had the same hardware specification. The VMWare servers were all within the DIT and contained varying numbers of virtualised instances of the ACN Nodes. All nodes were started within a few seconds of each other by the ACN-Control system. The average file processing rate is also given for each of the servers. The IBM servers are operating in line with previous experiments when running against an NFS server, but there is no degradation of the processing rate as the number of ACN nodes increased. Two other interesting observations can made by looking at Figures 4.12 and 4.13. Because of the introduction of the queue to the system, each of the nodes will clean as fast as it can then move on to the next file. Even when some of the nodes are considerably slower and cleaning less files as shown in Figure 4.13, all processing finished roughly at the same time.

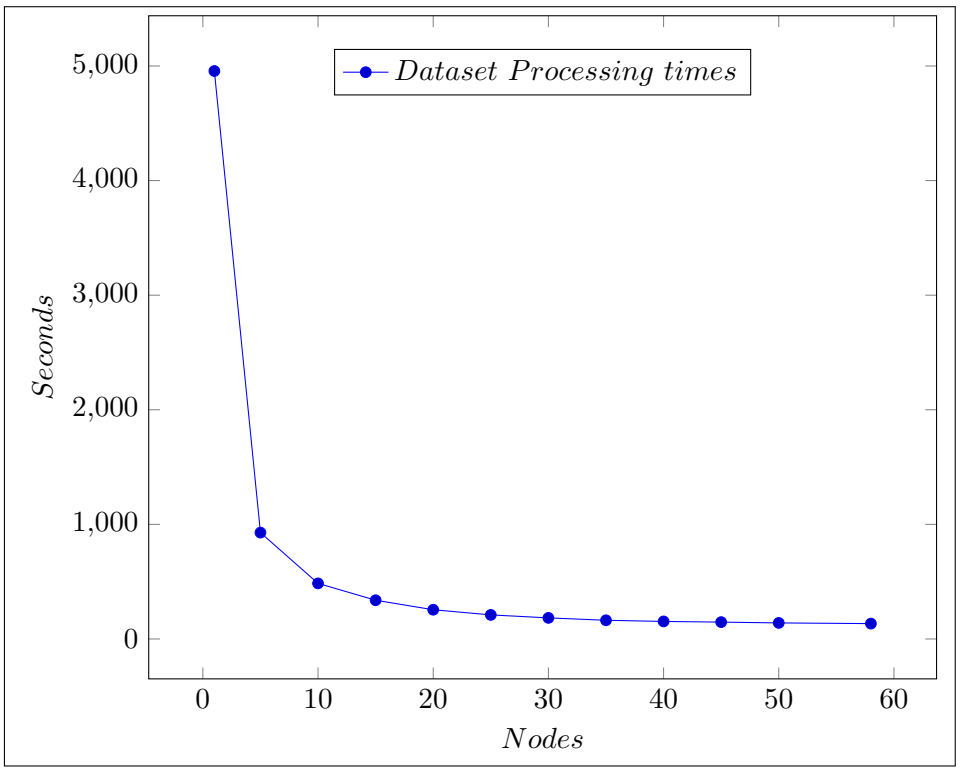


Figure 4.10: ACN5: Data cleaning for increasing numbers of ACN nodes in seconds. Includes compression time of 109 seconds and upload time of 200 seconds

Ref	Nodes	Compression	Upload	Processing	Total	File/s	GB/s
P1-4	1	00:01:49	00:03:20	01:22:36	01:27:45	0.699	0.005
ACN5-1	5	00:01:49	00:03:20	00:15:28	00:20:37	2.977	0.021
ACN5-2	10	00:01:49	00:03:20	00:08:05	00:13:14	4.637	0.033
ACN5-3	15	00:01:49	00:03:20	00:05:38	00:10:47	5.691	0.041
ACN5-4	20	00:01:49	00:03:20	00:04:15	00:09:24	6.528	0.047
ACN5-5	25	00:01:49	00:03:20	00:03:30	00:08:39	7.094	0.051
ACN5-6	30	00:01:49	00:03:20	00:03:04	00:08:13	7.469	0.053
ACN5-7	35	00:01:49	00:03:20	00:02:43	00:07:52	7.801	0.056
ACN5-8	40	00:01:49	00:03:20	00:02:33	00:07:42	7.970	0.057
ACN5-9	45	00:01:49	00:03:20	00:02:27	00:07:36	8.075	0.058
ACN5-10	50	00:01:49	00:03:20	00:02:20	00:07:29	8.200	0.058
ACN5-11	58	00:01:49	00:03:20	00:02:14	00:07:23	8.312	0.059

Table 4.7: ACN5: Clean rates per node and GB/s processing rate

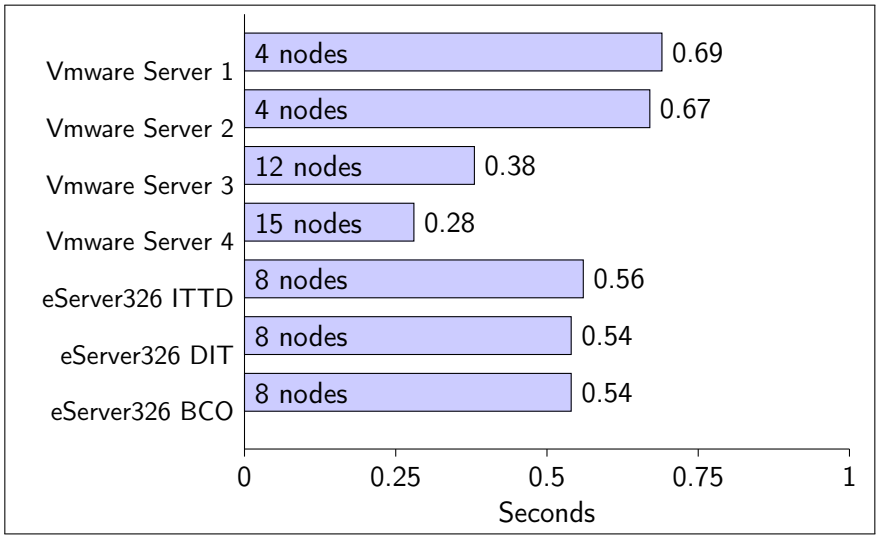


Figure 4.11: ACN5: Average file processing rate (files per second) per server type

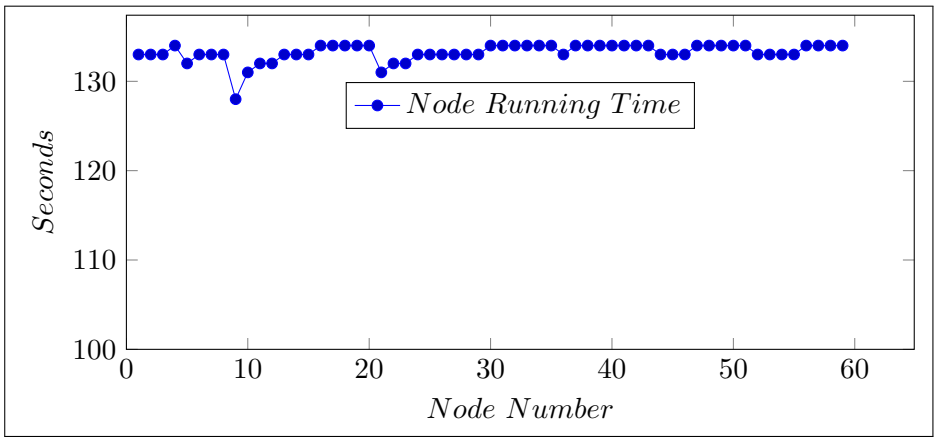


Figure 4.12: ACN5: Running time for each individual node.

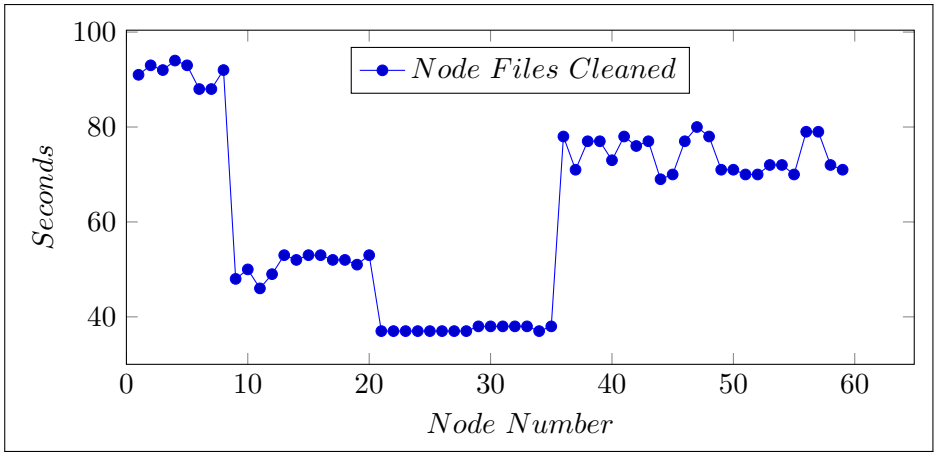


Figure 4.13: ACN5: Number of files cleaned by each individual node.

4.5 Conclusion

The ACN pipeline provides a strong argument for a storage device which can service requests at a consistent rate as the rate of requests increases. Experiments with local NFS devices proved to be inconsistent and required that multiple copies of the dataset had to be available in multiple locations to avoid network or read bottlenecks. The cost of compression and subsequent upload were not significant compared to the initial cleaning time of the data when a single node was used. By moving the dataset to a disturbed storage system, data replication was no longer required.

The NFS based queue, while simple and effective, limited access to worker nodes requiring that they remain within the private network. A more public queue would allow ACN nodes to join in the pipeline from other locations. The requirement to use SSH to access these nodes for the purpose of control is also a limiting factor for further scaling. SSH ports are often restricted. There is also a cost for Amazon S3 storage and an alternative Hadoop Distributed File System (HDFS) implementation or high performance static web server could provide similar functionality.

The pipeline does provide clear evidence that the parallelised processing approach does significantly reduce the processing time by allowing additional ACN Nodes to contributed. It has also been verified that this process does not impact the calculation of the instrument magnitude values and that the reduction in file read and write operations has also contributed to some of the performance enhancements observed.

The systems used to control the experiments required that operations could be run in parallel and in sequence for the purpose of comparison. Data compression and uploading should be incorporated into the data production cycle so that there is a constant flow of data into the pipeline. The worker nodes should also be able to play a more proactive role in monitoring the queue and ensuring they contain the latest software. This pipeline used a push model, but a pull model would suit a distributed system.

Using the ACN pipeline it has been demonstrated that the processing rate for this dataset can be reduced from the initial MATLAB time of 0.1 files per second to 8.3 files per second when using 58 nodes. If the overhead cost of the compression and uploading are removed that cleaning rate could increase to 27.48 files per second however the rate of performance improvements as nodes were added fell dramatically for larger numbers of nodes.

The lessons taken from this pipeline which are to be used within the NIMBUS system are as follows.

- The queue needs to be globally accessible
- The workers need to be more self configurable
- Data stores need to be more distributed and globally available
- The linear increase in computing resources did not produce a linear increase in performance,

most likely due to resource sharing.

- One pass pixel cleaning and magnitude generation speeds up individual ACN Nodes by reducing the number of I/O operations.
- Compression is a net contributor to the system and should be continued.

Chapter 5

The *NIMBUS* Pipeline

5.1 Overview

This NIMBUS pipeline, uses a public web queue to publish work to distributed computing nodes build explicitly for this pipeline which are referred to as *workers*. Workers are computing instances that can reside anywhere on the internet but are required to have internet access using port 80, as all services accessed are HTTP based. Each worker uses at its core the *acn-aphot.c* program used in the ACN Pipeline which runs in single step mode. For this pipeline the BCO dataset is also used but it has been renamed and replicated so that there are multiple terabytes of data available for processing, not just 26GB. The assumption with this pipeline is that the data is already staged for processing, and the emphasis is on processing image data rather than compression and uploading image files. From the previous experiments, the compression and upload costs were already established. The architecture is shown in Figure 5.1 and breaks into the following fundamental components which are described in more detail within this chapter. The code used for this pipeline is available on github in the following repository. `https://github.com/paulfdoyle/NIMBUS.git`

Figure 5.2 gives the basic work flow of the pipeline.

- Data Archive Cloud. A distributed collection of web servers which provide data for the pipeline to process. Data has already been compressed and servers within this cloud can be both virtual and physical. The archive is required to advertise the files it contains via a distributed web message queue, and to service those files via HTTP requests.
- Distributed Worker Queue Cloud. The entire pipeline is centred around a series of web message queues, and this is the central queue which contains work to be performed. The queue cloud is a series of communication protocols which allow the overall pipeline to orchestrate work in such a way that all workers operate at their peak performance by working as independent consumers of work to produce a series of result files. Data is advertised as a web

message, and workers are controlled using command queues. The worker queue specifically contains the messages for workers to process.

- Monitoring System. Due to the highly distributed nature of the pipeline, monitoring also needs to be distributed. Used primarily for experimental measurement monitoring is performed on queues and workers. All workers record their progress in a distributed log queue and queue sizes and rates of change are monitored. The web servers and worker nodes contained monitoring software to review CPU and Network performance.

- Global Processing Cloud. All workers which contribute to the processing are required to have an initialisation script within their boot sequence, and are assumed to be a Linux based system. This script initiates worker processes on the server by downloading a customer package which contains all instructions and tools for use by the workers. Workers can be globally located and are self contained processing units which are assumed to be transient resources.

- Results Cloud. When workers complete the processing for an image, the results are posted to a distributed storage service and a web message is constructed which provides a reference to the result. The reference can be used to reconstruct the sequence of the original images if required.

- Control System. A central system was created to control experiments. Using an AWS API in Python, workers and queues were created and monitored. All experiments are started and shutdown using well defined procedures to ensure consistency across experiments. The control system initialises the pipeline and the queues, instructs the storage nodes to advertise work and the processing nodes to start processing.

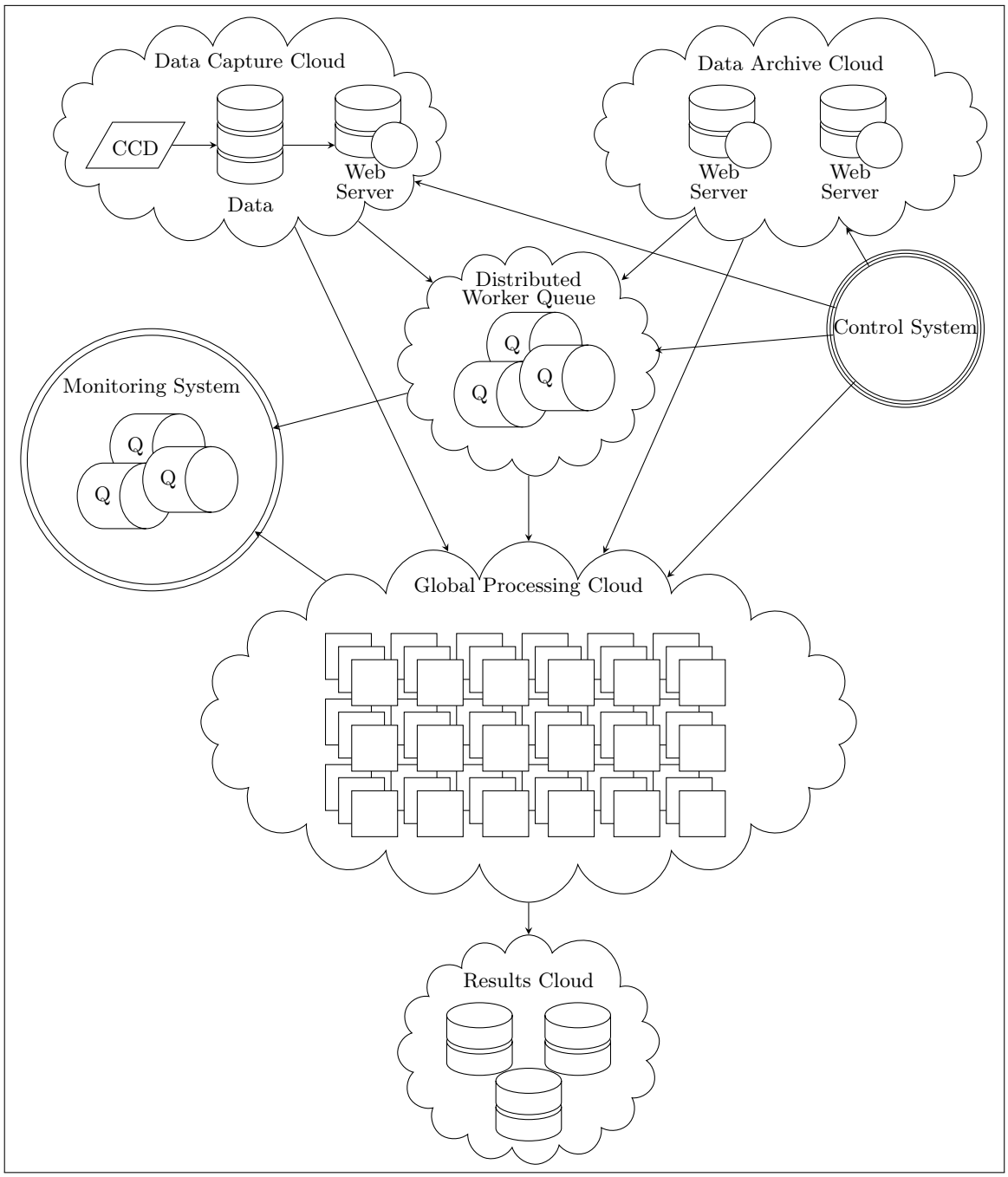


Figure 5.1: NIMBUS Architecture

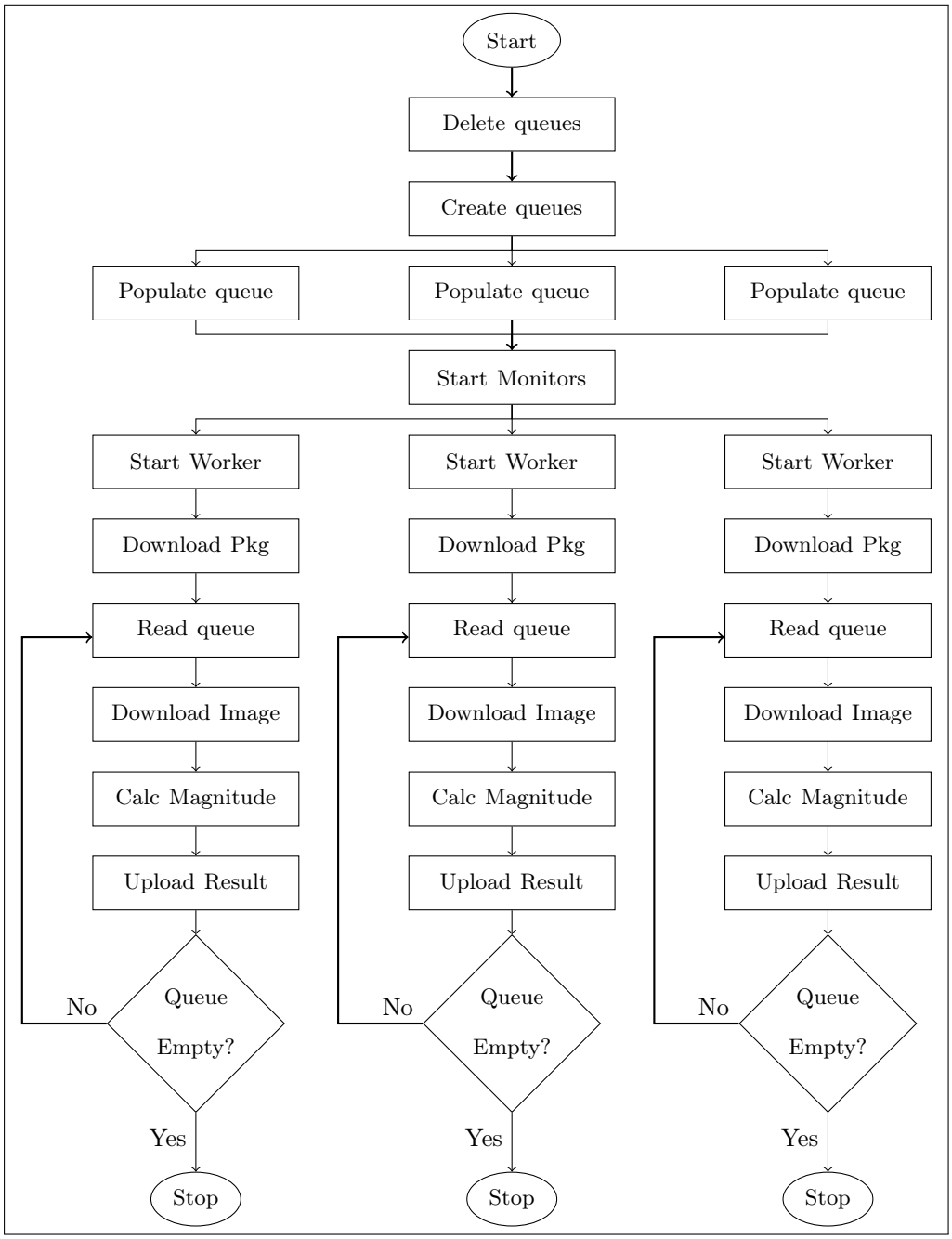


Figure 5.2: Distributed processing pipeline where worker nodes use the queue to work in parallel.

5.2 System Architecture

The NIMBUS pipeline is based on the requirement that all components operate in parallel with the ability to scale up or down as required. This is a distributed system with no requirements for locality built into the system, and where possible, sequential workflows are eliminated from the system. While this pipeline uses an archive data cloud, the only requirement for a facility producing new data to be included in the cloud is for data to be made available via a web server, and be advertised on one of the worker queues.

5.2.1 Data Archive Cloud

The data archive cloud is comprised of storage nodes which are distributed web servers with mounted file systems containing files to be processed by worker nodes. Each of the storage nodes within the cloud provides a set of services to the pipeline as shown in the Table 5.1. Each service is accessed either via the control system or as HTTP requests. The function of servicing web based image file requests separated from the storage of image data allows for a flexible method of data storage and upload. The storage node architecture is shown in Figure 5.3.

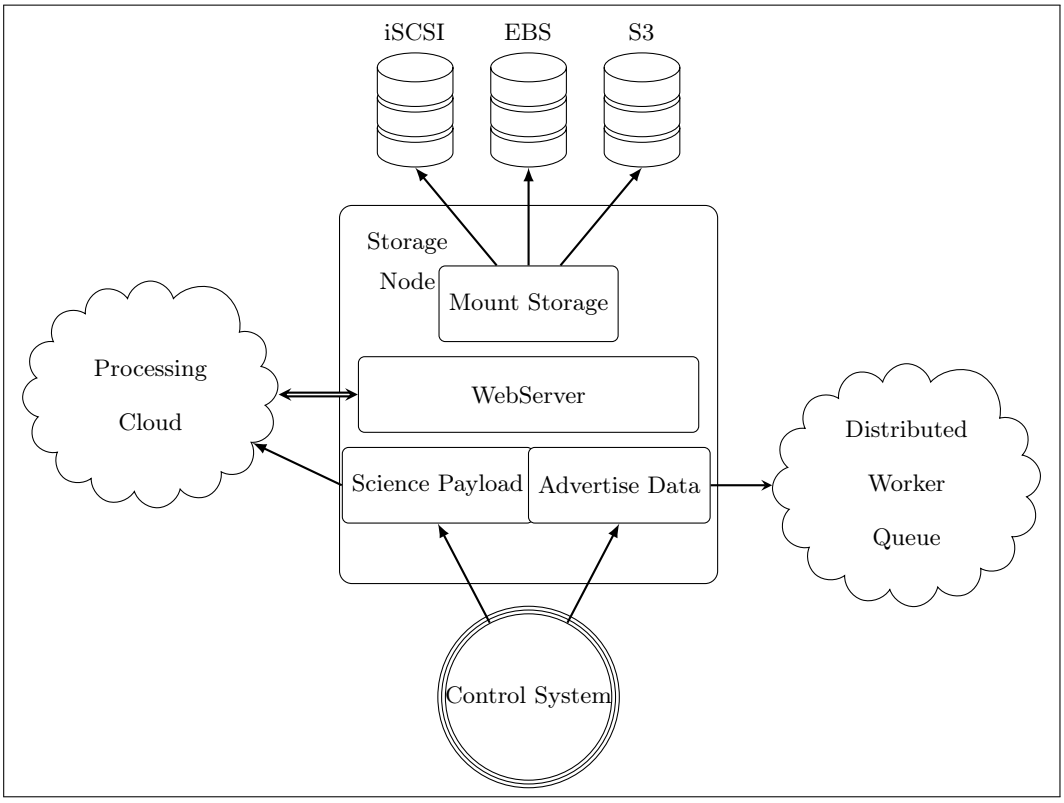


Figure 5.3: Storage Node Architecture

- **Upload** is the process of placing image files into a disk storage devices which is accessible

Service	Description
Upload	There are three primary methods of upload used, the first is for a storage device containing images to be mounted by the web server, the second is via HTTP POST requests to send data to the web server, the third is to instruct the web server to initiate a download via HTTP from another data source.
Storage	Image data can be stored on remote network storage system such as ISCSI devices, or it can be locally attached storage. For the purposes of the web server the method of mounting or attaching storage is irrelevant, however it may cause some delays in servicing files depending on the read time of the storage. Where possible files should be stored in a compressed state to reduce the file transfer times.
Advertise	A storage node will be required to populate an AWS Simple Queue Service (SQS) message queue with the URL of all of the files that it currently has stored. The storage node will do this via direct commands from the control system, but could also perform this via a HTTP command. Work is advertised and the contents of the message describe the location of the stored files and how they may be accessed.
Download	A web server is used to service HTTP requests providing access to files. This NginX webserver is highly optimised for servicing static pages and can be tuned to allow large number of simultaneous connections. Any web server however is permissible, including the Apache server and S3.
Science Payload	This service provides a static location for the downloading of the worker package to each worker. This is a small package containing all of the instructions a worker node will need to operate, including where to obtain work, where to put results, what work to perform and how to perform the work. The function of the payload and its operations are discussed within the Control system

Table 5.1: Storage Node Services.

to the web server of the storage node. No specific method is presented within the NIMBUS pipeline, but data transfer times have been reviewed in the ACN Pipeline in Figure 4.9. Within the NIMBUS pipeline the data archive cloud was populated with images downloaded from the ACN AWS S3 storage buckets using the s3fs FUSE system which mounts S3 storage buckets allowing files to be copied to mounted storage blocks.

- **Storage** within the pipeline is flexible to the point that any mounted storage device which can be accessed by a webserver can be accommodated. The speed at which a storage node serves file requests will be determined by the network, the number of concurrent request supported and the read speed of an image from the storage node. Three different configurations were constructed to demonstrate this flexibility within the implemented pipeline. The HEAnet iSCSI storage was mount on DIT based storage nodes while the AWS based storage systems used the AWS Elastic Block Storage service.

  An advantage of these devices is that through the use of the NFS file system, the storage device can be mounted for use by multiple servers. An NFS storage for example could be written to from a telescope location, but function as a read only mounted device running a web server. The performance of the read and write of the storage does depend on the configuration of the raw storage. If the storage uses multiple spindles then the write or read times may be quite reasonable. A disturbed file system such as S3 is likely to provide a more scaleable storage solution if there is a high rate of concurrent reading or writing.

- **Advertise** is the process of requesting a storage node to review the contents of its datastore and to create a message for each file found. That file is then written to an SQS queue as an advertisement of that file, indicating that it is accessible and ready for processing. The control system can instruct a storage node to review its storage and write the messages. In a production system it would be more realistic for the storage server to monitor for changes in storage and to create new messages when files were added to the data storage devices. The ability to reset and advertise everything rather than operate in an incremental fashion would also be a reasonable requirement. Message formats are explained further under the Distributed Queue Worker cloud section in this chapter.

- **Download** services the simple requirement of servicing static urls which provide access to files. Port 80 is a ubiquitously open port which allows all workers to access the image files. A worker node will read a message containing the URL of the image file and simply issue a request to download using a Python script. The NginX web server is a fast static web server which was used for most storage nodes. The entire pipeline works on the assumption that work is obtained through URL downloads.

- **Science Payload** is essential to the creation of workers which can be reconfigured easily. When creating $100's$ of workers it is a requirement that a worker can be dynamically recon-

figured before it starts performing any work. The science payload solution is initiated from the control system, but uses the storage node as a central point of advertising packages. When requested, the central web node responsible for package management removes the existing package from a standard location on the web server and creates and publishes a new version of the package. New package details are obtained from a central GIT repository.

5.2.2 Distributed Worker Queue Cloud

Central to the design of this architecture is the Amazon Simple SQS which is a distributed web based message delivery system. The service defines itself as *Reliable, Scalable, Simple, Secure* and *Inexpensive* and is one of many similar distributed web messages services such as RabbitMQ. A public message queue such as the AWS SQS system provides a transactional message system allowing for distributed processes to communicate. A transactional system ensures that messages can be read, held exclusively by one process and removed from the queue as required. The basic lifecycle of a message is shown in Figure 5.4. Messages are sent to a distributed queue where they are replicated and stored. When the message is read, the message visibility timer starts ensuring that nobody else can receive the message. During the visibility timeout the process which received the message can process the data and remove the message from the queue by deleting it. If the process fails to compete and does not delete the message, then the message will become visible on the queue once the visibility timeout has elapsed. The default timeout for messages within the system is 120 seconds.

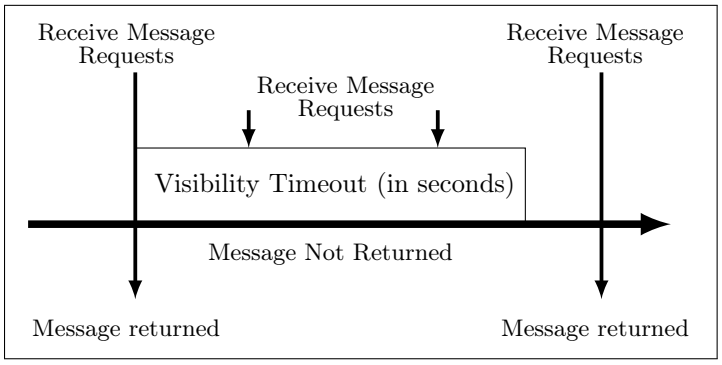


Figure 5.4: SQS message visibility.

A 2007 performance review of the SQS service supports the reliability of the service [114] and although these tests were run using EC2 instances, the service is available outside of the Amazon Cloud. It should be noted that these performance tests indicated a bottleneck reading messages at high speed, approximately 5 messages per second, however this was done using a single threaded system rather than using a distributed or multi-threaded system. As with many of the amazon services SQS offers distributed computing opportunities. A more recent study of the lag time for

messages being available from time of submission shows variability between 1 and 7 seconds [115], however for a queue that grows faster than consumption this is not necessarily an issue. Writing messages in parallel to the queue is also possible when there is little worry about the message order.

A key feature of a distributed queue is that it relies on the principle of eventual consistency, which means that given multiple SQS servers, each containing copies of the queue, when messages are written to one queue, there is a delay in syncing the message queues. It is also for this reason that message order is not preserved or guaranteed during delivery as shown in Figure 5.5. The advantage offered by the distributed system is that writing to a single queue is a distributed write operation allowing for the write operations to potentially occur in parallel. A series of tests were performed on message writing as shown later in the chapter.

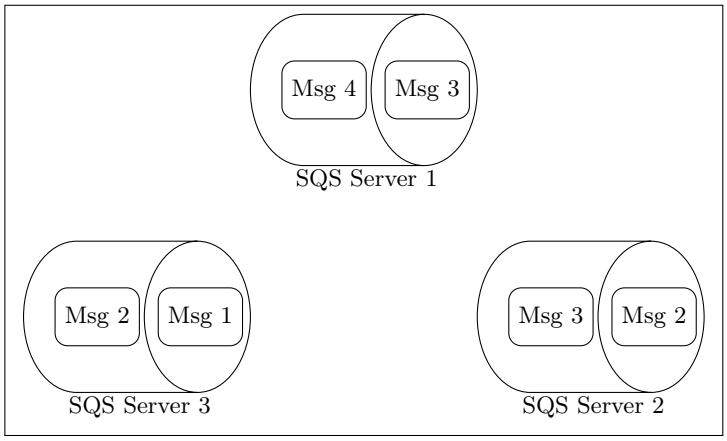


Figure 5.5: Distributed SQS Servers

Webservers were configured with data mounted from various data storage devices and used to advertise the raw images as URLs for download. An SQS message was constructed to represent each file in the data store which advertises a file available for processing, using the following form *http://webnode1.dit.ie/data/compressed/00-0001487.fits.fz*

A worker node will read the queue, download the file, and process it. There is an implicit relationship between the work performed and the work advertised. This is controlled by the work initialisation processes where workers download the worker scripts which include pointers to specific queues relevant to the worker script capabilities.

Within the NIMBUS architecture seven queues are used. Each queue allows for a worker process to work asynchronously while allowing a form of central control throughout the system. For each experiment each queue is deleted and then recreated in an empty state to ensure that it only contains data relevant to a specific experiment. Queues are deleted rather than emptied, for performance reasons, as it is possible for queues to contain hundreds of thousands of messages at the end of an experiment. Each of the queues and their function are explained below.

- **workerq** is the primary SQS queue containing the location of data files to be processed. Items are added to the queue by storage nodes which advertise their files by writing them to the queue as shown in Figure 5.6. Items from the queue are consumed by worker nodes which process the image and post the result prior to deleting the message from the queue. It is essential that the visibility timeout for the queue is set long enough for the worker node to complete the image processing before the message is visible. If a node fails to complete then the message reappears on the queue for another node to download and process. Storage nodes are instructed to access their storage directories and write a single message for each file found. The URL provided must be supported by the storage node through a web server running on the storage node. In all cases the NginX web server runs on each storage node and servers up files to requesting worker nodes. The message written to this queue is in the following form. *<NODEURL><filename>*. For example *http://webnode1.dit.ie/data/compressed/00-0001487.fits.fz*.

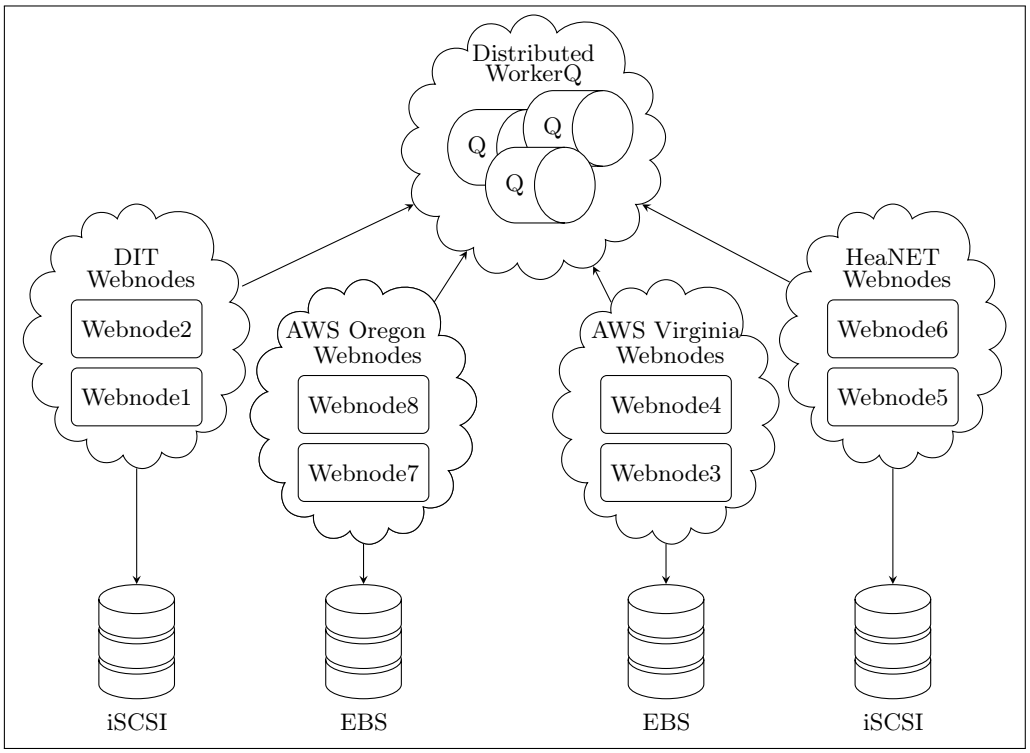


Figure 5.6: SQS Worker Queue constructed from 8 different storage nodes

- **supervisor** is an SQS queue read by each worker before it looks for work, and is tested before a worker checks the workerq. The supervisor queue contains approximately 100 messages all of which contain identical messages. The message visibility timeout for this queue is set to one second so that the message is put back onto the queue as soon as possible. At the start of an experiment the queue is deleted and reconstructed to ensure that all messages

Command	Description
REGISTER	This command requires that the worker writes a message to the register queue before performing any other actions. The register message format is as follows. <threadname><ipaddress><timestamp>
REBOOT	This command requires that the worker performs a hardware reset and reboots the machine
UPGRADE	This command is an essential service where a worker node is required to go to the service package location and download install an update to the software being used to perform work on the worker node. It is through this mechanism that a worker can be provisioned to service alternative jobs.
LISTEN	After a command has been performed the worker stops listening to avoid endless reboots, upgrades or registrations. Because the workers may come online at different times, the supervisor queue must continue to advertise its instructions, but workers should only perform the action once. This behavior can be over rotten by issuing the LISTEN command which allows workers to once again listen for a supervisor command and act upon it.

