Winter 2015

Hydrographic Data Processing on a Robust, Network-coupled Parallel Cluster

Rohit Venugopal
University of New Hampshire, Durham

HYDROGRAPHIC DATA PROCESSING ON A ROBUST, NETWORK-COUPLED PARALLEL CLUSTER

BY

ROHIT VENUGOPAL

Bachelors in Information Technology, University of Pune, 2008

THESIS

Submitted to the University of New Hampshire

in Partial Fulfillment of

the Requirements for the Degree of

Master of Science

In

Computer Science

December, 2015

This thesis has been examined and approved in partial fulfillment of the requirements of the degree of Masters in Computer Science by:

Thesis Director, Dr. Brian Calder
Associate Director, CCOM
Research Associate Professor, UNH

Dr. Radim Bartos
Associate Professor and Chair,
Department of Computer Science, UNH

Dr. Philip Hatcher
Professor,
Department of Computer Science, UNH

On 12th November 2015

Original approval signatures are on file with the University of New Hampshire Graduate School.

ACKNOWLEDGEMENTS

This work and my study at CCOM/JHC and UNH were made possible through the funding from the NOAA grant NA10NO54000073. I would like to express my greatest gratitude to my thesis advisor, Prof. Brian Calder, who spent many hours helping me understand the research problem, ideate the thesis and guide the solution. My time working with Brian was instrumental in my learning to think critically and present my ideas coherently. His patience and feedback, along with other committee members, Prof. Philip Hatcher and Prof. Radim Bartos, have been invaluable. Guiseppe Masetti, my peer and friend, has discussed my work with me numerous times and asked many right questions. His insights and encouragements were very important to the completion of this work. Finally, I would like to acknowledge my best friend, Yunwen Sun, who greatly strengthened my motivation to bring my research to fruition, in the face of my personal hardships. I am tremendously grateful to her timely and endless help.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

HYDROGRAPHIC DATA PROCESSING ON A ROBUST,

NETWORK-COUPLED PARALLEL CLUSTER

by

Rohit Venugopal

University of New Hampshire, December, 2015

There have been tremendous advances in acoustic sensor technologies and widespread adoption of multibeam echo-sounders in the recent past, which have enabled the efficient collection of large quantities of bathymetric data in every survey. However, timely dissemination of this data to the scientific community has been constrained by the relatively slow progress in the development of new data processing architectures. The current solutions for powerful, efficient and near-real time data processing systems entail high capital investments and technical complexities. Therefore, the installation base for these systems has been very small. The work presented here proposes a new architecture for bathymetric data processing based on parallel computing paradigms. The solution works by distributing the processing workload across a cluster of network-attached compute nodes. The success of using parallel processing for bathymetric data depends on the accurate measurement of the processing workload and its effective distribution across the compute nodes, thereby maximizing speedup and efficiency. These compute resources can be existing installations and other COTS components, such as blade servers, thereby reducing installation and maintenance expenditure.

For workload determination, an estimation algorithm was developed that uses stochastic sampling of the raw bathymetric data file. This produces a low cost and high accuracy estimate of the processing requirements for each line to be processed. This workload information, coupled with file and system metadata, is used as input to different load balancing algorithms - First Come First Served (FCFS), Longest Job First (LJF) and Contention-Reduction (CR). The performance of FCFS and LJF algorithms is highly dependent on the characteristics of the input dataset while CR scheduling aims to characterize the input and adjust load distribution for the best combination of speedup and efficiency. The choice of these algorithms depends on the requirements of the installation, i.e. prioritization of speedup or efficiency. To ensure robustness, watchdog mechanisms monitor the state of all the components of the processing system and can react to system faults and failures, through a combination of automated and manual techniques. Although not part of the current implementation, there is potential for adding redundant critical components and to enable live-failover, thereby reducing or eliminating system downtime.

The methods for workload estimation and distribution are templates for extending this framework to include additional types of bathymetric data and develop flexible, self-learning algorithms to deal with diverse datasets. This research lays the groundwork for the design of a ship-based system that would enable near-real time data processing and result in a faster ping-to-chart solution.

CHAPTER 1

INTRODUCTION

1.1 Background of the Problem

Bathymetry is the study of ocean depth with respect to sea level or some other reference surface; hydrography is the use of bathymetry and other information in support of safe navigation, usually of surface ships. Although traditionally the study of bathymetry was concerned with measuring critical depths, today it is also concerned with the shape of underwater terrain. Hence, bathymetry can be defined as the study of submarine topography.

One of the main purposes for the collection of bathymetric data is for the generation of nautical charts. These nautical charts are used by mariners and are a crucial resource for safe navigation (Figure 1). Besides navigation, bathymetric data has wide applicability, for example in understanding the impact of climate change and man-made disasters, and the study of aquatic life.

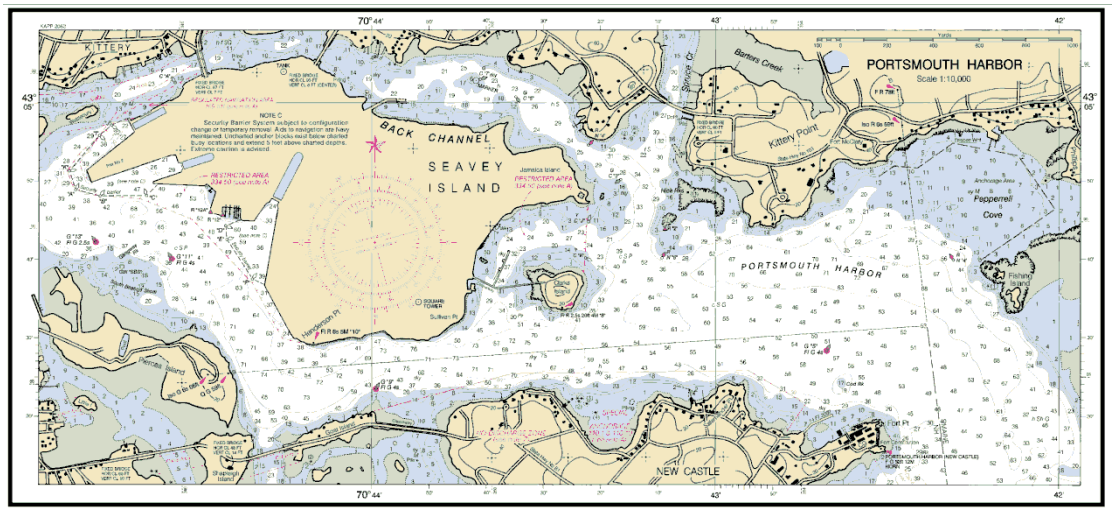

Figure 1: Nautical Chart of Portsmouth Harbor, NH (www.noaa.gov)

Since, historically, bathymetry was only concerned with the measurement of the critical depth of a water body, early devices that were used for the collection of bathymetric data only measured depth. One of the earliest instruments used for the collection of bathymetric data was a lead line. Lead lines are ropes, or lines, with graduated depth-markings and a lead weight attached to the end. Early hydrographic surveys involved measuring depths using a hand-held lead line with positions determined by three-point sextant fixes to mapped reference points. The use of a lead line involved an individual heaving it into the water, ahead of the moving vessel, as precisely as possible such that the lead line would become vertical within the water when the vessel reached the location (Figure 2). The lead line was allowed to sink till it touched the floor of the water body and using graduations on the line, the depth was determined. However, collecting hydrographic data by using lead lines was labor intensive and time consuming. Although the measurements were generally accurate, there was missing information between the points at which the readings were taken. Also, there was a limit to the depth at which this method could be applied. This was, under average conditions, 10 fathoms (18.29 m) for smaller vessels and 15 fathoms (27.43 m) for ships or auxiliary vessels [1].

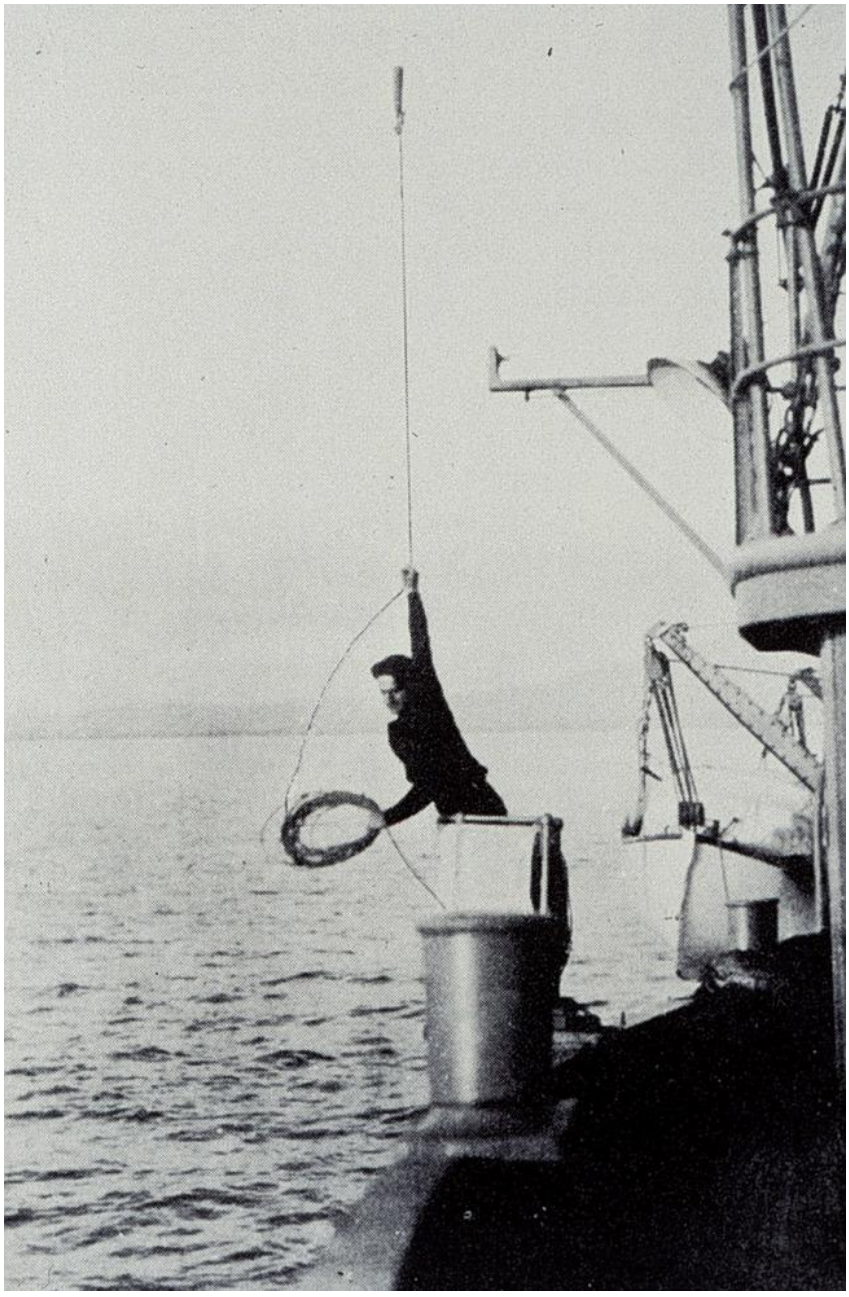

Figure 2: Use of a lead line (www.noaa.gov)

Technology has come a long way since the use of a lead line. Currently, a device called a Multibeam Echo Sounder (MBES) is most often used in the US for the collection of bathymetric data. The adoption of the MBES is increasing around the world [2]. The MBES works by transmitting a broad acoustic pulse through the water orthogonal to the direction of travel, waiting for the sound to bounce off the seafloor (or any other obstruction in the water) and return to the receiver acoustic array. The sonar fundamentally measures the two way

travel time between transmitting the acoustic energy and receiving the return. It uses phased array techniques in the receive elements to form multiple receive beams (often on the scale of 100 – 500 simultaneous beams) and measure travel time in each beam simultaneously [3]. If the speed of the sound is known for the entire water column as a function of depth, then the sonar can combine the travel time measurement, the angle of the receive beam and the speed of sound to determine the likely location of the acoustic echo relative to the transducer; combined with a positioning and attitude system, these locations can be translated into a position with respect to the earth.