Table 5.2: Supervisor Queue control commands.

are identical. The reason that the queue is used is to control the behavior of a worker node, usually on startup and messages need to be all the same so that all workers behave in the same manner. This queue contains one of four messages shown in Table 5.2 "REGISTER", "REBOOT","LISTEN", and "UPGRADE". The supervisor queue is checked before the worker queue is read, and after each worker node cycle. Once a command is performed the worker stops listening for REBOOT, UPGRADE or REGISTER commands to ensure that endless cycles are not initiated by the worker. To reset the worker node to listen for these commands the LISTEN command is used.

Command	Description
START	This command informs the worker node to start processing messages from the worker queue. A timestamp for the start of work is taken and sent to the log queue. Messages are downloaded and a "do_work" script is used to process the files.
STOP	This command suspends message processing. A timestamp for the stop event is taken and sent to the log queue.
SLEEP	This command is pauses the woker node for 60 seconds after which time it will continue to look for commands on the cmdq queue.
QUIT	Nodes can be requested to terminate although this is an unlikely use case as the node will require a reboot to automatically join the worker node cloud. A QUIT command will stop the primary worker node software from running and it will not check any queues, effectively leaving the cloud.

Table 5.3: Command Queue control commands.

- **cmdq** is an SQS queue read by the worker after the supervisor queue has been read. Assuming that the worker node is now registered and running the correct software, this queue gives explicit instructions to the worker on how to proceed. The supported queue commands are described in Table 5.3.
- **workerregister** is an SQS queue which is used to identify the workers as they become activated. For some of the larger experiments the number of workers dramatically increases as there are multiple workers per node. The message written to this queue is in the following form.*<threadname><ipaddress>TIMESTAMP<data:time>.*
- **logfile** is an SQS queue which is a source of data used in the analysis of the system performance. All workers write messages to the logfile queue to indicate progress and status. A worker writes to this queue for key events during processing. For all messages written to the logfile queue, core information about the worker is inserted with the message including the version of the software running within the worker node. The following events generate messages. *Message Received* from the queue which is targeted for processing. *Message Processing Rate* which keeps a count of the total number of files processed since the worker has started and a private processing rate after each batch has been completed. *Message Deleted* which is called just after a file has been processed, uploaded to the target storage node and the result has been written to the result queue. This queue also contains some

diagnostic information from the worker such as any supervisor or cmd queue events which are processed. This queue generates more than double the number of messages processed. Where M is the number of messages processed by a single worker node, and B is the number of files downloaded for processing as a batch[1] a worker node will produce an estimate of log messages $\widehat{l}$ where

$$\widehat{l} = M \times 2 + \frac{M}{B} \tag{5.1}$$

This estimate ignores the comparatively few diagnostic messages generated by each worker. To estimate $\widehat{L}$, the total number of log file messages created during an experiment, sum the values of all worker nodes threads where there are n worker nodes[2].

$$\widehat{L} = \sum_{i=1}^{n} (M \times 2 + \frac{M}{B}) \tag{5.2}$$

If it is assumed that all messages in a worker queue are processed across all worker nodes, then the sum of all messages processed by all workers is equal to the total number of messages in the worker queue T. Hence the equation can be simplified as follows, with the assumption that batch sizes are equal across all worker nodes.

$$\widehat{L} = T \times 2 + \frac{T}{B} \tag{5.3}$$

The number of log messages created for a worker message queue of $500,000$ is $2 \times 500,000 + \frac{500,000}{10}$ which is equal to $1,050,000$. The use of the log file queue creates additional work for the worker node, slowing it down slightly, and establishes a need to process the log file queue in order to review the results of the experiment. These log files are used as a monitor on the pipeline to provide an alternative measure of the processing rates and worker node behaviour. Similar to the writing of messages, a sequential approach to reading queues will result in poor performance. A distributed set of reading processes are required. The message written to the log queue is in the following form.
<threadname><ipaddress>TIMESTAMP<data:time><Message>

- **resultq** is the SQS queue which is written to once a processed file has been successfully posted to its S3 destination. There should be a single message for all files processed, so the number of messages in this queue should equate to the number of messages in the worker queue once all processing has been completed. The message written to this queue is in the

[1] Workers can be configured to download a specific number of messages before attempting to perform any work on them. This is referred to as the worker Batch Number

[2] A worker node may run multiple threaded versions of the worker nodes which operate as distinct worker node processes

following form.
<threadname><ipaddress>TIMESTAMP<data:time><resultfilename>.

- **canaryq** is used as an instrumentation queue to monitor the structure and behaviour of the system. When a storage node advertises its work to the worker it contains the URL and filename of an image. During the processing of the workerq it is a requirement that the message be deleted. While it is possible to extend the visibility of a message within the queue, only $120,000$ messages may be left in this condition. Messages in flight [3] cannot exceed this value and thus the information within the workerq cannot be saved without the use of another queue. The canaryq provides the opportunity to create a mirror of the workerq which can be analyzed after an experiment has been run as it contains a record of the writing order of the queue by using a timestamp. Further analysis of the queue is provided later in this chapter. The message written to this queue is in the following form.
TIMESTAMP<data:time><URL><filename>.

5.2.3 System Monitoring

The monitoring system is a collection of different subsystems each contributing to the overall monitoring and logging of the pipeline. This includes the logging queue which contains detailed information from each of the running worker processes, a Python monitor which calculates the rate that the workerq is being reduced over time, the canaryq which can be used to reconstruct message delivery sequence, web server statistics using a system called Munin, and processing node performance for AWS EC2 instances using their AWS monitoring service for special *canary worker nodes*. This node has additional monitoring configured for each experiment with the assumption that it is representative of other nodes performance. This is checked by seeing if the canary processes clean a similar number of files to the rest of the nodes. Given the number of instances potentially running in an experiment, this approach provides a snapshot of a single instance performance which can be used as a generalisation of the other instances performance. When the experiments are running, the primary monitor is the workerq monitor which estimates the cleaning rate over time. The components within the monitoring system summaries below.

- **Web Server Monitoring.** Each web server is configured with the Munin monitoring software. This is used to track system performance helping identify bottlenecks and resource utilisation. The web server data is collected and shown as a series of graphs of system performance covering a wide range of system resources including Disk I/O, CPU, Networking and Memory utilisation. Using these metrics an experiment can be reviewed to determine how the web servers were performing.

[3] Messages in flight are messages which have been read from the queue and are within a visibility timeout period

- **Canary Instance Monitoring.** The AWS service provides monitoring facilities for EC2 instances and for SQS queues. Detailed monitoring is a paid service which is not enabled by default. When activated the time resolution is for 1 minute time slots and graphs are provided for Network In, Network Out and CPU utilisation. Assuming that the work performed by the canary instance is similar to that of the other others, then it's system performance metrics are a reasonable estimate of all instances performance. For most experiments the canary monitor is configured for AWS cloud monitoring.

- **Canary Queue Monitoring.** For some experiments, knowing the order of messages processed is required to provide an accurate view of web server contribution to the experiment over time. The canaryq queue is a full copy of the messages as they are processed including a central timestamp so all messages can be reordered.

- **Logging Queue.** This has already been discussed, but it provides an alternative view to the processing rate by allowing to determine how many files the pipeline or individual worker processed. Differences in processing rates by web servers and workers can also be observed.

- **Workerq monitor.** Use a call to the workerq every second, requesting the size of the queue, the rate of processing since the experiment began can be estimated. Because messages which are consumed by workers nodes may not complete, this is not a cleaning rate, just a consumption of messages rate. If for any reason a worker node fails to delete a message then it will reappear on the queue. In a live pipeline with images potentially being added on a regular basis, this queue monitor would not necessarily be as useful. For these experiments however all data is in place and queues allowed to settle before monitoring begins.

5.2.4 Processing Cloud

Worker nodes, or worker instances, are servers which run worker processes. Worker processes are individual program threads running on a physical or virtual machine. The majority of instances in this pipeline, for the purpose of experimental control, are virtual machines hosted on the Amazon Web Service using the EC2 service. For an experiment all instances are usually set to be the same type as explained later in the Control System subsection in this chapter. Depending on the configuration of the instance it may be possible to run multiple worker threads which could result in an overall improvement in the amount of files processed by the instance. For simplicity, the worker process is not written to be multi-threaded, rather the operating system is instructed to run multiple copies of the same program which are kept in insolation from each other. Resource sharing is handled by the operating system.

As with the previous pipelines the *acn-aphot.c* program is run at the heart of the worker process. A control system implemented in Python wraps this program in a service which provides image data for cleaning. The core cleaning program was not modified for this pipeline.

In order for a server to join the global processing cloud it must first install an initialisation script which is run when the server is first started. This script call be installed into an existing server, or as in the case of this pipleine a reference virtual machine was constructed with this script installed and an amazon machine image (AMI) produced from which large number of virtual machine instance can be created.

This initialisation script will look for a science payload on a public web server then download and execute it. The payload is designed to clean the instance directory spaces, install scripts and utilities, and clone itself to create multiple programming processes if required and then begin whatever tasks are set by the worker utility script.

To ensure that all experiments started in a consistent manner all experimental worker instances were deleted and new instances were created from the reference AMI. The primary lifecycle of a worker instance is shown in Figure 5.7 and described below.

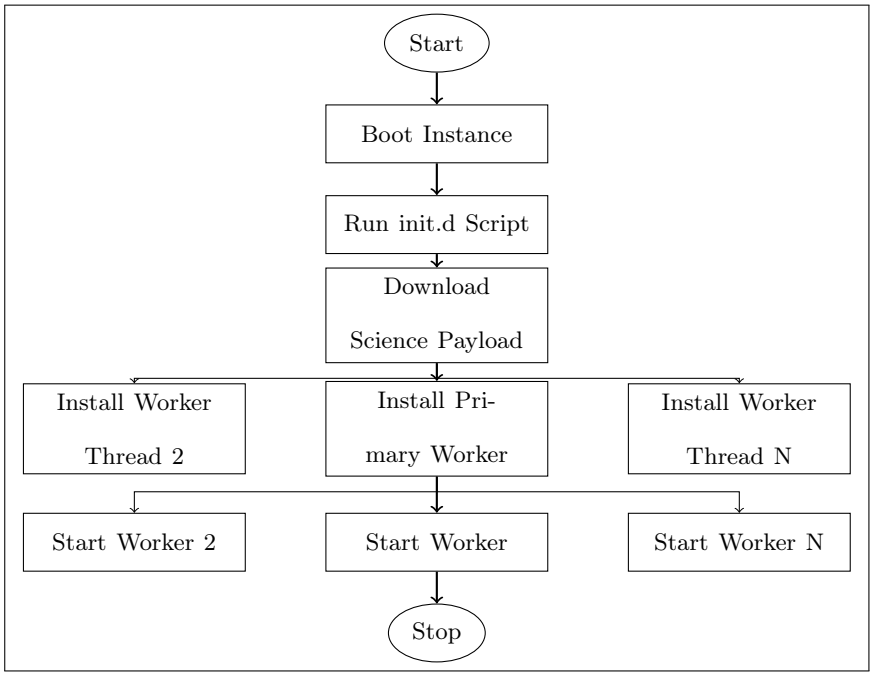


Figure 5.7: Flow Chart showing worker node initialisation during the boot up process

- **Start** The control system is responsible for starting the worker instances. A standard preconfigured AMI is used. In some experiments physical instances were also used.
- **Boot Instance** The standard Linux boot sequence brings the instance onto the network and mounts the local storage.
- **Run init script** The nimbus-worker init script is run which ensure that the server is cleaned of any previous experimental data.

- **Download Science payload** The science payload is downloaded, moved to the worker home directory on the server and installed. The work performed by a worker is detailed in the this package. This ensure that there is a flexibility to the system allowing for multiple types of processing to be performed depending on the payload. In most cases the worker will perform image cleaning and calibration of the FITS files, however any tasks are feasible. For the purpose of this pipeline the payload was altered to test network performance and web server performance by downloading files but not processing them, and by doubling the processing in an alternative payload to show the effects of a more CPU bound problem.
- **Install Worker thread 1-N** The installed system will create the required number of independent threads which are then run.
- **Start worker thread**. The workers are ready to process data and listen to the supervisor and command queue for the instruction to start downloading data to process.
- **Stop/Delete** The worker threads will sleep when there are no more messages to process, periodically checking for work. When an experiment is finished the control system may stop or terminate the running instance.

The flowchart for a running worker thread is shown in Figure 5.8. Each worker instance may have multiple worker threads all operating at the same time. All threads are isolated from each other however, with their own storage space and directory structure on the worker instance. The control system which operates this thread is a Python script inside which any science payload could potentially be inserted. The function of the worker thread is to listen to the sqs command queues for instructions on how to operated. Instructions such as start, stop, and sleep provide simple interfaces into the system. What to download and where to get it, is provided by the configurable message queue, and how to process it is determined by a central *do_work* script which then performs the required work. The output of the processing is then uploaded to a central web server and information on its location is pushed to a sqs based results queue.

5.3 Experimental Methodology

The function of the control system is to initiate all experiments and ensure that all systems are available and functioning correctly. It is important that experiments can be compared, and to do this the starting state must be consistent in all cases. The control system runs a Python script which tears down the experimental infrastructure, then rebuilds it before the start of the experiment. All systems must be accessible from the control system which resides on a virtual machine within the AWS cloud, running an Ubuntu instance on the EC2 service. A batch script contains the series of experiments to run, which in turn calls a script to start and experiment. The batch script contains a series of calls to the *run-experiment.sh* script, which takes a set of

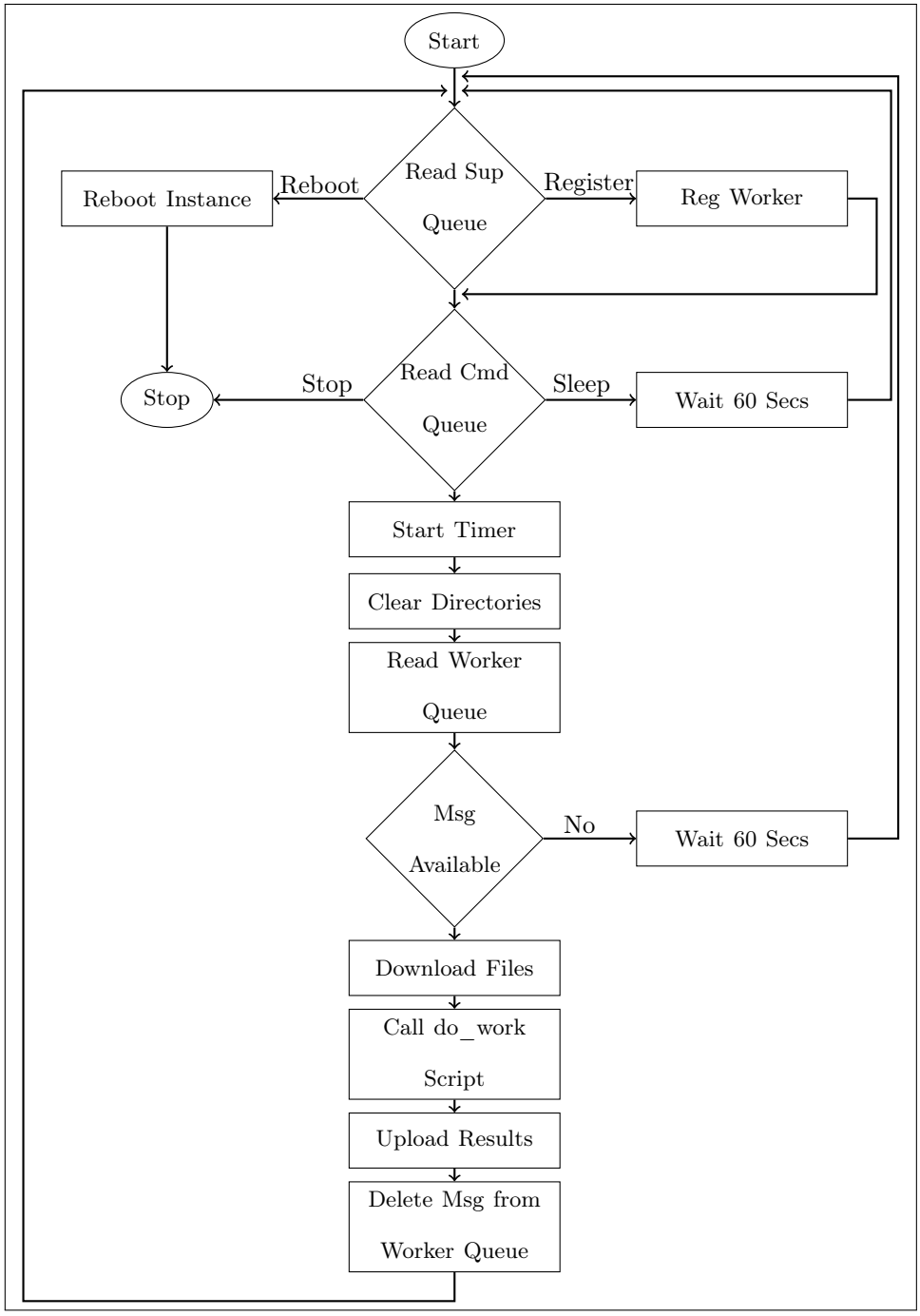


Figure 5.8: Worker control script managing the flow of work based on message reading status

parameters, shown in Table 5.4. The pipeline can also be left in a running state, which means that it will continue to process image files as they appear on queues one it has started, and new workers can be added or removed dynamically. The full workflow of the experimental run system *run-experiment.sh* is given in Figure 5.9. The experimental run script was required to ensure that the starting point of each experiment was consistent for each experimental iteration. A sample of the batch script used to run multiple experiments is shown in Appendix D.5.

Option	Description
-a | -d | -n | -x	Identify the combination of web servers to use in the experiment. -d uses the DIT based web servers, -a indicates the use of the AWS based web servers, -n uses the heanet based web servers and -x indicates that all web servers should be used. For a web node to be used in an experiment it will advertise the files it has storage to the SQS worker queue.
Instances	The number of instances of workers to run for an experiment less the monitoring node which always runs. The maximum number of concurrent instances is set to 100 within these experiments. This limit required explicit permission from AWS Ireland to run instances within the Irish region.
Time	The maximum amount of time in seconds that the worker nodes should be allowed to run. In most cases experiments were set to 20 minutes. An experimental timer was set only when the EC2 instances were initiated and confirmed to be running.
Name	The name of the experiment so that it can be identified.
Number	The number of web servers to run per type. Webservers are configured in pairs, so if a DIT web server is selected then either 1 or 2 will be allowed to run while 1-4 was the range allowable for AWS based web servers.
Size	The size of the AQS EC2 instance to run. The parameter conforms to the specific reference name that AWS uses for its instances. Most experiments used either the *t1.micro* the *m1.large* or in some cases the *m3.2xlarge*
Package Payload	In addition to changing the experimental options above, the package used by a worker node can also be reconfigured. The most significant change is specifying the number of threads a worker instance is allowed to initiate when running. This ranged from 1 to 100. The number of files downloaded by a worker before it begins processing the can also be specified. The processing batch size for nearly all experiments was set to 10.

Table 5.4: Experimental execution options for the NIMBUS pipeline.

The experimental run script is a Unix bash script which call a series of Python scripts on remote system to control a specific experiment. The Python programming API used to control AWS services is BOTO, which provides access to primary services such as S3, EC2 and SQS. To ensure that an experiment starts consistently each time it is run efforts are made to ensure there is little or no system caching being performed by any system. The S3 buckets are emptied to ensure that no previous results are being counted for the experiment. Queues are deleted and recreated to eliminate the chance of old messages contaminating an experiment. All worker instances are created from a standard AMI instance and are deleted at the end of each experimental run.

The experimental pipeline performance has also been considered where possible to ensure that steps in the process for creation or tear down of the system are as efficient as possible. Some of the steps in the experiments are primarily focused on experimental integrity but are not essential to a production pipeline, such as queue deletion, and S3 result deletion. Where large number of entries are placed on queues a multi-threaded queue reader is employed to extract all messages. As previously discussed the logfile contains the largest number of entries within an experiment and as such the reader for this queue downloads messages via multiple threads. The process of recombining these are sorting data into a sequential stream is the final generate metrics step in the flow chart.

In preparation for the experiment a science payload is created and pushed to a web server as shown in Figure 5.10. The payload will include details on how many independent workers are to be created on the instance and what work each of these workers should perform and how it should perform that work. The science payload consists of the following components:

- Datafiles directory. The specific location to download batches of files.
- Results directory. The location where results files are held until they are uploaded to the S3 service.
- Masterfiles directory. Contains the mastervbias and the master flat images for use in image cleaning.
- bin directory. Contains the acn-aphot compiled utility used to clean image files and the funpack utility to uncompress images.
- scripts directory. Contains the do_work.sh script which performs the image uncompress and calls the acn-aphot program to perform cleaning, and the start-worker.sh script which installs the required number of worker threads and starts them.
- sql-reader.py. The central control script which monitors queue, downloads images, passes them to the do_work script, and uploaded results.

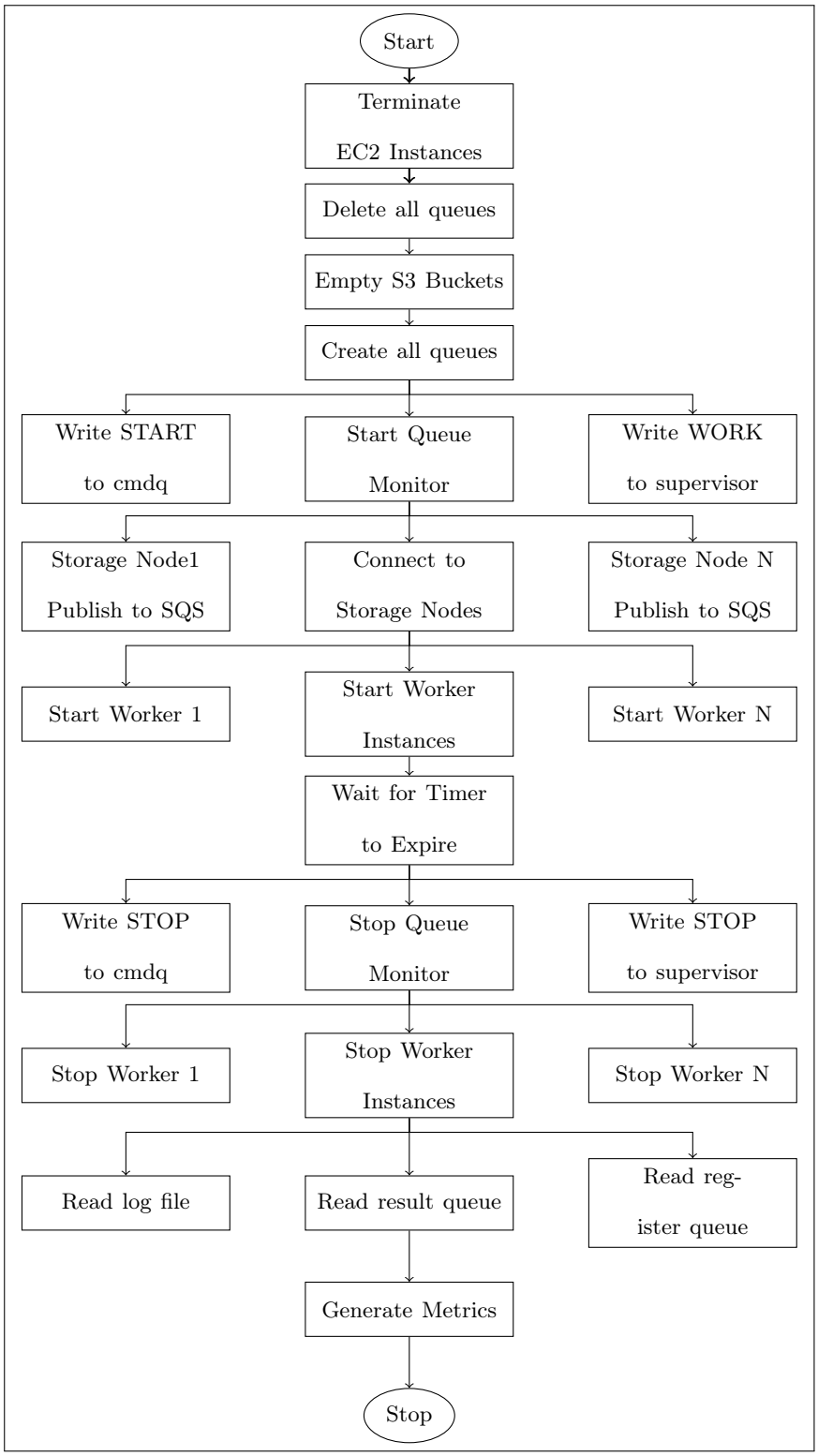


Figure 5.9: NIMBUS Experimental run script flow chart

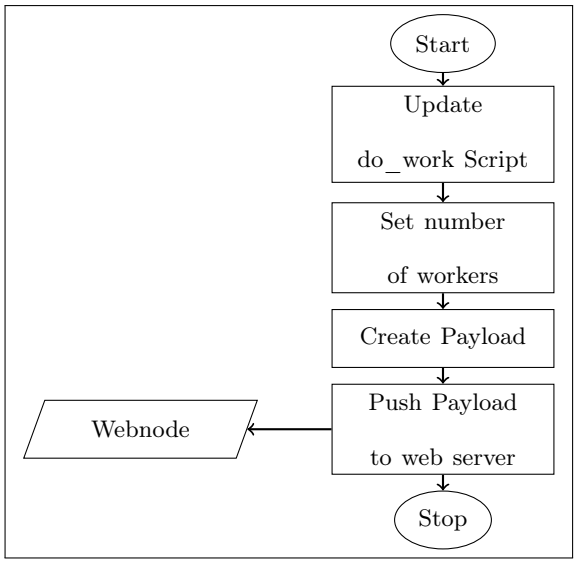


Figure 5.10: Flow Chart showing creation of payload for distribution to worker nodes

5.3.1 Experiment Metrics System

At the end of each experiment when the messages are written to the log file queue, the monitor queue, results queue and worker registration queue, a series of metrics are extracted to characterise the experimental behaviour. These metrics are used in the analysis of the experiment performance and to check that everything ran correctly.

- **log file queue** is a combination of all messages written by each of the workers to identify key events during the processing cycle. The metrics available from this queue are given in Table 5.5.
- **Result queue** contains messages written by workers after a message is successfully posted to the results storage system S3. Messages in this queue are either equal to or just slightly less than then total number of files found in the results directory. The metrics available from this queue are given in Table 5.6.
- **Worker register queue** is written to by each of the worker threads for all instances. This is used to ensure that the correct number of workers are active within the system to ensure that rate calculations are correct based on the number of workers. In most cases the workers all start, however if for any reason they fail, the registration message will not be present. The metrics available from this queue are given in Table 5.7.
- **Derived Metrics** are generated from the raw data using a spreadsheet to compare the data from multiple sources. Given the distributed nature of the data and the queues, and considering that workers may be still processing data when an experiment is concluded

differences in processing rates may be calculated using different metrics. It is also useful to measure specific attributes of the data processing performance such as Results-Not-Posted, which shows the number of files downloaded but not yet completed processing across workers. Details of these derived metrics are shown in Tables 5.8 and Table 5.9

- **Monitor queue** is not a specific queue but rather the processing of monitoring the worker queue throughout the experiment. Once the experiments starts the monitor will keep a record on the reported number of messages in the worker queue and perform a calculation of the rate of message processing over time. The metrics available from this queue are given in Table 5.10 although the raw data can also be used to plot the performance over time.

Metric	Description
Results-Posted	The confirmed total of all files posted successfully to the S3 bucket. This is an absolute count of work fully completed by all workers
Total-Downloads	The total of all files pull from all web servers.
Web-Server-Downloads	The total of all files pull from a specific web server.
Total-Instances	The total number of virtual or physical instances performing work during an experiment.
Total-Workers	The total number of worker threads running across all instances performing work during an experiment. This value is the same as instances if only 1 worker thread is running on an instance.
First-Start	The time that the first worker is first seen in the log file. Times are UTC based.
Last-Start	The time that the last worker is first seen in the logfile. Times are UTC based.
Last-Download	The time that the last file download occurred within a worker
Last-Upload	The time that the last result file was uploaded to S3 by a worker

Table 5.5: Metrics extracted from worker logfile.

Metric	Description
Results-Posted	The confirmed total of all files posted successfully to the S3 bucket. This is an absolute count of work fully completed by all workers
Total-Per-Thread	Each worker instance is a physical or virtual machine, and within each of the instances a number of threads can be activated to a maximum of 10. This total is the number of results posted for a specific thread across all instances
Total-Workers	The total number of virtual or physical instances performing work during an experiment.
Min-Worker	The minimum number of files processed by a worker thread across all instances
Max-Worker	The maximum number of files processed by a worker thread across all instances
Avg-Worker	The average number of files processed by all worker thread across all instances
STD Dev	Standard deviation of files processed by all worker thread across all instances
Total-Instances	The total number of virtual or physical instances performing work during an experiment.
Min-Instance	The minimum number of files processed by an instance thread across all workers
Max-Instance	The maximum number of files processed by an instance thread across all workers
Avg-Instance	The average number of files processed by an instance thread across all workers
STD Dev	Standard deviation of files processed by an instance thread across all workers

Table 5.6: Metrics extracted from Results Queue.

Metric	Description
Results-Posted	The confirmed total of all files posted successfully to the S3 bucket. This is an absolute count of work fully completed by all workers
Total-Per-Thread	Each worker instance is a physical or virtual machine, and within each of the instances a number of threads can be activated to a maximum of 10. This total is the number of results posted for a specific thread across all instances
Total-Workers	The total number of virtual or physical instances performing work during an experiment.
Min-Worker	The minimum number of files processed by a worker thread across all instances
Max-Worker	The maximum number of files processed by a worker thread across all instances
Avg-Worker	The average number of files processed by all worker thread across all instances
STD Dev	Standard deviation of files processed by all worker thread across all instances
Total-Instances	The total number of virtual or physical instances performing work during an experiment.
Min-Instance	The minimum number of files processed by an instance thread across all workers
Max-Instance	The maximum number of files processed by an instance thread across all workers
Avg-Instance	The average number of files processed by an instance thread across all workers
STD Dev	Standard deviation of files processed by an instance thread across all workers

Table 5.7: Metrics extracted from worker register queue.

Metric	Description
Results-Not-Posted	A count of the number of files confirmed to have ben downloaded from a web server but have not been posted to the results folder on S3. This difference is present when workers are either intentionally or otherwise terminated during processing of a file downloaded. It should be closely related to the messages in flight as the message is deleted after a result it posted to S3.
Results-Not-Posted %	The Results-Not-Posted metric expressed as a percentage of the total number of files downloaded
MAX-Time	The maximum amount of time that all of the workers were active. This is take as the time in seconds, between the last registration time of a worker (indicating that all workers were active) and the last download time from a web server from any worker. This is check to ensure that the approximate running time of an experiment is as expected.
AVG-Logfile-Rate	The total reported files uploaded to S3 by log file queue divided by the experimental time (1200 seconds).Setup time per worker may be included which is evident when reviewing the MAX-Time metric above which allows for workers to register and start downloading.
AVG-Result-Rate	The total reported files uploaded to S3 by result queue divided by the experimental time. A workers failure to write to either the log file or results queue during processing may result in differences with AVG-Logfile-Rate, however it should be a nominal difference.
AVG-Monitor-Rate	The total difference in the start queue size and the end queue size divided by the experimental time. This metric may be higher than AVG-Logfile-Rate and AVG-Result-Rate as it will include any messages in flight. In cases of high volumes of workers the queue size may not be accurate at that point in time.
AVG-Rate	Using the averages above an average of the overall average rate across the three metrics can be obtained.

Table 5.8: Composite metrics derived from multiple raw metrics.

Metric	Description
Best-Rate	The best cleaning rate achieved using the 3 average methods. In most cases this should be the AVG-Monitor-Rate as explained above. With larger volumes of process, or larger number of workers the monitoring system may not return an accurate queue size and may return a smaller size. This is due to the *eventual consistency* feature of the SQS queues.
Logfile-Rate-Var	The variance of the AVG-Logfile-Rate from the AVG-Rate. This can be used to determine if there are larger differences which may have skewed the calculation of the AVG-Rate.
Result-Rate-Var	The variance of the AVG-Result-Rate from the AVG-Rate. This can be used to determine if there are larger differences which may have skewed the calculation of the AVG-Rate.
Monitor-Rate-Var	The variance of the AVG-Monitor-Rate from the AVG-Rate. This can be used to determine if there are larger differences which may have skewed the calculation of the AVG-Rate.

Table 5.9: More composite metrics derived from multiple raw metrics.

Metric	Description
Start-Q-Size	Size of the worker queue at the beginning of the experiment. If the monitor observes the queue size increasing that it waits for it to decrease before beginning.
End-Q-Size	Each worker instance is a physical or virtual machine, and within each of the instances a number of threads can be activated to a maximum of 10. This total is the number of results posted for a specific thread across all instances
Start-Time	The total number of virtual or physical instances performing work during an experiment.
End-Time	The minimum number of files processed by a worker thread across all instances

Table 5.10: Metrics extracted from Monitor queue.

5.3.1.1 Experimental Parameters

Given the number of variables within the experimental setup, an exhaustive experimental approach would involve thousands of experiments. In total approximately 100 experiments were conducted with the primary aim of identifying the key variables within the system and to use that to maximise the performance of the NIMBUS pipeline within the given constraints of cost. The following seven variables were identified and where possible controlled for within each experiment.

- **VAR1: Web Server Location.** There are potentially significant impacts associated with the location of the web server providing files to the pipeline. While the configuration of 6 of the web servers was similar, the network between them and the workers was quite different. Network performance could have significant impact on the performance of a worker. The locations of the web servers has already been highlighted in Figure 5.6. For most experiments, the AWS US East (Virginia region) web servers were used while the remaining web servers provided scaling options for larger scale experiments. The fastest performing web server was provided on loan from HEAnet [4], a high performance balanced web server highly tuned for large data transfer when the experimental requirement was to eliminate the web server performance from an experiment. This resource was used in a number of experiments producing the highest system performance.
- **VAR2: Number of web servers.** Initial testing reviewed the impact of increasing the number of servers that served files to workers. Multiple servers allows for the testing of the network performance when running small numbers of workers. There are 8 Nginx web servers and 1 FTP cluster used.
- **VAR3: Number of Instances.** An instance is a virtual or physical machine running a unix environment capable of running 1 or more workers. The instances are primarily AWS EC2 ubuntu machines and the number run in an experiment ranged from 1 to 100.
- **VAR4: Type of Instance.** The EC2 instance types available range in size and configuration. Experiments can be run with a number of EC2 instance types such as T1.Micro, T1.Large, M1.Large, M1.XLarge and C1.Xlarge. For a subset of experiments two physical machines were also used. A Sun/Oracle x4150 and an IBM i326e Server. Details of the specification of these given in Appendix Table D.3
- **VAR5: Workers per Instance.** For both single and multicore systems the use of multiple threads running independent workers allows for an investigation of the balance of CPU and Networking resources on an instances. Instances with more CPU cores should potentially improve the performance of the system if multiple worker instances are run.

[4] HEAnet is Irelands National Education and Research Network

- **VAR6: Experimental Time.** Most experiments ran for 20 minutes which was deemed long enough for the processing rate to be determined, and to ensure that a large volume of data could be processed. Some exceptions to this occurred near the final experiments to demonstrate that the processing rate could indeed be maintained over longer periods of time.
- **VAR7: Batch Size.** The C program which processes images can process all files it finds in a directory. If there is a single data file found then the file is read in along with the Master Files. The images are processed, the results posted and the C program terminates and is restarted for the new batch. If the batch size is 10 then the Master Files are read in and held in memory while each of the 10 files are processed. This reduces the number of times the C program is started and stopped and reduces the file I/O by only reading the Master Files once per batch instead of once per data file.