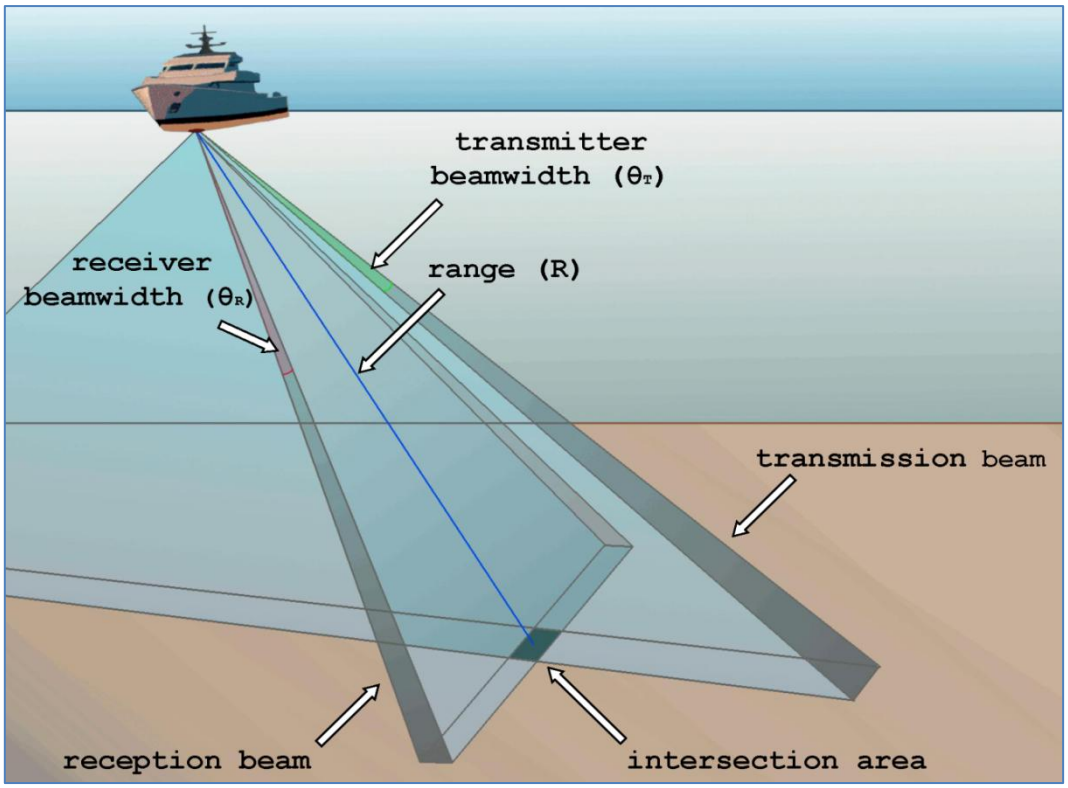

Figure 3: Principles of operation of a MBES [4]

Use of acoustic methods has resulted in greater coverage, and increased efficiency with respect to time and labor. Also, bathymetry has come to include the features of the underwater terrain in addition to the primary task of measurement of critical depth (Figure 4).

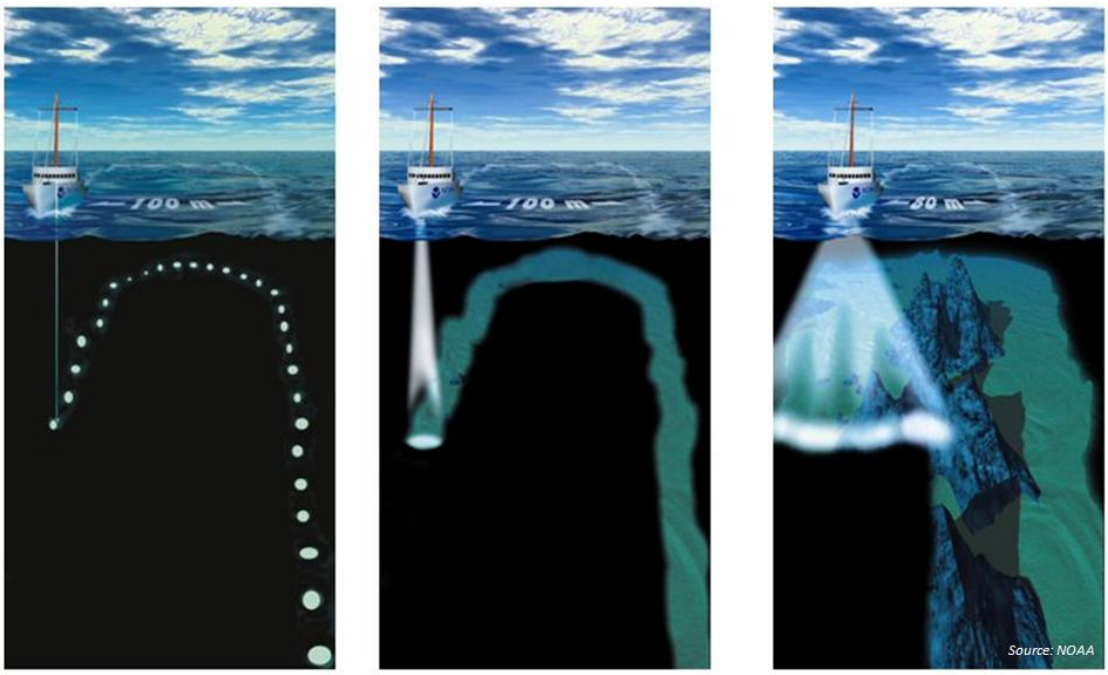

Figure 4: Evolution of Bathymetric Data Collection (www.noaa.gov)

The caveat is the significant increase in the volume of data being collected, and as higher resolution MBES have become standard on most survey cruises, more data is being collected per cruise than ever before. The jump in the data volume is very prominent as depicted in Figure 5. This graph illustrates the effect of the adoption of MBES into the survey fleet of the Naval Oceanographic Office (NAVOCEANO) on the volume of data being collected. It depicts the increase in bathymetric data volume for different generations of MBES systems (oldest to youngest, and deepest to shallowest from left to right on the legend). If this graph were to be brought up to date, we would expect to see a similar 'jump' between the years 2004 and 2012, owing to the improvements in high resolution MBES and adoption of newer technologies.

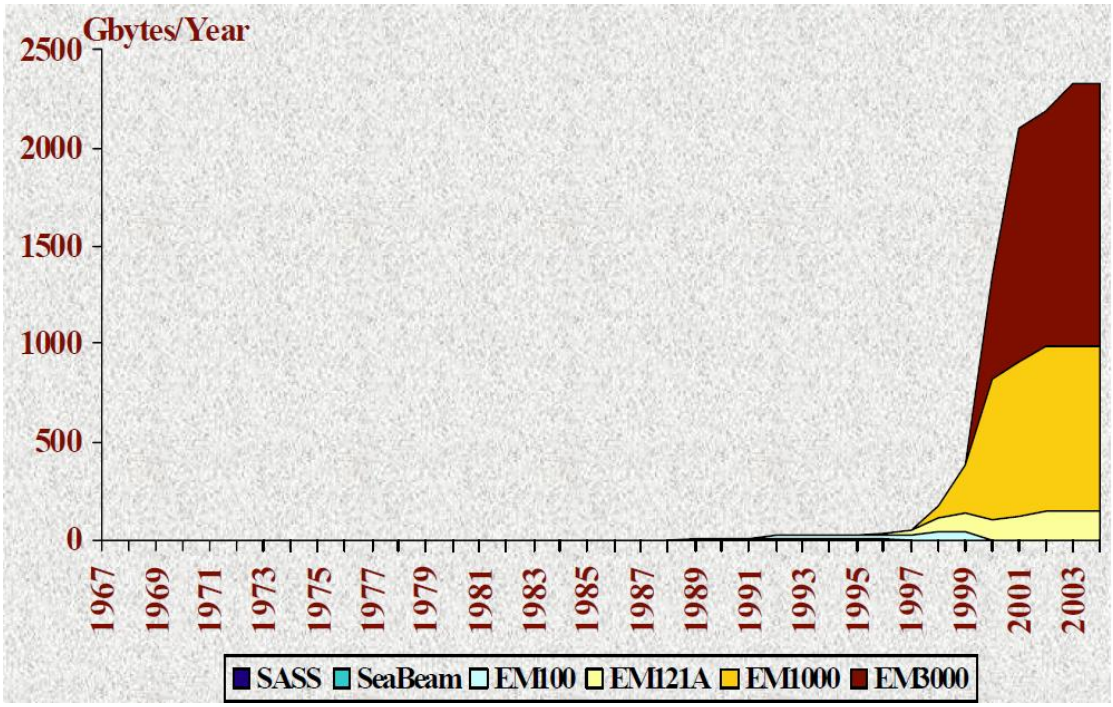

Figure 5: Increase in Bathymetric Data Volume[3]

1.2 Problem Definition

The data collected by a MBES is written onto a data-storage medium, on the survey vessel, in a binary format. This binary bathymetric data has to undergo a series of transformations before it reaches a stage where it is comprehensible to the end user. Typically, the processing of bathymetric data is performed on workstations, one file at a time, i.e., serially. This conversion of raw data to the final product is a resource intensive process in terms of I/O, network bandwidth and processor cycles. In the current scenario, there is limited data processing capacity on board the survey vessels. Adding processing capacity, in the form of workstations, to deal with increasing amounts of raw data is non-trivial because of the limited resources (particularly human resources) and space constraints on board the survey vessels.

With modern shallow water systems running at up to 9600 soundings/second, data collection at the rate of approximately 250 million soundings/day/system is possible[5]. With development of new techniques of hydrographic mapping, the amount of data collected is

expected to increase. Improved and affordable technology will result in more efficient survey activities that would bring back more data per cruise than ever before. This necessitates the development of a faster data processing system that can handle increasing amounts of data efficiently.

Near-real time processing is the capability to process the raw bathymetric data at or very close to the time it is collected. This requires a faster and more efficient data processing infrastructure that can match the capabilities of the data collection system. With survey cruises costing hundreds of thousands of dollars and the massive amounts of data collected (on the order of several hundred gigabytes), any activity that allows streamlining or improving the efficiency of the data collection and processing activity would lead to significant monetary gains/savings.

Survey vessels plan their track lines based on an area/object of interest. If the data collected by the survey vessel can be analyzed in real time or in a timely manner, it would help the surveyors assess the quality of the survey and identify new objects of interest. The significance of having a near real-time processing capability results from the fact that the cost of surveying/resurveying an area increases by orders of magnitude as the survey vessel moves away from the area of interest. Hence, the goal is to deal with survey quality issues while the survey vessel is within the area of the survey.

It is also imperative that the data collected be processed in a timely manner such that it can be distributed to and assimilated by the scientific and nautical community and benefit all the stakeholders involved. A faster and more efficient data processing system presents the opportunity to experiment with the transformation algorithms, e.g., parameter tuning would be more efficient as the response of the algorithms to user input would be quicker, enabling a better feedback loop. Opportunities such as these, in turn, would lead to development of new techniques and faster algorithms. These factors would lead to greater return on investments

for the survey activities. Hence there is an urgent need for new developments in the field of hydrographic data processing if the pace of the advances in hydrographic exploration is to be maintained.

The next generation hydrographic data processing system must be

- Faster – The increased throughput would facilitate more rapid processing of the ever increasing volume of data and help attain near-real time processing capabilities.
- Scalable – To overcome the constraints of the workstation-based processing paradigm and move towards a solution where processing capacity can be adjusted to meet demands of the data acquisition system and the data processor.
- Robust – Data processing systems on board survey vessels need to be fault tolerant and easily recoverable from failures, as there are limited repair capabilities and technical expertise available when the survey vessel is out at sea.