5.4 Results and Discussion

This Chapter defined and executed a series of experiments to determine the overall performance of the NIMBUS system architecture presented in this chapter. The results of each experiment are broken down and analysed to provide a comprehensive view of the system. In some cases the experiments cover multiple components of the system, but in all cases the context of the results and their contribution to the overall pipeline are discussed. There are four basic sets of experiments performed, message queue performance, single and multi node instance scaling, and pipeline limit testing. Table 5.11 shows the high-level experiments performed which are further broken down within this chapter. The web queues must be fast enough to service high levels of concurrent requests and the globally distributed computing nodes should be able to scale linearly until bottlenecks are observed. If bottlenecks are found then the architecture should be flexible enough to work around them. All data sources for all experiments and graphs are identified in Appendix Table D.2 which references an accompanying supplementary USB disk which contains raw and processed data relevant to these experiments.

Limits imposed on the experiments were based on limits of available resources, although where possible indications of scaling opportunities were identified. For the pipeline to be active a minimum of 1 worker is required to perform image cleaning and reduction. Multiple workers processes can run on a worker node/instance, which is typically a virtual AWS instance. The maximum number of instances activated within the experiments was 100, but the maximum number of workers was $10,000$. In some cases multiple runs of the same experiment were performed to ensure results were repeatable.

It is required that the processing rates used within these experiments are expressed correctly and consistently. While the original data set is stored as multiple images per file, and the raw

Reference	Measure	Description
Exp:NIM1	SQS Performance.	Testing the read and writing times of the web message queues
Exp:NIM2	Single Instance Node Performance.	Determine the variables which affect the performance of the overall processing power of a single instance.
Exp:NIM3	Multi-Instance Node Performance.	Focus on scaling the number of instances up to 100 looking for factors which could affect the scalability of the system.
Exp:NIM4	System Limits.	Identify the full scalability of the pipeline and to identify strategies to continue improving the system performance

Table 5.11: NIMBUS Experiments Overview

results from the experiments were measured as files per second, or images per second, a more useful representation of the processing rates is the amount of data processed over time. The conversion from files to bytes also needs to take into account that the data being processed is compressed so the concept of equivalent uncompressed data rate is also given. The following values and calculations are central to correctly determine processing rates.

- F Uncompressed File Size. An unprocessed image data cube is 7.297920 MB.
- F_c Compressed File Size. An unprocessed compressed data cube is 1.247040 MB.
- F_i Images per File. The number of images within an image data cube is 10.
- I Images size. An unprocessed compressed image.

$$I = \frac{F_c}{F_i} \tag{5.4}$$

- C Compressed Rate. The size reduction of an image using the *fpack* utility.

$$C = \frac{F_c}{F} \tag{5.5}$$

- P_{fps} Processing rate in files per second. The number of files processed per second using the NIMBUS pipeline.
- P_{gps} Processing rate in GB per second.The number of gigabytes processed per second of compressed data using the NIMBUS pipeline.

$$P_{gps} = \frac{P_{fps} * F_i * I}{1024} \tag{5.6}$$

- $\widehat{P}_{gps}$ Processing rate in equivalent uncompressed GB per second. The number of gigabytes processed per second of equivalent uncompressed data using the NIMBUS pipeline.

$$\widehat{P}_{gps} = \frac{P_{gps}}{C} \tag{5.7}$$

Reference	Measure	Description
Exp:NIM1-1	SQS Write Performance Single Node	Testing the writing time of a single storage node using a series of threaded applications
Exp:NIM1-2	SQS Write Performance Multi-Node	Testing the performance of the queue when multiple sources writing to it
Exp:NIM1-3	SQS Distributed Read Performance	Testing the distribution of messages from multiple nodes and the impact this has on queue read performance
Exp:NIM1-4	SQS Queue Read Rates	Testing the read speed of an SQS queue using a series of threaded approaches

Table 5.12: NIMBUS SQS Performance Experiment Overview

- $\widehat{P}_{gph}$ Processing rate in equivalent uncompressed GB per hour. The number of gigabytes processed per hour of equivalent uncompressed data using the NIMBUS pipeline.

$$\widehat{P}_{gph} = \widehat{P}_{gps} * 3600 \tag{5.8}$$

- $\widehat{P}_{tph}$ Processing rate in equivalent uncompressed TB per hour. The number of terabytes processed per hour of equivalent uncompressed data using the NIMBUS pipeline.

$$\widehat{P}_{tph} = \frac{\widehat{P}_{gph}}{1024} \tag{5.9}$$

5.4.1 Simple Queue Service (SQS) Performance

To achieve a data cleaning rate of terabytes per hour it is essential that the queuing mechanism is able to advertise data sufficiently quickly to present work at a rate higher that the expected cleaning rate, and to ensure that work creation rates are expandable as the number of files to be cleaned increases. This requires that the storage nodes within the NIMBUS architecture can collectively create messages on the SQS worker queue at a rate of over 100 messages per second[5]. In addition to writing messages to the queue to generate work, the architecture of the system requires that queues are also used for monitoring and obtaining the results of an experiment. Experiments were devised to determine the sqs queue read performance. A full list of the sqs experiments are given in Table 5.12.

[5] 100 messages for a 10 image data cubed file represents, in this system, 700Megabytes of raw data where each message points to a file of 7MB. 700MB per second $\approx$ 2.4 Terabtyes per hour

5.4.1.1 Exp:NIM1-1 SQS Write Performance Single Node

This test investigates the performance of a single server writing to an SQS queue to understand how quickly a storage node can advertise files to be cleaned. A series of experiments was executed on a single storage node with modifications to the utility used to write to an SQS queue. The storage node platform used was the IBM i326 server with 4GB of RAM and a 1 Terabyte remote mounted iSCSI storage drive.

Four different approaches were used for writing messages. Sequential writing using a single threaded application, sequential writing spawning a new process per message write, multi-threaded processing for varying numbers of threads and connections to the SQS system. The results are given in Figure 5.11.

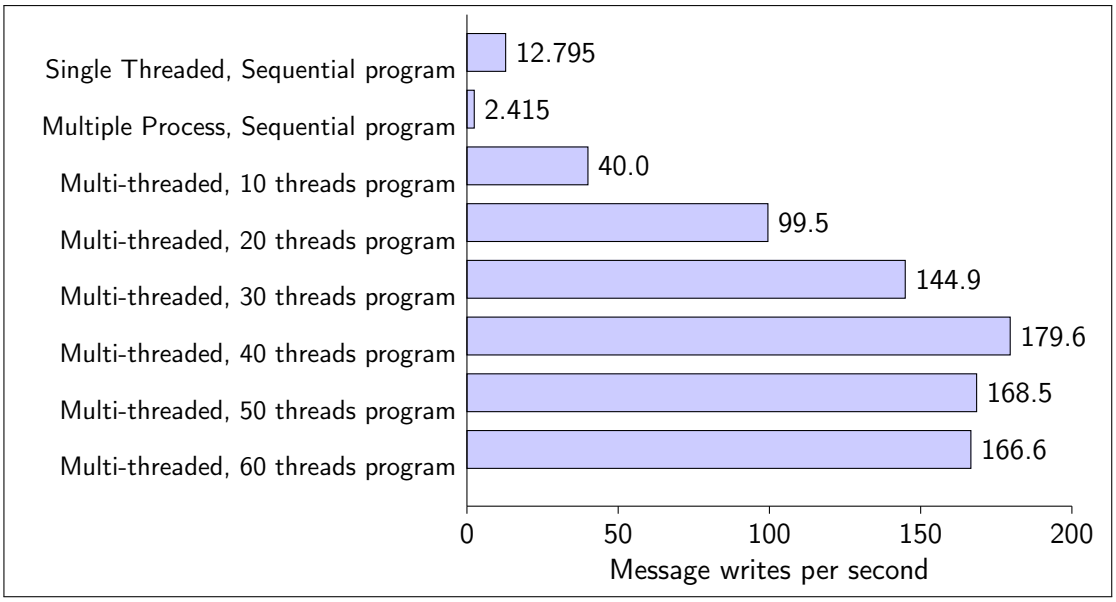


Figure 5.11: Exp:NIM1-1 SQS Write Performance Single Node. Average message writing time per second from a single web server node using varying methods

The multi-threaded application used can be found in the Appendix D.2. Further description of the various writing approaches are described below.

Single Threaded, Sequential program

This approach uses a single Python program which connects to the SQS queue and writes, in strict sequence, 1,000 messages. The message writing rate is the time on average for a single message to be written. This time takes into account the connection time which is spread evenly across all of the writes. As more messages are written the connect time to the queue would become less significant.

Multiple Process, Sequential program

Using the Linux environment to create multiple processes, a program was written which

forked 1 processes per message. The poor performance of this approach can be attributed to the creation time of the process and the fact that every single message requires a new connection to be made to the SQS queue. Each forked process also requires its own memory footprint within the server.

Multi-threaded

A multi-threaded program where 10 threads are created, each of which writes $\frac{1}{10}$ of the messages to the queue. In this case there is no cost for creating a full forked process for the entire program but 10 connections are set-up to the SQS queue, one per thread.

Multi-threaded single connection

Further optimisation of the Python program allows for the multi-threaded system to spawns an arbitrary number of processes. For the purposes of this experiment a set was selected ranging from 20-60 threads which the results posted. In this program there is a single connection made to the SQS which is shared by each of the threads. A Python queue is created which loads all image filenames into it and allows the threads to use that queue to generate messages. Threads now balance their work rate using the Python queue, with faster threads writing more messages. This modification also allows the system to take arbitrarily large numbers of files to the Python program. (see Code Listing in Appendix: D.2. Results for varying threaded values are shown in Figure 5.11 with the optimal performance shown around 40 threads. For the system tested this result provided the fastest message writing rate but this thread number is not the optimal number of threads for all systems, as that would be a factor of the number and performance of CPUs in the system.

The single connection with multiple threads allowed for almost 180 messages written per second which on its own is greater that the required writing performance to generate a work list of over 3TB of data per hour. While a larger server may perform even faster, the next experiment focused on having multiple servers write to the queue at the same time from different locations.

5.4.1.2 Exp:NIM1-2 SQS Write Performance Multi-Node

The results from Exp:NIM1-1 show a message creation rate of approximately 180 messages per second, however the testing environment relied upon a local data store from a single node. Further experimentation was required to consider the effect of different network storage configurations and server types within the NIMBUS architecture, and the effect of multiple nodes concurrently writing to the SQS queue form different locations. This experiment is designed to test the scalability of the SQS queue to determine what message writing limitations may exist or may be relevant to this pipeline.

The previous approaches implemented strict sequential message writing per thread, and due to the fact that the threads were on the same system, an element of performance balancing between

threads is to be expected by the underlying operating system. If the threads were running on different storage nodes then a better understanding of the SQS central queue performance can be obtained. Of primary concern is the rate of message writing performance when the number of overall processes writing messages to a single queue is increased. Using our multi-threaded Python message writing script on a number of different web nodes two things are testable. The first is to see if the overall message write rate for the queue will increase as the number of nodes writing is increased, and secondly will the rate of increase be a factor of the number of nodes running.

To exploit the distributed nature of the SQS queue in Exp:NIM1-2, 8 web servers were configured, each containing image data which can be advertised to the worker queue. This experiment attempts to see if a maximum write time could be obtained for the queue. Each of the 8 web servers contained approximately $73,000$ raw image files. Initially they were run in sequence to get a baseline of the speed at which they could write messages to the queue, next all of the nodes were run in parallel with the primary objective of all 8 writing messages onto the SQS queue concurrently. Given that each of the configurations and locations of the web servers were different as shown in Figure 5.6, the network connectivity to the SQS queue was varied. It would be expected that the difference in the performance of writing would arise as a result of issues such as network latency and processor performance.

In Figure 5.12 the message write time is shown for a storage node run in isolation, and when all of the storage nodes are running. It can clearly be seen that the performance of the SQS queue was a function of the number of nodes. As the number of nodes increased, the number of messages written also increased. Running in isolation, or with other storage nodes running and writing messages, the performance of storage nodes writing was relatively unchanged. Using the longest write time for a single node, an estimate of the average message write performance W_{msg} can be obtained where n is the number of nodes used, and $\max T$ is the maximum time in seconds for one of the nodes to complete writing. The total number of messages M written by all 8 nodes in this example is 596,3658 with the last node finishing after 9 mins giving a message processing rate of 1104.5 messages per second (See Appendix Table D.1 for data details) which is equivalent to 26TB of data advertised per hour.

$$\widehat{W}_{msg} = \frac{M}{\max T} \tag{5.10}$$

On consideration of the results presented in Figure 5.12 there are clear variations in storage servers writing rates but that introduction of additional servers writing to the queue does not impact the rate of writing for individual nodes. The message write rates for each server are shown in Figure 5.13. The rate of writing for each node can be expressed as the number of messages m written over time t.

$$\overline{w} = \frac{m}{t} \tag{5.11}$$

Figure 5.12: Exp:Nim1-2 Comparison of sqs queue write rates per second web node for standalone or multi-node writing. Source Data in Appendix Table D.1

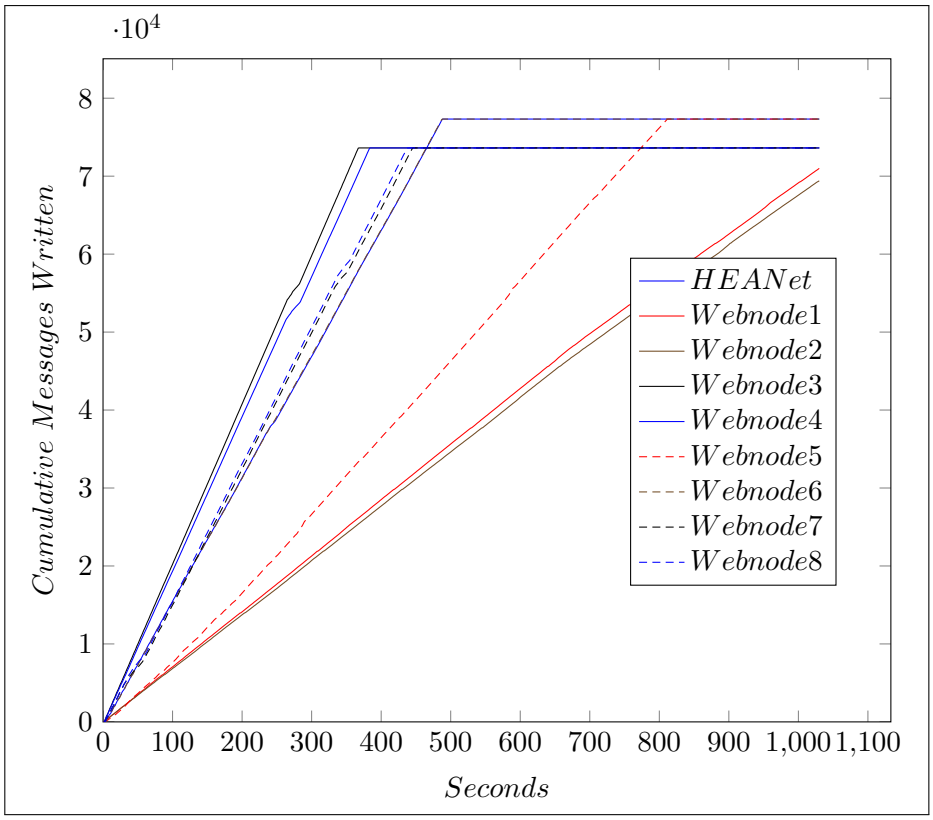


Figure 5.13: Exp:NIM1-2 Messages written to the queue over time for each storage node.

A more accurate estimate of the system queue write rate is the sum of n storage nodes individual write rates, $\overline{W}$ as shown in equation 5.12.

$$\overline{W} = \sum_{1}^{n} \overline{w} \tag{5.12}$$

The data used in these graphs was based on the retrieval of messages from the SQS canary queue, which was written to by the web nodes at the same time as messages are written to the SQS worker queue. This data was taken from an experimental run where data was being processed using 9 web nodes. Because worker queue messages are deleted as they are processed the canary queue offered a mirror of the queue. When reading the canary queue the SQS attribute *SentTimestamp* was requested which provides the Coordinated Universal Time (UTC) for the message write time to the queue. Message write order is reconstructed by combining all messages and sorting them into time buckets (in this case seconds), and for that time period determine the number of messages written from each web node. See code listing D.1. Figure 5.14 plots the same data with a simple moving average with an interval of 9 using a central moving average. The raw data plot for this image can be seen in Appendix Figure D.1. The reduction in the total write rate over time is due to the fact that the node writing faster have run out of messages, so are no longer contributing to the system.

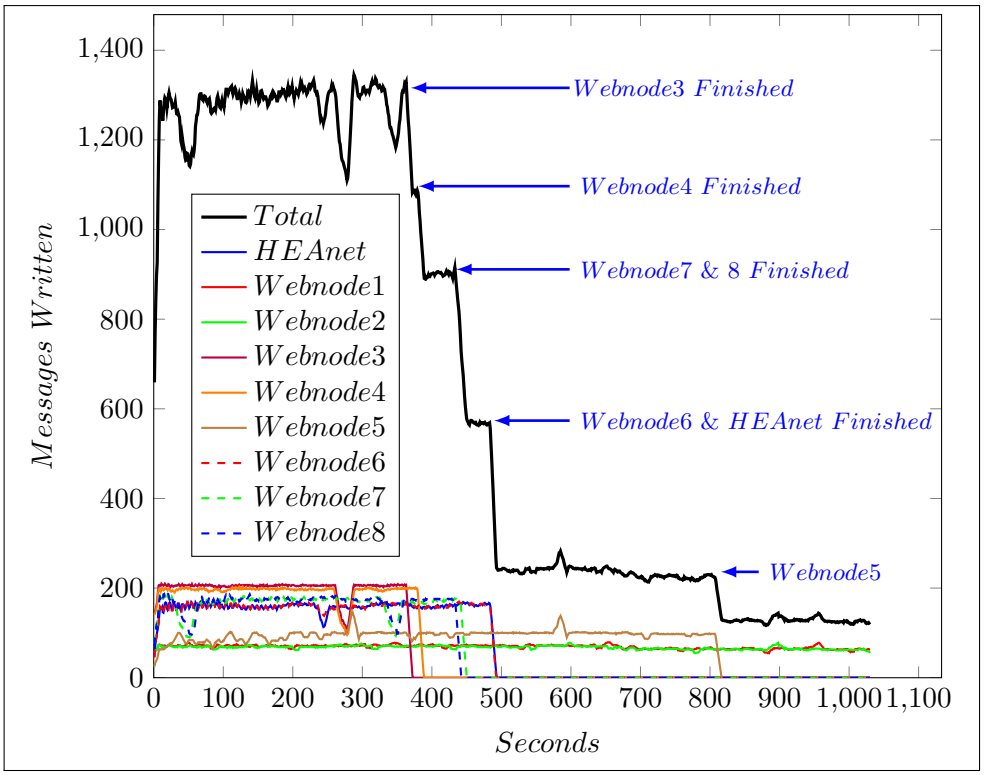


Figure 5.14: Exp:NIM1-3 Simple moving average of canary queue SQS message *write rate* for all storage nodes.

5.4.1.3 Exp:NIM1-3 SQS Distributed Read Performance

As the SQS queue does not guarantee any specific order to message delivery it is important to understand the mix of messages on the queue as presented to multiple readers, and to have an appreciation of how that can affect a running system. From results presented in Figure 5.14 it can be seen that the system wide message write rate reduces for the experiment as faster web nodes exhaust the number of messages they have to put on the queue. If the queue delivered messages in the same order as received then the same distribution of messages would be expected when the queue is being read. A complication is that there is no single read point for this queue as messages are read by multiple nodes. Given the distributed nature of the queue the message order will now be split over these reading nodes. The SQS system does not provide a message read timestamp so this cannot be used to reconstruct message read order. It is possible however to reconstruct an approximation of the message delivery order to the reading worker nodes.

To simulate the worker node read behavior all messages are read using a multithreaded Python program[6] which spawn 40 threads, each of which creates it's own log file that preserves the order in which the messages were read. The population standard deviation of the number of messages

[6] Code Listing D.3

read by each thread is calculated [7] and the relative standard deviation is obtained, which in this case is 0.25%. It can be assumed that each of the threads is downloading messages at a similar rate, so by grouping each of the files into bins of 100 messages and combining them[8], It can be determined for each bin how many messages were downloaded from each storage node. Figure 5.15 plots this data using a simple moving average with an interval of 17 with a central moving average. Consider webnode3 which wrote messages to the queue faster than any other node, it provides consistently higher rate of messages early on in the graph and has all messages read before other nodes. The slower webnode1 starts slower and finishes last. There is evidence of more random behaviour midway through the read process. This may be attributed to a combination of the distributed nature of the SQS queue and the method used for combining messages logs.

While it has been shown that the read order is not identical to the write order it is reasonable to assume that nodes which contribute messages to the queue at a higher rate than other storage nodes will have their messages presented to worker nodes more frequently than those that contribute at a slower rate.

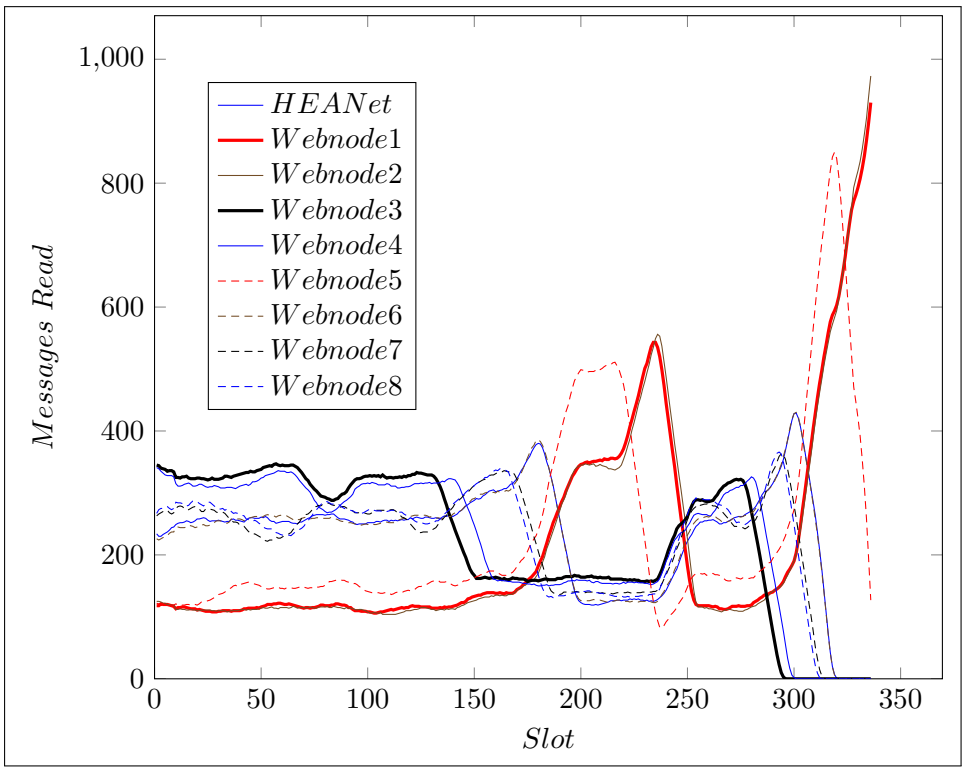


Figure 5.15: Exp:NIM1-3 Simple moving average representation of messages read within 350 sequential time slots with 40 threads reading the SQS canary queue.

[7] $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$

[8] Code listing D.4

5.4.1.4 Exp:NIM1-4 SQS Queue Read Rates

The issues of reading queue message in a none distributed or parallel manner are consistent will the issues associated with writing messages to a queue. For each message read there is a connection overhead, but in addition to this there is also the overhead of deleting the message from the queue creating a double cost to message reading. When a message is read, a visibility timeout it set leaving the message invisible to others. Messages within the visibility timeout are referred to as a messages in flight, and the limit for the number of messages in flight is set currently at $120,000$. This provides a potential limit on the performance of a system using a single worker queue. A system which has the capacity to read more that that number of messages at one time, will be blocked reading messages until the total number of messages in flight is reduced below that threshold. This requires that either a message is returned to the queue or deleted by the reading worker.

While this is a limitation of the AWS SQS service it is not necessarily a limiting factor on the pipeline as it is possible to operate a larger number of queues, each containing file information for processing. If a pipeline was limited to operating at $120,000$ messages per second this would equate to $1.2 Million$ images per second using image data cubes of 10 stacked images, which in this pipeline would represent about 1 Terabyte per second, or 2.8 Petabytes per hour.

The issue of reading queues quickly is more related to the processing of post experiment analysis than with the running of the system. With the pipeline's distributed nature, and the use of batch downloading per worker node, the processing time of the images is the limiting factor for the image processing rate, not the message download time for each worker. The use of the queues for distributed sharing of log files and result files however does require some thought on queue read performance. The reason a queue system is used to record result information and log file information is that it allows the worker nodes to be more independent. By centralising key information about their processing to a central queue, all of the pipeline log messages are available in a single queue, although there is no specific reason why multiple queues could not be used.

If the log files are being monitored for issues with workers or with processing then they need to be constantly read and monitored with key value pairs [9] being sought to identify issues. The read rate of a log file queue must therefore have a similar read rate to the worker processing rate. A monitoring process reading the queue will be doing considerably less processing that a worker node, so it should be able to process messages considerably faster, or allow for multiple monitors to operate simultaneously (which is the case). A single monitoring server reading queue messages can read approximately 100 messages per second when running multiple threads, as shown below in Figure 5.16. Multiple monitoring servers could therefore reasonably be assumed to be able to read all log or result messages produced by a fully functional operating pipeline. Using Equation

[9] A key/value pair could be a specific pattern within the log file data indicting an unusual or important state for a worker, such as WorkerID:STOP

5.3 it can be estimated that the message read rate would have to be slightly greater than twice the message processing rate of a system since each worker generates just over 2 messages per image file processed. If a single monitoring system can read 100 images per second, then the total monitoring node requirements is the message processing rate of the system per second divided by 100.

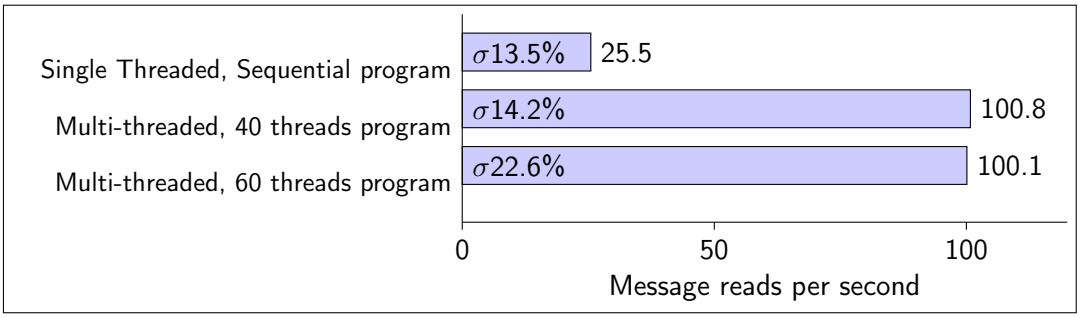


Figure 5.16: Exp:NIM1-4. Messages read per second from a single monitor server node using varying levels of threads running with the standard deviation shown.

A number of strategies were considered for log file message reading which are briefly discussed.

- **Serial Download** requires a single-threaded application to read messages from the queue one at a time. It is possible to increase the number of messages taken from the queue at a single read, however this did not provide any significant performance improvements on the message read performance. This approach, as shown in Figure 5.16 is not an optimal solution.
- **Multi-Threaded Download** provided considerable improvements in the overall message download rate with the number of threads being a configurable number similar to the Python listings showing message writing. In this case messages are deleted once they have been read so the queue is constantly reducing over time.
- **Multi-Threaded Download with messages in flight** is a faster message read given that the message is not deleted, but rather messages are given an exceptionally long visibility read time when downloaded. When a message is read, the time it remains off the queue can be set. If no messages are deleted then the message limit of $120,000$ is a system bottleneck after which no messages can be read until that number is reduced. This method only works for queues which are relatively small. In the experiments performed, queue messages can reach over 1 Million messages.
- **Multi-Threaded Download with messages in flight hybrid** is a compromise solution which estimates the number of messages within the queue and has a policy of deleting a proportion of them to take advantage of the messages-in-flight mechanism, while never allowing the maximum number of messages to be in flight. For large queues however, the advantage is diminished over time.

5.4.1.5 Analysis

The message queuing system provides a number of advantages to the overall pipeline.

- **Exp:NIM1-1**. A single web node can advertise data at a rate of approximately 3TB per hour.
- **Exp:NIM1-2**. Using multiple web nodes writing at the same time, the advertising rate for the pipeline is over 26TB per hour, although this is unlikely the limit as write rates were linear with the number of web nodes added.
- **Exp:NIM1-3**. Due to the fact that some nodes may write to the queue faster than others, in the experiments where there are fixed numbers of messages written by each worker node, the faster nodes will initially contribute more to the queue and reduce over time. This may cause the process rate to appear to slowdown over time.
- **Exp:NIM1-4**. A single node read performance for messages is similar to the single node write performance. Downloading of messages is naturally distributed for the pipeline. A limit per queue existing which is equivalent to a processing rate of 2.8 PB per hour. All that is required to overcome this is to increase the number of queues being used for reading.

5.4.2 Single Instance Node Performance

This group of experiments focuses on understanding the behaviour of a single instance by looking at variables within the pipeline to determine how to optimise the instance performance. Give the low volume of data being processed the results of the baseline experiments are shown as P_{fps}, files processed per second.

There are three different variables which are tested in these experiments to determine their impact on a single instance performance, the size of the instance (Memory, Network, CPU), the number of worker threads running on the instance, and the web servers being used to provide images to clean. All experiments run batches of size ten (downloading 10 messages before processing begins), and all work performed by a worker is identical. Experiments are broken down by variable under test and shown in Table 5.13

5.4.2.1 Exp:NIM2-1 Single Instance Webserver Performance

For this experiment the web servers used are those identified earlier in this chapter, and shown in Figure 5.6. NginX web servers were run in multiple location, on both physical (DIT & HEANET) and virtual environments (AWS EAST containing 1 webserver, and AWS WEST which had 4 webservers), in addition to a high performance FTP server from HEAnet which also serviced HTTP requests. The worker instance was run on an AWS EC2 instance and the instance type was T1.Micro, which is the smallest of the free tier system available. The worker instance ran 5

Reference	Measure	Description
Exp:NIM2-1	Single instance web-server performance	Testing the impact of different web server configurations to service a single worker instance.
Exp:NIM2-2	Single instance performance by type	Testing the impact of selecting different instance machine types.
Exp:NIM2-3	Single instance multi-worker performance	Testing the impact of increasing the number of worker threads run on an worker instance.

Table 5.13: NIMBUS Single Instance experiments

worker threads. Figure 5.17 show the files per second processing rate of the instance (combining the processing of each of the individual workers). From this graph it would appear that the web server location, number of web servers and type has little impact on the performance of the server instance.

A more detailed look at the breakdown between the time to download and the time to process provides further information as shown in Figure 5.18. Using a kernel density function to plot the download times for each worker during an experiment, it is clear that the AWS web servers takes longer to download files compared to the DIT, HEANet or FTP web servers. Given that there are no other demands being placed on the web servers these results are good indicators of the maximum performance capabilities of the worker instance. In Table 5.14 the mean, standard deviation and variance for file downloads is presented, showing a clear difference in the mean for the AWS servers. It is also evident from the low variance that the web server response times are reasonably consistent. In Table 5.15 the mean processing times of the workers is reasonably similar, but the variance is quite high. Given that amazon report that the T1.micro instance has CPU throttling, this would appear to be evident with some workers experiencing considerable longer delays than the mean. With the mean values for processing times higher for the experiments where faster downloads occur it would seem to indicate that all workers are more likely to be processing data at the same time. As processing times become longer, then the CPU is most likely closer to being close to a bottleneck.

It can be concluded from these experiments that there is a performance benefit from using the non-AWS web servers for single instance processing. With limits inherent within the capability of the T1.Micro instance, these benefits my not be evident, however using faster performing server instances, improvements in file processing rates could be expected. This will be explored further in Exp:NIM2-2.

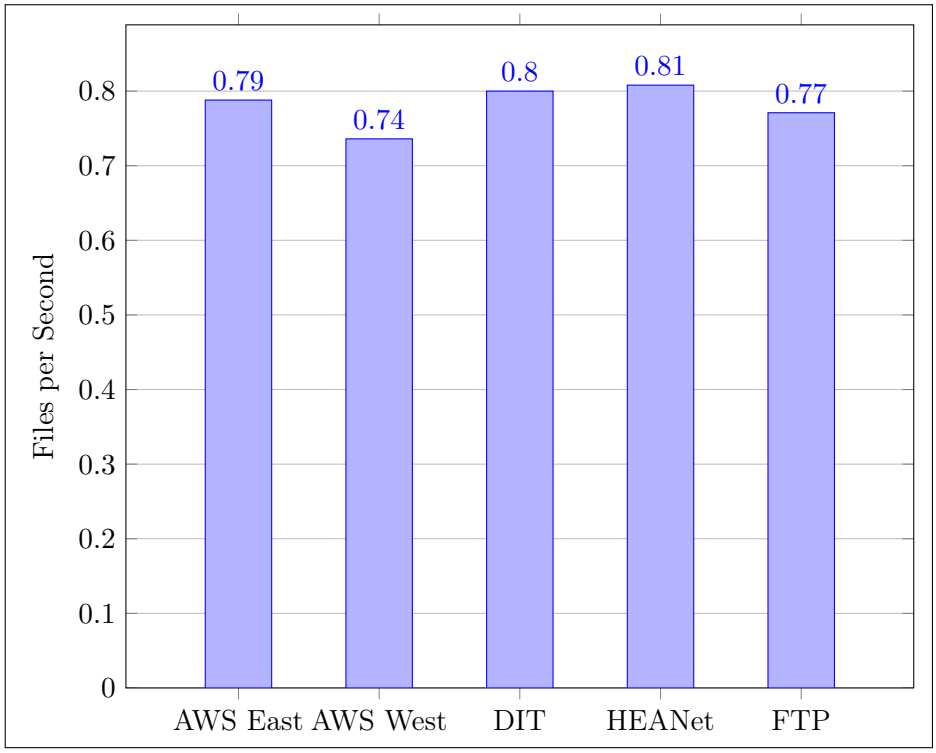


Figure 5.17: Exp:NIM2-1 Files Per Second: Varying Web servers and their impact on T1.Micro Instance performance

Web Server	Std. Dev.	Mean	Variance
FTP	1.565	4.59	2.451
AWS East (1)	2.759	22.49	7.616
AWS West (4)	3.278	25.00	10.746
HEANet	2.775	4.92	7.704
DIT	2.791	5.05	7.795

Table 5.14: Exp:NIM2-1 T1.Micro Single Instance 5 Worker statistics for image downloads per web server

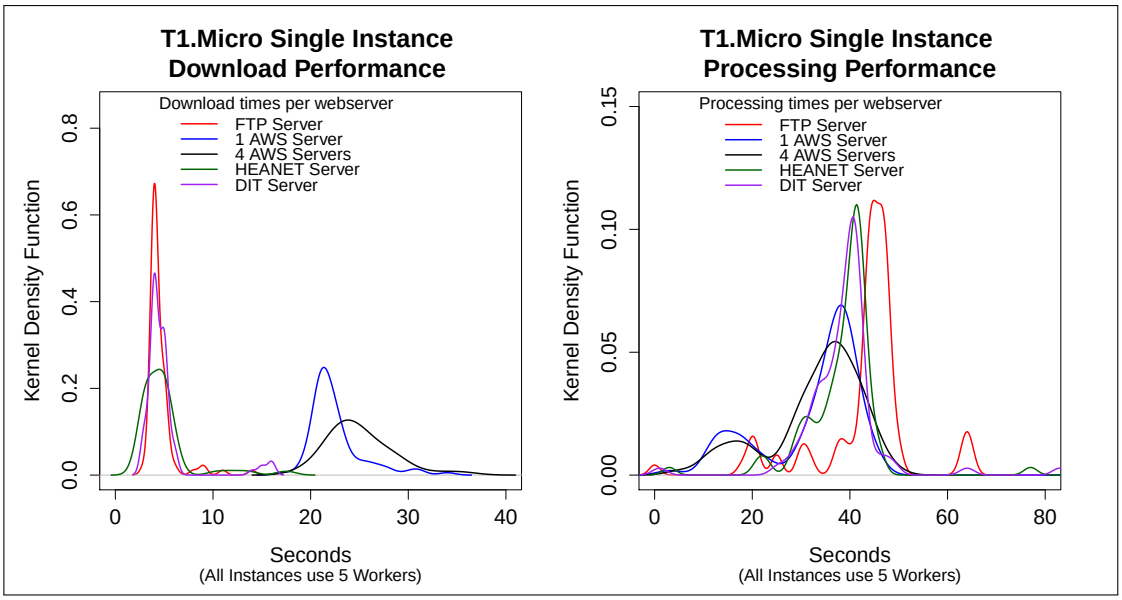


Figure 5.18: Exp:NIM2-1 Single Instance Performance breakdown for different Web Servers, with 5 workers.