1.3 Previous Research

The workstation-based model for bathymetric data conversion and limited resources on board survey vessels have made it challenging to implement near-real time data processing solutions which are fast and cost-effective. One solution to this issue is to utilize a network connection between the survey vessel and a data processing centre. The core concept in this scheme is to have the data collected by the survey vessel sent to an onshore data processing centre using a telecommunications link which, in most cases, would be a satellite link due to the remoteness of the area at which the survey activities are performed. The processing centre would analyze the data and report on any issues with the surveying operations and provide the necessary instructions. The main adopter of this scheme has been NAVOCEANO, located at the Stennis Space Center, Mississippi. The idea was to establish a Survey Operations Center (SOC) that would exercise technical control on the Navy's fleet of

survey vessels that have been forward-deployed all over the world [6]. A combination of International Telecom Satellite (INTELSAT) and commercial service providers in C (4-8GHz) and K_u (12-18GHz) bands were chosen to provide the network link between the survey vessels and the SOC [7]. It enabled access to survey systems, data file transfers, video teleconference and other communications by using an asymmetrical (2048 kbps from ships, 384 kbps to ships) satellite communications link. However, the primary role of this system was in monitoring the data that affected the quality of the survey (e.g., sound speed profile, water column data, etc.) thereby reducing the cost of potentially having to resurvey the entire area at a later time. Although raw bathymetric data was also sent back to the SOC, it was of low priority due to the tremendous volume of the files. There have been several claimed benefits of the establishment of the SOC, most notably being able to monitor survey quality within 24 hours of the data collection as compared to the previous one to two month scenario. Configurable access to the ship's sensors is also claimed to help maintain survey quality, reduce ship down-time, and in most cases eliminate engineer travel costs associated with rectifying these issues [8]. Financially, the initial investment and maintenance costs are very high. In addition, one also needs to factor in the costs that are associated with employing the necessary technical expertise for maintaining the communication equipment on board each survey vessel and at the SOC. Hence only large entities, such as the US government, with important tasks like mapping strategic areas to aid military activities, could justify this kind of expenditure.

The solution engineered by NAVOCEANO suffers from the delay in data processing due to time required to transfer the huge amount of data from the survey vessels to the onshore data processing center. Another idea, focused on avoiding this heavy data transfer and its associated costs, is to move only the data analysts onshore. Hare *et al*. [9], at the Canadian Hydrographic Service, expanded on the work of Peyton *et al*. [10], who had been

processing hydrographic data by logging onto a centralized server from a shore-based remote location. The focus of their effort was to develop a near-real time processing system by improved utilization of human resources and assets, while keeping the costs minimal. The solution implemented consisted of a processing system (centralized server) on the survey vessel, which was controlled by data analysts who were based onshore. The communication, carried over a cellular data network, was highly optimized to conserve bandwidth. The issue with this solution, which centered on using the remote-desktop paradigm, was that it was heavily dependent on a reliable form of communication between the vessel and the data processing center. As one moves to more remote locations, the cost of establishing a communication link (via satellite) to the shore rises exponentially. Furthermore, this method only moves the individual processing of the data from the vessel to an onshore location and hence performance is still limited by on board processing capacity.

Both of the above solutions aim at improving the collection of bathymetric data by implementing near-real time processing mechanisms. The underlying concept behind both these solutions is to overcome limitations in on board processing capacity by offloading the transformation tasks and/or the data analyst to an onshore location Although NAVOCEANO has adopted their system, it is likely that the tremendous costs (on the order of millions of dollars) incurred in setting up and maintaining such a system is prohibitively expensive for commercial survey entities. This is probably the most important factor that has prevented the widespread adoption of such systems. The solution pioneered by Peyton *et al*. and Hare *et al*. focuses on improving the effectiveness of the data analyst but does not improve or enhance the utilization of the on board processing capacity and suffers from range limitations if costs were to be controlled.

With the advent of affordable and reliable multi-core and multi-processor desktop architectures, there have been various attempts to implement multi-threaded versions of

common hydrographic processing algorithms. To some extent, these have been successful in increasing the throughput of traditional data processing schemes[11]. However, a single system poses many constraints, like memory, disk and network bandwidth, that limit the scalability of these multi-threaded solutions. Hence, the gain from these solutions has been limited.

There has been very limited research into improving the hardware and software capabilities of the current processing infrastructure. Most of the solutions try to bypass this processing bottleneck by offloading processing tasks to more powerful onshore facilities. There is tremendous potential to incorporate the advances in computational technologies to develop the next generation processing platform that is faster, more efficient, robust and flexible for implementation on the survey vessel itself.

1.4 Thesis Statement

The aim of this work was to devise a processing scheme for raw bathymetric data that would fulfill the requirements described in the problem definition. The core concept of the work was based on parallel processing, which led to the following thesis statement.

There is a way to parallelize the processing of bathymetric data (on a network-coupled cluster of nodes) such that the distribution of workload across the computational resources is balanced and the system exhibits high efficiency and throughput, scalability, and fairness in job distribution, and demonstrates robustness against failure.

The application of parallel processing allows implementation of a data processing system that overcome the limitations of the workstation-based processing paradigm and allow for a scalable processing resource, where capacity (in the form of nodes) can be added or removed depending on the requirements. Robustness is also ensured as the failure of a particular processing node does not bring down the entire processing framework and faulty nodes can be easily replaced. These characteristics mean that a fast data processing platform can be implemented on the survey vessel itself rather than transferring data to an onshore data processing site. This addresses the cost and coverage limitations experienced in the solutions from NAVOCEANO and Hare *et al*. It also allows for a more efficient near-real time processing capability, with a faster turn-around time as a faster data processing capacity is located on the survey vessel itself.

1.5 Structure of a Parallel Solution

Raw bathymetric data has to be processed into a format that is easy to investigate, manipulate and transform. There are many software packages, provided by corporations like QPS, CARIS, HyPack, and others, that aid in the transformation of raw data to processed data, but the conversion in each is essentially the same. These conversions are a sequence of steps in which the raw bathymetric file is converted from one form to another, each step creating an intermediate result that is used as input for the next step of the processing (Figure 6). Here SSP is the Sound Speed Profile and TPU is the Total Propagated Uncertainty.

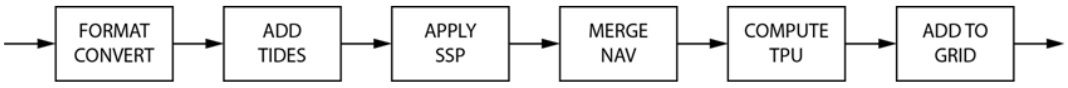

Figure 6: Basic Processing Flow

In this processing flow of raw bathymetric data, each binary data file, when provided with ancillary information like sound speed profile, tidal data, etc., proceeds independently of

other binary data files. This absence of interdependencies between different raw data files makes the processing chain an ideal candidate for parallelization.

With the advent of affordable, high performance, parallel processing architectures, the trend of having computationally intensive threads of execution (tasks) split up amongst various computational resources has become the norm. Here, a thread of execution is called a *task* and a *collection of tasks* will be referred to as a *job*. An algorithm that takes advantage of parallelism can benefit from (a best-case of) a super-linear increase in throughput, i.e., a speedup of greater than *n*, for a parallel cluster of *n* computational resources, due to greater memory availability and bigger composite caches across multiple processors[12].

The first stages of the bathymetric processing chain are clearly embarrassingly parallel. Embarrassingly parallel applications benefit the most from the parallel computing architecture because there are no inter-dependencies between the computational tasks which eliminates the wait time as the processing moves from one stage to another.

Based on the concept of parallel processing, the following system was designed.

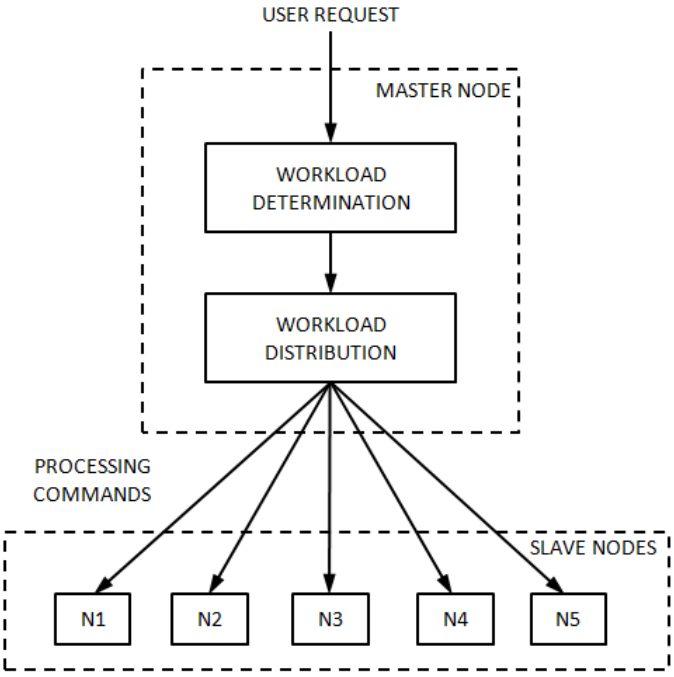

Figure 7: Block Diagram for Parallel Processing System

The above diagram is a simplified representation of the parallel data processing system that was developed as part of this thesis. A detailed explanation of the technical aspects of this work is provided in Chapter 2 of this document. It covers the different technologies used to implement this model and the reasons for choosing them. It is important to emphasize that the system design is a proof of concept, and not a specific solution, that demonstrated the performance of various algorithms under different conditions. Hence, not all the components of a complete processing system, e.g., a fully functional user interface, porting the code to a parallel cluster, etc. were implemented, since these were not primary areas of interest in this research.

In the first module of the system, the user request for processing files is analyzed and the processing workload involved in transformation of the file to the final product is estimated. For this research, the examples were limited to use of Extended Triton Format (XTF) files. XTF, although proprietary, is a common format used for collection of raw bathymetric data and many MBES manufacturers provide software to directly record in or convert their own proprietary format to the XTF format. The general mechanism of recording raw data is very similar across different formats however and hence the solution here can be used as a framework for the design and implementation of systems that deal with other types of raw bathymetric data.

Analysis of the processing chain revealed that each raw data packet undergoes similar transformations as it is converted to the final product and hence the number of bathymetric packets contained in the file could be used as a workload metric. For reasons of efficiency and speed, the number of bathymetric packets is estimated rather than being computed directly. After experimenting with various algorithms, the data show that stochastic sampling provides a reasonably accurate and consistent estimate of the number of bathymetric packets in the file. Chapter 3 provides complete details regarding the work in this area.

Workload estimates from the previous module are utilized for load balancing and task scheduling purposes. Because this implementation is not a fully functional processing system, the slave nodes simulate the processing of data files. The system maintains statistics regarding the processing tasks assigned to each node and depending on factors such as current processing load, outstanding requests, node performance, etc., decides on the best available node for each task submission, aiming to increase throughput while maintaining fairness. Chapter 4 discusses in detail the various experiments that were performed, their results and the scheduling algorithms that could be utilized depending on the needs of the installation.

The final goal of the proposed system is to be deployed on board survey vessels. Limited on board technical expertise during the cruise means that the system has to be robust. It should be able to tolerate faults and recover from failures, with minimal or no human intervention. Chapter 5 discusses identification of various points of failure within the proposed system and investigates the solutions for the same. These include backing up of critical data structures and databases, and the capability of the system to detect node failure and offload pending tasks to the remaining nodes.

The proposed system is a proof of concept and the aim is to utilize the latest developments in computer technologies to improve the data processing activities. Chapter 6 elaborates on the findings of the work presented here. It also discusses the various methods by which this work could be extended and utilized as a guide for processing various other types of data files and implementing a fully functional processing system.

1.6 Contributions

This work aims to integrate advances in computation technologies with the hydrographic sciences. A data processing system comprising of a network-coupled cluster of

nodes will allow the development of a faster, more efficient, robust, and near real-time data processing system. This work investigates the challenges faced in incorporating the parallel processing paradigm into bathymetric data processing and provides solutions for the same. The work shows that it is possible to reliably determine the processing workload of a raw bathymetric data file through a fast and efficient stochastic estimation algorithm. It also tackles the problem of distribution of processing tasks across nodes and proves that this can be done in a way that maintains throughput and avoids starvation. Robustness is also ensured through strategies like critical data replication and node fault tolerance capabilities. The hydrographic community can utilize this work as a framework for development of an on board data processing system. This advanced data processing and transformation capability would help make survey activities more efficient, reduce expensive resurvey cost overheads and help the data processing technology keep pace with, and spur advances in, sensor technologies and data processing algorithms.

CHAPTER 2

SYSTEM DESIGN

The core idea behind this work is to integrate the parallel processing paradigm and enhance the existing method of transformation of raw bathymetric data. This improved processing framework consists of two stages:

- Job scheduling and load balancing of tasks among the computational resources
- Conversion (processing) of the raw bathymetric data files

The research, as part of this thesis, deals with the first stage. This includes workload determination, distribution of workload among the available computational resources using various algorithms, and issues regarding robustness of the system. The second stage deals with the transformation of the raw bathymetric data and includes the API from vendors such as CARIS.

2.1 Design Considerations

Designing a parallel processing system for bathymetric data, based on the requirements stated previously, meant that there were some questions to be answered. The main idea behind the use of parallel processing is the division of workload among the processing nodes, with the aim of increasing the overall throughput of the system. Hence, for efficient use of parallelization, analysis of the raw bathymetric files and determination of a metric, which could be used as an indicator of the amount of processing for transformation, was required. Based on the metric, it would then be possible to determine workload for all

processing tasks submitted to the system. With this information, it becomes possible to distribute the workload among the computational resources so as to ensure optimal distribution, avoid starvation and increase efficiency.

Due to the fact that survey vessels spend most of the time out at sea, maintaining a team of technical professionals to operate a specialist parallel machine would mean a very significant monetary and space overhead. Having such a contingency group would not only put a strain on the vessel's resources but also they would replace other, and probably more useful, scientists on the survey expedition (owing to the limited capacity on board). Also, network bandwidth limitations on board the survey vessels severely reduce the feasibility of having remote diagnostics rectify any major issues. Hence, it was imperative that the system designed be fault tolerant and resistant to failures. Even in the unlikely event of a system failure, the recovery process should be simple enough that a person with minimal training be able to perform it. Hence, robustness is another key aspect that the solution incorporates.

Since the architecture proposed here is a network-coupled cluster, the delay encountered when all the computational resources access a shared resource, e.g., network attached storage, etc. is an essential consideration [13]. The effect of these resource contention problems are varied and depend on the nature of the computational task, frequency of contention, and other factors that are unique to the problem being solved. Hence, the effects of parallel access to shared resources also influenced the design of the parallel processing system.

2.2 Component Layout

Figure 8 represents the layout of various components in the parallel processing system.

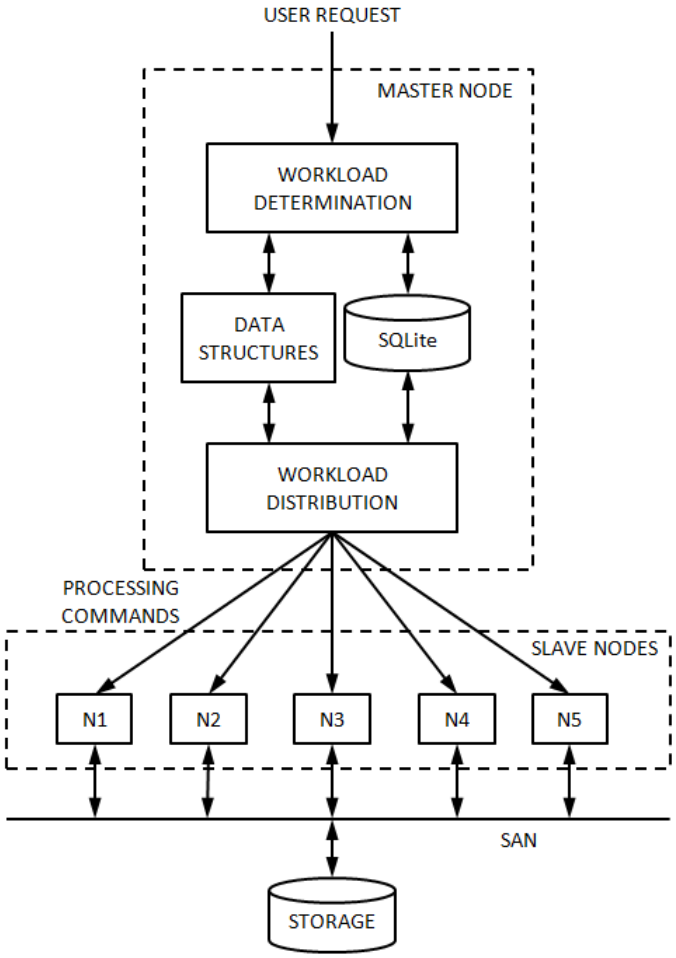

Figure 8: System Diagram

As explained in the previous chapter, the user request for processing raw bathymetric data files is received by the workload determination module. The processing load is determined for each file (task). This information, in combination with data regarding the nodes' current outstanding requests, performance, etc., is used by the workload distribution module, to determine the appropriate node to which the processing task is to be dispatched.

All modules in the proposed system are multi-threaded and inter-thread communication is through shared memory, in the form of data structures. The entire code base is written in C. The reasons for the choice of C was familiarity, comfort with programming in C and its interfaces with SQLite, which was used for managing persistent data. SQLite was chosen for its zero-configuration installation, server-less structure and

multiple access support, and simplicity. All the features of a complex database system were not required and the functions provided by SQLite fit in with the requirements. However, it is important to note that, if need be, it would be fairly straightforward to adopt any other database system since all the queries used are standard SQL. Speed and efficiency are addressed by using threads and eliminating any inter-process communication. Robustness is ensured by backing up the data structure onto disk and also by creating a backup copy of the database on a separate disk. Finally, the system was designed such that all raw bathymetric data, to be accessed by the nodes, resided on the SAN.

The following chapters delve into details of the different components that have been specified in the system design diagram.

CHAPTER 3

WORKLOAD DETERMINATION

The primary requirement for the operation of the Job Scheduling/Load Balancing sub-system was to establish a metric that could be used to determine the amount of processing required for each raw bathymetric file. Workload quantification enables determination of resource requirements, which is the central factor for load balancing. This allows the code to allocate tasks on the target nodes, and control and improve job throughput and resource utilization[14]. Balancing the processing workload across the available compute nodes facilitates efficient resource management. Determination of the workload requires analysis of the structure of the raw bathymetric file and determination of the factors that influence the data size, data distribution and packet types.

3.1 Raw Bathymetric Data Formats

3.1.1 Data Recording and File Structure

There are a number of formats that allow the recording of raw bathymetric data. Depending on the model of the MBES, its settings and type of information being collected, data can be recorded in one of the formats prescribed by the manufacturer, e.g., Triton (.XTF), Kongsberg (.ALL), Reson (.PDS, .S7K), HyPack (.HPX), etc. Each format has its own standards that prescribe the file headers and their sizes, data packets identifiers, relative positions of data, and so on. Although the formats vary, the general mechanism of writing

raw bathymetric data into files is very similar. The data is added to the file at any time without needing to synchronize the data packets. This type of asynchronous data collection means that storage space is utilized effectively and no “holes” are created in the saved data stream. Hence, the file can be considered to be a “pool” of data.

However, due to the asynchronous nature of data collection, there is no way to determine the byte offset of a particular packet or calculate the number of data packets within the file without performing a complete read[15]. Depending on factors such as, for instance, the depth, the number of sensors, packet types, model of the MBES and survey settings; the size of the resulting raw data file, the number of packets in the file and the packet density can vary. The file size of raw data files can vary from a few kilobytes (kB) to a few gigabytes (GB). It is important to note that due to the asynchronous nature of data collection, settings on the MBES and user preference, the size of a data packet can vary, and hence the size of a raw data file has a complex relation with the number of packets contained within that file. As mentioned before, depending on the file format, there are standards (headers containing metadata for each packet type) which are followed while writing data asynchronously to the raw data file. Having such a mechanism greatly aids the processing of raw bathymetric data, as the processing algorithms can use a fairly deterministic approach even when dealing with non-deterministic, raw data. The raw data formats are designed in such a way that, when supplied with the ancillary information like the sound speed profile, tidal information and others, a single file contains all the necessary information required for its transformation to processed data. There are no inter-line dependencies and hence each file can be processed independently.

3.1.2 Packet Count Metric

During the transformation to the final product, the data packets are processed from one stage to another, where the data from the previous stage acts as the input for the next. The processing path for MBES data applies, generally, the same processing to the data packets in the raw bathymetric file. Although there might be a slight variation in the processing time for packets of different sizes, the time required to process a data packet is fairly constant for a particular stage in the processing chain. This implies that the time required to process a data packet through the processing chain is also fairly constant. Hence, the number of raw data packets within a bathymetric file would indicate the time required for processing that particular file and therefore can be used as an approximate measure of the workload required for processing.

3.1.3 XTF Format

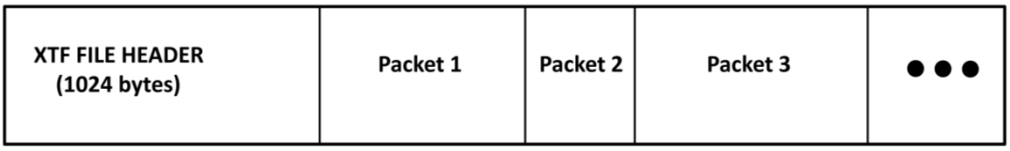

Figure 9: XTF File Structure

Like other formats that record raw bathymetric data, XTF files record data asynchronously. An XTF file begins with an XTF header and is followed by one or more data packets (Figure 9)[15]. The header does not store any information about the number or size of packets to follow. This implies that the number of data packets cannot be determined without a complete read of the file. An end-to-end analysis of the file for the purpose of counting the number of bathymetric packets will provide accurate metadata that will help with very precise load balancing amongst the compute nodes. However, this task is resource intensive because the file will have to be pulled into cache and main memory from the disk or over the

network from the SAN (depending on where the file resides) only to be swapped out once the reading process is complete. This would have resulted in having to read the entire file, byte by byte, in at least two separate instances, the first being for workload determination and the second for the actual processing. To ensure that processing capacity is utilized for data transformation rather than pre-processing, various more resource efficient means of determining the number of data packets within the file were investigated. Since counting the number of data packets is only possible by performing a complete read, the work focused on ways to estimate the number of packets contained in an XTF file.

3.2 Concept of Workload Estimation

There are multiple methods to estimate the number of data packets in the raw bathymetric file, all of which revolve around sampling a small part of the file and determining the number of bathymetric data packets in that section. Using this density (bathymetric packets per unit size), the number of data packets in the whole file can be estimated through extrapolation. Three methods were examined.

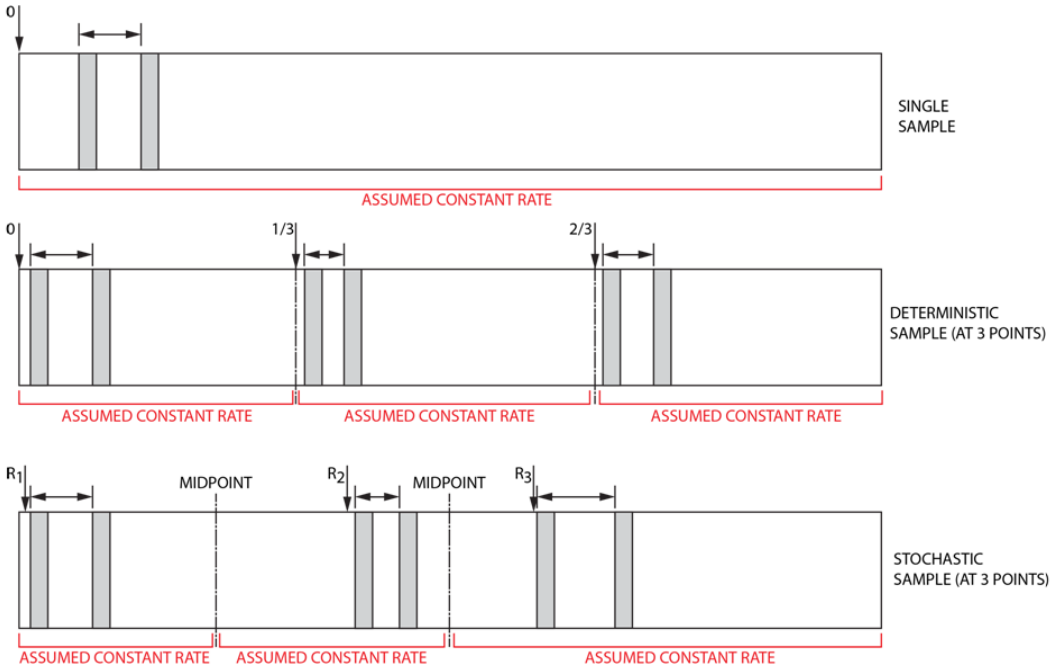

Figure 10: XTF Sampling Methods

In the deterministic approach, the file is sampled at pre-determined indices (within the file) and the results from each sample are used to estimate the number of bathymetric packets in the file. The indices for sampling in the deterministic approach could be packets from the start of the file or indices that represent any pre-determined position into the file which could provide good samples, suitably representing the density of data packets in the file (Figure 10). Even within a raw XTF file, there is a significant variation in the organization of data packets. Factors like, for instance, type of data being collected, the depth of the water column being surveyed, presence of ancillary data between bathymetric data packets, etc., result in inconsistent density of bathymetric packets in the raw file. To compensate for this non-determinism, in the stochastic approach, the indices are randomly generated and the file is sampled at these random indices (Figure 10).