5.4.2.2 Exp:NIM2-2 Single Instance Performance by Type

Given that the FTP web server from HEANet provides a high performance load balanced web server, that variable can remain constant for this next set of experiments. Looking at the performance of a single instance running a single worker in Figure 5.19 it is clear that varying the type of server instance makes little difference. In this experiment 1 physical server was used (the IBMe326) and two different AWS instance types (T1.Micro, and M1.Large). There is a slightly longer delay in download times from the IBM server but this is likely to do with the network differences between the location of the virtual amazon instances and the physical IBM server instance. If the differences between the web servers is processing power then further experiments

Web Server	Std. Dev.	Mean	Variance
FTP	19.59	47.05	383
1 x AWS	18.66	36.17	348
4 x AWS	9.70	32.94	94
HEANeet	23.05	45.38	531
DIT	23.18	45.65	537

Table 5.15: Exp:NIM2-1 T1.Micro Single Instance 5 Worker statistics for image processing per web server

are required which increases the processing demand of the webserver. In Exp:NIM2-1 there were 5 workers running per server, while in this experiment there was only 1 worker running.

To verify that the web server would have an impact on the instance, a comparison of the T1.Micro and the M1.Large virtual instances performance using both the FTP and the AWS web server is shown in Figure 5.20. In this diagram the processing remains similar but the networking time for the downloads is clearly slower when using the AWS web servers. To further explore the instance type performance the number of workers running to determine how to take advantage of the additional server performance must be considered. This will be explored further in Exp:NIM2-3.

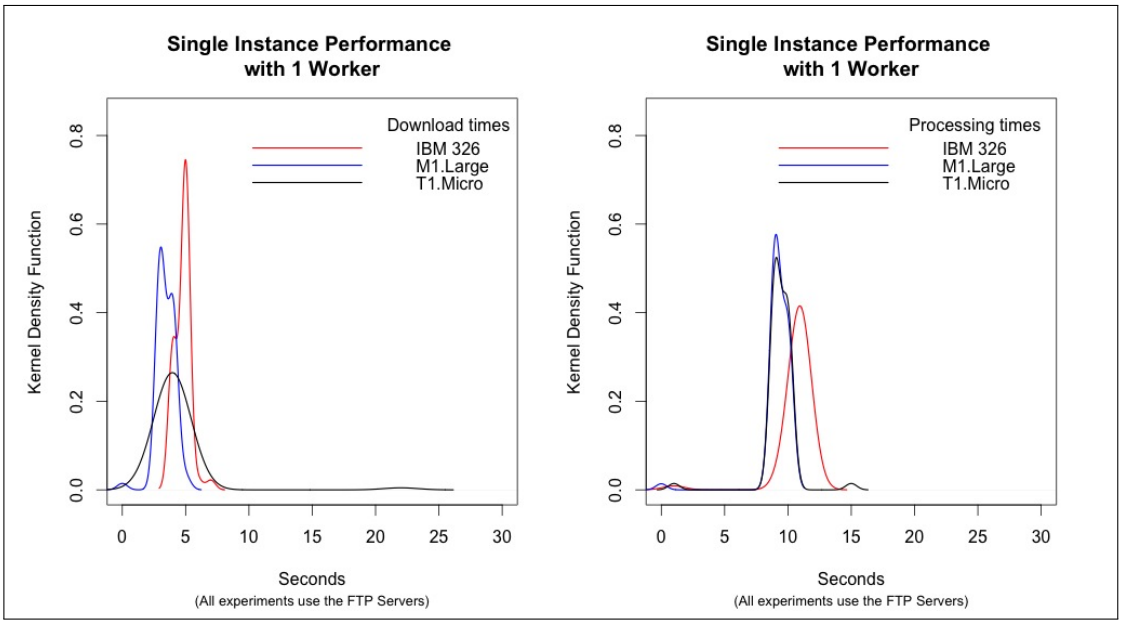


Figure 5.19: Exp:NIM2-2 Single Instance 1 Worker Performance breakdown for different Instance types, using FTP Webserver

5.4.2.3 Exp:NIM2-3 Single Instance Multi-worker Performance

For this group of experiments a variety of physical and virtual machine instances are used to look at the impact of running multiple workers on the same instance. The assumption is that if an instance is busy downloading an image then the CPU resource is not being used. To fully utilise the CPU, additional workers can run to balance the load of the CPU over time. Workers are designed to cycle through downloading batches of files, processing them, and then uploading them. A single worker will not use all of the instance resources fully at the same time. By increasing the number of workers it would be reasonable to assume that the overall resources are being more fully used, but that there is a point beyond which the number of workers being added does not increase the performance of the instance.

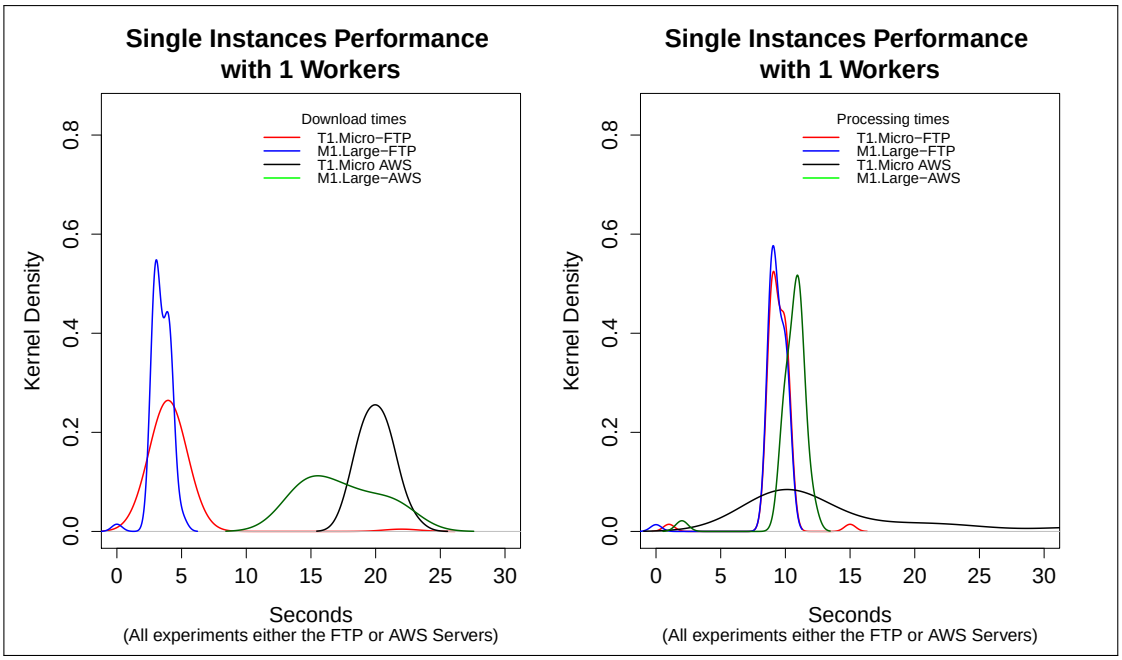


Figure 5.20: Exp:NIM2-2 Single Instance 1 Worker Performance breakdown, varying web server and instance type

Since different instances have different performance characteristics such as CPU, RAM and Networking, the performance characteristics should be varied across different server types and configurations. Using the T1.Micro, M1.Large and the IBM 326 physical server for 1, 5, and 10 workers the processing rate can be drawn as shown in Figure 5.21. An increase in the rate of files processed is seen across all instance types as the number of workers increases from 1, however as the rate goes from 5 to 10, the effect is less significant in all cases. The instances with more resources show the greater gain.

To verify the relationship between increased CPU capacity and an increase in the capacity to run more worker threads, an additional experiment (see Figure 5.22) was run on an x4150 server which has 4 dual quad core CPUs compared to the M1.Large which has 2 CPU cores. The x4150 was run with 10 and 50 workers. The larger server runs significantly faster using 10 workers compared to the M1.Large, but while running 50 workers there is a levelling off of performance. Assuming that the network and the web server were not a bottleneck at this point, then the CPU is the most likely limiting factor.

Figures 5.23 and 5.24 compare, at a high level, the rate of cleaning of files per second by both the T1.micro and the M1.large instances, varying web servers and the number of workers per instance, with the data shown in Table 5.16. As expected the AWS versus the non-AWS web servers have an overall impact on the instance performance, but the increase in the number of workers is initially significant for an increase to 5 servers, but as the number of workers grows the performance improvement does not improve in a linear manner.

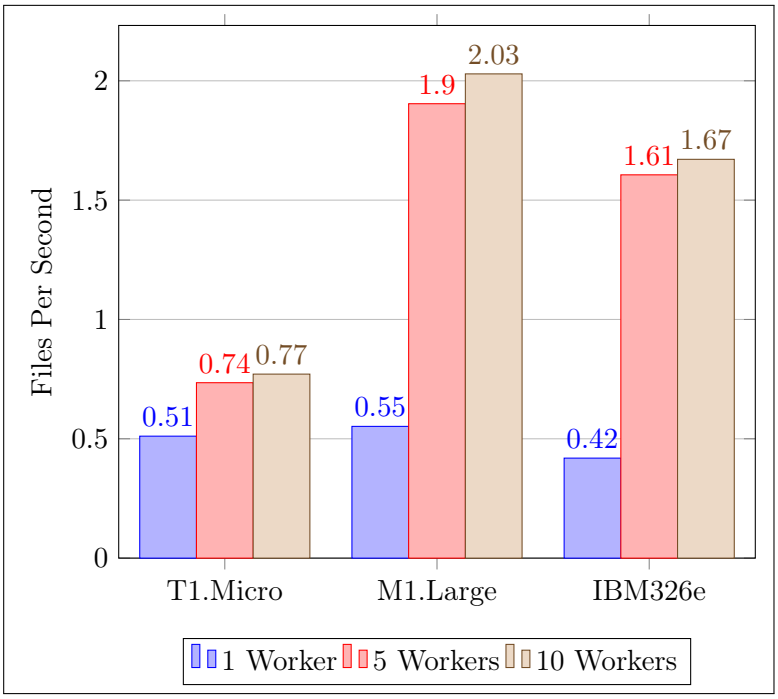


Figure 5.21: Exp:NIM2-3: Single instance performance by type, running multiple number of workers.

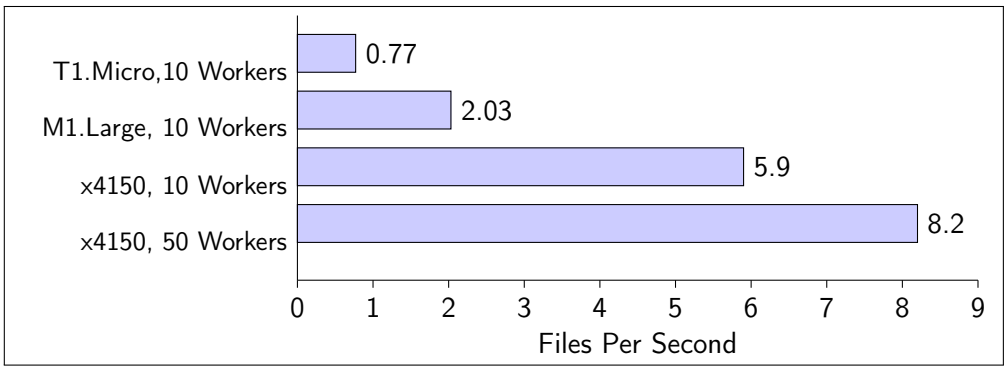


Figure 5.22: Exp:NIM2-3 Increasing the number of workers on faster CPU servers

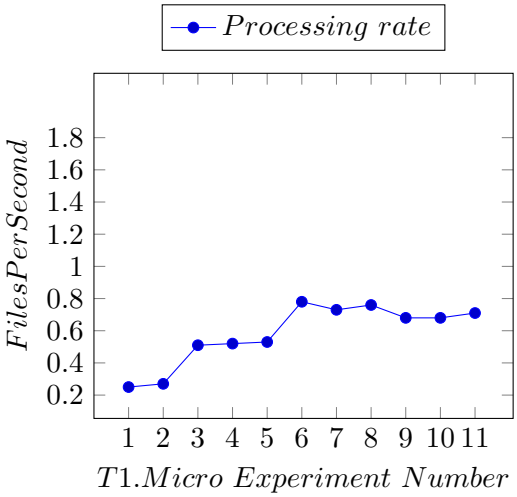


Figure 5.23: Exp:NIM2-3 Single T1.Micro Instance file cleaning rate per second.

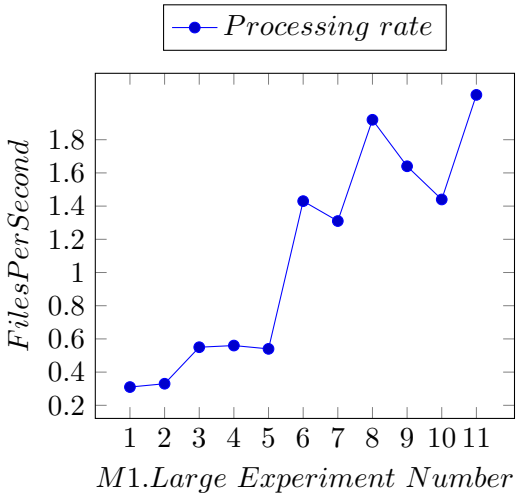


Figure 5.24: Exp:NIM2-3 Single M1.Large Instance file cleaning rate per second.

Exp. Num.	Web Server	Num. Workers	Num. Web Servers	t1.micro fps	m1.large fps
1	AWS East	1	1	0.25	0.31
2	AWS East	1	2	0.27	0.33
3	FTP	1	1	0.51	0.55
4	DIT	1	1	0.52	0.56
5	DIT	1	2	0.53	0.54
6	AWS East	5	1	0.78	1.43
7	AWS East	5	2	0.73	1.31
8	FTP	5	1	0.76	1.92
9	AWS East	10	1	0.68	1.64
10	AWS East	10	2	0.68	1.44
11	FTP	10	1	0.71	2.07

Table 5.16: Single Instance Experimental Results Table for Fig 5.23 and 5.24

A more detailed comparison of the experiments is shown in Figure 5.22 where the same web server is used in all cases, and selecting the first three experiments where the same number of worker threads are running. This ensures that the primary variation is the CPU capability of the server. In Figure 5.25 a density probability plot of time is shown, where the worker performance is split into download times and processing times. The processing power of the instance becomes the primary bottleneck when the web server and network are relatively unconstrained as with the t1.micro, but not for the x4150. It is clear that each instance will require a different number of workers to maximise their performance.

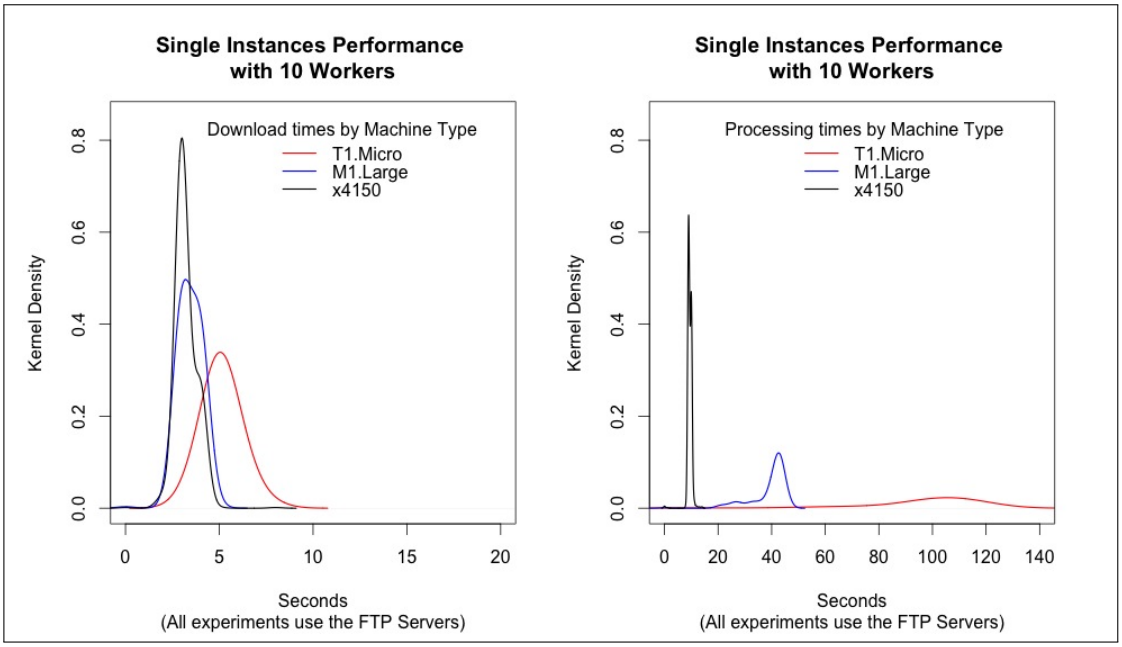


Figure 5.25: Exp:NIM2-3 Single Instance Performance breakdown with increasingly powerful servers

Finally a comparison between the T1.Micro and the M1.Large is shown in Figure 5.26, breaking down the performance of each worker by time to download files and time to process them, showing results for 1, 5 and 10 worker experiments.

From these experiments it has been shown that the number of workers on a server instance has the ability to grow the overall performance of the server w.r.t file processing rate. The rate of increase is dependant on the exact configuration and capabilities of the server, but number of CPU cores does appear to be relevant. There is a point beyond which the number of workers added will cause a negative impact on the servers processing rate, For future work, it should be possible for an instance to modify the number of worker threads it runs to maximise the overall instance processing rate. The would require a local instance monitor to operate with the ability to start-up new workers or shut them down. This could be accomplished by either multi-threaded modifications to the existing workers or to create additional sandboxes for the single threaded

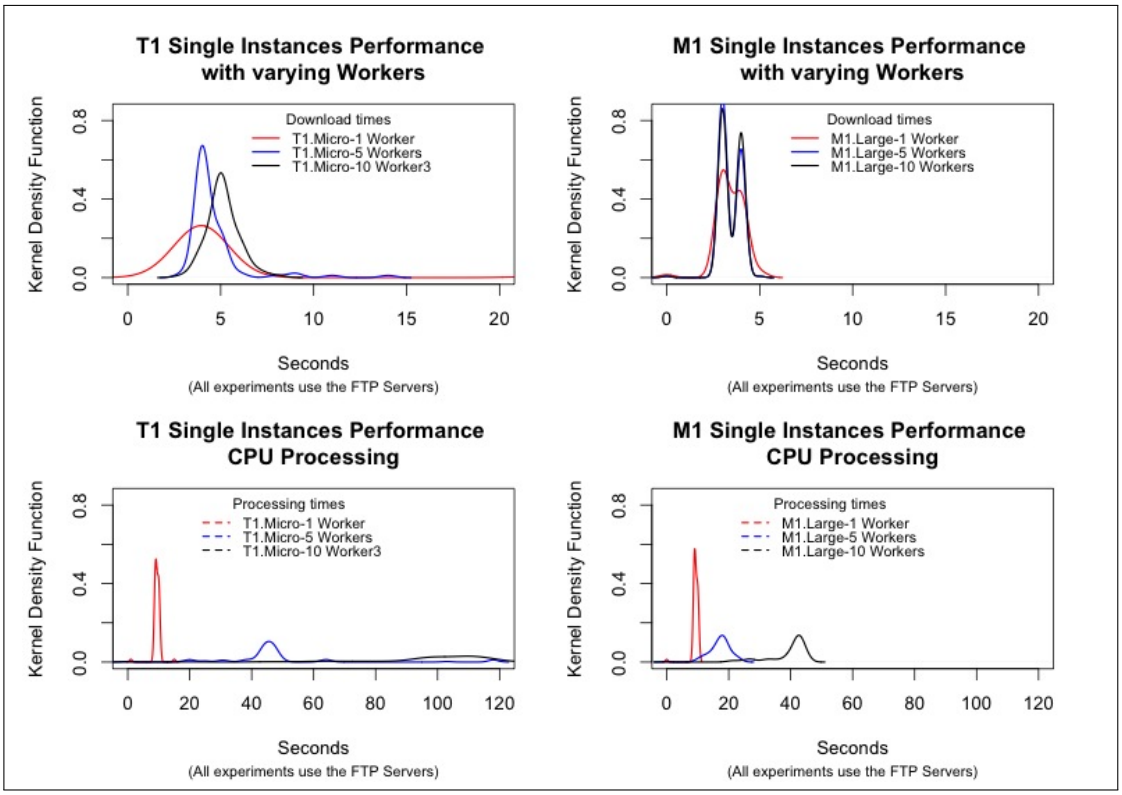


Figure 5.26: Exp:NIM2-3 Single Instance Performance breakdown with increasing number of workers

version to run inside as it currently the case. For single instance systems the web server is unlikely to create a bottleneck, however faster web servers will ensure that the optimal number of workers is reached sooner as the CPU is kept busy due to consistently quick file downloads.

5.4.2.4 Analysis

These experiments provide basic information regarding the performance of a single instance within the pipeline architecture.

- Exp:NIM2-1. For a single worker instance, running a single worker there is clearly a difference observable in the download time from the AWS based web servers used.
- Exp:NIM2-2. For different worker server types, different worker processing rates are observed. As the instance type CPU processing capability is increased, the rate CPU quickly becomes a bottleneck. By increasing the CPUs on the server the number of workers can continue to grow. If the worker nodes are increased the is a point of diminishing returns.
- Exp:NIM2-3. Each worker type will contain different characteristics such as CPU performance. If the number of workers is increased then providing there are sufficient CPU resources, the processing rate will improve. The overall pipeline will therefore run faster as more powerful servers are utilised until the ability to download becomes a bottleneck.

5.4.3 Multi-Instances Node Performance

The experiments outlined in Table 5.17 are focused on understanding the effect of increasing the number of server instances in an experiment. Experiments range from 1 to 100 EC2 amazon instances. Building on the observations of running a single instance, the aim is to determine if the increase in instances results in an overall improved performance of the system and to determine if any specific limits have been reached.

The three variables being tested in these experiments are Web Servers, number of instances and worker impact. Experiments are shown where the number of workers per server is consistently set to 5 for T1.Micro server instances, and 10 for M1.Large server instances, batch downloads are set to ten and all work performed is the same. Experiments are broken down by the variable under test.

5.4.3.1 Exp:NIM3-1 Multi Instance Webserver Performance

When dealing with a single instance it was clear the the AWS web server delivered files to the worker nodes slightly slower than the FTP, DIT and HEANet based web servers. As the number of instances running is increased, the AWS versus Non-AWS web server performance are compared to determine if the performance difference is sustained over larger number of file requests per

Reference	Measure	Description
Exp:NIM3-1	Multi instance web-server performance	Testing the impact of different web server configurations to service multiple worker instances.
Exp:NIM3-2	Multi instance performance by type	Testing the impact of selecting different instance machine types for multiple worker instances.
Exp:NIM3-3	Multi Instance Scaling Analysis	Analysis of the rate of scaling from a single instance to 100 instances.
Exp:NIM3-4	Limit Testing	Investigating the limits of the experimental setup

Table 5.17: NIMBUS Multi Instance experiments

second. Figure 5.27 shows a histogram of the overall file processing rate for 100 T1.Micro servers running 5 workers against different web servers. It is clear that the DIT and single AWS East server do not appear to scale with the increases in requests while the 4 AWS West web server combination, the HEANet Servers and the FTP servers seem to be performing well. If the cleaning rates of a single instance running 5 workers as given in Figure 5.17 are multiplied by 100, the cleaning rates where the web server is not a bottleneck scale almost linearly, based on observation. It is interesting to note that the web servers in the DIT and HEANet are identical in hardware and software, possibly indicating that the network is a factor in the experiment.

Looking at the breakdown of the experiments based on download and processing time in Figure 5.28, the FTP, HEANet server configuration gives consistent download times for data, while the other servers are consistently slower. The mean, standard deviation and variance in Table 5.18 from the FTP and HEANet servers indicate consistent reliable network performance throughout the experiment. The processing performance in Table 5.19 again shows similar behaviour for the experiments with fast downloads, while the slower web server based experiments have lower variance, standard deviation and mean for the overall slower experiments as the CPU is less loaded due to slower download of data.

It can be concluded from these experiments that there is a performance benefit from using the non-AWS web servers when scaling worker instances. When using the FTP HEANet server the increase in processing rates has been almost linear. Clearly there are also network effects given different network configurations but identical hardware configurations for the DIT and HEANet web servers. Combinations of multiple web servers such as the AWS West 4 server setup will also allow the performance of the pipeline to continue to grow with the number of instances.

Given the limited processing performance of the T1.Micro instance, it is possible that bottlenecks in the web servers have not yet been reached. As the larger M1.Large instance type puts

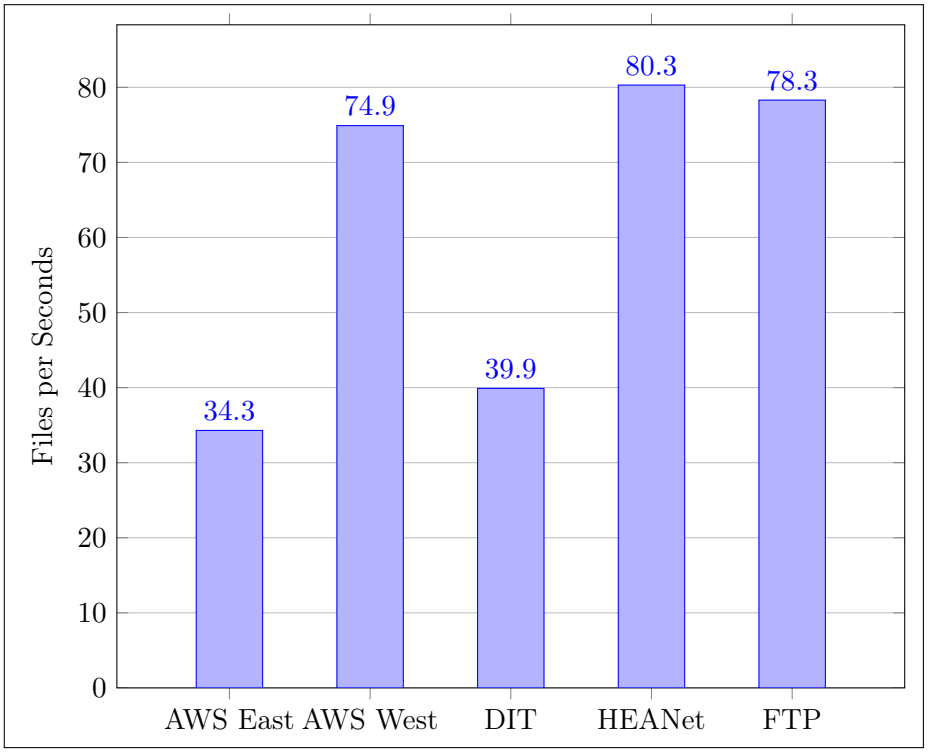


Figure 5.27: Exp:NIM3-1 Files Per Second: Varying Web servers and their impact on T1.Micro 100 Instance performance

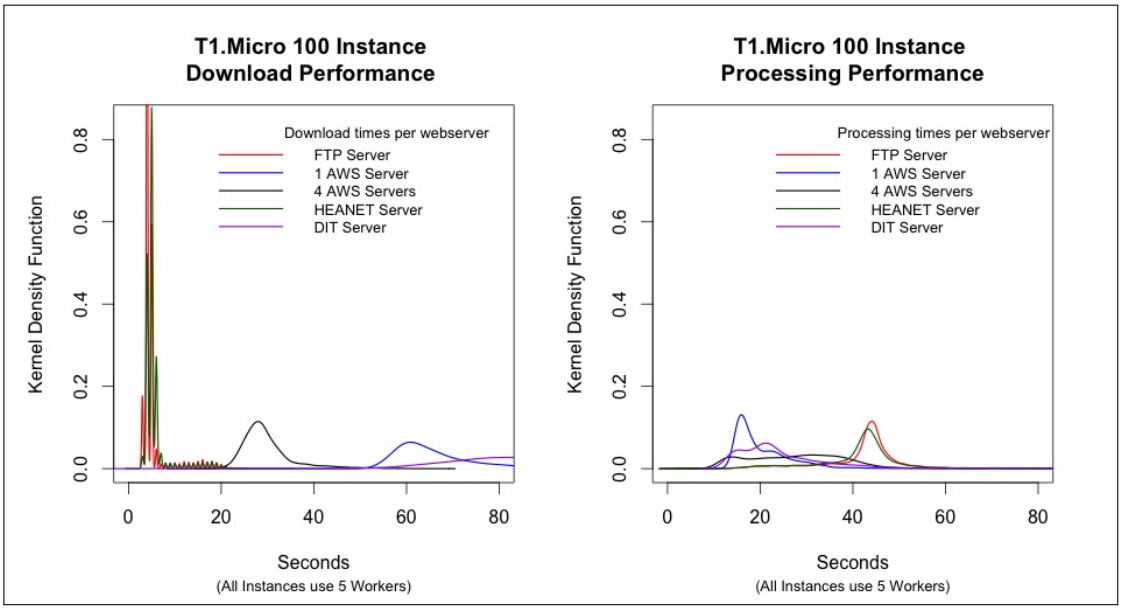


Figure 5.28: Exp:NIM3-1 100 Instance Performance breakdown for different Web Servers, with 5 workers.

Web Server	Std. Dev.	Mean	Variance
FTP	3.148	5.12	9.91
1 x AWS	10.813	66.62	116.90
4 x AWS	5.509	29.72	30.35
HEANeet	3.065	5,72	9.39
DIT	15.585	84.35	242.91

Table 5.18: T1.Micro 100 Instance 5 Worker statistics for image downloads per web server

Web Server	Std. Dev.	Mean	Variance
FTP	28.67	52.85	822
1 x AWS	13.63	22.35	186
4 x AWS	20.15	32.27	405
HEANeet	28.06	51.54	787
DIT	13.24	25.45	175

Table 5.19: T1.Micro 100 Instance 5 Worker statistics for image processing per web server

higher demands on the web servers due to the higher processing rates, additional experiments are considered in Exp:NIM3-2 comparing the performance of the two instance types.

5.4.3.2 Exp:NIM3-2 Multi Instance Performance by Type

These experiments look at the ability of the system to continue increasing performance as the number of instances is increased. The different instance types are explored to see if the increase in performance is linear for all instance types using the FTP web server to service all requests.

In Figure 5.29 the T1.Micro and the M1.Large instance types are compared while running 10 workers per instance. If the mean time for workers to download and process data to those of a single instance is compare in Table 5.20 and Table 5.21 it is clear that the T1.Micro network mean is similar, but there is more fluctuation in the download times, while the T1.Large mean is increasing. This might suggest that the increase in demand on the FTP server is starting to indicate some pressure on its ability to service file requests. The CPU processing mean remains similar for the T1.Micro and the M1.Large, but there is considerably more variance introduced into both instance types. This could be due to the larger sample size. In Figure 5.30 the CPU of one of the running instances of the M1.Large instance type is shown, indicating that the CPU of the M1.Large is close to maximum for the length of the experiment while Figure 5.31 shows the additional load generated by the experiment on the ftp.heanet.ie web server.

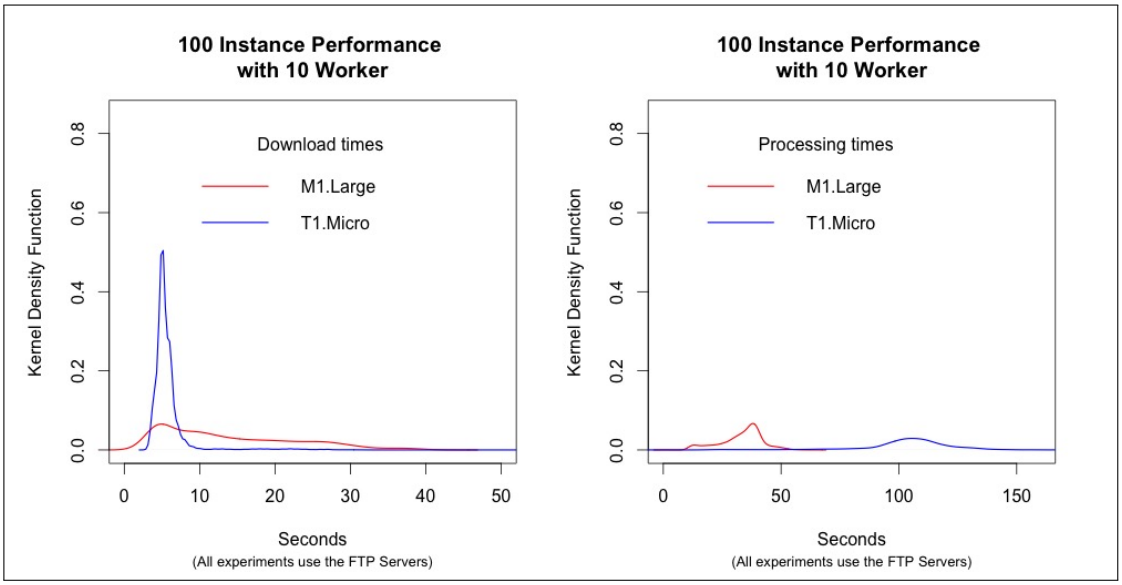


Figure 5.29: Exp:NIM3-2 100 Instance Performance breakdown for different instance types running 10 workers using the FTP web server.

The overall processing rate of the T1.Micro is just over 72 files per second, while the M1.Large is almost 192 files per second. From Table 5.16 the single instance processing times for 10 workers for the T1.Micro was 0.71 files per second, and the M1.Large was 2.07 files per second indicating

Instance Type	Num. Instances	Std. Dev.	Mean	Variance
T1.Micro	1	0.8	5.1	0.7
T1.Micro	100	3.2	5.8	10.4
M1.Large	1	0.6	3.5	0.3
M1.Large	100	8.9	13.8	79.4

Table 5.20: Comparing 1 vs 100 Instance 10 Worker network statistics using the FTP web server

Instance Type	Num. Instances	Std. Dev.	Mean	Variance
T1.Micro	1	21.9	97.0	479.3
T1.Micro	100	57.7	120.8	3338
M1.Large	1	6.6	39.3	44.1
M1.Large	100	9.0	33.2	81.8

Table 5.21: Comparing 1 vs 100 Instance 10 Worker CPU statistics using the FTP web server

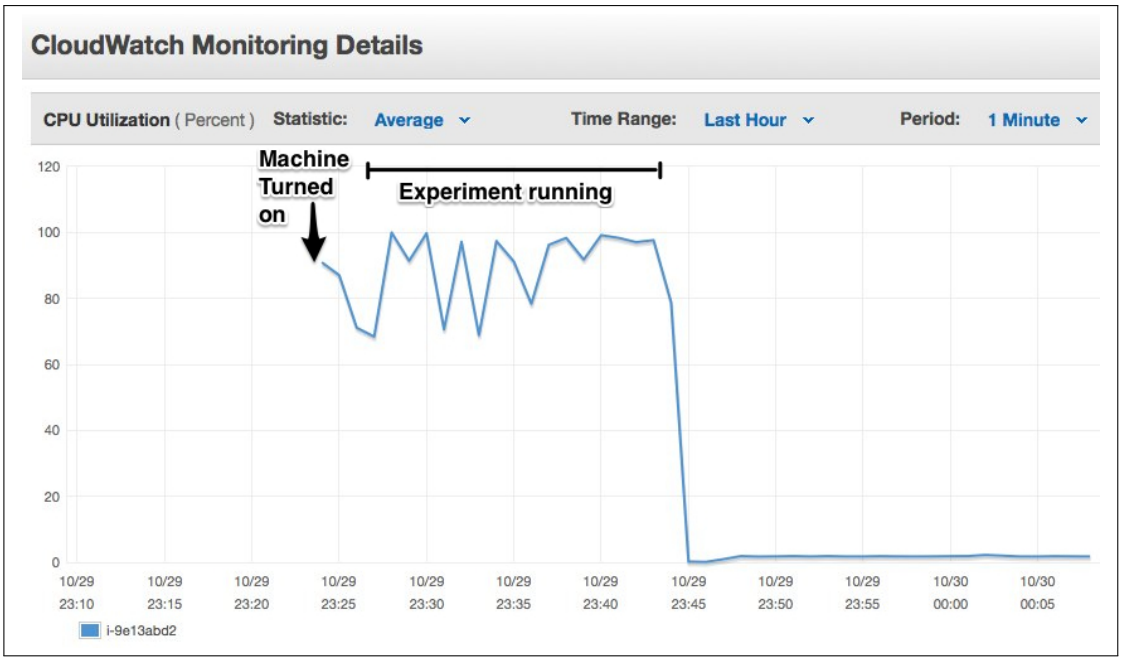


Figure 5.30: Exp:NIM3-2 CPU Performance for M1.Large canary instance during 100 instance run, with 10 workers.