The choice of either of the two approaches (deterministic or stochastic) for providing estimates depends on the accuracy of the estimate and the cost of providing the estimate. The time required for generation of estimates will also depend on the number of samples used, since it directly influences the number of file blocks read from disk and transferred over the network. A larger number of samples will lead to more accurate estimates, but also higher computational time, I/O and network delay. It had to be determined if the accuracy of the estimate increased significantly to justify the additional delay and processing cost. A cost-benefit analysis helped in choosing the best method for the generation of the estimates of the data packet count. Additionally, a very high degree of accuracy in the estimation of the bathymetric packets (workload) is not required so long as over a large number of estimates, the mean of the errors would be zero (or close to zero). Since estimates are bipolar, this is likely the case everywhere, and is borne out by the experiments reported in section 3.4.

3.3 Pre-Experiment Calibration

To determine which of the proposed estimation algorithms could be used as part of the system, the algorithms had to be compared on the basis of the accuracy of the estimates and the processing costs for generating those estimates. The calibration for these tests involved selection of datasets, determination of true count of bathymetric packets in the files of the datasets and a scheme to measure the processing required for estimate generation.

For the purpose of these experiments two distinct datasets were considered. The first dataset (DS1) originates from a survey conducted by the NOAA Ship *Fairweather*, in Ernest Sound, Alaska, USA and consists of 84 raw data files with a total size of approximately 100 GB. The second data set (DS2) is the result of a survey done by R2Sonic in and around Wellington, New Zealand, in support of the 6th International Conference on High Resolution Surveys in Shallow Water (Shallow Survey 2012). This dataset has 224 raw data files with a total size of 22.5 GB. These datasets were chosen as they provide good variation in the number of files, size of the individual files and density of bathymetric packets within the files.

The true count of packets in each file was determined by skipping over the file header and channel information structures and counting the number of bathymetric packets in the payload section. This algorithm was implemented in C and was run against the files of DS1 and DS2. The filenames and the corresponding count of bathymetric packets were recorded, which would be used for determining accuracy of the estimation methods.

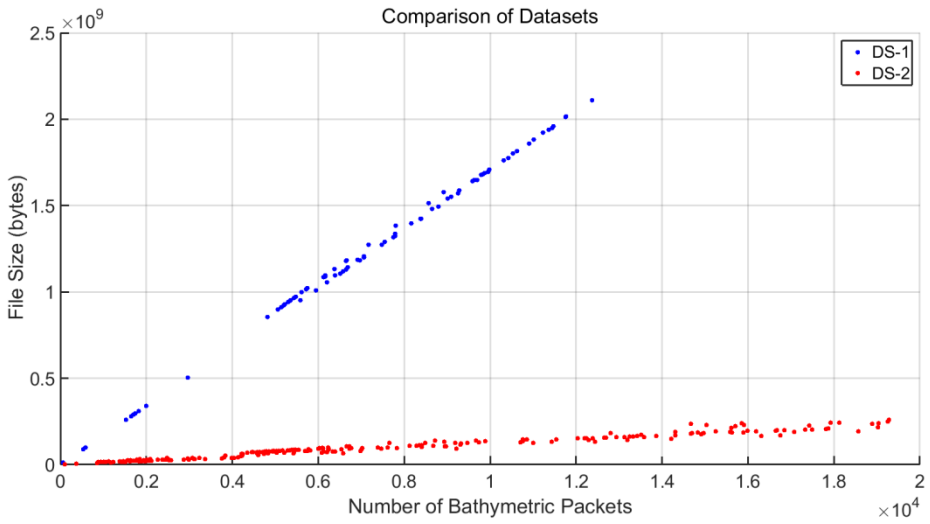

Figure 11: Comparison of DS-1 and DS-2

Figure 11 depicts the differences in characteristics of DS1 and DS2 in terms of the file size and the number of bathymetric packets contained within each file. In general, the files in DS1 exhibit a fairly linear increase in the number of bathymetric packets as the size of the file increases. DS2 also depicts a linear pattern with a higher rate of increase in packet count with respect to file size. Additionally, the density of packets in the files of DS1 is much lower as compared to the files in DS2. Both these observations point to a complex relationship between factors like file size, number of data packets and packet density. These indicate that the settings of the MBES or characteristics of the surveyed area have a critical effect on the structure and contents of the raw bathymetric data file. This results in a complex relationship between the file size and the number of bathymetric packets, and this relationship could vary between datasets.

Since the time required to generate the estimate is a reasonable indicator of the processing required, estimation time can be used as one of the metrics for comparison of the various estimation algorithms. For determining this estimation time, Windows high resolution timer functions were used. These were

- QueryPerformanceCounter – Used to count the number of current elapsed ticks

- QueryPerformanceFrequency – Used to count the number of the ticks per second, which is used to convert ticks to wall-clock time

3.4 Estimation Experiments

The goal of the estimation experiments was to determine the accuracy of each of the proposed estimation algorithms. Each algorithm was tested using both datasets (DS1 and DS2) so as to analyze their performance on bathymetric files that exhibit diverse characteristics. For these experiments, the datasets were stored on the local disk of the simulator because the focus of the experiments was to determine estimation accuracy and measure estimation time. Given the true count of the bathymetric packets in each file, the error in the estimates was calculated. Estimation time for the algorithms was calculated using the, previously described, Windows high resolution timer functions.

In the following sections, for each method, a description of the algorithm used is provided, followed by the observations and overall analysis.

3.4.1 Deterministic – 1 Sample

In this method, a single sample was collected from the beginning of the raw bathymetric file, which represented the distance between two consecutive bathymetry packets. The underlying assumption was that the density of bathymetric packets remains constant throughout the file.

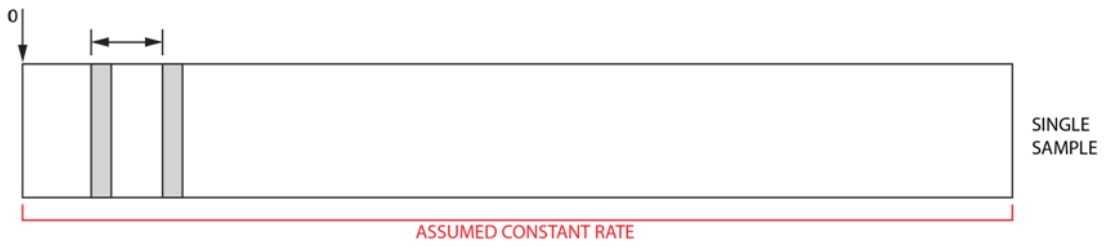

Figure 12: Deterministic Method - 1 Sample

The following algorithm was used for this method:

1 Open the file to be analyzed, skip the file header and any existing channel information structures.

2 Read the data packets, one at a time. If it is a bathymetric data packet, start recording the number of bytes between this bathymetric packet and the next bathymetric packet.

3 Use the distance between the two consecutive data packets and the payload size in the file to calculate the number of bathymetric packets.

By running this algorithm over the files in DS1 and DS2, the number of bathymetric packets within each file was estimated. Using the true count of the packets, obtained from the pre-experiment calibration, the percentage error in the estimation in each of the estimates was determined.

For DS1, mean of the percentage errors over the whole dataset was -0.01% and standard deviation was 0.21%. The mean estimation time was 24.22 milliseconds (Figure 13).

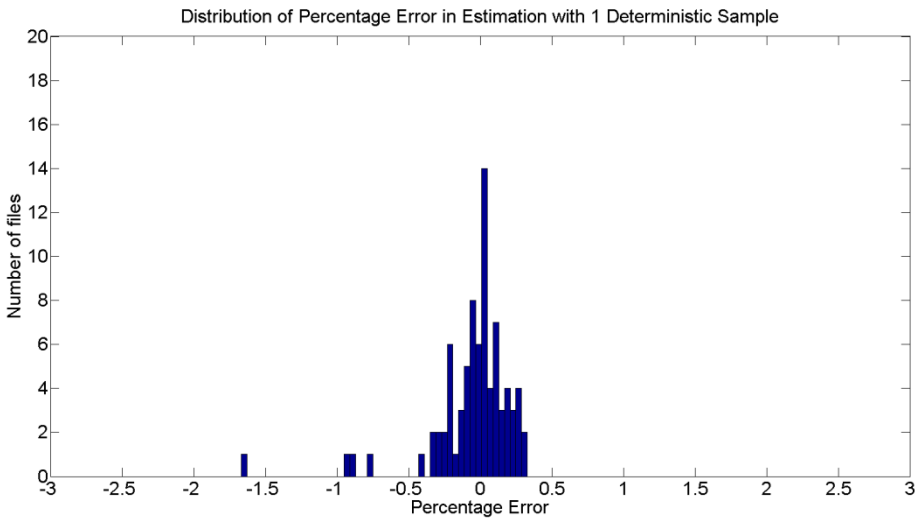

Figure 13: Error Distribution with 1 Deterministic Read, for DS1

For DS2, the mean of the percentage errors over the files of the dataset was 8.12% with a standard deviation of 18.85%. The mean estimation time was 16.94 milliseconds (Figure 14).

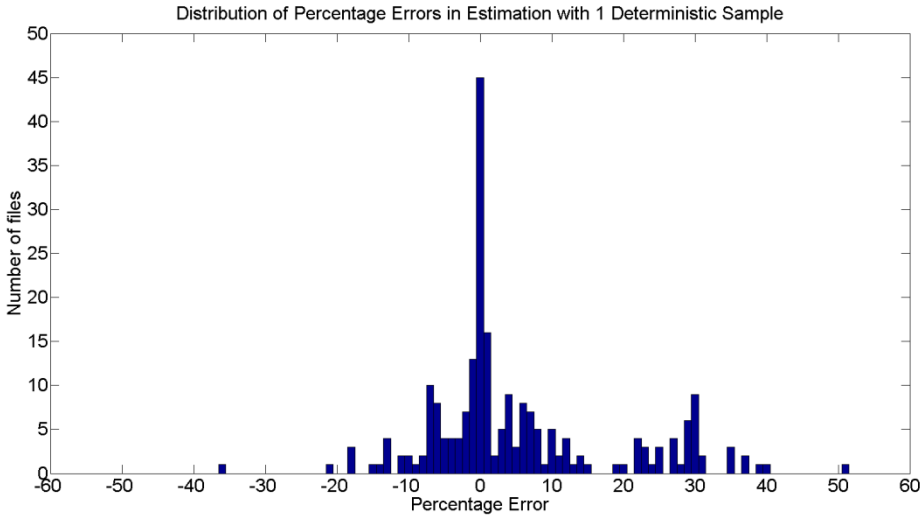

Figure 14: Error Distribution with 1 Deterministic Read, for DS2

An extension to this method was to increase the number of bathymetric packets sampled at the beginning of the file and analyze the effects on the accuracy of the estimates. For this set of experiments, 2, 5, 10 and 20 bathymetric packets were sampled and their inter-packet distances were measured. Given the total size of the file, the average inter-packet distance was used to generate an estimate of the number of bathymetric packets in the raw data file.

For DS1, as the number of samples increased, the mean percentage error decreased initially, reaching its lowest value when 10 samples were collected, after which it begins to increase. The standard deviation in percentage error follows the same trend but its variation remained very small (Figure 15) (Figure 16).

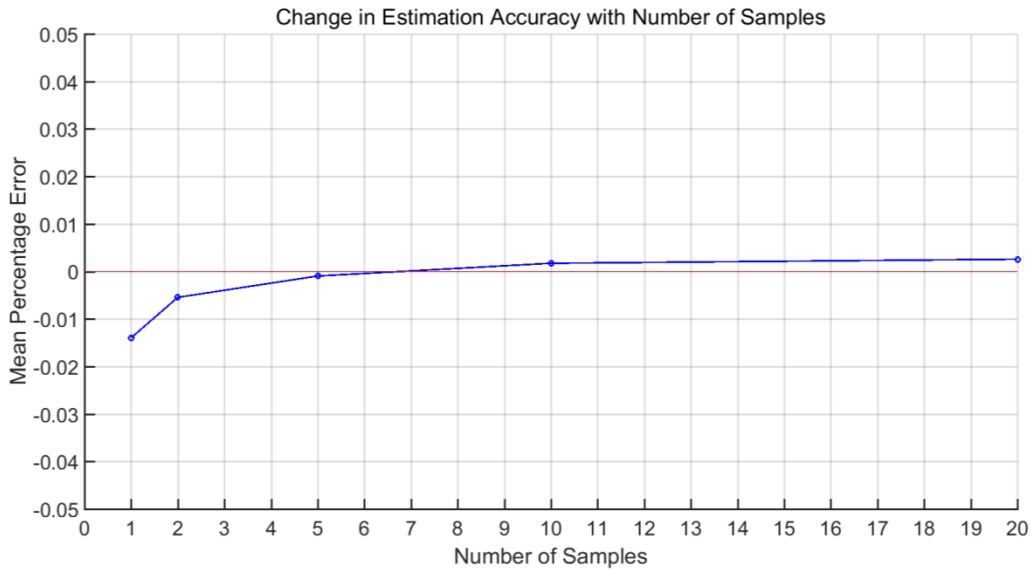

Figure 15: Change of Mean Percentage Errors, for DS1

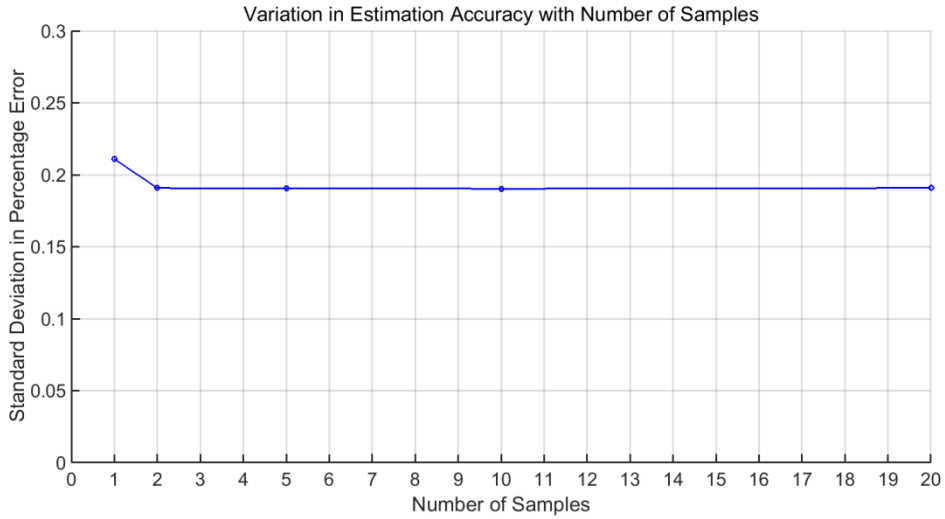

Figure 16: Change in Standard Deviation, for DS1

The effect of increased sampling was more significant on the file in DS2. The average percentage error decreased considerably as the number of samples increased. The standard deviation in percentage errors, though declining, remained high (Figure 17) (Figure 18).

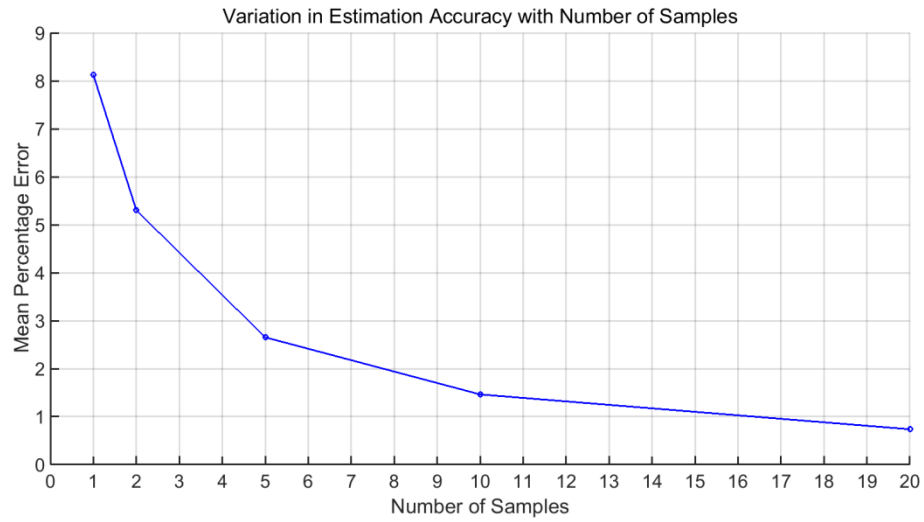

Figure 17: Change of Mean Percentage Errors, for DS2

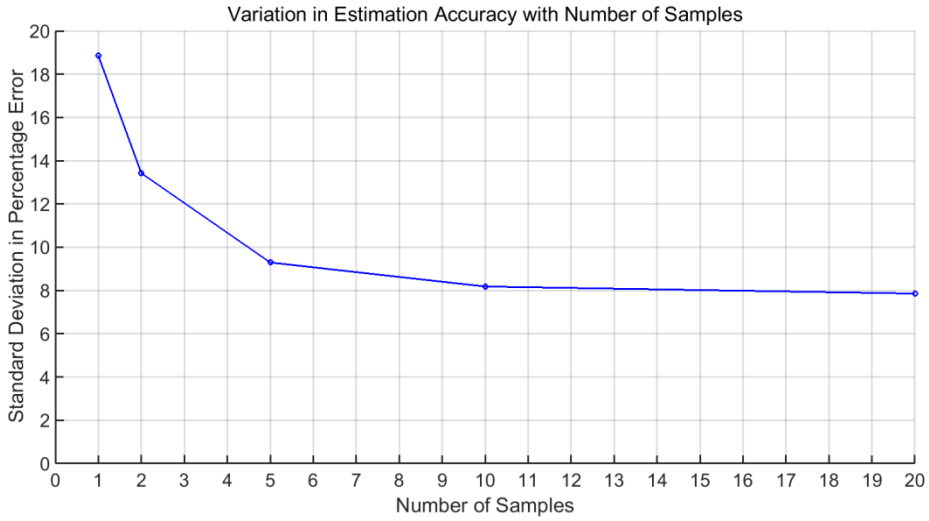

Figure 18: Change of Standard Deviation, for DS2

The estimation time increases as the number of samples increases, as the algorithm has to read and process greater number of data blocks. This is illustrated in Table 1.

Dataset	Method	Avg. Percentage Error	Standard Deviation in Percentage Error	Avg. Estimation Time (ms)
DS-1	1 Sample	-0.0137	0.2109	24.2262
	2 Samples	-0.0054	0.1907	31.0587
	5 Samples	-0.0009	0.1906	44.2246
	10 Samples	0.0017	0.1903	68.2772
	20 Samples	0.0025	0.1908	115.3321
DS-2	1 Sample	8.1252	18.8528	16.9493
	2 Samples	5.3025	13.4221	17.9803
	5 Samples	2.6533	9.2929	18.6289
	10 Samples	1.4633	8.1825	21.4565
	20 Samples	0.7370	7.8572	26.0493

Table 1: Comparison of 1 Sample Estimation Methods

3.4.2 Deterministic – 3 Samples

In this method, the raw bathymetric data file was sampled at three separate locations. Samples were taken at the beginning of the file, one-third and two-thirds through the file. It was assumed the packet density remained constant through these intervals and the samples taken represented these densities.

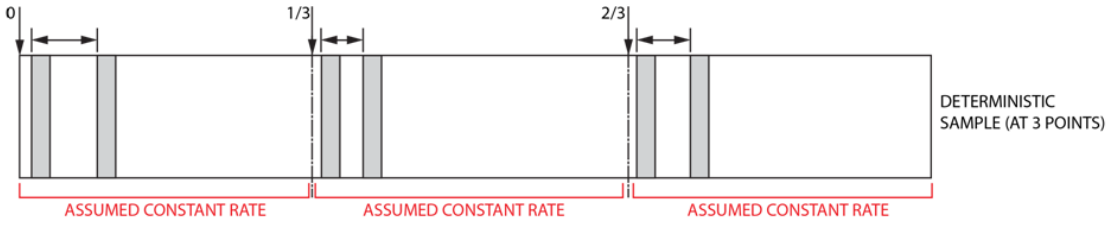

Figure 19: Deterministic Method - 3 Samples

The following algorithm was used:

1 Open the file to be analyzed, skip the file header and any existing channel information structures.

2 Read the data packets, one at a time. If it is a bathymetric data packet, start recording the number of bytes between this bathymetric packet and the next bathymetric packet. This

length is used to calculate the number of bathymetric packets in the first one-third of the file.

3 Jump to one-third and then two-thirds of the file. Each time syncing the file stream until a bathymetric packet is found, measuring the distance between consecutive bathymetric packets and calculating the number of bathymetric packets in each section.

4 The sum of bathymetric packets in each section gives the estimate of the total number of bathymetric packets in the whole file.

For DS1, the mean of percentage errors was 0.04%, with a standard deviation of 0.16%. The mean estimation time was 57.32 milliseconds (Figure 20).

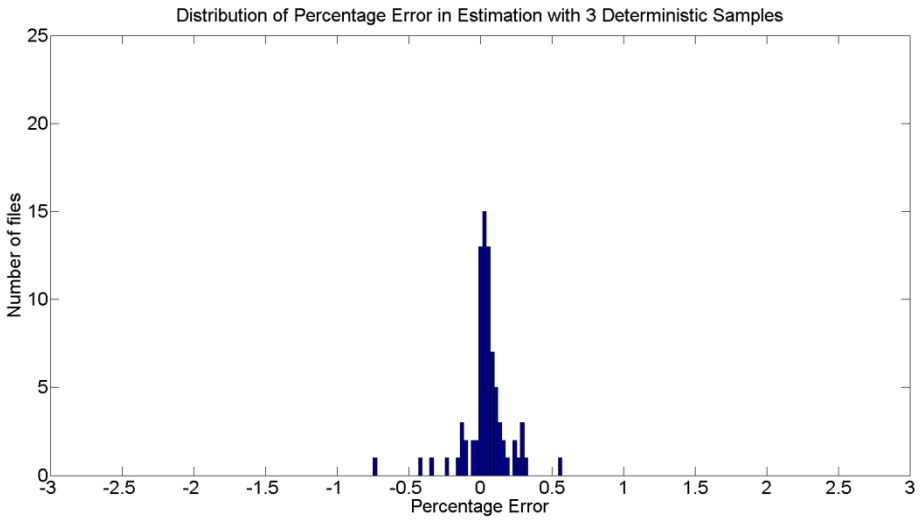

Figure 20: Error Distribution with 3 Deterministic Reads, for DS1

For DS2, the mean of percentage errors was 2.95%, with a standard deviation of 6.63%. The mean estimation time was 38.94 milliseconds (Figure 21).