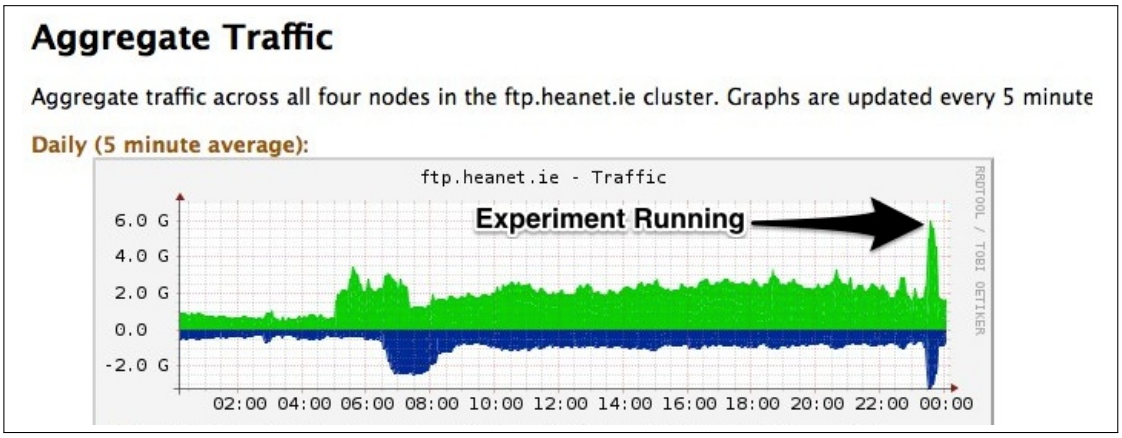


Figure 5.31: Exp:NIM3-2 Network Performance for FTP.heanet.ie web server during M1.Large 100 instance run, with 10 workers per instance.

that the increases in instances is indeed scaling linearly however fluctuations in the performance of the workers is starting to occur.

5.4.3.3 Exp:NIM3-3 Multi Instance Scaling Analysis

From the previous section it can be seen that the increase from 1 to 100 instances appears to provide an almost linear increase in performance for both the T1.Micro and the M1.Large instance types, although the larger instance experiments are seeing more fluctuations in the worker performance. To test the statistical significance of the increase in overall system performance a set of statistical

tests have been run on the M1.Large instance type. This instance type was selected as it does not have the reported T1.Micro CPU throttling issues reported by AWS, which eliminates yet another variable from the results. As already shown, the web server performance has an effect on the outcome of the experiments for larger numbers of instances. To eliminate this bottleneck from experiments when seeking a correlation between the experiment performance and the number of instances, the selection of experiments for larger instances was restricted to the faster web servers configurations, (FTP, 4xAWS servers, HEANEet web servers). Experiments with either 5 or 10 workers were included. So only for the larger instances have the single AWS and DIT web servers been removed.

Before running a correlation or a T-test a test for normality of the data must first be performed. Taking two experiments, both using the FTP server and 10 workers per instance, where the first has a single instance running and the second has 100 instances running, a density plot and the corresponding Normal Q-Q plot is show in Figure 5.32. From this it can be assumed that the data is reasonably normally distributed and it is appropriate to run a correlation test and T-test.

The Pearson product-moment correlation coefficient is used to measure the dependance between instance numbers and files processed and the scatter plot along with the pearson coefficient is given in Figure 5.33. With a value of $\rho = 0.95$ it can be determined that there is a *strong and positive* correlation between the number of instances and the number of files process, for the M1.Large instance type experiments. It is important to note that ρ provides a measure of the linear relationship between these variable, however it does not in itself indicate a causal relationship.

Assuming the the data is normally distributed, a one-way ANOVA test could be run to perform an analysis of the variance, however with a skewness of 0.859 (skewed to the right) and a kurtosis of 2.149, implying the data is platykurtic, it is prudent to transform the total number of files process using a Log10 function. Testing can now be done testing the null hypothesis, that the means are all equal. The P-value is calculated to help determine if the null hypothesis should be rejected. The result of the one way ANOVA in Figure 5.34 is considered significant with a P value < 0.001, so a pairwise comparison is performed to test if the differences are statistically significant, while adjusting for Type 1 errors. The results of the pairwise test are shown in Figure 5.35 and with p-values < 0.001 in most cases, it can be concluded that there is a statistically significant difference comparing instance numbers to files processed.

5.4.3.4 Exp:NIM3-4 Limit Testing

The majority of experiments run and presented so far have been based on the T1.Micro instance and M1.Large instance servers running within the amazon AMS environment. Due to physical hardware limitations, some additional experiments were completed that have shown the ability of the system to process higher rates of data when using more powerful servers, but the number of physical servers was limited to 1.

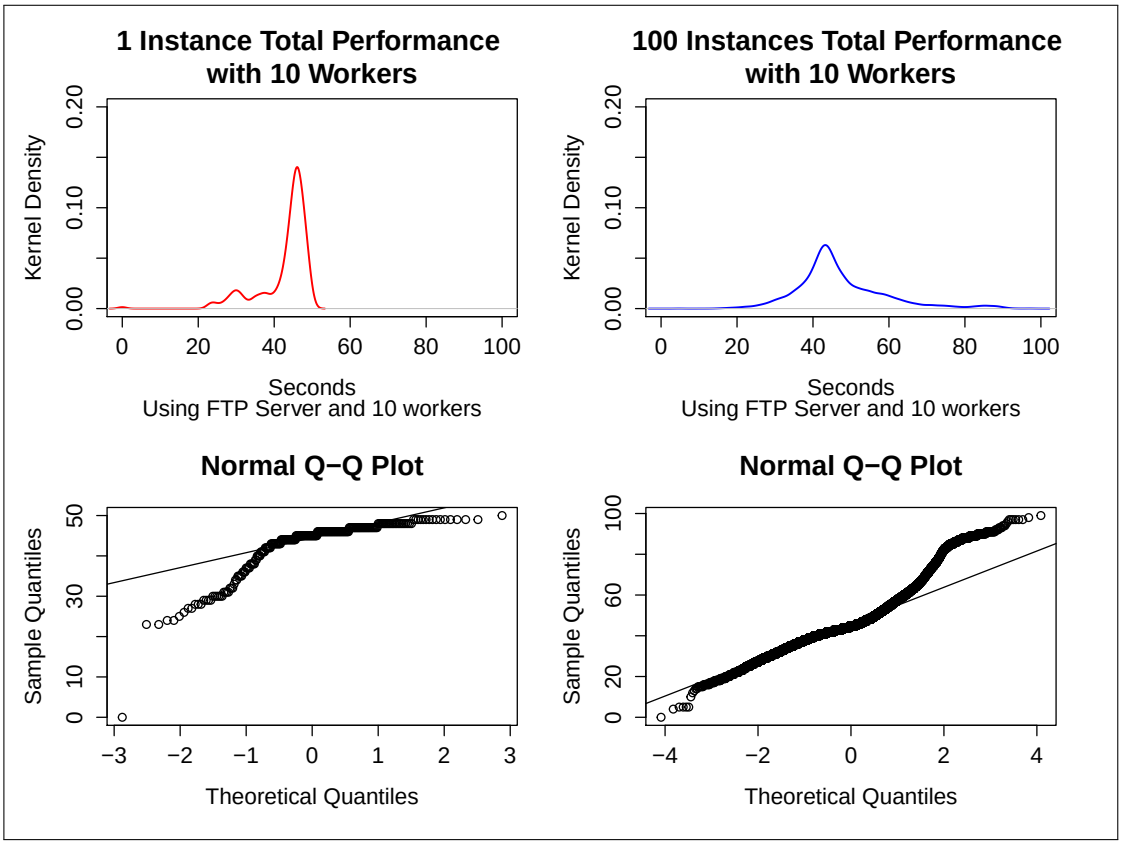


Figure 5.32: Exp:NIM3-3 Testing for normal distribution of worker performance for 1 and 100 M1.Large instance experiments

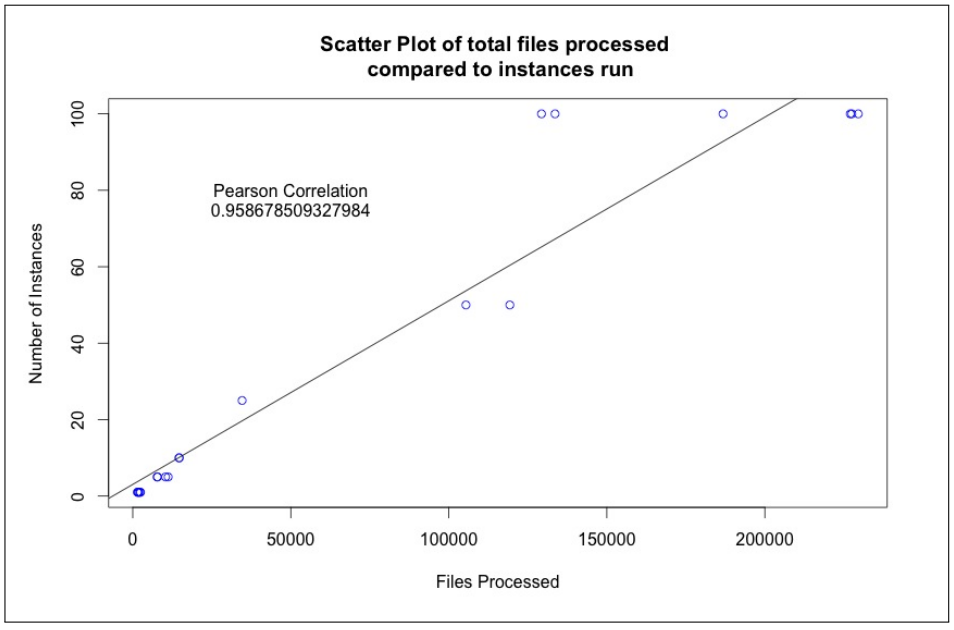


Figure 5.33: Exp:NIM3-3 Testing for a correlation between instances and files process. Data for the scatter plot given in Appendix Table D.5

```
Analysis of Variance Table

Response: data1$log
                        Df  Sum Sq Mean Sq F value    Pr(>F)
factor(serverinstances)  5 13.5643 2.71286  309.96 1.398e-14 ***
Residuals               15  0.1313 0.00875
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 5.34: Exp:NIM3-3 One Way ANOVA Table for increasing instances

```
Analysis of Variance Table

Response: data1$log
                        Df  Sum Sq Mean Sq F value    Pr(>F)
factor(serverinstances)  5 13.5643 2.71286  309.96 1.398e-14 ***
Residuals               15  0.1313 0.00875
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 5.35: Exp:NIM3-3 Pairwise T Test

It has also been shown that the impact of the number of workers and the web servers serving data provides a dynamic which can often have smaller number of workers performance faster, or comparable to those with higher numbers depending on how fast data is available to the server and the CPU load generated on the instance. The T1.Micro and the L1.Large performances for 100 instances can be summarised in Figures 5.36 and 5.37, with the data for the figures given in Table 5.22. In line with previous observations, experiment 6, which used the DIT web servers has reduced performance across instance types, and while the T1.Micro instances have a CPU bottleneck around 80 files per second, the M1.Large instances continue to increase the processing rate as more workers are run per instances, and the web server capacity is increased.

To determine if the observations about the increased performance of the M1.Large can continue as larger instances are available (as shown with the x4150 server instance previously), two additional AWS instance types were selected to continue a limited set of additional experiments. This limitation was primary due to the cost of these systems. If the best performance obtained using the T1.Micro instance and the M1.Large instance it taken, then as an instance type increases its capacity to run more workers it can affect the overall performance of the instance. Where the number of workers increases beyond the capacity of the CPU, a processing bottleneck is created and the instance slows down as evidenced by the T1.Micro having poorer performance when running 10 workers per instance compared to 10 instances. It is also clear that the FTP.heanet.ie server acts as a high-performance web server for these experiments without providing a significant bottleneck on file availability for workers. By continuing to use the ftp.heanet.ie server and by running additional experiments, all of which execute 100 instances, it can be shown that larger capacity instances can yield increased performance. Regardless of how large an instance is however there

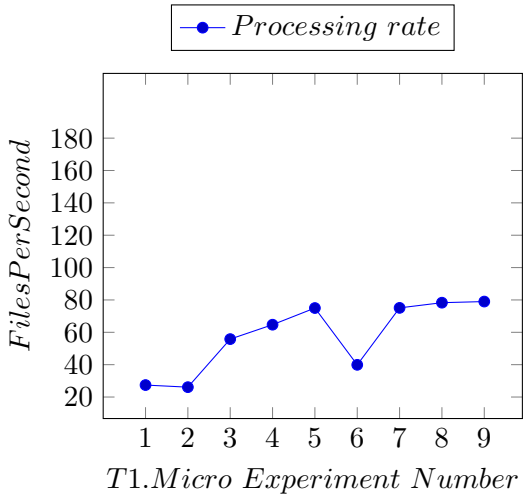


Figure 5.36: Exp:NIM3-4 100 T1.Micro Instances, file cleaning rate per second.

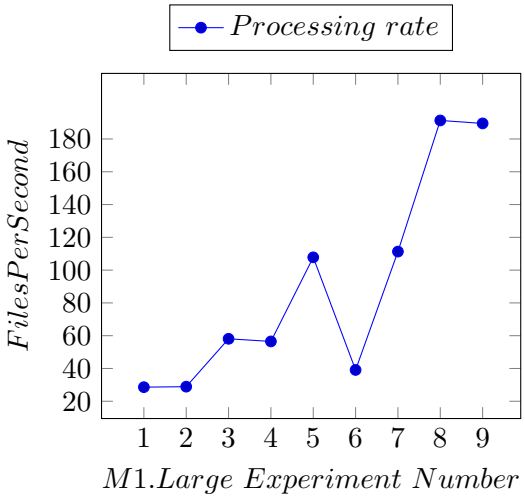


Figure 5.37: Exp:NIM3-4 100 M1.Large Instance file cleaning rate per second.

Exp. Num.	Web Server	Num. Workers	Num. Web Servers	t1.micro fps	m1.large fps
1	AWS East	1	1	27.4	28.6
2	AWS East	1	2	26.1	28.9
3	FTP	1	1	55.8	58.1
4	AWS East	5/10	2	64.7	56.5
5	AWS East and West	5/10	4	75.0	107.8
6	DIT	5/10	2	39.9	39.1
7	HEANET	5/10	2	75.1	111.3
8	FTP	5/10	1	78.3	191.3
9	ALL	10/10	10	79.0	189.5

Table 5.22: 100 Instance Experimental Results Table for Figure 5.36 and 5.37

should be a point at which a CPU bottleneck is observed. Figure 5.38 provides a histogram showing the best processing rates obtained for all 100 instance experiments including the M1.XLarge and the C1.XLarge instance types showing the number of workers per instance. All experiments used the ftp.heanet.ie web server. As can be seen in Experiment 5 in Table 5.23, there is reducing performance for the M1.Xlarge instance type as the number of workers is increased from 20 to 50.

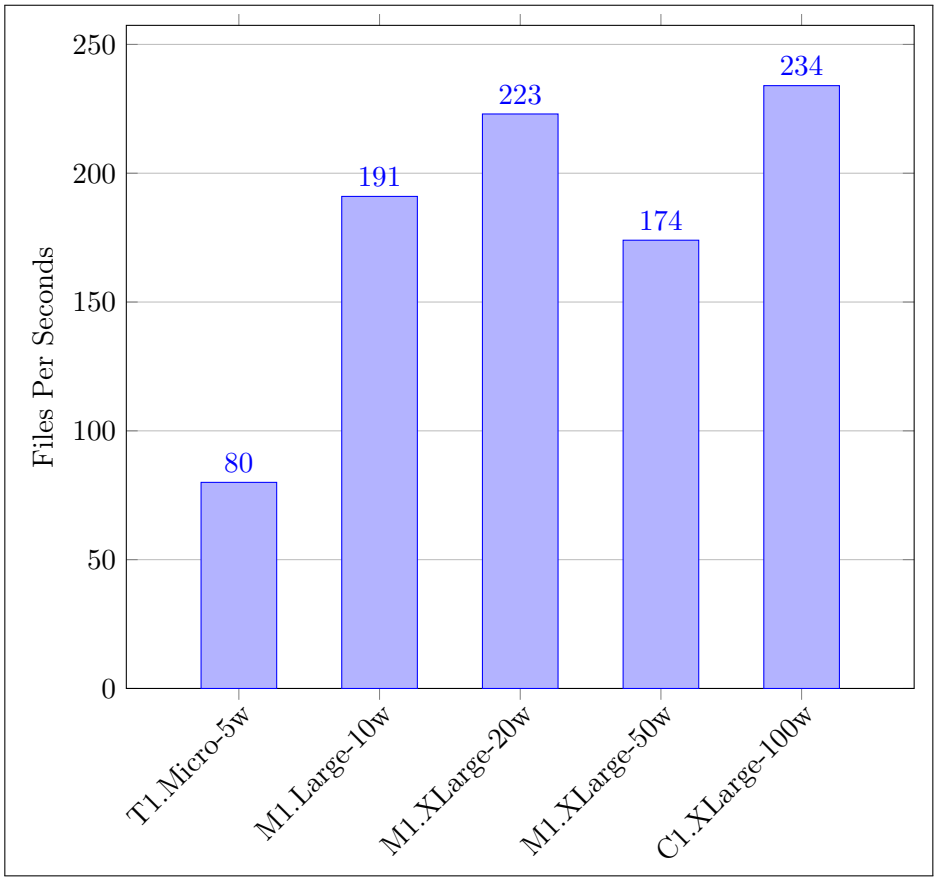


Figure 5.38: Exp:NIM3-4 Best file processing rates per second, for different instance types using 100 Instance experiments. See Table 5.23

To better understand the behavior of the running experiment, the processing performance of each experiment shown in Table 5.23 was plotted over time, showing the cumulative total of result files written to the S3 Storage. Each experiment was 1,200 seconds in duration, and each ran 100 instances using the FTP server. Figure 5.39 shows the cumulative number of files posted over time. From the graph, the throttle of the T1.Micro CPU is evident over time as the rate of processing actually slows down after a period of time. The same effect is not seen by the other instance types.

As the number of workers increases there is a significant delay in starting the C1.XLarge instances as there is a sequential process for copying and initialising each of the workers. This minor sequential process causes no significant delays then the number of workers is small, however it is increasingly significant as the number of workers increases. This would require additional

Exp. Num.	Web Server	Instance	Num. Workers	Files per Sec	GB per Hr
1	FTP	T1.micro	5	78.3	343.1
2	FTP	M1.Large	10	191.3	838.7
3	FTP	M1.XLarge	20	193.3	847.4
4	FTP	M1.XLarge	20	223.1	978.0
5	FTP	M1.XLarge	50	173.6	761.0
6	FTP	C1.XLarge	100	212.2	930.5
7	FTP	C1.XLarge	100	233.8	1024.1

Table 5.23: 100 Instance Experimental Results for different instance types using the FTP web server. Table for Fig 5.38

modification of the worker software to eliminate this behaviour. The C1.XLarge processing rate however, once established is slightly better that the other workers. The minor tail at the end of the experiments is due to either the message queue becoming empty or the experiment concluding.

To ensure that the processing rates seen in Figure 5.39 are sustainable, the best performing experiment, the C1.XLarge with 100 workers, was selected to run for 3,000 seconds. Figure 5.40 shows a consistent processing rate once all of the workers have started, with no indication of an unsustainable process rate. The sequential initialisation process is evident as the instances start up.

To investigate the slow start to the processing rate, further analysis was performed. The time stamp used per instance which determine the time at which a result was posted to S3, are based on the internal time of the instance, which may be slightly out of sync with other instances. As it is possible that these times may be different across instances, two different plots are shown indicating the time a worker first posts a result using two different times. The first is the time as recorded by the instance, the second is an offset from when the instance initially registers itself. So in the case where the actual time recorded by the instance is used, each worker first data publish time is shown in black in Figure 5.41. An alternative plot is also shown in red, where the earliest result across all of the instances is used, and using an assumption that all instances started approximately the same time, each timestamp from each instance is calibrated back to align with the start time of the earliest result posted.

Given that the Figure 5.40 uses the worker time and not a calibrated time, a more elongated start time for workers would be expected in Figure 5.41, which is in fact observed.

Table 5.23 gives the details for Figure 5.38 which presents the most significant results of these

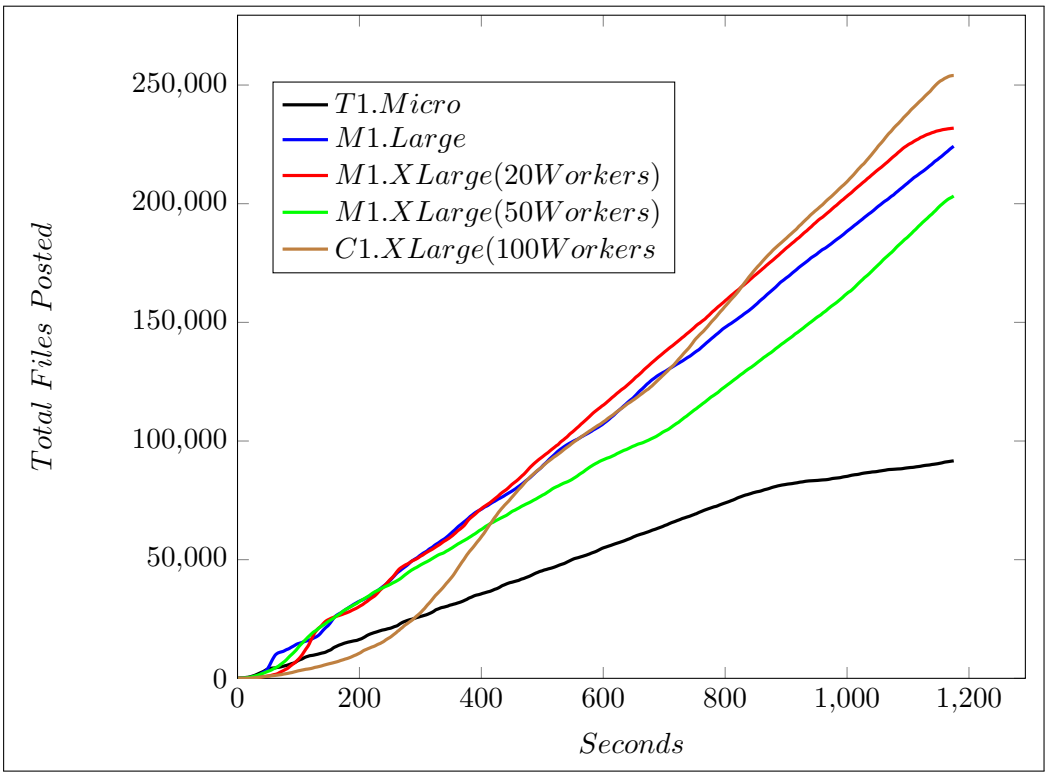


Figure 5.39: Exp:NIM3-4 Cumulative files processed over time.

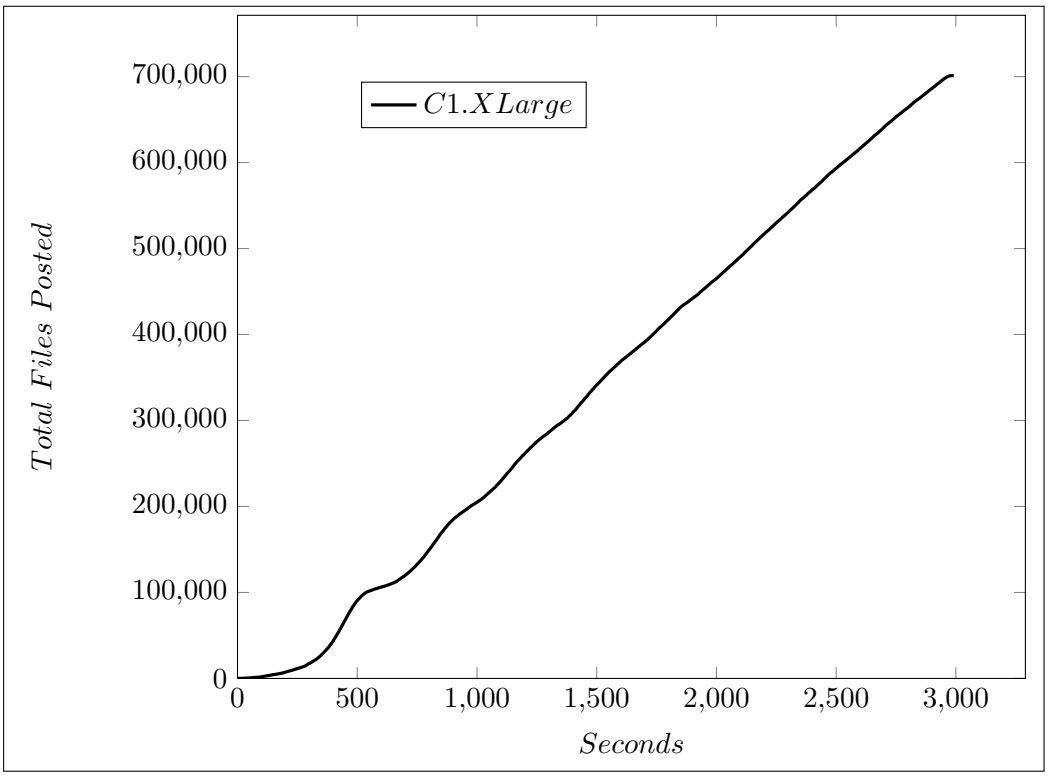


Figure 5.40: Exp:NIM3-4 Cumulative files processed over time for 100 C1.XLarge Instances, running 100 workers for 3,000 seconds.

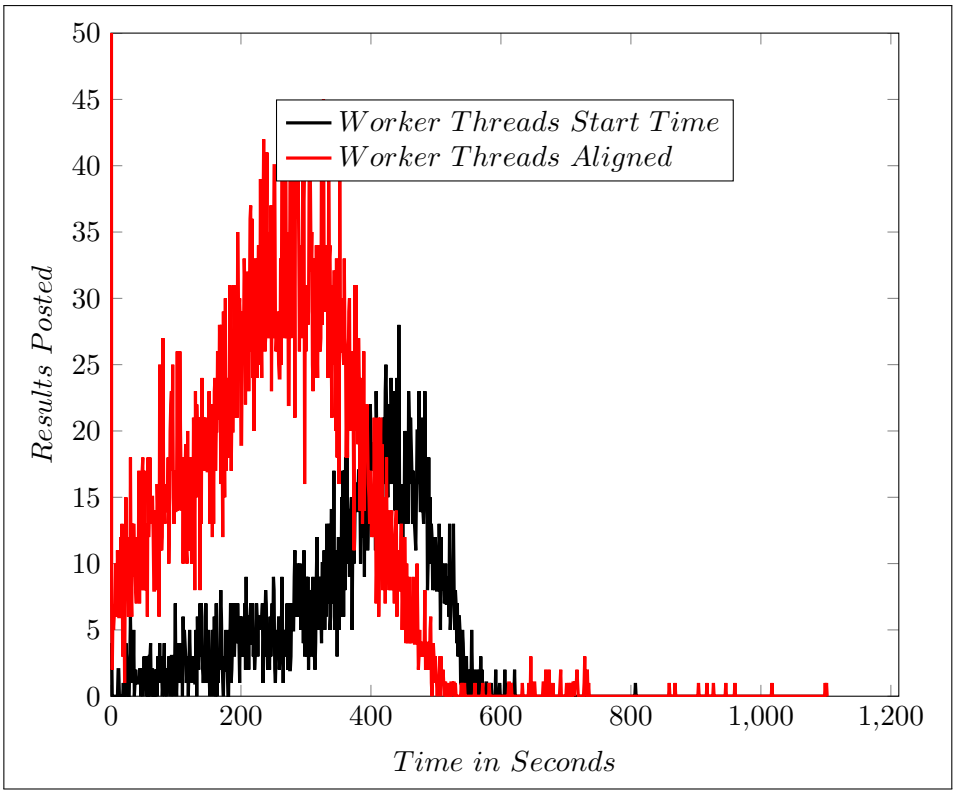


Figure 5.41: Exp:NIM3-4 Start times for each threaded worker across all Instances.

experiments. Further increases in either the instance types or the number of instances requires additional funding and investment. As the instance type is increased, so is the overall performance. At some point the web server providing the data will become the bottleneck for the system, however this is not an issue once the data being processed is distributed across different locations.

Data generation rate must be slower than the data publication rate to ensure that work can be advertised as quickly as it is generated. Work must also be processed faster then the generation rate. So simply put $D < W < P$ where D is the rate at which data is generated, W is the rate at which work is processed and P is the rate at which work can be published.

For all experiments which have 100 instances running, the FTP web server is required or there is a noticeable decrease in the performance of the overall system as the web server becomes the bottleneck. For the system to scale, the source of the data being processed must be distributed. Lower performance web servers which grouped together can match the high performance web server from HEANet for example.

5.4.3.5 Analysis

The pipeline allows for instance types to vary, for the number of instances to increase and for the number of web servers to be extended. Each of the these components contribute to the overall performance of the system, and these experiments review the core factors as each of these

components in increased. The key findings of these experiments are summarised below.

- Exp:NIM3-1. As the number of instances increases some of the web servers failed to scale and download time increased on average. Where multiple slower web servers were used, they maintained good performance due to the load balancing across them.
- Exp:NIM3-2. The differences in machine types performance continues as the number instances increases, but there is a higher variance observerd most likely due to the increase in the number of overall requests to the server. The M1.Large instance type is still maintaining faster processing rates consistently compared to the T1.Micro.
- Exp:NIM3-3. As the number of instances increases so does the overall processing of the pipeline. It is shown the this increase is statistically significant, and there is a high correlation between the number of worker instances and the performance of the pipeline.
- Exp:NIM3-4. Limit based experiments demonstrate the core factors in system scaling. The optimal number of worker threads on an instance is different for each instance type used, and the web server ability to respond to increases in the number of requests varies by configuration. If more web servers are used the load is spread out allowing the processing rates to continue to grow literally with number of instances. Using the larger machine types and the more powerful web servers high processing rates are observed, but the similarity in resuls may indicate that despite the high processing rate, a limit may be emerging within the system.

5.4.4 System Limits

Of interest in Figure 5.39 is the similarity of the processing rates across machine types. To determine if this is a CPU bottleneck or a web sever download issue, a more detailed look at the split between these two activities for the worker node within each experiment is shown in Figure 5.42. As expected the large amazon EC2 instances process data the fastest, however considering that they are each using the same web server for download, there are clearly difference in download times for each experiment. Given an overall lower processing rate for the T1.Micro experiment, the FTP web server performance consistently well with fast download times, while the M1.XLarge running 20 workers experiences considerably slower download rates. The M1.XLarge processing times are faster than the download times, leaving the primary bottleneck as the web server. This is reversed for the T1.Micro which has fast downloads but slower processing times. Of primary concern in this graph is the increase in the download times for the C1.XLarge instances. As the machine times get faster the mean download time gets longer, going from 5 seconds to 300 seconds. The reduction in the download time could be an indication of a bottleneck with the overall system. To investigate this issue, the following possible restrictions were considered and experiments de-

signed to identify and eliminate the bottleneck to ensure that there were no underlying scalability issues within the architecture. Table 5.24 contains details of these experiments.

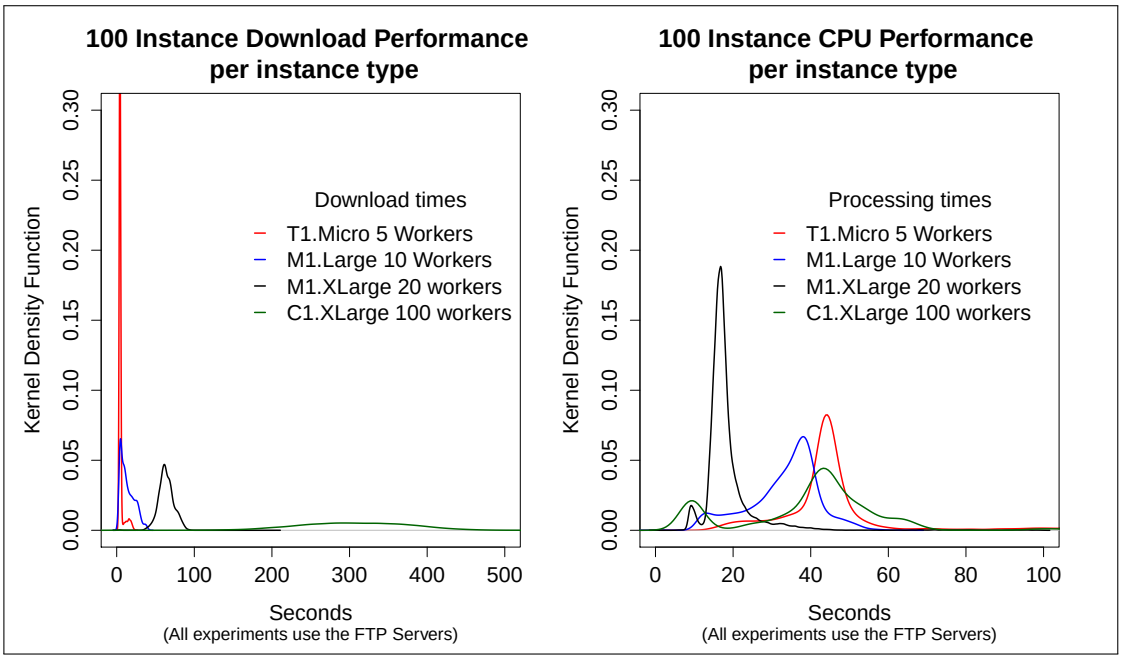


Figure 5.42: Breakdown of CPU and Network performance on fastest experiments by instance type.

5.4.4.1 Exp:NIM4-1 Instance limitations based on workers

For these experiments a single M3.2XLarge instance type was chosen. Comparable to the C1.XLarge instance type, this is an 8 VCPU instance with 30GB of RAM using SSD drives. This instance type is described as a balance between compute, memory and network performance with the CPU usually a high frequency Intel Xeon E5-2670 v2 processor.

In Figure 5.43 the breakdown of network download and performance for a single instance is shown. The web server used was the ftp.heanet.ie server and as can be seen quite clearly, the web server offers consistent performance, with the mean download time ranging from 3.39 to 3.76 seconds for a batch of 10 images with a standard deviation of less than 0.7 seconds (Table: 5.25). The processing rate in seconds, for cleaning the downloaded batch of 10 images, demonstrates a different behaviour showing a clear relationship between the number of workers and the processing time for a batch of images. As the processing times increase as more workers are added, the overall processing rate remains about the same for the experiment run. This relationship will be dependant on the system configuration used, and may require additional monitoring and adjustment to obtain an optimal rate over time. Assuming similar performance, less workers per system is preferred purely for the reduction in complexity in post experimental analysis to provide a smaller data set per instance for analysis. The file processing rate per second P_{fps} is calculated using Equation

Reference	Title	Experimental Objective
Exp:NIM4-1	Instance limitations based on workers	As the number of workers increases on a server instance, even a larger server, at some point there may be diminishing rates of returns for processing and/or downloading files.
Exp:NIM4-2	Virtual Machine sharing as a bottleneck	Depending on the configuration and deployment of the virtual machine instance it is possible that the service of the network could degrade as virtual machine instances shared physical networks. While is is not knowable exactly how virtual instances are deployed on physical machines, the basic transfer rates can be looked at.
Exp:NIM4-3	Bandwidth as a Bottleneck	This would indicate that the network either had bandwidth throttles or limited bandwidth available, and that the overall experiments were exceeding these values.
Exp:NIM4-4	Web Server as a bottleneck	Determine if the service form the web server had some hard limit for servicing file requests.
Exp:NIM4-5	System Scalability.	The scalability and flexibility of the system is tested taking into account any limits observed in previous experiments.

Table 5.24: NIM4: Testing the pipeline architecture for system bottlenecks

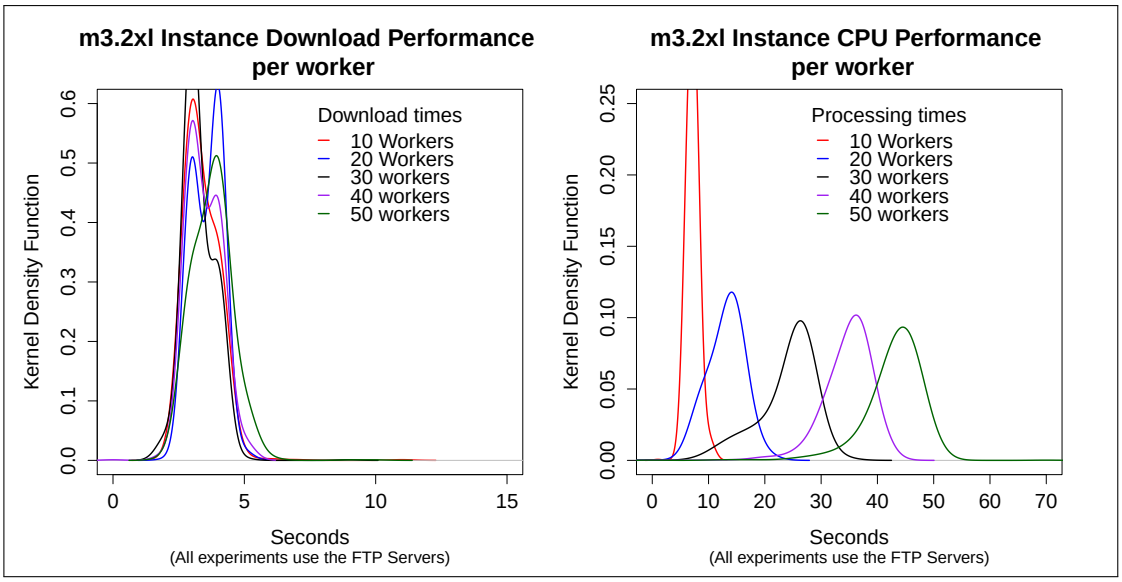


Figure 5.43: Exp:NIM4-1 Breakdown of CPU and Network performance by number of workers

5.13, where the experimental time E_{time} was limited to 1200 seconds, and F_{total} is the total number of files processed by the experiment.

$$P_{fps} = \frac{F_{total}}{E_{time}} \tag{5.13}$$

Num Workers	Download Mean	Download Std. Dev.	Download Variance	CPU Mean	CPU Std. Dev.	CPU Vari-ance	fps
10	3.39	0.59	0.36	7.1	1.0	1.1	6.4
20	3.57	0.52	0.27	13.1	2.9	8.7	9.0
30	3.26	0.54	0.29	24.0	4.8	23.4	9.0
40	3.45	0.58	0.34	34.7	4.3	18.2	9.0
50	3.76	0.69	0.47	43.0	5.4	28.9	9.3

Table 5.25: Comparing single instance M3.2XLarge performance statistics and file processing rate per second for different workers.

5.4.4.2 Exp:NIM4-2 Virtual Machine sharing as a bottleneck

The aim of this experiment is to scale the number of instances until a bottleneck is evident using instances from a single physical AWS location and to then determine if the limitation can be overcome by ensuring that additional instances are created which do not share physical hardware. To accomplish this multiple AWS server locations around the world are used, and in this way any resource sharing at the network or physical machine level are eliminated. While the advertised performance of the M3.2XLarge instance is set to "Fast" there is no specific performance service level agreement given. The expectation is that the network performance of the instance should be approximately 1GBits per second. From the specifications of the AWS services it is unclear if this is a maximum burst capacity, or a sustainable data transfer rate. The possibility exists that virtual machine deployment could dictate that sharing of network capacity exists to some extent. From the experiments on a single instance with 20 workers, as shown in Table 5.25, and using the files processed per second P_{fps} for a single instance, the data throughput for the network in Gigabits per second, B_{Gbps} is calculated as follows, where n is the number of instances being run, and D is the size of a data cubed image file in Megabytes. To convert from bytes to bits, the result is multiplied by 8.

$$B_{Gbps} = \frac{D}{1024} \sum_{i=1}^{n} P_{fps} * 8 \tag{5.14}$$

For the images used in these experiments the size is 2.4Mbytes, and unless otherwise stated, experiments run for 1200 seconds. While the file process rate for an instance may vary to some extent, it can be estimated that P_{fps} for 50 instances should be approximately 8.4 Gbps. This should be well within the limits of the individual instance specifications which is 1 Gbps per server and where a potential maximum network performance of 50Gbps (1 Gbps per instance). There is a possibility that the virtual instances may not be able to perform at sustained data transfer speeds or that the network connection between the instances and the HEANnet ftp server are not sufficient to sustain these transfer rates. To determine if a limit of sustainable Gbps transfer speed exists a set of experiments were run using varying numbers of M3.2XLarge instances each running 20 workers. Figure 5.44 clearly shows a roughly linear increase in files processed per second as instances are initially scaled, however as the number of instances rises, this linear increase in performance does not continue after about 25 instances.

If this is a bottleneck due to physical resource sharing by virtual instances, the bottleneck can be circumvented by running 25 of the instances from the initial location, and 25 additional instances from alternative locations. If the linear scaling of the architecture is to continue, file processing rates for 50 Instances should be twice that of 25 instances. Looking at Figure 5.44, the label 50 Instances* represents an experiment where half of the instances were run in the primary AWS location used for all experiments, while the remaining 25 instances were run in a different

physical location. The bottleneck observed after 25 instances is still evident in the result, and as such the sharing of virtual resources as the cause for this can be ruled out based on these results.

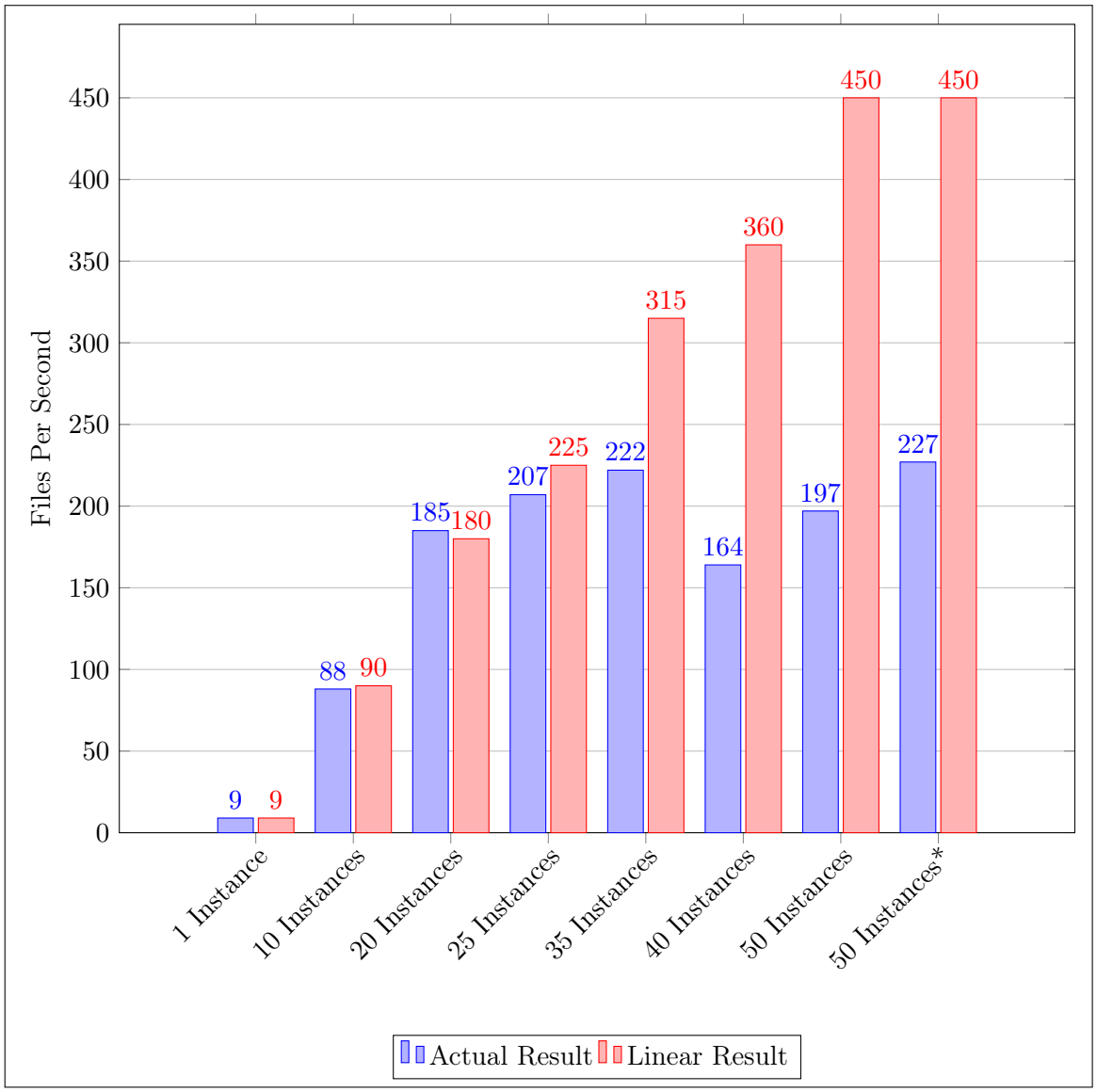


Figure 5.44: Exp:NIM4-2 File processing rates for different numbers of Instances using the M3.2XLarge instance type using 20 workers.

5.4.4.3 Exp:NIM4-3: Bandwith as a Bottleneck

The issue of a bottleneck existing as a result of the shared physical network between the AWS and the HEANet needs also to be considered. If the maximum connection speed between AWS and HEAnet was reached after processing at a rate of 230 Files per second then no additional increase in instances would affect the overall result. Using equation 5.14, the bandwidth required, in Gbps, for 300 file per second for files of size 2.4Mbytes, is less than 4.5 Gbps. Further investigation revealed

that the actual connection between these two networks at the edge for the AWS networking within Ireland, was 10Gbps. Data was limited to 1Gbps from AWS regions outside of Ireland which in total 11Gbps. While network sharing would be expected to occur, experiments using the HEAnet FTP server consistently showed a limit of approximately 4 to 4.5 Gbps regardless of the HEANet FTP server load. The network bandwidth limitation between the instances and the server would appear not be responsible for the bottleneck in performance.

5.4.4.4 Exp:NIM4-4: Web Server as a Bottleneck

Considering that experiments have shown that moving instances to different regions around the AWS network allowed the architecture to use additional bandwidth and to ensure that machine resource sharing was not an issue, the final set of tests considers the web server as the primary bottleneck. As the number of requests increases it is possible that the web server's ability to service the requests is limited. The ftp.heanet.ie service uses a load balancing service which is supported by four apache web servers. As requests are made to the ftp.heanet.ie url the load balancer will offload requests to one of four web servers each of which has a replicated copy of the data. The configuration of each of these web servers should allow for more concurrent connections than requested from the processing pipeline. To eliminate the load balancer as an issue, tests were run against one of the four web servers. With 50 instances running, the test used the same network and instances as pervious tests. The result was that the total number of files processed in 1200 seconds was 66 thousand, which is approximately a quarter of the total processed when all four of the web servers are utilised by the load balancer. From this observation it would appear that the bottleneck in the experiments provided is the ftp.heanet.ie web server. While its performance was superior to the other web servers used, its ability to process more that 250 file per second would appear to be limited. As the 100 instances used in earlier experiments with the C1.2XLarge virtual machine type was about the limit of the entire system, this bottleneck was not initially evident. Only by increasing the instance performance, and splitting the download versus processing time was this bottleneck evident.

5.4.4.5 Exp:NIM4-5 System Scalability

To ensure that the proposed architecture is scalable, modifications were made to the system to address the possible bottlenecks die tidied and the actual web server bottleneck already discussed. Three key changes were introduced.

- Diversity in the location of the virtual machines to eliminate resource sharing concerns. In this case 4 AWS regions were chosen to run virtual machines. Ireland, Virginia, Tokyo and San Paolo.
- Increase in the number of web servers. In the event of a single web server being a bottleneck

due to the high rate of request, data should be spread across multiple servers. In this case two additional servers were used which were also within the HEANet network.

- Increase the number of SQS queues. Although there was no reduction in the performance of the get message requests, additional worker queue were created to demonstrate the flexible nation of this approach. If worker instances have a choice of sqs queue to read, then they could in theory self load balance by moving queues if the response time for a message started to degrade.

Using these modifications, a final experiment was run which consisted of 25 instances in the Ireland AWS region and 15 instances in the three additional AWS regions mentioned above. Each instance was of type M3.2XLarge, and ran 20 workers threads each. One workerq contained work for the FTP.HEANet.ie web server which was used by the Ireland based instances, and a second wokerq pointing and two additional web servers which was used by the 3 non-Irish based locations, each of which ran 15 instances for a total of 45 instances, giving 70 instances in all for the experiment. Figure 5.45 shows consistent network download performance and CPU performance. Within the experiment time a total of almost 400,000 files were processed, which equates to 322.9 files per second. Using the formula 5.9 this is an equivalent processing rate of 194 TB per 24 hours.

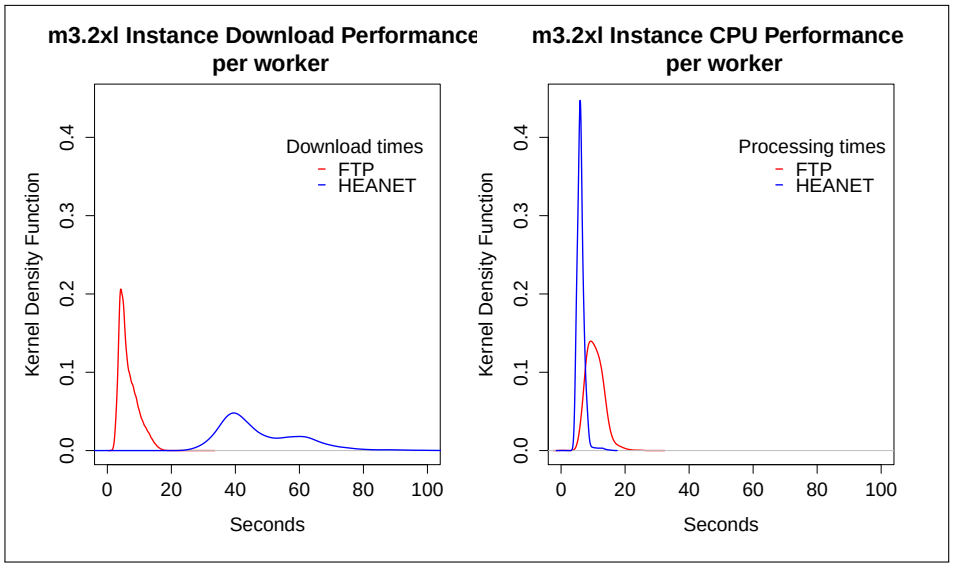


Figure 5.45: Exp:NIM4-5 Breakdown of CPU and Network performance for 20 worker per Instance experiment where Instances are located in multiple AWS regions and use multiple web servers.

Figures 5.46, 5.47, and 5.48 plot different aspects of 43 experiments. Figure 5.46 shows the increase in files per second for each experiment, with the final experiment giving the fastest processing rate of 322.9 files per second. If this is cross referenced to Figure 5.47, which shows the total number of workers operating within each experiment, it can be shown that the total num-

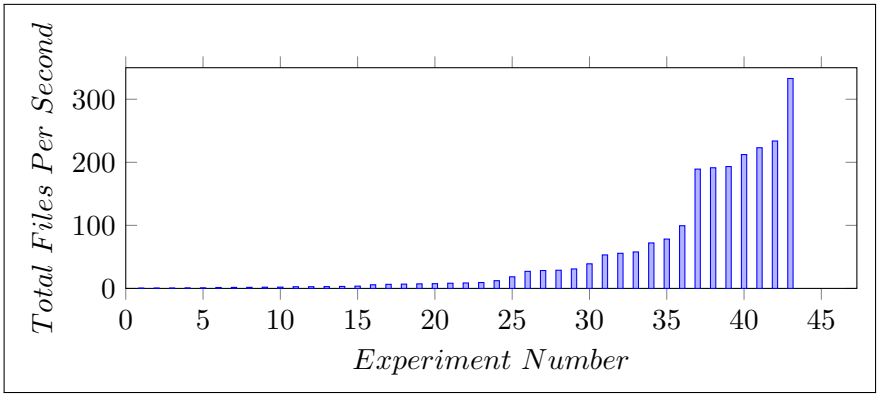


Figure 5.46: Exp:NIM4-5 Files Per Second. See Appendix Table D.4.

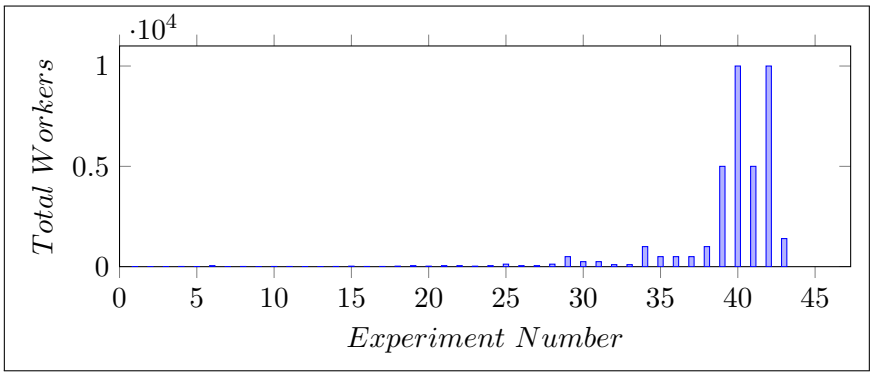


Figure 5.47: Exp:NIM4-5 Total Workers Per Experiment. See Appendix Table D.4.

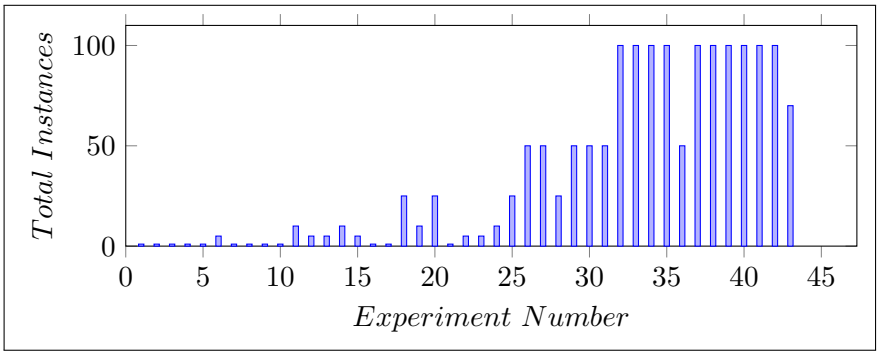


Figure 5.48: Exp:NIM4-5 Total Instances Per Experiment. See Appendix Table D.4.

ber of workers does not necessarily correspond to faster process rates. The fastest experiment for example ran 1400 worker instances, while previous experiments ran as many as 10,000.

Figure 5.48 shows the number of instances run per experiment and again the fastest experiment had only 70 instances. While the number of instances was important, the number of workers per instance was a function of the power of the server, with more powerful servers being able to run more workers. The major difference however between the fastest experiments was that with the introduction of additional web servers, the pipeline could continue to increase its processing rate, whereas previous experiments were hitting a web server bottleneck. So it is shown that by altering certain parameters of the pipeline, but without changing the underlying architecture, an experiment with 70 globally distributed worker instances, running a total of 1400 worker threads, outperformed all other experiments.

5.4.4.6 Analysis

The pipeline allows for an instance type to vary, for the number of instances to increase, the number of worker queues expanded and for the number of web servers to be extended. Each of the these components contribute to the overall performance of the system, and these experiments review the core factors as each of these components in increased. The key findings of these experiments are summarised below. Ultimately when all parameters were reviewed any observable limitations were identified and the architecture of the pipeline allowed these to be eliminated.

- Exp:NIM4-1. As the number of worker threads per instances increases these experiments show that the web server delivers images at a consistent rate, but the processing time as the workers increase goes up. If the web server is not a limit then worker threads will eventually become their own bottleneck if they continue to increase.
- Exp:NIM4-2. To eliminate the possibility that the virtual machine instances used with the AWS, which were based in the same geographical region had limits on their collective shared bandwidth, experiments were run with EC2 instances starting in multiple geographical regions. Using tests with 50 instance, there was no significant difference on the overall result based on the location of the virtual machines.
- Exp:NIM4-3. The bandwidth required for a processing rate of 300 files per second is less then 5GBs, the bandwidth connecting AWS to HEANet is 10Gbs. Experiments were run when the HEANet server was both heavily and lightly loaded, and in all cases the data transfer from the web servers was less 4Gbps. The HEANet bandwidth clearly could operate at a higher rate, but the pipeline could not get better performance rates that about 230fps.
- Exp:NIM4-4. The HEAnet FTP web server was tested to see if it was a bottleneck. The web server is load balanced, so testing was performed agains one of the servers and the processing

rate dropped to one quarter of the previous results. It can be concluded that this websever had finally hit a data serving limit, possibly due to disk access read limitations.

- Exp:NIM4-5. The final experiment was run taking all results into account. Using 70 virtual machines distributed around the world, and using multiple web servers and worker queues the processing rate achieved was close to 200TB per day. Further scalability may be possible, but the budgets used were limited.

5.5 Conclusion

The NIMBUS pipeline extends the ACN pipeline into a global environment. The use of the publicly accessibly SQS web queue service from AWS enabled computing resources from around the globe to join the processing cloud which was a key modification to the ACN architecture. The SQS is a distributed architecture which scales horizontally allowing for a maximum of 120,000 messages in flight at a time for a single queue. NIMBUS allows for the used of multiple queues both for advertising work, controlling the behaviour of the worker instances, and monitoring running experiments.

Most of the worker instances used for experiments were EC2 virtual machines within the AWS cloud. This was a convenience for experimentation rather than a primary required feature of the architecture. Physical machines were also integrated into the pipeline for various experimental purposes, specifically when reviewing maximum performance for individual servers. Experiments ranged from single instances to a maximum of 100 instances. Each worker instance had multiple threads activated to determine optimal configuration of the instance type.

The experimental variables modified while testing the architectures ability to scale to terabytes per hour were, virtual machine location, virtual machine size, number of worker threads per instance, web server type and number of web servers, and number of queue for advertising work.

The data processed by the system is assumed to be part of an existing data archive and is already compressed. The requirement for a datastore to participate in the pipeline is to create an SQS message for each file to be processed.

The final experiment demonstrated that horizontal scaling of all primary components is possible, and that this approach will ensure system bottlenecks are overcome.

The following is a summary of the experiments used to evaluate the performance of the pipeline.

- The SQS queue performance was evaluated to ensure that its ability to have messages written was sufficiently fast to advertise work for processing, and for messages to be read by workers within the processing cloud as it scales to thousands of worker threads. Work advertising needs to be significantly faster than work processing. Multi-threaded Python programs were required to ensure that the speed was sufficiently fast.

- Before scaling to multiple worker instances, the performance limit of a single instance was explored. By varying the type of instance, the number of worker threads run and the web server used, a baseline was obtained against which the scaling architecture could be compared. Multiple threads increased performance, but bottlenecks were identified after too many threads were run. With a sufficiently fast web server to service requests, higher performance from the worker instance was based on the number and speed of the CPUs used.

- Scaling from a base of good performing single instances, the experiments reviewed the ability of the web servers to maintain data rates. Multiple web servers provided the best solution to ensure that data requirements from the scaling instances could be serviced. It was shown that there was a strong correlation between instance numbers and the processing speed of the pipeline.

- Despite the good performance of the scaling experiments, further investigation revealed possible limitations in the processing rate. A series of tests identified the possible sources of these limits and a final experiment was run which increased the horizontal scaling of each of the primary architectural components. This resulted in an overall processing rate improvement using less instances. While funding limited the ability to run additional experiments, the result of this final experiment were such that 192TB of data per 24hours processing could be achieved, with evidence that further improvement would be possible through the use of more instance types, and more web servers providing data.

Chapter 6

Conclusion

The research hypothesis proposed within this thesis was to determine if a globally distributed astronomical CCD image reduction pipeline can process data at a rate which exceeds existing data processing requirements and is scalable such that it can meet future data processing challenges. The hypothesis has been proven to be true in Chapter 4 (the ACN pipeline), and Chapter 5 (the NIMBUS pipeline). The ACN demonstrates that data can be compressed and uploaded to a central storage service at a rate higher than it can be generated. The NIMBUS pipeline demonstrates that data can be distributed and processed using a horizontally scalable elastic architecture at proven rates of 200 terabytes per day.

This thesis focused on the processing of CCD image reduction and photometry in a distributed global network to determine if an architecture could be devised which had the ability to process terabytes of data per day to meet the growing demands of data production for projects like the LSST which are expected to reach tens of terabyte per day within the next few years. It has been shown that a distributed horizontally scalable cloud based architecture can process hundreds of terabytes of data per day, and that the architecture is flexible enough to continue to scale to petabytes per day by allowing individual processing nodes to join and leave the pipeline without impacting the integrity of the system. The form of CCD data used is such that it is suitable for parallel processing, and the pipeline developed can be applied to alternative science areas with similar characteristics.

Whilst existing solutions to CCD data processing for large scale projects, such as space crafts or large ground-based observatories, are processed either serially or using large computing data centres, much of the archive data made available to researchers still requires reprocessing. A solution, such as NIMBUS, which is flexible and scalable, is needed to assist with this requirement as the volume of data increases. With the increased availability and sensitivity of CCD or CMOS devices, further innovative research opportunities exist to support high-speed photometry, and robotic telescope farms. Each of these areas will require data processing rates to keep pace with data generation rates.

The NIMBUS processing pipeline architecture is presented as the final stage in an iterative process, where a series of architectures were proposed and experiments designed to evaluate limits of each of the bottlenecks identified through the research. The initial stage focused on a pilot system called FEBRUUS, which performed basic calibration of CCD images to establish a benchmark of performance for a sequential processing pipeline. Work on this pipeline clearly demonstrated that parallel processing was a suitable solution that could clearly be used. The second stage was the consideration of IRAF instances implemented within the cloud. This architecture considered virtualisation instances of existing reduction software, communicating via a centralised queue. From the analysis of this potential solution, the queuing design was identified as an essential communication feature. A light-weight utility based on the FEBRUUS pilot, which also performed photometry, was implemented for the ACN pipeline that focused on compression and uploading of data to a distributed storage node, for a distributed processing private cloud to access. The private process cloud was constructed across three Irish institutes of technology. Using a private NFS queuing system limited the system to processing nodes which were logically within the same network and hence failed to incorporate globally distributed nodes. The final pipeline, NIMBUS, incorporated web based queues for advertising work and for controlling the processing cloud. With work being advertised using a globally available message queue, the number of instances, and the number of worker threads within the instances, were expanded and experiments were devised to test the scaling nature of the system. Final experiments demonstrated the ability to process hundreds of terabytes per day with limits of budget finally restricting additional experimentation. No evidence was found that this final experiment was in any way the maximum processing rate possible by the pipeline. Evidence suggests that further expansion of the system is likely through an increase in available processing units.

6.1 Summary of Contributions and Achievements

The NIMBUS and ACN [110] architectures forms the core contribution of this thesis. The NIMBUS architecture is focused on data processing of archive systems, while the ACN pipeline focused on the compression and uploading of data to an archive. The NIMBUS system can support both modes of operation, requiring only that all data to be processed is advertised via a central web queue. While Amazon Web Services (AWS) were used in many of the experiments, this was for convenience only and the NIMBUS architecture does not have to be AWS based. The following are the core features of the pipeline, which form the contributions of this thesis.

A globally distributed data processing architecture

The global architecture is a combination of the ACN pipeline which was further extended by the NIMBUS pipeline. An overview of NIMBUS is shown in Figure 6.1. Data was produced either within a dispersed cloud of CCD generating detectors which compress data

and service the data via http web requests, or is available from a data archive which contains compressed images. Data availability was advertised via a distributed worker web queue, which is readable by all workers within the data processing cloud. Worker Instances read from the web worker queue, download data from the data provider, process it and upload to a defined storage cloud containing a distributed storage system. A central controlling server acts as a broker, publishing the location of the workers and the queues. A series of monitoring systems are provided to help assess the overall system performance. Workers and web servers can easily be added to the overall system allowing it to scale up as required.

A self configuring, balancing system that is scalable and resilient to failure

The use of web based message queues, as shown in the NIMBUS pipeline, decouples the processing cloud from the individual worker performance. The extent of decoupling is that different worker instance types can process work at different rates without holding up the overall system. If for any reason an instance fails, the message is not lost but restored back to the queue for another worker to process. The distributed set of web queues allows for management of thousands of workers through a series of instructions which provide instructions on where work can be found.

Reconfigurable for multiple science payloads

The NIMBUS pipeline works as a distributed data processing system that can be used whenever workers can process files in parallel with no specific ordering required. Once the files are advertised as available from a web server, the processing workers initialise themselves by looking for a science payload to download, install and run. The pipeline provides the necessary supporting architecture and calls the downloaded control scripts to perform the work. This pipeline could be used for any process, which simply requires files to be processed in parallel, where a worker performs a specific task on the file and uploads results when completed.

Enabling real-time data processing

The processing speed of the NIMBUS pipeline was tested to just almost 200TB of data per day, which equates to 3,220 images processed per second. Given multiple data sources such as a robotics telescope farm, this processing speed would enable a process where real feedback could be provided to a targeting system to modify the location they are searching, thereby facilitating a real-time feedback loop.

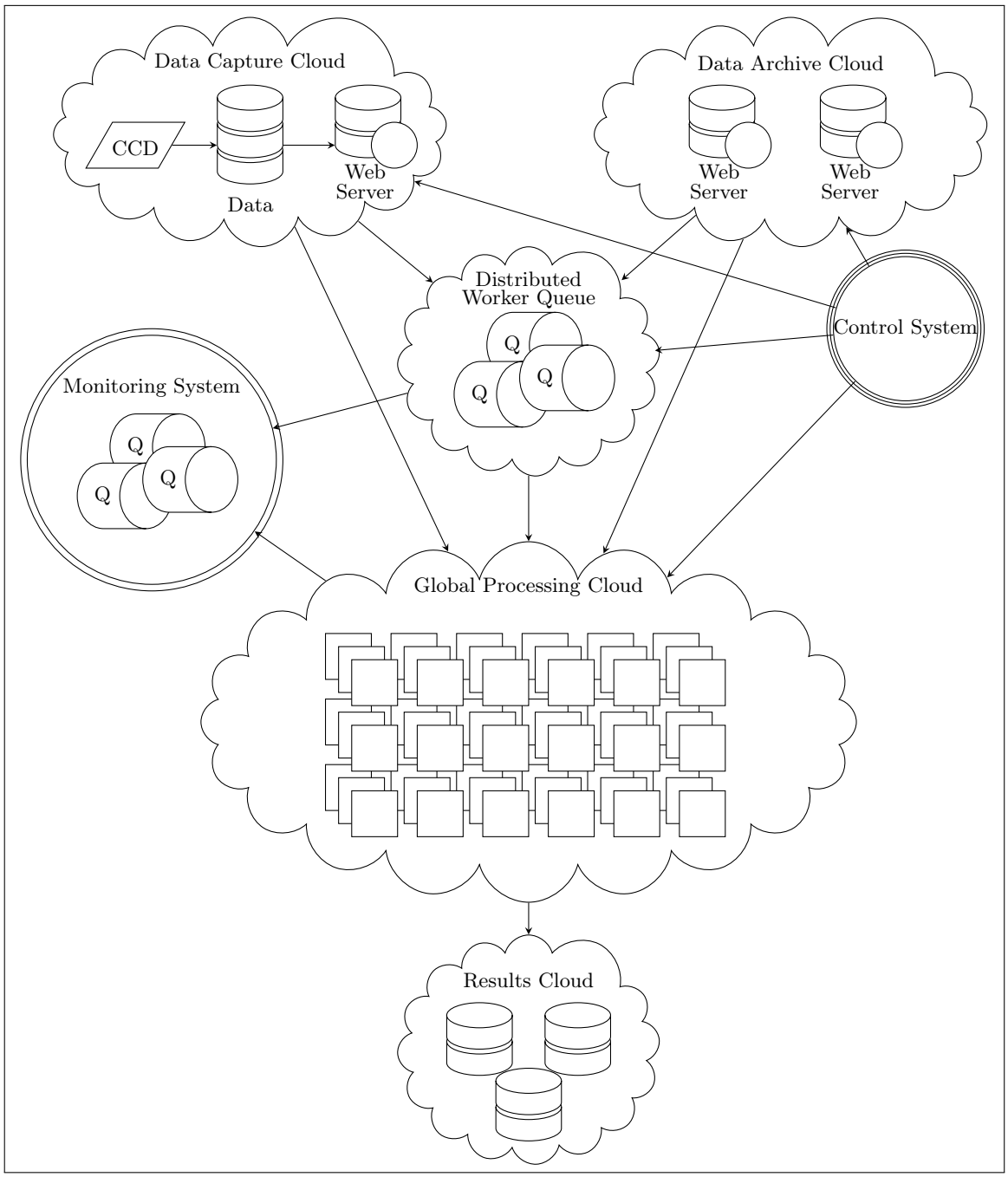


Figure 6.1: NIMBUS Architecture

6.2 Future Work

Further optimisations of the pipeline are considered as possible future areas of research. Much of the experimentation performed in this thesis demonstrated the extensive capability of a distributed system and identified the key factors within the system. It is possible to take these factors and monitor them such that a machine learning approach could be used to optimise a running system.

Further investigation into data compression and clipping optimisation could reduce the amount of data to be transferred. Clipped images, if not stacked deeply enough, have considerable readwrite costs, so data cubes could be constructed at source which are optimised for data movement to reduce the data transfer requirements. Rather than providing a static measure of how many files should be contained within the data cube, a machine learning approach could be used to find optimal sizes for data given different characteristics of the processing pipeline.

With the increase in the number of survey telescopes and robotic telescope farms, data processing requirements continue to grow. It should be possible to incorporate the NIMBUS pipeline into such systems for data processing to be ongoing and automated. A full implementation of a user interface would be required to provide flexibility regarding the science payloads to be used within the pipeline. A more sophisticated identification of stars using the world coordinate system would ensure that data could be categorised for future use. These science packages would be required to extend the basic photometry of such a system, which performed fundamental analysis, from which control commands could be incorporated back to the robotics farm.

Research should be performed into data storage and subsequent retrieval of processed data. Integration with the Virtual Observatory would be a requirement for both raw and processed data using search terms which were suitable for large scale data base queries such as NoSQL Column Databases [116]. Objects should be identified by type, position and time index. This would allow for research to be performed on objects observed at different times and with different instruments. A review of the search capabilities and indexing of astronomical data sets would provide context for the results of this pipeline.

To dramatically reduce the overall data movement where live telescopes are being used, data processing at the telescope site, using a GPU system, could result in the transmission of processed data, instead of the raw image. Work on light curve generation within the pipeline could also be incorporated into the worker nodes. Further research would be required into data reduction at the source of data production which would ensure that the NIMBUS pipeline could increase the overall processing rates by changing the ratio between data movement and data processing.

The NIMBUS pipeline is applicable to other areas within science. A data analysis of exiting science processing pipelines would be required to determine their suitability for data processing using NIMBUS. The structure and format of the dataset would require that it facilitated parallel processing at some level. If large volumes of data could be processed independently then a map-

ping of data files to worker nodes is possible via the queue generation sequence, where the files are advertised by each of the web stores where the data resides. Opportunities for distributed processing also exists where data is naturally distributed and data must be categorised or processed independently. Post processing of results similar to the map-reduce module would allow for results to be recombined to form new datasets. The migration of existing science projects such as the Solar Monitor, `http:\\solarmonitor.org` to the NIMBUS platform would demonstrate the flexible nature of the architecture and its applicability to multiple scientific disciplines. The Solar Monitor was developed by the Solar Research Group, Trinity College Dublin, led by Peter Gallagher and is an IDL based system which integrates data from multiple sources to produce near real time monitoring of active regions of solar flare activity [117].

NIMBUS and the ACN pipelines do not contain a security layer, and while the data being processed is not sensitive for the purposes of this thesis, other datasets are likely to require some form of security. Security within the pipeline should be researched, but where possible it should incorporate the cost of additional processing into the distributed components of the system. Public/private encryption could be performed with all data transferred over http port 80 or 8080 thus ensuring that firewall restrictions would not impact the overall connectedness of the pipeline. Data could either be encrypted at source, or by the web server before the data is advertised via a web queue.

While the current pipeline demonstrated that data can be processed by distributed computers, the process of preparing and advertising data also lends itself to opportunities for processing data using citizen science. For example `http://galaxyzoo.org` is a galaxy classification project where volunteers are presented with images within browsers, which they must then classify. The NIMBUS pipeline could incorporate a browser based application which downloads messages from the web queue which would include both the questions to be asked about the images, and links to the images themselves. In this way a browser could act as a generic system providing a per message citizen science question for the volunteer from a generic web based application. In this system the data processing is done by the end user.

A final thought to possible expansions of the NIMBUS architecture is to enable a global data processing cloud which uses the processing power of the mobile phone community. Given the proliferation of these devices a cloud could be constructed within minutes which could provide the processing power of a super computing thorough the use of millions of mobile devices. Possible options to encourage the deployment of the application on the mobile application could be the inclusion of a micro-payment scheme for applications that process specific amounts of data. Given the public nature of this approach it is also likely that additional security would be required to ensure that values returned were correct. Typical methods to ensure accurate data processing requires data to be processed by multiple clients and only accepting results which are consistent between clients. Using this method clients could download data while the phone is being charged

and within WIFI range so there are no data download charges and the performance of the phone was not compromised. A global flash cloud could be created easily which would be transient and potentially cost controlled by ensuring processing was done using a pricing module similar to the AWS spot price option where researchers could bid for computing resources at specific prices.