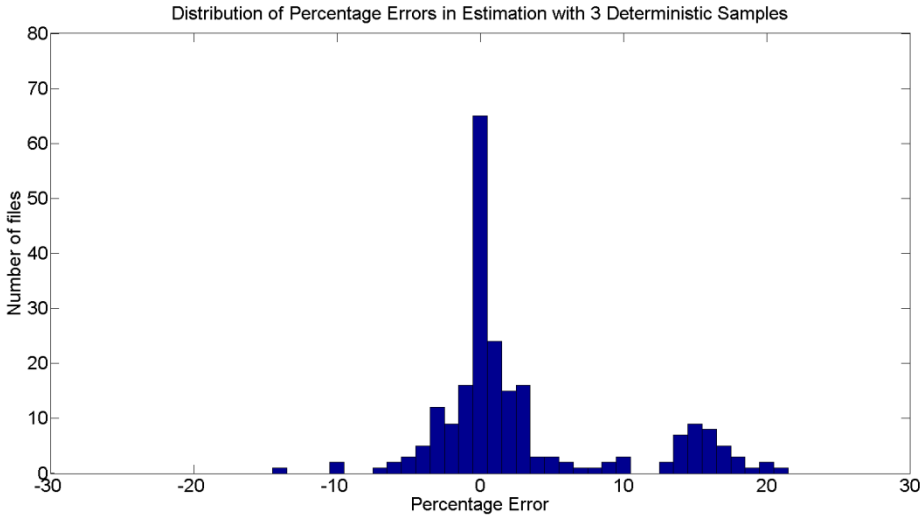

Figure 21: Error Distribution with 3 Deterministic Reads, for DS2

3.4.3 Stochastic – 3 Samples

In the method, samples are collected from three random positions within the raw bathymetric data file. The height of the water column being sampled, which is one of the factors on which size of the data packet depends, can vary along a line. This method allows us to potentially examine the points within the stream of packets where the density changes.

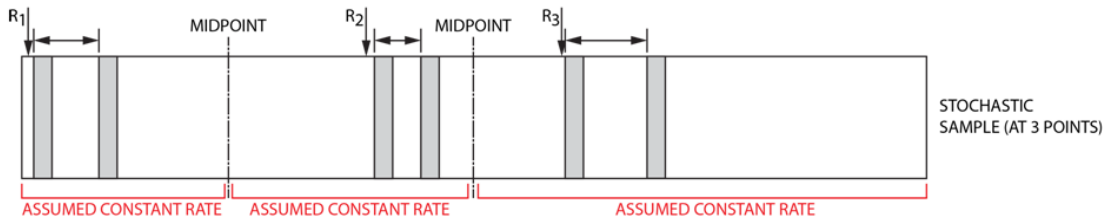

Figure 22: Stochastic Method - 3 Samples

The following algorithm was used:

1 Open the file to be analyzed, skip the file header and any existing channel information structures.

2 Generate three random indices into the data section of the file.

3 Seek to the first index generated and sync till the first bathymetric packet is found. Record the distance between this and the next bathymetric packet. This is the first sample,

representative of the section between the first index and midpoint of first and second indices. Calculate the number of bathymetric packets in the section by using the size of the section and size of the sample.

4 Collect samples from the second and third indices and calculate the number of bathymetric packets in the remaining sections.

5 The estimate of the total number of bathymetric packets is the sum of estimates of bathymetric packets in the three sections.

Being a stochastic algorithm, the samples chosen from the files for estimation of bathymetric packets vary between different runs of the algorithm. It was observed that, over 100 runs of the algorithm, the distribution of the percentage errors over the files of the dataset was similar.

For each dataset, two graphs are presented. The first depicts the distribution of percentage errors in estimation, for a particular run of the algorithm, over all the files of the dataset. The second graph visualizes the distribution of percentage errors in estimations for a single raw bathymetric data file over multiple runs of the algorithm.

For DS1, the standard deviation of percentage errors in estimation was 0.39% and mean percentage error was 0.072% (Figure 23). The average estimation time was 50.40 milliseconds.

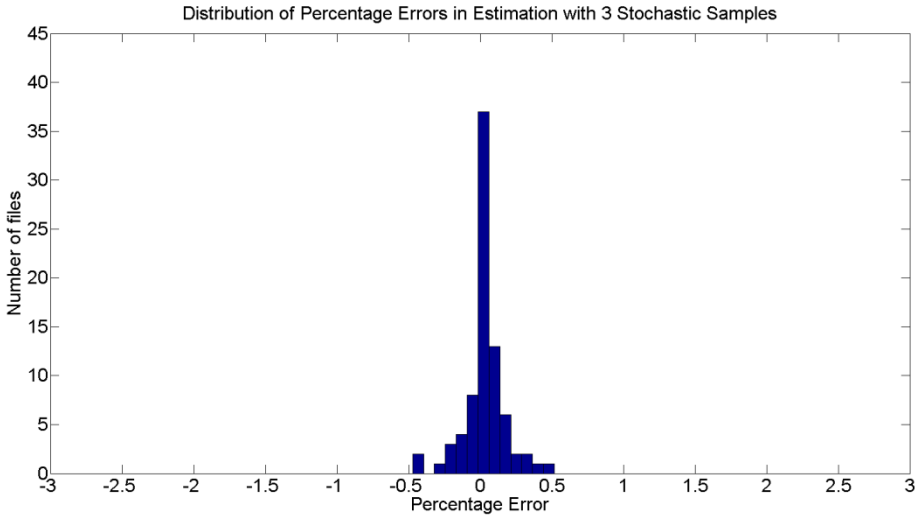

Figure 23: Error Distribution - 3 Stochastic Samples (Single Run, DS1)

Figure 24 depicts the distribution of percentage errors for a single file in DS1, over 100 runs of the stochastic estimation algorithm. The standard deviation of the percentage errors was 0.04% and mean percentage error was -0.10%.

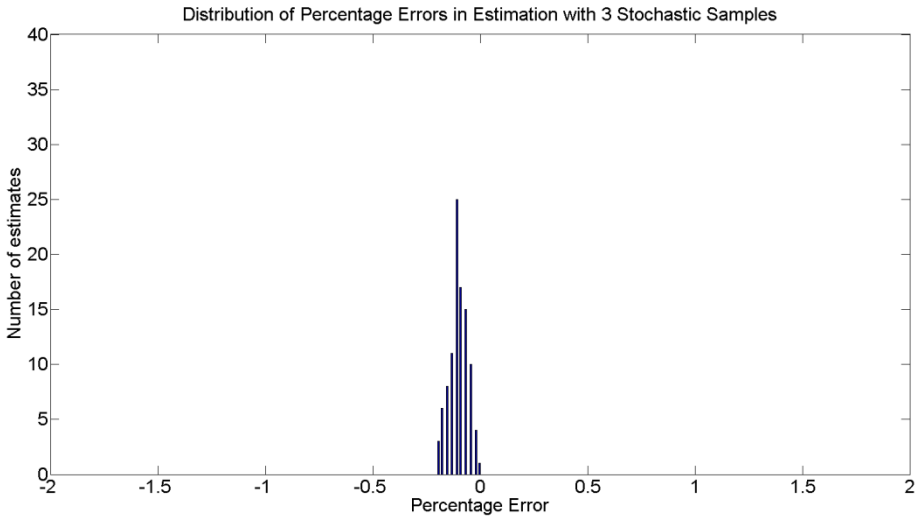

Figure 24: Error Distribution - 3 Stochastic Samples (Single File, DS1)

For DS2, the standard deviation of percentage errors in estimation was 3.53% and mean percentage error was 0.08% (Figure 25). The average estimation time was 37.50 msec.

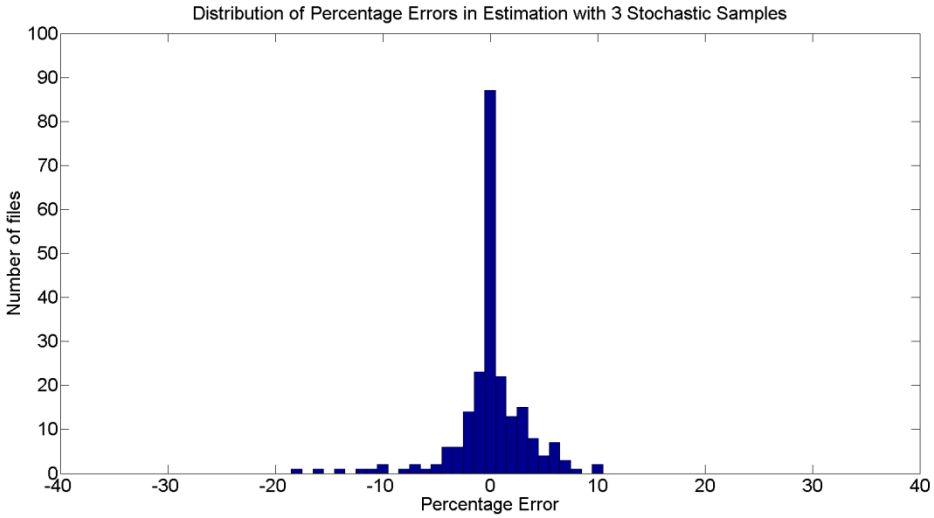

Figure 25: Error Distribution - 3 Stochastic Samples (Single Run, DS2)

Figure 26 depicts the distribution of percentage errors for a single file in DS2, over 100 runs of the stochastic estimation algorithm. The standard deviation of the percentage errors was 0.82% and mean percentage error was -0.17%.

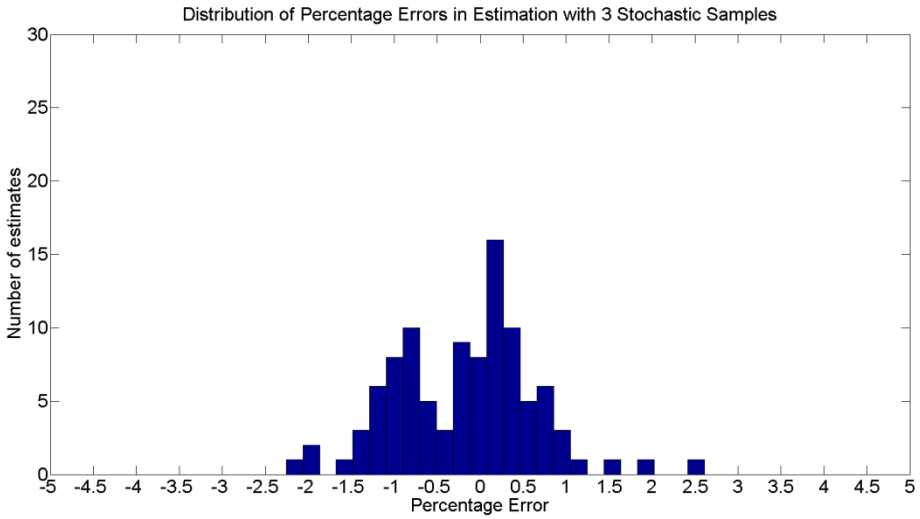

Figure 26: Error Distribution - 3 Stochastic Samples (Single File, DS2)

3.4.4 Observations

Table 2 summarizes the results from the previous experiments.

Method	Dataset	Avg. Percentage Error	Standard Deviation in Percentage Error	Avg. Estimation Time (ms)
DS-1	Deterministic 1 Sample	-0.0137	0.2109	24.2262
	Deterministic 3 Sample	0.0401	0.1609	57.3296
	Stochastic 3 Sample	0.0728	0.3954	50.4014
DS-2	Deterministic 1 Sample	8.1252	18.8528	16.9493
	Deterministic 3 Sample	2.9551	6.6379	38.9485
	Stochastic 3 Sample	0.0873	3.5320	37.5058

Table 2: Estimation Algorithm Results

The experiments highlighted the importance of the need for datasets obtained through different surveys to the assessment of accuracy for these algorithms. The files in each dataset had different characteristics, in terms of file size, count and density of bathymetric data packets within the files, types of bathymetric packets, etc. This allowed for the performance analysis of the various algorithms under diverse input data.

The deterministic class of algorithms samples the files at pre-defined indices within the file to estimate the number of bathymetric packets. In the one sample method, the sample is collected from the beginning of the file. This method assumes that the density of bathymetric packets remains fairly constant throughout the file. Hence, for DS1, it produced good estimates as the files exhibited a fairly uniform distribution of data packets. The errors in estimates for files in DS2 were many times higher due to the variation in data density within the files. Even by sampling an increased number of bathymetric packets at the start of the file, the impact on performance was not comparable to the other estimation methods. For DS1, there was almost no improvement in accuracy of the estimates. This was expected as the data density within the DS1 files was fairly constant. The gains in accuracy were more

significant for DS2, as the larger sample size helped to better normalize the diverse sizes of the data packets with the files. However, the minor and inconsistent estimate improvements (between DS1 and DS2) did not justify the additional processing overhead.

The second deterministic algorithm, which sampled at 3 regular intervals within the file, was able to handle variation of data packet density better as compared to the single sample algorithm. The diversity in the sampled packets meant that it could produce estimates with lower errors for both datasets.