This research set out to determine if a distributed processing pipeline could scale to meet the existing, and future demands, of image processing for astronomical photometry, which for optical photometry is projected to be tens of terabytes of CCD image data per day in the near future. NIMBUS is presented as an architecture that is truly global in scale, with a demonstrated capability for processing hundreds of terabytes of CCD data per day. The limits of the scalability of this architecture were not fully determined due to financial constraints, but evidence suggests that significant scaling is still possible, in the order of petabytes per day. NIMBUS is also resilient, and flexible enough to address the processing requirements of other large scientific datasets.

Acronyms

ACN Astronomical Compute Node. 13

AMI amazon machine image. 121

BCO Blackrock Castle Observatory. 45

CCD Charge-Coupled Devices. 1

CMOS Complementary Metal Oxide Semiconductor. 2

CPL ESO Common Pipeline Library. 38

CPU Central Processing Unit. 2

CTE Charge Transfer Efficiency. 17

DSN Deep Space Network. 29

EC2 Elastic Compute Cloud. 33

EMCCD Electron Multiplying Charge Coupled Device. 45

ESO European Southern Observatory. 36

EVALSO Enabling Virtual Access to Latin-American Southern Observatories. 29

FITS Flexible Image Transport Systems. 23

GTC Gran Telescopio Canarias. 34

HDFS Hadoop Distributed File System. 103

HPC High Performance Computing. 29

HST Hubble Space Telescope. 30

IRAF Image Reduction and Analysis Facility. 30

ITTD Institute of Technology Tallaght in Dublin. 68

JMS Java Message Service. 43

JPL Jet Propulsion Laboratory. 7

JWST James Webb Space Telescope. 30

LBT Large Binocular Telescope. 34

LSST Large Synoptic Survey Telescope. 10

MAST Mikulski Archive for Space Telescopes. 30

MPT Message Passing Toolkit. 33

NFS Network File System. 13

NHPPS NOAO High-Performance Pipeline System. 37

NIST National Institute of Standards and Technology. 40

NOAO National Optical Astronomy Observatory. 36

OpenMP Open MultiProcessing. 33

OPUS Operational Pipeline Unified System. 28

S3 Simple Storage Service. 68

SDSS Sloan Digital Survey Systems. 35

SKA Square Kilometre Array. 15

SOC Space Operations Center. 31

SQS Simple Queue Service. 110

STScI Space Telescope Science Institute. 28

STSDAS Space Telescope Science Data Analysis System. 30

TDRSS Tracking and Data Relay Satellite System. 29

UTC Coordinated Universal Time. 144

VO Virtual Observatory. 27

Appendix A

Additional Material for Chapter 1

Time(secs)	fps	pixels	bits	terabytes
28800	100	5.5 MegaPixel	5.07E+14	63.36
28800	50	5.5 MegaPixel	2.53E+14	31.68
28800	25	5.5 MegaPixel	1.27E+14	15.84
28800	10	5.5 MegaPixel	5.07E+13	6.34
28800	5	5.5 MegaPixel	2.53E+13	3.17
28800	1	5.5 MegaPixel	5.07E+12	0.63
28800	100	4 MegaPixel	3.69E+14	46.08
28800	50	4 MegaPixel	1.84E+14	23.04
28800	25	4 MegaPixel	9.22E+13	11.52
28800	10	4 MegaPixel	3.69E+13	4.61
28800	5	4 MegaPixel	1.84E+13	2.30
28800	1	4 MegaPixel	3.69E+12	0.46
28800	100	2 MegaPixel	1.84E+14	23.04
28800	50	2 MegaPixel	9.22E+13	11.52
28800	25	2 MegaPixel	4.61E+13	5.76
28800	10	2 MegaPixel	1.84E+13	2.30
28800	5	2 MegaPixel	9.22E+12	1.15
28800	1	2 MegaPixel	1.84E+12	0.23
28800	100	1 MegaPixel	9.22E+13	11.52
28800	50	1 MegaPixel	4.61E+12	5.76
28800	25	1 MegaPixel	2.30E+12	2.88
28800	10	1 MegaPixel	9.22E+12	1.15
28800	5	1 MegaPixel	4.61E+11	0.58
28800	1	1 MegaPixel	9.22E+11	0.12

28800	100	0.24 MegaPixel	3.02E+12	3.02
28800	50	0.24 MegaPixel	1.51E+12	1.51
28800	25	0.24 MegaPixel	7.55E+11	0.75
28800	10	0.24 MegaPixel	3.02E+11	0.30
28800	5	0.24 MegaPixel	1.51E+11	0.15
28800	1	0.24 MegaPixel	3.02E+10	0.03

Table A.1: CCD data generation raw data for 32bit pixel precision for varying resolutions and frames per second over 8hr period

Appendix B

Additional Material for Chapter 3

The FEBRUUS pilot is a series of CFITSIO files which are used to perform pixel calibration on CCD images. The flowcharts for these programs are presented within this section of the appendix.

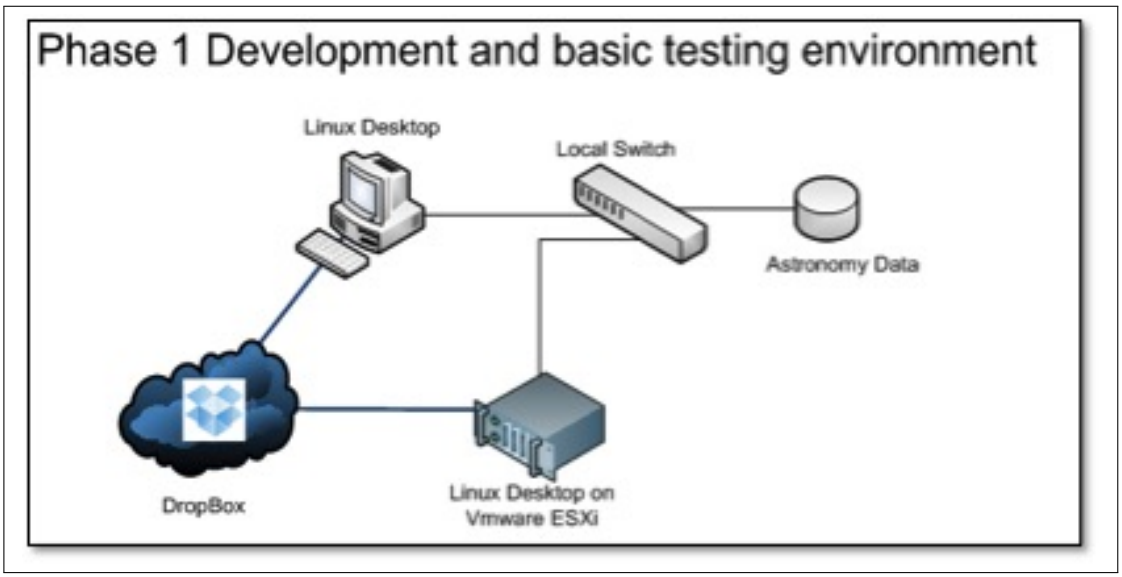


Figure B.1: FEBRUUS: Initial configuration for the Pilot Pipeline

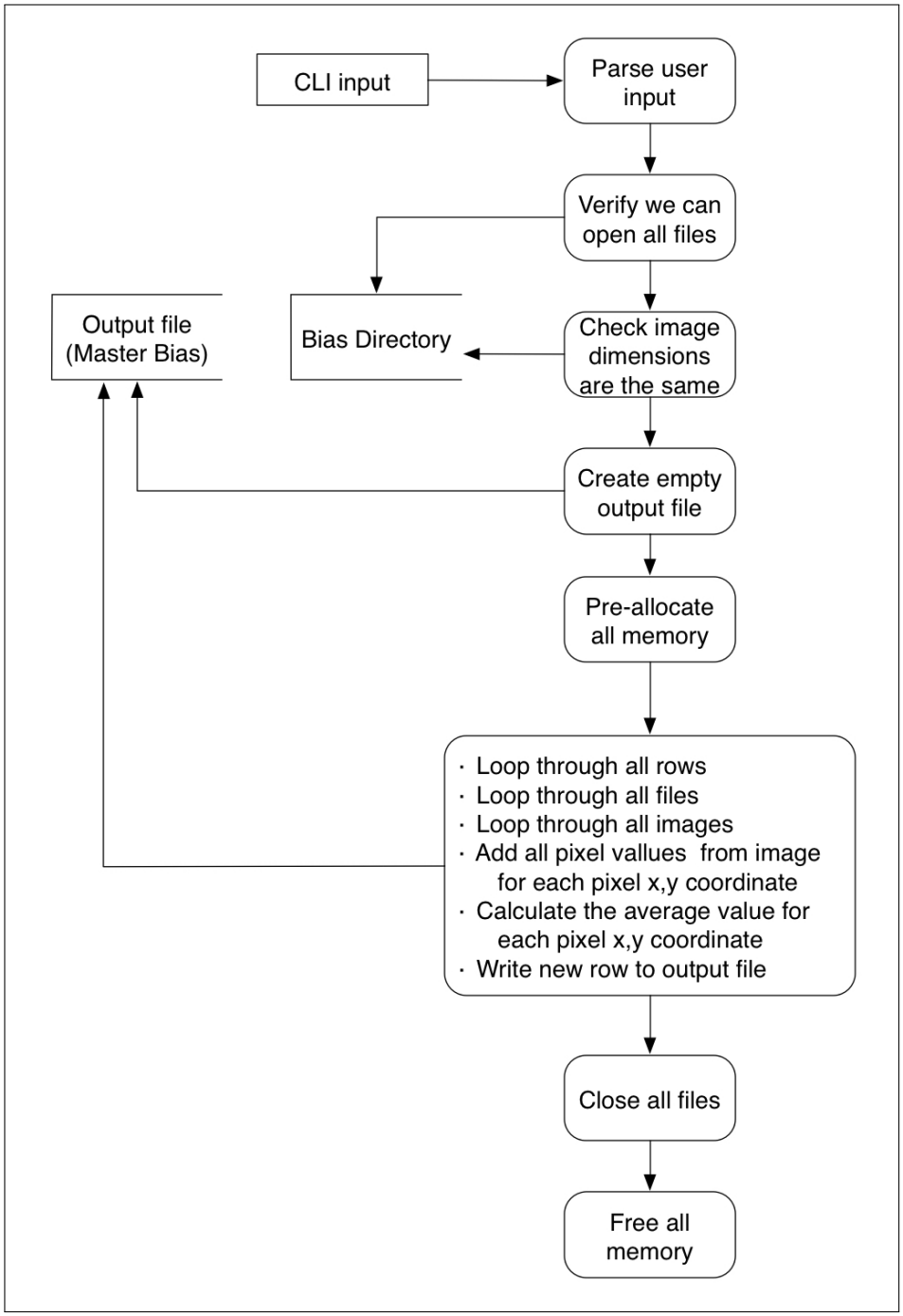


Figure B.2: FEBRUUS: The GMB.c program flowchart

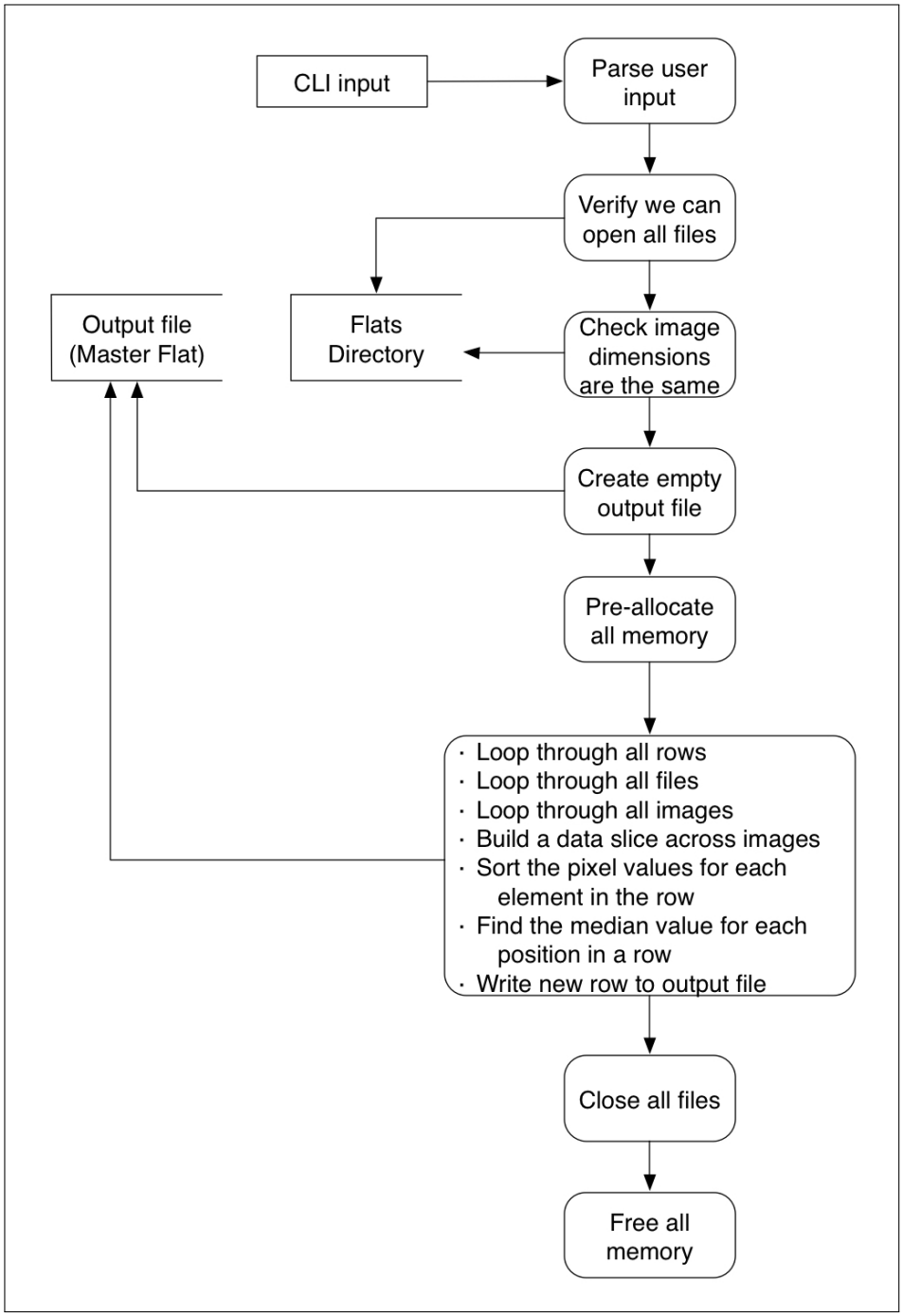


Figure B.3: FEBRUUS: The GMF program flowchart

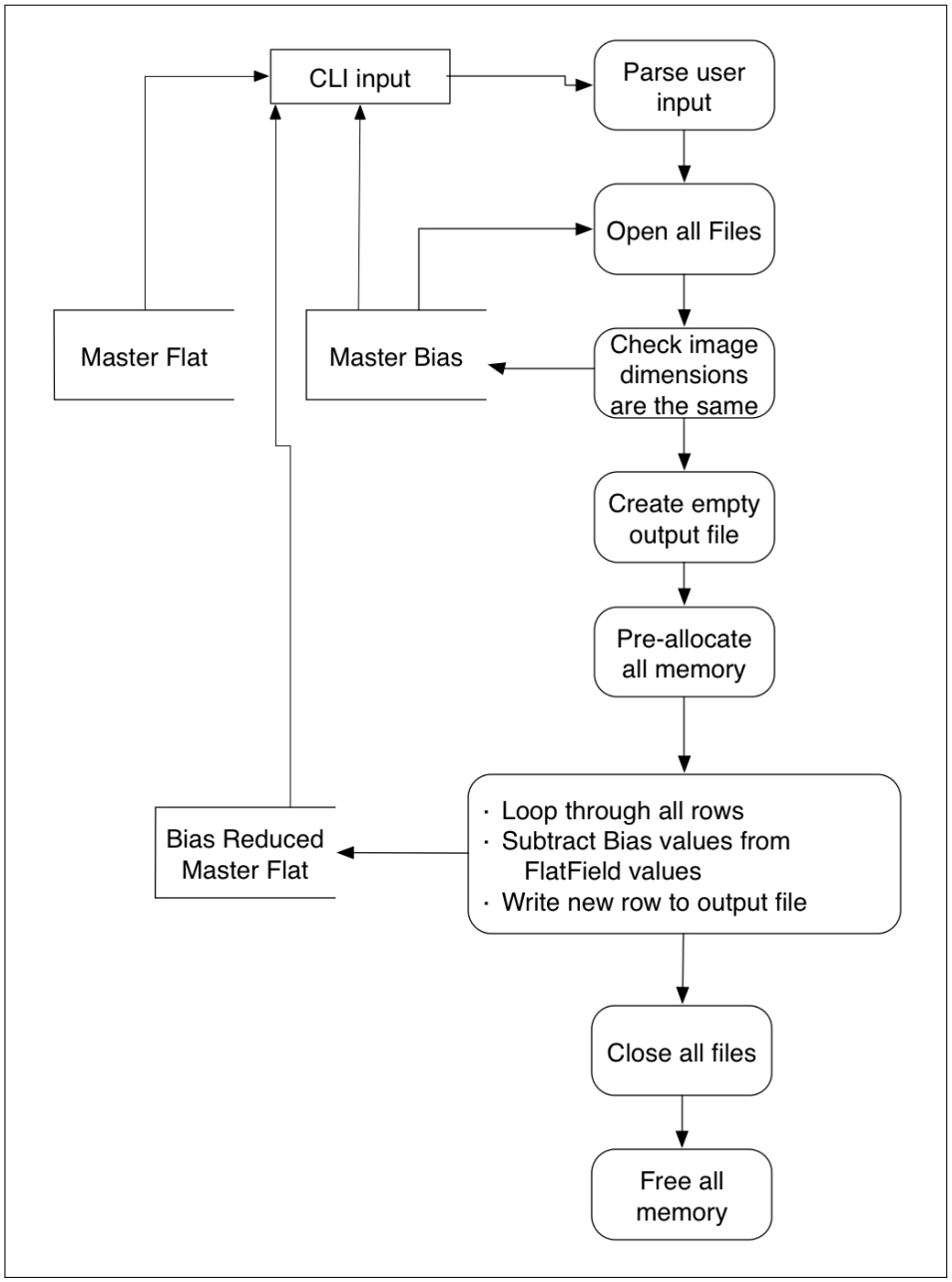


Figure B.4: FEBRUUS: The BMF program flowchart

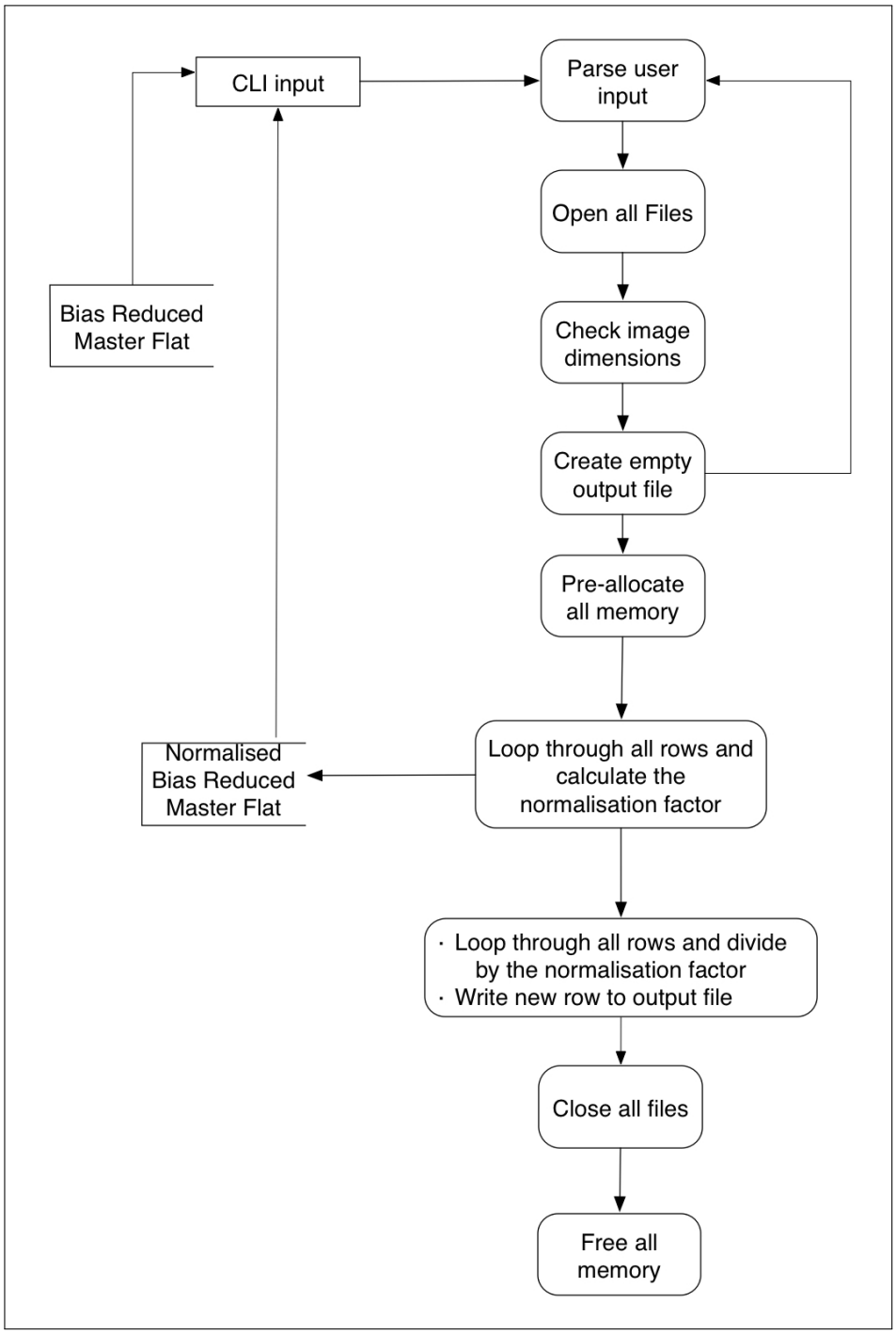


Figure B.5: FEBRUUS: The NMF program flowchart

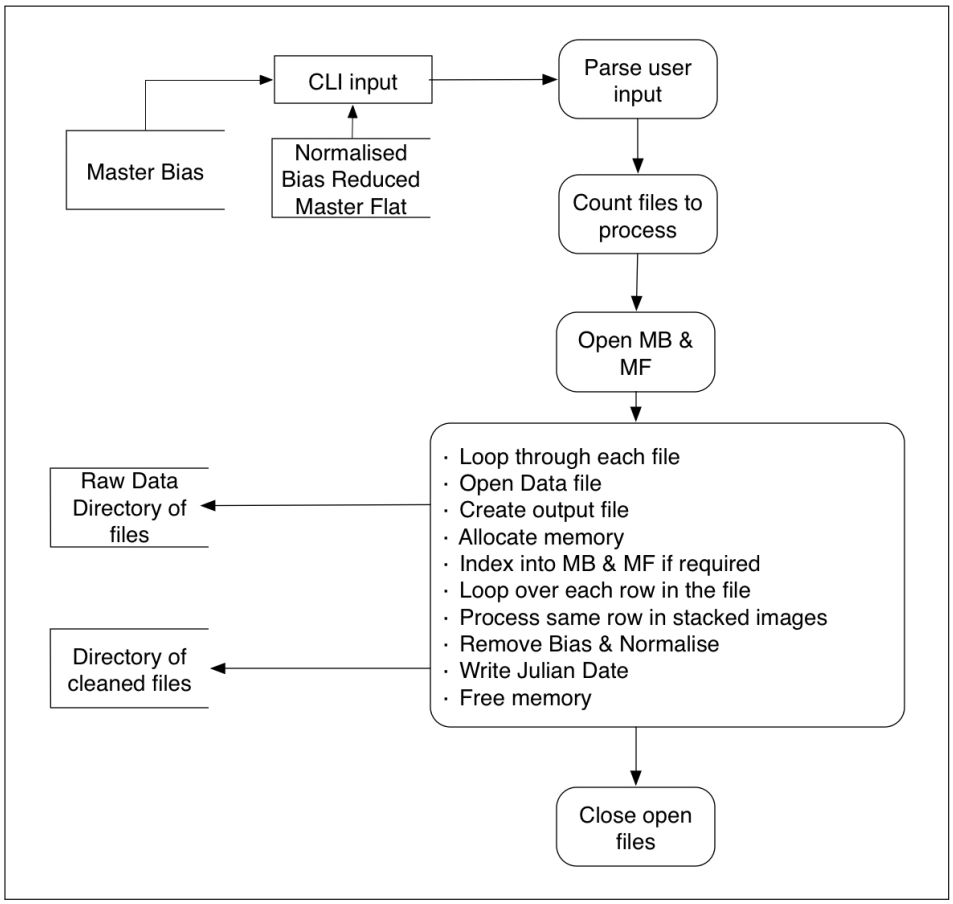


Figure B.6: FEBRUUS: The RRF program flowchart

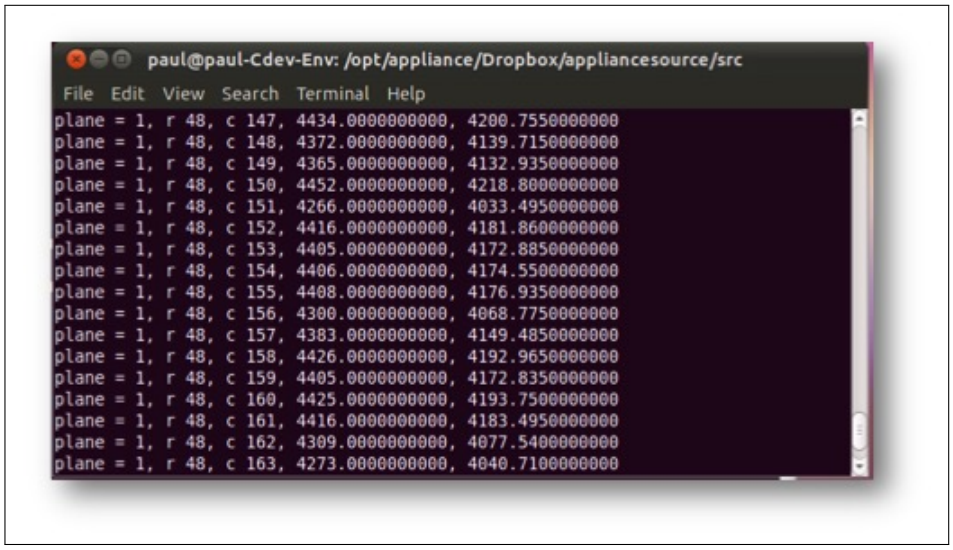
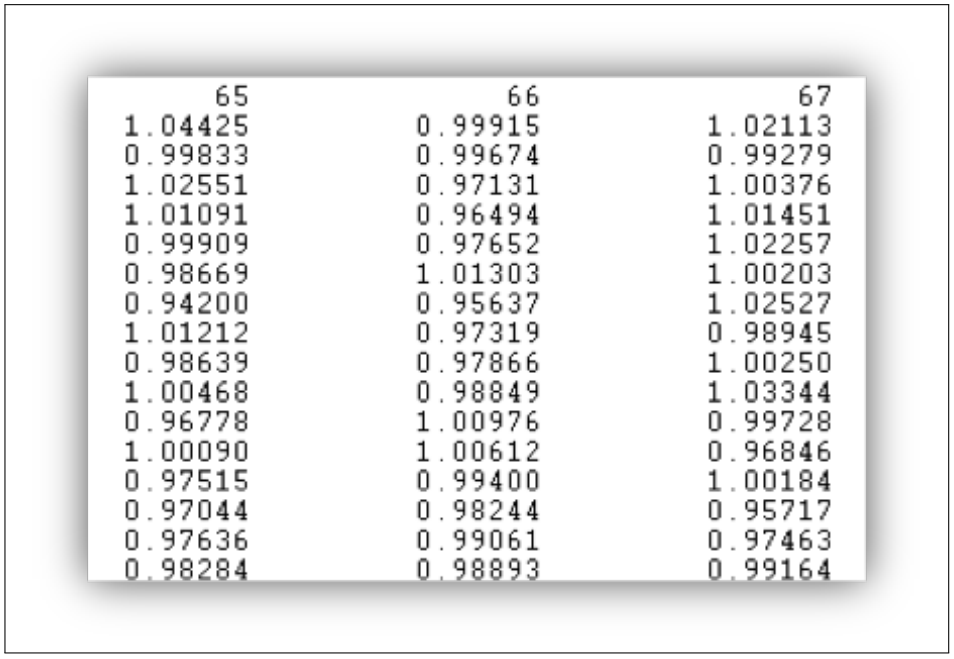


Figure B.7: FEBRUUS: bias reduced master flat output.



65	66	67
1.04425	0.99915	1.02113
0.99833	0.99674	0.99279
1.02551	0.97131	1.00376
1.01091	0.96494	1.01451
0.99909	0.97652	1.02257
0.98669	1.01303	1.00203
0.94200	0.95637	1.02527
1.01212	0.97319	0.98945
0.98639	0.97866	1.00250
1.00468	0.98849	1.03344
0.96778	1.00976	0.99728
1.00090	1.00612	0.96846
0.97515	0.99400	1.00184
0.97044	0.98244	0.95717
0.97636	0.99061	0.97463
0.98284	0.98893	0.99164

Figure B.8: FEBRUUS: Normalised master flat output.

Appendix C

Additional Material for Chapter 4

Command	Description
Activate-ACN -r NODEFILENAME	This command will activate all ACN Nodes in the nodes file so they start looking to a queue for image files to process. It is possible to have ACN nodes wait for a queue, but this option provides better control for experimentation so it is clean which nodes are active. Once a node is running, it checks the queue for unlocked files and locks them as shown in Figure 49. Once locked, the full image file is downloaded from an S3 bucket and cleaned using the acn-aphot program. This program once for each file downloaded.
Activate-ACN -c NODEFILENAME	This command will reset all ACN Nodes in the nodes file so they stop running, remove all temporary data and download the latest utility files, configuration files and Master Bias/Master Flat images ready for the next round of processing.
Activate-ACN -p NODEFILENAME	This command will Ping all of the nodes to ensure that they are accessible to the pipeline.
Activate-ACN -x NODEFILENAME	Reboot all of the nodes to ensure that they have flushed all caches. If a node processes the same data multiple times, it may operate faster on the subsequent executions due to caching of data in memory.

Table C.1: Control Node commands.

Command	Description
fits-compress -p DATADIR	This script has the option of performing compression in parallel or in sequence. The parallel execution spawns off processes and requires a machine with good RAM and processing capabilities. A comparison of performance for this script running in both modes is given in the next chapter.
s3-upload -p DATADIR	This script has the option of performing upload in parallel or in sequence. The parallel execution spawns off processes and requires a machine with good RAM and processing capabilities. A comparison of performance for this script running in both modes is given in the next chapter. Compressed or uncompressed data (depending on what is in the data directory) is uploaded to an S3 bucket. The data transfer rate through the server Ethernet card is increased using this approach. A comparison of performance for this script running in both modes is given in the next chapter.

Table C.2: Storage Control commands.

Command	Description
Activate-ACN -q compressed | standard | clipped	This command will create a list of empty files in a directory, which can be used as a simple queue. Files are named with a prefix source and traverse a dataset creating an entry in the queue for all files found. A compressed source file of 0000001.fits.fz has a corresponding queue entry of Queued-0000001.fits.fz. When successfully locked for processing by an ACN-Node this is changed to LOCKED-0000001.fits.fz. The NFS file system ensures only one lock can be obtained.

Table C.3: Queue Control commands.

Hardware	Storage Type	Reduction Steps	Step 1 Time	Step 2 Time	Total Time	Files per Sec
Macbook Pro	SATA	2	11:21:45	00:54:48	12:16:33	0.8332
Macbook Pro	SATA	1		0055:53	00:55:53	1.09812
eServer326	SCSI	2	00:49:00	01:15:23	02:04:23	0.49337
eServer326	SCSI	1		01:22:30	01:22:30	0.8332
eServer326	NFS	2	00:49:53	01:15:08	02:05:01	0.49087
eServer326	NFS	1		01:17:25	01:17:25	0.79268

Table C.4: Single Node performance data for 1 and 2 step reduction processing

Node Type	Node Ref	Seconds running	Files Cleaned	Cleaning Rate
VM-Tokyo	1	133	91	1.4615
VM-Tokyo	2	133	93	1.4301
VM-Tokyo	3	133	92	1.4456
VM-Tokyo	4	134	94	1.4255
VM-Sydney	5	132	93	1.4193
VM-Sydney	6	133	88	1.5113
VM-Sydney	7	133	88	1.5113
VM-Sydney	8	133	92	1.4456
VM-Paris	9	128	48	2.6666
VM-Paris	10	131	50	2.62
VM-Paris	11	132	46	2.8695
VM-Paris	12	132	49	2.6938
VM-Paris	13	133	53	2.5094
VM-Paris	14	133	52	2.5576
VM-Paris	15	133	53	2.5094
VM-Paris	16	134	53	2.5283
VM-Paris	17	134	52	2.5769
VM-Paris	18	134	52	2.5769
VM-Paris	19	134	51	2.6274
VM-Paris	20	134	53	2.5283
VM-London	21	131	37	3.5405

VM-London	22	132	37	3.5675
VM-London	23	132	37	3.5675
VM-London	24	133	37	3.5945
VM-London	25	133	37	3.5945
VM-London	26	133	37	3.5945
VM-London	27	133	37	3.5945
VM-London	28	133	37	3.5945
VM-London	29	133	38	3.5
VM-London	30	134	38	3.5263
VM-London	31	134	38	3.5263
VM-London	32	134	38	3.5263
VM-London	33	134	38	3.5263
VM-London	34	134	37	3.6216
VM-London	35	134	38	3.5263
ITTD	36	133	78	1.7051
ITTD	37	134	71	1.8873
ITTD	38	134	77	1.7402
ITTD	39	134	77	1.7402
ITTD	40	134	73	1.8356
ITTD	41	134	78	1.7179
ITTD	42	134	76	1.7631
ITTD	43	134	77	1.7402
DIT	44	133	69	1.9275
DIT	45	133	70	1.9
DIT	46	133	77	1.7272
DIT	47	134	80	1.675
DIT	48	134	78	1.7179
DIT	49	134	71	1.8873
DIT	50	134	71	1.8873
DIT	51	134	70	1.9142
BCO	52	133	70	1.9
BCO	53	133	72	1.8472
BCO	54	133	72	1.8472
BCO	55	133	70	1.9
BCO	56	134	79	1.6962
BCO	57	134	79	1.6962
BCO	58	134	72	1.8611

BCO	59	134	71	1.8873

Table C.5: ACN Multi-Node processing raw data

Appendix D

Additional Material for Chapter 5

Webnode	No. Msgs	Single Node Write Time	Msg Write Rate	Multi Node Write Time	Msg Write Rate
Webnode1	73620	8m47.279s	139.7	8m40.200s	141.6
Webnode2	73620	9m0.286s	136.3	8m57.302s	137.1
Webnode3	73620	3m7.055s	393.7	3m13.622s	381.5
Webnode4	73620	3m10.238s	387.5	3m21.267s	366.3
Webnode5	77342	7m11.410s	179.5	7m0.083s	184.2
Webnode6	77342	7m28.142s	172.6	7m26.385s	173.4
Webnode7	73620	3m49.490s	321.4	3m46.775s	325.8
Webnode8	73620	3m47.520s	324.3	3m40.521s	334.7

Table D.1: SQS queue write performance data

Exp. Num.	Web Server	Instance	Num. Instances	Workers per Inst.	FPS
1	FTP	IBM326	1	1	0.42
2	FTP	T1.Micro	1	1	0.51
3	FTP	M1.Large	1	1	0.54
4	FTP	T1.Micro	1	10	0.71.
5	HEANET	T1.Micro	1	5	0.81

6	FTP	T1.Micro	5	10	1.45
7	FTP	IBM326	1	5	1.61
8	FTP	IBM326	1	10	1.66
ğ 9	FTP	M1.Large	1	5	1.91
10	FTP	M1.Large	1	10	2.04
11	AWS	T1.Micro	10	1	2.64
12	FTP	T1.Micro	5	1	2.70
13	FTP	M1.Large	5	1	2.81
14	AWS	M1.Large	10	1	3.13
15	FTP	T1.Micro	5	5	3.60
16	FTP	x4150	1	10	5.73
17	FTP	x4150	1	10	6.47
18	AWS	T1.Micro	25	1	6.79
19	AWS	T1.Micro	10	5	7.19
20	AWS	M1.Large	25	1	7.52
21	FTP	x4150	1	50	8.20
22	FTP	M1.Large	5	10	8.64
23	FTP	M1.Large	5	5	9.33
24	AWS	M1.Large	10	5	12.25
25	AWS	T1.Micro	25	5	18.42
26	FTP	T1.Micro	50	1	27.04.83
27	FTP	M1.Large	50	1	28.37
28	AWS	M1.Large	25	5	28.83
29	FTP	T1.Micro	50	10	30.87
30	FTP	T1.Micro	50	5	39.00
31	AWS	M1.Large	50	5	53.11
32	FTP	T1.Micro	100	1	55.66
33	FTP	M1.Large	100	1	57.82
34	FTP	T1.Micro	100	10	72.11
35	FTP	T1.Micro	100	5	78.26
36	FTP	M1.Large	50	10	99.42
37	FTP	M1.Large	100	5	189.20
38	FTP	M1.Large	100	10	191.31
39	FTP	M1.XLarge	100	10	193.28
40	FTP	C1.XLarge	100	100	212.27
41	FTP	M1.XLarge	100	50	223.10
42	FTP	C1.XLarge	100	100	233.79

43	FTP/AWS	C1.2XLarge	70	20	332.93

Table D.4: Big Picture processing rate summary data

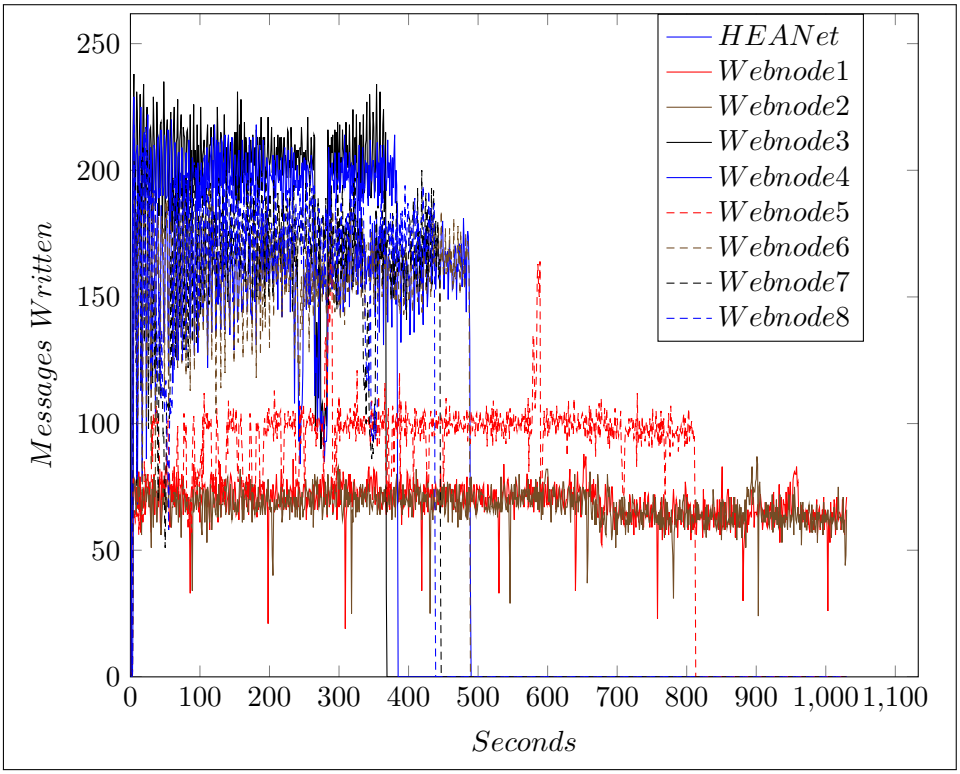


Figure D.1: Cumulative writing of messages to the canary queue

Ref.	Directory	Comment
RAW DATA	DataSet-BCO/22-9-2003	BCO 26GB Raw CCD Data
FEBRUSS	FEBRUUS-Pilot/Source Files	C programs for processing FITS files
FEBRUSS	FEBRUUS-Pilot/FITS-Testfiles	Sample FITS files for accuracy testing
ACN	ACN-Pipeline/ACN-C-Source	Core CCD Image processing C files
ACN	ACN-Pipeline/ACN-Controller	BASH scripts for controlling experiments
ACN	ACN-Pipeline/ACN-Worker	BASH scripts for controlling Worker Nodes
ACN	ACN-Pipeline/Experimental Data	Excel Spreadsheets with Result Data
NIMBUS	NIMBUS-Pipeline/ACN-APHOT-SRC	C Program for cleaning and Photometry
NIMBUS	NIMBUS-Pipeline/NIMBUS Controller	Python source for experimental execution
NIMBUS	NIMBUS-Pipeline/NIMBUS Worker	Python source for Worker execution
NIMBUS	NIMBUS-Pipeline/acn-science.pkg.tar.z	Compressed Pkg an utilities tools
NIMBUS	NIMBUS-Pipeline/Experimental Data	Raw data with R-files and Excel

Table D.2: Data Sources for experiments and graphs

Listing D.1: Generate Metrics: writes per node per second from RAW canaryq source files.

```
#!/bin/bash
# This generates a pair of values – epoch time and webserver
SERVERLIST=$(cat SQS* | awk -F "," '{print $5 }' | grep http | awk -F "http://"
    '{print $2 }' | awk -F "/" '{ print $1 }' | sort | uniq)

# Time is rounded to nearest second
TIMELIST=$(cat SQS* | grep SentTime | awk -F "," '{print $1 $5}' | awk -F "http://"
    '{print $1 "," $2 }' | awk -F "/" '{print $1}' | sed
    s/\{u\'SentTimestamp\'\:[[:space:]]u\'// | awk -F "\'\}," '{print $1}' | sed
    's/\(^.\{10\}\).\{3\}\(.*\)/\1\2/' | sort -n -k 1 | uniq)

for times in $TIMELIST
do
        echo -n $times ","
        for servers in $SERVERLIST
        do
                counter1=$(cat SQS* | grep $times | grep $servers | wc -l)
                echo -n $counter1 ","
        done
        echo
done
```

Family	Type	CPU	Memory (GB)
Virtual-Micro	t1.micro	1	0.65
Virtual-General	m1.large	2	7.5
Virtual-General	m1.xlarge	4	15
Virtual-Compute	c1.xlarge	8	7
Physical	x4150	8	64
Physical	IMB326e	1	4

Table D.3: Instance Types and their Specification

Listing D.2: Python code extract for multi-threaded single node SQS testing.

```
sqs_queue = conn.get_queue(args.queuearg)
canary_queue = conn.get_queue("canaryq")

class Sender(threading.Thread):
        def __init__(self):
                threading.Thread.__init__(self)
        def run(self):
                global sqs_queue,canary_queue,queue
                while True:
                        try:
                                msg = queue.get(True,3)
                                m = Message()
                                m.set_body(msg)
                                status = sqs_queue.write(m)
                        except Queue.Empty:
                                return
                        except:
                                return
queue = Queue.Queue(0)

for file in sys.stdin:
        file = ipadd+file
        queue.put(file)

threads = []
for n in xrange(40):
        t = Sender()
        t.start()
        threads.append(t)

for t in threads:
        t.join()
```

Listing D.3: Python code extract for multi-threaded SQS queue reading.

```
1
2  class Sender(threading.Thread):
3          def __init__(self):
4                  threading.Thread.__init__(self)
5
6          def run(self):
7                  global sqs_queue,queue
8                  name = args.experiment+str(queue.get())+"-"+args.queuearg+".csv"
9                  f = open(name,'w')
10
11                 while True:
12                         try:
13                                 m = sqs_queue.read(60)
14                                 m = sqs_queue.get_messages(num_messages=1,
                                       attributes='SentTimestamp')
15                                 print "This is the message->",
                                       m[0].attributes,m[0].get_body()
16                                 f.write(str(m[0].attributes)+str(m[0].get_body())+"\n")
17                                 sqs_queue.delete_message(m[0])
18                         except:
19                                 if sqs_queue.count() < 1:
20                                         f.write(args.queuearg + " is empty\n")
21                                         return
22 queue = Queue.Queue(0)
23
24 threads = []
25 for n in xrange(40):
26         queue.put(n)
27         t = Sender()
28         t.start()
29         threads.append(t)
30
31 for t in threads:
32         t.join()
```

Listing D.4: Bash code combining canary queue messages into bins.

```
1
2  #!/bin/bash
3  # This generates a pair of values – epoch time and webserver
4  FILELIST=$(ls SQS*)
5  SERVERLIST=$(cat SQS* | awk -F "," '{print $5 }' | grep http | awk -F "http://"
       '{print $2 }' | awk -F "/" '{print $1 }' | sort | uniq)
6
7  for servers in $SERVERLIST
8  do
9          echo -n $servers ","
10 done
11 echo
12 x=1
13 y=$(($x+100))
14 filelen=$(cat SQS-canary9-* | wc -l)
15 loops=$(($filelen/100))
16
17 for (( loopcount=1; loopcount<=$loops; loopcount++ ))
18 do
19         for servers in $SERVERLIST
20         do
21                 counter=0
22                 for files in $FILELIST
23                 do
24                         counter1=$(cat $files | sed -n "$x,$y p" | grep $servers | wc -l)
25                         counter=$(($counter+counter1))
26                 done
27                 echo -n $counter ","
28         done
29         echo
30         x=$(($x+100))
31         y=$(($x+100))
32 done
```

Listing D.5: Experimental Batch Script for running multiple experiments.

```
#!/bin/bash
# Configuration options for running experiments
# -a Use Amazon Web servers
# -d Use DIT Web servers
# -n Use HEANT Web servers
# -x Use ALL Web servers
#
# Num Instances to Run
# Time in Seconds to run the experiment
# Name of the Experiment
# Number of Web Servers to run (1 or 2)
# Size of the instances to use in the experiment
# All experiments are formally names to correspond to a specific configuration
# Format of the run command below is as follows.
#
# webservertype instancenum seconds expname webservernum instancetype
#
#AWS Experiments group 1, Workers= 1 per instance, BatchSize = 10
#
# The following parameters must be set in the Worker Package
#  Workers per instance in this case is 1
#  Batch Size per worker is set to 10
#
#  The canary1.nightsky.ie should also be running rounding up the
# number of instances by 1 in all cases
./run-experiment.sh -a 9 1200 SetB5w.2 1 t1.micro
./run-experiment.sh -a 24 1200 SetB5w.3 1 t1.micro
./run-experiment.sh -a 49 1200 SetB5w.4 1 t1.micro
./run-experiment.sh -a 99 1200 SetB5w.5 1 t1.micro
./run-experiment.sh -a 4 1200 SetB5w.6 1 m1.large
./run-experiment.sh -a 9 1200 SetB5w.7 1 m1.large
./run-experiment.sh -a 24 1200 SetB5w.8 1 m1.large
./run-experiment.sh -a 49 1200 SetB5w.9 1 m1.large
./run-experiment.sh -a 99 1200 SetB5w.10 1 m1.large
```

Num. Instances	Web Servers.	Files per Second
1	1AWS	1470
1	1AWS	1560
1	1AWS	1733
1	1AWS	1971
1	FTP	2295
1	FTP	2446
5	1AWS	7694
5	1AWS	7845
5	FTP	10366
5	FTP	11197
10	1AWS	14635
10	1AWS	14702
50	FTP	105423
50	FTP	119318
100	1AWS	129333
100	HEANET	133612
100	FTP	186777
100	FTP	227039
100	ALL	227514
100	FTP	229577

Table D.5: Files Processed by varying number of M1.Large instances

Bibliography

[1] Merriam-Webster. *The Merriam-Webster dictionary.* Merriam-Webster, Springfield, Mass, 2014.

[2] C Sterken. *Astronomical photometry : a guide.* Kluwer Academic Publishers, Dordrecht Boston, 1992.

[3] Robert R Newton. The crime of Claudius Ptolemy. *Baltimore : Johns Hopkins University Press*, 1977.

[4] T. B. McCord and J. P. Bosel. Potential usefulness of CCD imagers in astronomy. In *Charge-Coupled Device Technology for Scientific Imaging Applications*, pages 65–69, June 1975.

[5] Nausicaa DELMOTTE. Astronomical observatory sites by latitude and longitude @ONLINE. `http://www.eso.org/~ndelmott/obs_sites.html`, October 2007.

[6] James Janesick and Gloria Putnam. DEVELOPMENTS AND APPLICATIONS OF HIGH-PERFORMANCE CCD AND CMOS IMAGING ARRAYS. *Annual Review of Nuclear and Particle Science*, 53(1):263–300, December 2003.

[7] Robert R White, James Wren, Heath R Davis, Mark Galassi, Daniel Starr, W T Vestrand, and P Wozniak. Talon: the telescope alert operation network system: intelligent linking of distributed autonomous robotic telescopes. *Advanced Software*, 5496:302–312, September 2004.

[8] G Bruce Berriman and Steven L Groom. How will astronomy archives survive the data tsunami? *Communications of the ACM*, 54(12):52–56, 2011.

[9] J Gantz and D Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the Future*, 2012.

[10] Eugene F Milone and C Sterken. *Astronomical Photometry.* Past, Present, and Future. Springer Science and Business Media, April 2011.

[11] 2nd century Ptolemy, Christian Heinrich Friedrich Peters, and E B B 1841 Knobel. *Ptolemy's Catalogue of Stars*. A Revision of the Almagest. Nabu Press, September 2010.

[12] JB Hearnshaw. The measurement of starlight. *The Measurement of Starlight, by JB Hearnshaw, Cambridge, UK: Cambridge University Press, 2005*, 1, 2005.

[13] Erik Zackrisson, Pat Scott, Claes-Erik Rydberg, Fabio Iocco, Bengt Edvardsson, Göran Östlin, Sofia Sivertsson, Adi Zitrin, Tom Broadhurst, and Paolo Gondolo. Finding high-redshift dark stars with the james webb space telescope. *The Astrophysical Journal*, 717(1):257, 2010.

[14] R Miles. A light history of photometry: from hipparchus to the hubble space telescope. *Journal of the British Astronomical Association*, 117:172–186, 2007.

[15] Steve B Howell. *Handbook of CCD astronomy*, volume 5. Cambridge University Press, 2006.

[16] Jason W Barnes. Transit lightcurves of extrasolar planets orbiting rapidly rotating stars. *The Astrophysical Journal*, 705(1):683, 2009.

[17] G A Antcliffe, L J Hornbeck, J M Younse, J B Barton, and D R Collins. Large-area CCD imagers for spacecraft applications. *Charge-Coupled Device Technology for Scientific Imaging Applications*, -1:125–136, June 1975.

[18] J Janesick and T Elliott. History and advancement of large array scientific CCD imagers. *... CCD Observing and Reduction ...*, 1992.

[19] James R Janesick. *Scientific charge-coupled devices*, volume 117. SPIE press Bellingham, WA, 2001.

[20] W J Borucki and A L Summers. The photometric method of detecting other planetary systems. *Icarus (ISSN 0019-1035)*, 58:121–134, April 1984.

[21] David Charbonneau, Timothy M Brown, David W Latham, and Michel Mayor. Detection of planetary transits across a sun-like star. *The Astrophysical Journal Letters*, 529(1):L45, 2000.

[22] David G Koch, William J Borucki, Larry Webster, Edward W Dunham, Jon M Jenkins, John Marriott, and Harold J Reitsema. Kepler: a space mission to detect earth-class exoplanet. In Pierre Y Bely and James B Breckinridge, editors, *Astronomical Telescopes and Instrumentation*, pages 599–607. SPIE, August 1998.

[23] Vik Dillon. Schematic cross-section through a ccd pixel@ONLINE. `http://www.vikdhillon.staff.shef.ac.uk/teaching/phy217/detectors/phy217_det_structure.html`, 2013.

[24] M J Graham. Astronomy 2020: A Pragmatic Approach. *Astronomical Data Analysis Software and Systems XVIII*, 2009.

[25] H Ferguson, Perry Greenfield, Tim Axelrod, Stefi Baum RIT, Alberto Conti, Dennis Crabtree, Eric Feigelson, Mike Fitzpatrick, Wendy Freedman, Kim Gillies, et al. Astronomical data reduction and analysis for the next decade. *The Astronomy and Astrophysics Decadal Survey*, 2009.

[26] PJ Teuben. Astronomical data analysis software and systems iv. In *ASP Conf. Ser*, volume 77, 1995.

[27] Fionn Murtagh, Jean-Luc Starck, and Mireille Louys. Distributed visual information management in astronomy. *Computing in Science & Engineering*, 4(6):14–23, 2002.

[28] Ronald L Gilliland. Details of noise sources and reduction processes. In *Astronomical CCD observing and reduction techniques*, volume 23, page 68, 1992.

[29] Paul A Abell, Julius Allison, Scott F Anderson, John R Andrew, J Roger P Angel, Lee Armus, David Arnett, SJ Asztalos, Tim S Axelrod, Stephen Bailey, et al. Lsst science book, version 2.0. *arXiv preprint arXiv:0912.0201*, 2009.

[30] Peter E Dewdney, Peter J Hall, Richard T Schilizzi, and T Joseph LW Lazio. The square kilometre array. *Proceedings of the IEEE*, 97(8):1482–1496, 2009.

[31] D L Jones, K Wagstaff, D R Thompson, L D'Addario, R Navarro, C Mattmann, W Majid, J Lazio, R Preston, and U Aerospace Conference 2012 IEEE Rebbapragada. Big data challenges for large radio arrays. In *Aerospace Conference, 2012 IEEE*, 2012.

[32] A Fowler, P Waddell, and L Mortara. Evaluation of the rca 512x320 charge-coupled device (ccd) imagers for astronomical use. In *Solid State Imagers for Astronomy*, pages 34–44. International Society for Optics and Photonics, 1981.

[33] Radu Corlab. Gcx user's manual @ONLINE. `http://astro.corlan.net/gcx/html/node7.html`, December 2004.

[34] P Massey and G H Jacoby. CCD data: The good, the bad, and the ugly. *Astronomical CCD observing and ...*, 1992.

[35] D C Wells, E W Greisen, and R H Harten. FITS - a Flexible Image Transport System. *Astronomy and Astrophysics Supplement Series*, 44:363, June 1981.

[36] Mark Adams. *Stellar magnitudes from digital pictures.* AURA, Association of Universities for Research in Astronomy, 1980.

[37] Otto Klotz. Magnitude of Stars. *Journal of the Royal Astronomical Society of Canada*, 15:289, October 1921.

[38] RJ Hanisch and Peter Quinn. The international virtual observatory. *Space Telescope Science Institute, Baltimore, MD*, 21218, 2003.

[39] ALMA Explorer.

[40] Douglas J Mudgway. Uplink-downlink: a history of the nasa deep space network, 1957-1997. *Uplink-downlink: a history of the NASA Deep Space Network, 1957-1997. Washington, DC: National Aeronautics and Space Administration, 2000. The NASA history series*, 1, 2000.

[41] Matthew E Grein, Matthew Willis, Andreew Kerman, Eric Dauler, Barry Romkey, Danna Rosenberg, Jung Yoon, Richard Molnar, Bryan S Robinson, Daniel Murphy, et al. A fiber-coupled photon-counting optical receiver based on nbn superconducting nanowires for the lunar laser communication demonstration. In *CLEO: Science and Innovations*, pages SM4J–5. Optical Society of America, 2014.

[42] D Lindler. Astronomical Data Reduction Pipelines at NASA: 30 Years and Counting. *The 2007 ESO Instrument Calibration Workshop*, page 177, 2008.

[43] Stephen T Bryson, Jon M Jenkins, Todd C Klaus, Miles T Cote, Elisa V Quintana, Jennifer R Hall, Khadeejah Ibrahim, Hema Chandrasekaran, Douglas A Caldwell, Jeffrey E Van Cleve, et al. Selecting pixels for kepler downlink. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 77401D–77401D. International Society for Optics and Photonics, 2010.

[44] N A Renzetti. DSN Functions and Facilities. *Deep Space Network Progress Report*, 1975.

[45] J Teles, MV Samii, and CE Doll. Overview of tdrss. *Advances in Space Research*, 16(12):67–76, 1995.

[46] J Rose, R Akella, S Binegar, TH Choo, C Heller-Boyer, T Hester, P Hyde, R Perrine, MA Rose, and K Steuerman. The opus pipeline: a partially object-oriented pipeline system. In *Astronomical Data Analysis Software and Systems IV*, volume 77, page 429, 1995.

[47] C.E Leitherer. *HST Data Handbook*, volume 2. Baltimore: STScI, 1995.

[48] Alan Johns, Bonita Seaton, Jonathan Gal-Edd, Ronald Jones, Curtis Fatig, and Francis Wasiak. James Webb Space Telescope: L2 communications for science data processing. In Roger J Brissenden and David R Silva, editors, *Astronomical Telescopes and Instrumentation: Synergies Between Ground and Space*, pages 70161D–70161D–7. SPIE, July 2008.

[49] Michael R Haas, Natalie M Batalha, Steve T Bryson, Douglas A Caldwell, Jessie L Dotson, Jennifer Hall, Jon M Jenkins, Todd C Klaus, David G Koch, and Jeffrey Kolodziejczak. Kepler science operations. *The Astrophysical Journal Letters*, 713(2):L115, 2010.

[50] Todd C Klaus, Miles T Cote, Sean McCauliff, Forrest R Girouard, Bill Wohler, Christopher Allen, Hema Chandrasekaran, Stephen T Bryson, Christopher Middour, Douglas A Caldwell, and Jon M Jenkins. The Kepler Science Operations Center pipeline framework extensions. In *Software and Cyberinfrastructure for Astronomy*, pages 774018–774018–11. SPIE, July 2010.

[51] Jennifer R Hall, Khadeejah Ibrahim, Todd C Klaus, Miles T Cote, Christopher Middour, Michael R Haas, Jessie L Dotson, Brett Stroozas, Michael Wu, Jeneen Sommers, and Paresh Bhavsar. Kepler Science Operations processes, procedures, and tools. In *Software and Cyberinfrastructure for Astronomy*, pages 77370H–77370H–12. SPIE, July 2010.

[52] J Van Cleve, DA Caldwell, et al. Kepler instrument handbook. *NASA Ames Research Center*, 2009.

[53] Elisa V Quintana, Jon M Jenkins, Bruce D Clarke, Hema Chandrasekaran, Joseph D Twicken, Sean D McCauliff, Miles T Cote, Todd C Klaus, Christopher Allen, Douglas A Caldwell, and Stephen T Bryson. Pixel-level calibration in the Kepler Science Operations Center pipeline. In *Sensors, Cameras, and Systems for Industrial/Scientific Applications XI*, pages 77401X–77401X–12. SPIE, July 2010.

[54] Todd C Klaus, C Henze, J D Twicken, J Hall, S D McCauliff, F Girouard, M Cote, R L Morris, B Clarke, J M Jenkins, D Caldwell, and Kepler Science Operations Center. Computational Challenges in Processing the Q1-Q16 Kepler Data Set. *AAS/Division for Planetary Sciences Meeting Abstracts*, 45, October 2013.

[55] Piyush Mehrotra, Jahed Djomehri, Steve Heistand, Robert Hood, Haoqiang Jin, Arthur Lazanoff, Subhash Saini, and Rupak Biswas. Performance evaluation of amazon ec2 for nasa hpc applications. In *Proceedings of the 3rd workshop on Scientific Cloud Computing Date*, pages 41–50. ACM, 2012.

[56] H. Siddiqui, S. G. Els, R. Guerra, N. Cheek, A. Mora, and W. O'Mullane. Gaia downlink data processing. In *SPIE Astronomical Telescopes+ Instrumentation*, volume 9149, pages 91492E–91492E–9, 2014.

[57] F. Mignard, C. Bailer-Jones, U. Bastian, R. Drimmel, L. Eyer, D. Katz, F. van Leeuwen, X. Luri, W. O'Mullane, X. Passot, D. Pourbaix, and T. Prusti. Gaia: organisation and challenges for the data processing. In *A Giant Step: from Milli- to Micro-arcsecond Astrometry*, volume 3 of *Proceedings of the International Astronomical Union*, pages 224–230, 10 2007.

[58] Michèle Péron. Control Software & Data Reduction and Analysis@ONLINE. *ESO JENHAM 2010*, page 28, September 2010.

[59] Jordi Cepa. Osiris imaging and spectroscopy for the gtc. *Astrophysics and Space Science*, 263(1-4):369–372, 1998.

[60] J. Cepa, M. Aguiar, E.J. Alfaro, J. Bland-Hawthorn, H.O. Castañeda, F. Cobos, S. Correa, C. Espejo, A. Farah, A.B. Fragoso-López, J.V. Gigante, F. Garfias, J.J. González, V. González-Escalera, J.I. González-Serrano, B. Hernández, A. Herrera, C. Militello, L. Peraza, R. Pérez, J.L. Rasilla, B. Sánchez, M. Sánchez-Portal, and C. Tejada. Osiris: Status and science. In Francesca Figueras, JosepMiquel Girart, Margarita Hernanz, and Carme Jordi, editors, *Highlights of Spanish Astrophysics IV*, pages 71–80. Springer Netherlands, 2007.

[61] John M Hill and Piero Salinari. Large Binocular Telescope project. *Large Ground-based Telescopes. Edited by Oschmann*, 4837:140–153, February 2003.

[62] M Fukugita, T Ichikawa, J E Gunn, M Doi, K Shimasaku, and D P Schneider. The Sloan Digital Sky Survey Photometric System. *Astronomical Journal v.111*, 111:1748, April 1996.

[63] Ž Ivezić, RH Lupton, D Schlegel, B Boroski, J Adelman-McCarthy, B Yanny, S Kent, C Stoughton, D Finkbeiner, N Padmanabhan, et al. Sdss data management and photometric quality assessment. *Astronomische Nachrichten*, 325(6-8):583–589, 2004.

[64] J E Gunn, M Carr, C Rockosi, M Sekiguchi, K Berry, B Elms, E de Haas, Z Ivezic, G Knapp, R Lupton, G Pauls, R Simcoe, R Hirsch, D Sanford, S Wang, D York, F Harris, J Annis, L Bartozek, W Boroski, J Bakken, M Haldeman, S Kent, S Holm, D Holmgren, D Petravick, A Prosapio, R Rechenmacher, M Doi, M Fukugita, K Shimasaku, N Okada, C Hull, W Siegmund, E Mannery, M Blouke, D Heidtman, D Schneider, R Lucinio, and J Brinkman. The Sloan Digital Sky Survey Photometric Camera. *The Astronomical Journal*, 116(6):3040–3081, December 1998.

[65] LSST. New windows on the universe | lsst @ONLINE. `http://www.lsst.org/lsst/public`, 2015.

[66] LSST. Lsst: A new telescope concept @ONLINE. `http://www.lsst.org/lsst/public/tour_software`, 2015.

[67] Martino Romaniello. Eso's science archive facility. In *Bulletin of the American Astronomical Society*, volume 1, page 30503, 2011.

[68] P Quinn, A Lawrence, and B Hanisch. The Management, Storage and Utilization of Astronomical Data in the 21st Century—A Discussion Paper for the OECD Global Science Forum. *OECD Global Science Forums*, 2004.

[69] R J Hanisch. Distributed Data Systems and Services for Astronomy and the Space Sciences. *Astronomical Data Analysis Software and Systems IX*, 216:201, 2000.

[70] Harvey R Butcher. Lofar: First of a new generation of radio telescopes. In *Astronomical Telescopes and Instrumentation*, pages 537–544. International Society for Optics and Photonics, 2004.

[71] K Begeman, Andrey N Belikov, DR Boxhoorn, F Dijkstra, H Holties, Z Meyer-Zhao, GA Renting, EA Valentijn, and W-J Vriend. Lofar information system. *Future generation computer systems*, 27(3):319–328, 2011.

[72] Andreas Kaufer and Florian Kerber. *The 2007 ESO Instrument Calibration Workshop*. Proceedings of the ESO Workshop Held in Garching, Germany, 23-26 January 2007. Springer, January 2008.

[73] Doug Tody. The IRAF Data Reduction and Analysis System. In Lawrence D Barr, editor, *1986 Astronomy Conferences*, pages 733–748. SPIE, August 1986.

[74] D Scott, F Pierfederici, R A Swaters, B Thomas, and F G Valdes. The NOAO High-Performance Pipeline System: Architecture Overview. *Astronomical Data Analysis Software and Systems XIV*, 376:265, October 2007.

[75] F Valdes, T Cline, F Pierfederici, M Miller, B Thomas, and R Swaters. The NOAO High Performance Pipeline System. *NOAO DPP Document PL001*, 950, 2006.

[76] K Banse, P Ballester, C Izzo, Y Jung, L K Lundin, D J McKay, A Modigliani, R M Palsa, M Kiesgen, and C Sabet. The Common Pipeline Library - a silver bullet for standardising pipelines? *Astronomical Data Analysis Software and Systems XIV*, 314:392, July 2004.

[77] L de Bilbao, L K Lundin, P Ballester, K Banse, C Izzo, R Palsa, and C E García-Dabó. Multi-Threading for ESO Pipelines. *Astronomical Data Analysis Software and Systems XIV*, 434:241, December 2010.

[78] RA Shaw and HA Bushouse. An iraf port of the new iue calibration pipeline. In *Astronomical Data Analysis Software and Systems VII*, volume 145, page 103, 1998.

[79] Seathrún O Tuairisg, Michael Browne, John Cuniffe, Andrew Shearer, John Morrison, and Keith Power. Webcom-g: Implementing an astronomical data analysis pipeline on a grid-type infrastructure. In *ASTRONOMICAL SOCIETY OF THE PACIFIC CONFERENCE SERIES*, volume 347, page 325. Citeseer, 2005.

[80] John P Blakeslee, Kenneth R Anderson, GR Meurer, N Benitez, and D Magee. An automatic image reduction pipeline for the advanced camera for surveys. *arXiv preprint astro-ph/0212362*, 2002.

[81] A Rotem-Gal-Oz. Fallacies of distributed computing explained@ONLINE. `http://www.rgoarchitects.com/Files/fallacies`, 2006.

[82] Peter Mell and Timothy Grance. The nist definition of cloud computing. 2011.

[83] Lee Badger, Tim Grance, Robert Patt-Corner, and Jeff Voas. *Cloud Computing Synopsis and Recommendations: Recommendations of the National Institute of Standards and Technology.* CreateSpace Independent Publishing Platform, 2012.

[84] D Borthakur. The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 2007.

[85] Robert S Huckman, Gary P Pisano, and Liz Kind. Amazon web services. *Harvard Business School Case*, (609-048), 2008.

[86] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3):38–42, 2012.

[87] Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai, and Thomas Sandholm. What's inside the cloud? an architectural map of the cloud landscape. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 23–31. IEEE Computer Society, 2009.

[88] Ronald C Taylor. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC Bioinformatics*, 11(Suppl 12):S1, 2010.

[89] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, Bruce Berriman, Benjamin P Berman, and Phil Maechling. Scientific workflow applications on amazon ec2. In *E-Science Workshops, 2009 5th IEEE International Conference on*, pages 59–66. IEEE, 2009.

[90] Lizhe Wang, Jie Tao, Marcel Kunze, Alvaro Canales Castellanos, David Kramer, and Wolfgang Karl. Scientific cloud computing: Early definition and experience. In *2008 10th IEEE International Conference on High Performance Computing and Communications (HPCC)*, pages 825–830. IEEE, 2008.

[91] Kate Keahey, Renato Figueiredo, Jos Fortes, Tim Freeman, and Maurcio Tsugawa. Science clouds: Early experiences in cloud computing for scientific applications. *Cloud computing and applications*, 2008:825–830, 2008.

[92] G B Berriman, E Deelman, G Juve, M Rynge, and J S Vockler. The application of cloud computing to scientific workflows: a study of cost and performance. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1983):20120066–20120066, December 2012.

[93] Pei Fan, Zhenbang Chen, Ji Wang, Zibin Zheng, and Michael R Lyu. Topology-aware deployment of scientific applications in cloud computing. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 319–326. IEEE, 2012.

[94] Greg Wilson, D A Aruliah, C Titus Brown, Neil P Chue Hong, Matt Davis, Richard T Guy, Steven H D Haddock, Katy Huff, Ian M Mitchell, Mark Plumbley, Ben Waugh, Ethan P White, and Paul Wilson. Best Practices for Scientific Computing. *arXiv.org*, October 2012.

[95] Lavanya Ramakrishnan, Piotr T Zbiegel, Scott Campbell, Rick Bradshaw, Richard Shane Canon, Susan Coghlan, Iwona Sakrejda, Narayan Desai, Tina Declerck, and Anping Liu. Magellan: experiences from a science cloud. In *ScienceCloud '11: Proceedings of the 2nd international workshop on Scientific cloud computing*. ACM Request Permissions, June 2011.

[96] Richard Holland. The Magellan Report for Cloud Computing in Science | Eagle Genomics. *Eaglegenomics*, February 2012.

[97] Jeffrey Dean and Sanjay Ghemawat. MapReduce. *Communications of the ACM*, 51(1):107–113, January 2008.

[98] Jianwu Wang, Daniel Crawl, and Ilkay Altintas. Kepler + Hadoop: a general architecture facilitating data-intensive applications in scientific workflow systems. In *WORKS '09: Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*. ACM Request Permissions, November 2009.

[99] Keith Wiley, Andrew Connolly, Jeff Gardner, S Krughoff, Magdalena Balazinska, Bill Howe, Y Kwon, and Yingyi Bu. Astronomy in the cloud: using mapreduce for image co-addition. *Astronomy*, 123(901):366–380, 2011.

[100] Todd C Klaus, Sean McCauliff, Miles T Cote, Forrest R Girouard, Bill Wohler, Christopher Allen, Christopher Middour, Douglas A Caldwell, and Jon M Jenkins. Kepler science operations center pipeline framework. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 774017–774017. International Society for Optics and Photonics, 2010.

[101] John F Shoch and Jon A Hupp. The "worm" programs—early experience with a distributed computation. *Communications of the ACM*, 25(3):172–180, March 1982.

[102] David P Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, November 2002.

[103] E Korpela, D Werthimer, D Anderson, J Cobb, and M Leboisky. SETI@home-massively distributed computing for SETI. *Computing in Science & Engineering*, 3(1):78–83, 2001.

[104] Adrian Collins, Stephen O'Driscoll, and Niall Smith. A high performance distributed system for astronomical image and data analysis. In *Irish Machine Vision and Image Processing Conference 2006*, pages 191–218, 2006.

[105] Niall Smith, Alan Giltinan, Aidan O'Connor, Stephen O'Driscoll, Adrian Collins, Dylan Loughnan, and Andreas Papageorgiou. Emccd technology in high precision photometry on short timescales. In *High Time Resolution Astrophysics*, pages 257–279. Springer, 2008.

[106] Niall Smith, Alan Giltinan, Aidan O'Connor, Stephen O'Driscoll, Adrian Collins, Dylan Loughnan, and Andreas Papageorgiou. EMCCD Technology in High Precision Photometry on Short Timescales. *High Time Resolution Astrophysics*, 351:257, n/a 2008.

[107] Bram Cohen. The bittorrent protocol specification @ONLINE. `http://www.bittorrent.org/beps/bep_0003.html`, 2008.

[108] Doug Tody. Iraf in the nineties. In *Astronomical Data Analysis Software and Systems II*, volume 52, page 173, 1993.

[109] Amazon Web Servces. Amazon sqs @ONLINE. `http://aws.amazon.com/sqs/`, January 2015.

[110] Paul Doyle, Fred Mtenzi, Niall Smith, Adrian Collins, and Brendan O'Shea. Significantly reducing the processing times of high-speed photometry data sets using a distributed computing model. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 84510C–84510C. International Society for Optics and Photonics, 2012.

[111] FreeNAS. Freenas setup and user guide@ONLINE. `http://doc.freenas.org`, 2015.

[112] Werner Vogels. Eventually consistent. *Communications of the ACM*, 52(1):40–44, 2009.

[113] WD Pence, R Seaman, and RL White. Lossless astronomical image compression and the effects of noise. *Publications of the Astronomical Society of the Pacific*, 121(878):414–427, 2009.

[114] Simson L. Garfinkel. An evaluation of amazon's grid computing services: Ec2, s3, and sqs. Technical report, Center for Research on Computation and Society, 2007.

[115] Alexandru Iosup, Nezih Yigitbasi, and Dick Epema. On the performance variability of production cloud services. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 104–113. IEEE, 2011.

[116] Jing Han, E Haihong, Guan Le, and Jian Du. Survey on nosql database. In *6th international conference on Pervasive computing and applications (ICPCA), 2011*, pages 363–366. IEEE, 2011.

[117] Peter T Gallagher, Y-J Moon, and Haimin Wang. Active-region monitoring and flare forecasting–i. data processing and first results. *Solar Physics*, 209(1):171–183, 2002.