The stochastic algorithm aimed to incorporate the non-determinism in packet density by randomly selecting samples from the data file. Owing to this random selection of samples, the performance of the algorithm varies with each run, but as Figure 24 and Figure 26 illustrate, the variation in accuracy of estimates for a particular file is low. For the chosen run, the accuracy of the estimates produced by the stochastic algorithm, on DS1, was slightly worse that the three-sample deterministic algorithms. However, the accuracy of estimates was significantly higher for the stochastic algorithm when run against DS2.

The cost of estimation, in all the above algorithms, is the time required to generate the estimate, which is the sum of the disk access time and the computation time. Computation time is dependent on the number of packets sampled for estimation, the size of each individual packet and the stream synchronization operation when the estimation algorithm seeks into the file. It is important to note that the values for estimation time are to be used for comparison for different algorithms, operating on the same dataset.

The estimation time for DS1 is greater than DS2, irrespective of the algorithm used, because of the presence of numerous ancillary data packets, of large sizes, between the two bathymetric packets being sampled. The estimation algorithm has to recognize and skip these ancillary data packets, which requires additional processing, resulting in higher estimation

time. These packet characteristics affect both the time required for sampling and stream synchronization.

The estimation time for the one-sample method was the least due to the minimum sample size. For the remaining two algorithms, the estimation time was similar because both were required to collect three samples from distinct locations within the file, resulting in an identical set of operations, which consisted of seeking to a specific byte-location within the file, synchronizing to a bathymetric packet and measurement of inter-bathy packet distance. The small variation in the estimation time was a result of the uncertainty of the disk seek time, resynchronization required, type and size of packets encountered as part of the analysis.

3.5 Conclusion

The accuracy in estimation of the deterministic algorithms is high for datasets that exhibit a fairly uniform data density. However, the performance degrades very significantly as the complexity of the organization of data packets in the files increases. As the uniformity of data packets within the raw bathymetric files cannot be guaranteed and there is a high probability that the proposed system will be exposed to datasets with varying attributes, the stochastic method for estimation of the number of bathymetric packets was selected. The stochastic algorithm would provide a good estimate, irrespective of the input dataset. This would allow the system to determine the workload of the processing task being submitted and use this information as one of the factors in the load balancing tasks across the compute nodes.

CHAPTER 4

WORKLOAD DISTRIBUTION

The primary purpose of this work was to facilitate the processing of bathymetric data by application of the parallel processing paradigm. This would increase the throughput of processing raw data, over a set of files, while maintaining high efficiency. The previously stated goals can be achieved through an appropriate workload distribution mechanism. This component would be responsible for determining the priority of tasks, i.e., conversion of bathymetric files, over a set of compute nodes. The prioritization of tasks would be dependent on a number of factors, which include, for instance, the number of bathymetric packets in a raw data file, time spent by the task waiting for processing, priority of the user who submitted the task, etc.

4.1 Factors Influencing Load Distribution

The workload distribution module aims to match a particular processing node to a raw bathymetric file that needs to be processed. The selection of the ideal compute node to which the processing task is dispatched depends on a variety of factors in the input and the current state of the processing system:

- Number of bathymetric packets in the raw data file: This estimate would directly correspond to the time required for transformation of the file to the final product and quantifies the workload requirement for a particular task.

- Number of pending packets: This number represents the count of data packets in the processing queue at the compute nodes. The number is updated as processing tasks are added to the node or a particular processing task completes. This metric is used to keep the nodes balanced in terms of processing tasks assigned.
- Performance: This number indicates the number of bathymetric packets processed per unit time by a compute node.

The count of the bathymetric packets in the raw data file is the basis from which a significant number of the factors are derived. These factors are used by the various load balancing algorithms to determine the best possible distribution of tasks among the processing nodes. It is important to note that not all algorithms mentioned in this work use all the factors. Depending on the algorithm used, only a subset of the factors might be drawn on. The next section describes the procedure to obtain and store the various metrics that were listed before.

4.2 Module Design

The design of this module revolves around the components that enable collection and storage of task and job metadata, and scheduling and monitoring of processing tasks. Figure 27 shows the layout of different components within the module and highlights their interactions.

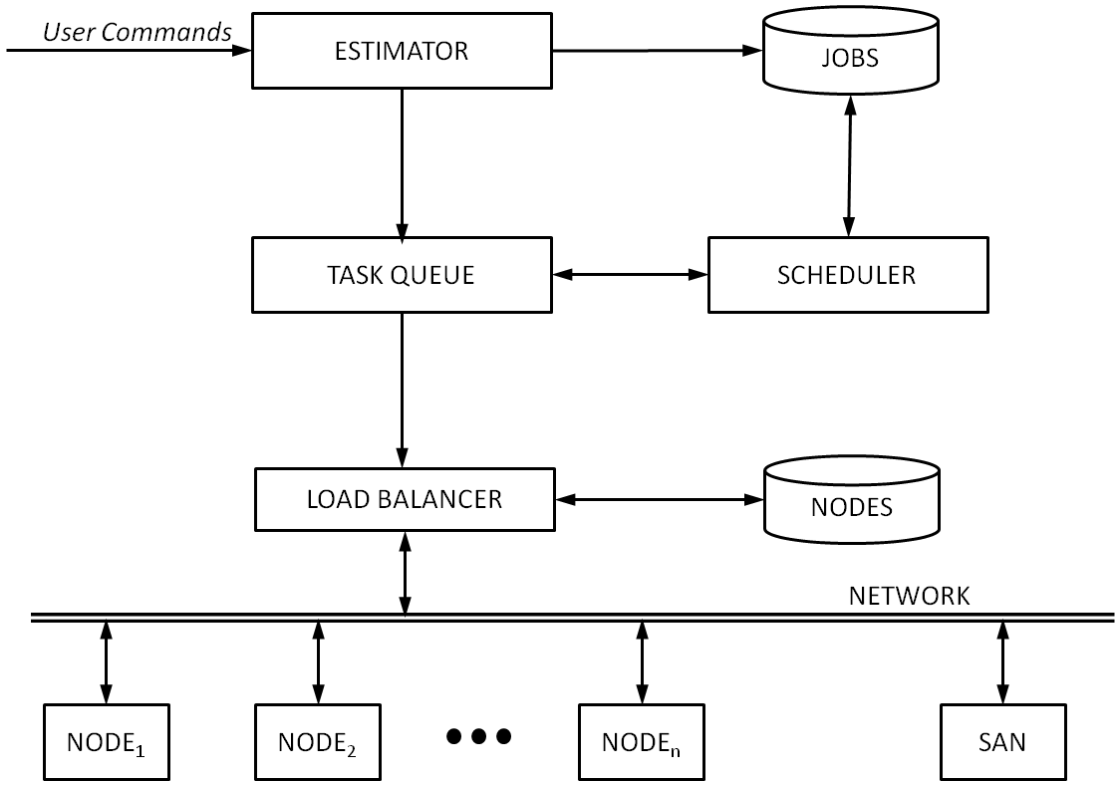

Figure 27: Scheduling and Load Balancing Module Design

The major components are:

- Job Table: This table is stored in the SQLite database and holds information regarding the jobs that are submitted to the processing system. The input from the user is analyzed and broken down into its component tasks. Each job is assigned a unique identity and all its related metadata, e.g., number of tasks, total number of bathymetric packets, time submitted, etc., is stored in the job table.
- Task: This is a data structure that represents a processing task, i.e., conversion of raw bathymetric data to the final product. It consists of information regarding the file corresponding to the task, the number of bathymetric packets, its priority, etc.
- Task Queue: The task queue holds tasks that are ready to be submitted to the compute nodes. It consists of multiple arrays, one for each priority level. The priority of a task is determined by the scheduling algorithm installed using one or more of the factors mentioned in the previous section. The position of tasks within the task queue can be

altered as required by the scheduling algorithm and the best candidate task for processing is always at the head of the queue. This allows the load balancer to pick the next task in constant time, regardless of the size of the queue.

- Node Table: This is a SQL table that holds information regarding the compute nodes. For each compute node there is a corresponding entry in the table which provides metrics that are used for load balancing and performance analysis. These metrics consist of the number of bathymetric packets processed, number of pending bathymetric packets, state of the node (up/down), its performance (number of bathymetric packets processed per second), etc.
- Load Balancer: This component aggregates information from all of the above mentioned components and determines the computing node to which the processing task is to be submitted. The working of this component is dependent upon the type of scheduling algorithm being used.

The following sections describe the algorithms used as part of this work and how the information and the modules presented previously fit into their working.

4.3 Task Scheduling Algorithms

The various components described in the previous section provide data regarding the processing tasks assigned to the system and the state of the various components of the processing system. This information is aggregated by the scheduling algorithm and is used for assigning a processing task to a particular compute node.

The aim behind this work was to maximize throughput and efficiency of a parallel compute cluster while avoiding any resource starvation for any particular processing task. As part of this work, three separate algorithms were investigated, all of which utilized the factors mentioned in the previous section in different ways.

First Come, First Served (FCFS) scheduling was the first algorithm chosen as part of the experiment. The idea behind this algorithm is to prioritize resources based on the arrival time of the request. All the requests are given equal priority and the resource is only allocated when the preceding requests are fulfilled. The simple logic behind this algorithm leads to an easy implementation and minimal scheduling overhead.

The implementation of the FCFS algorithm here uses the arrival time of the processing request to schedule the various conversion tasks. Each incoming job, and its component tasks, is given the same priority and placed on the task queue for processing. The remaining task metadata are not considered in the calculation of the priority of a task. Once the task is ready to be processed, it is pulled off the task queue and assigned to a compute node with the fewest bathymetric packets to be processed. In Figure 28, each block represents a processing task and the number inside the block denotes the estimated conversion time for that particular task.

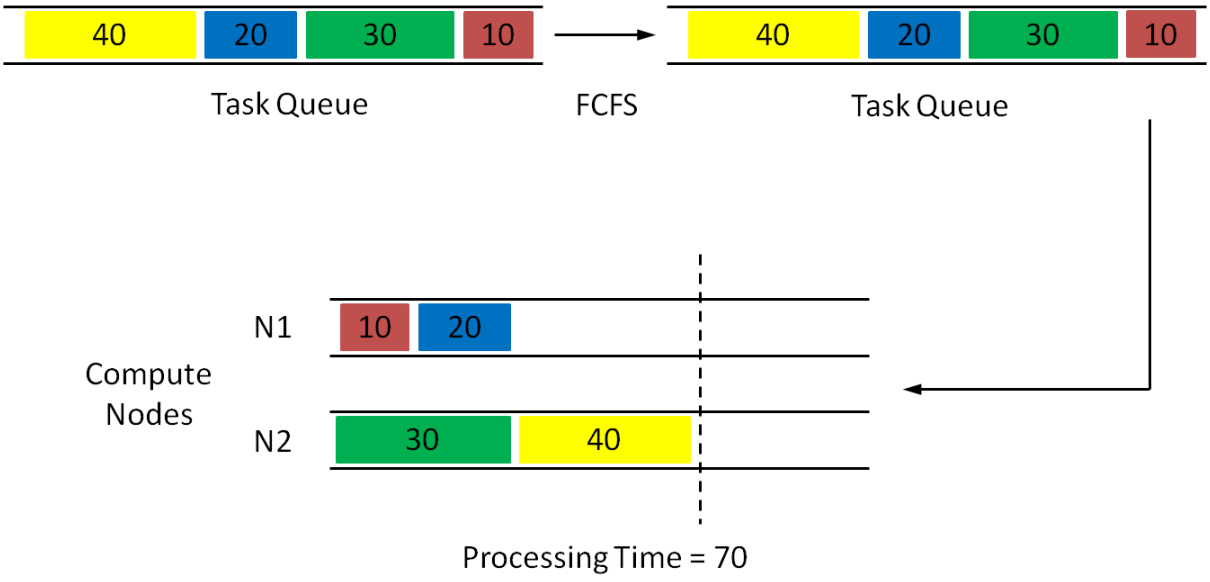

Figure 28: Working of FCFS

Largest Job First (LJF) is one of the many scheduling algorithms that take into account the processing time when allocating resources to a particular job. The structure and

effectiveness of this algorithm is dependent on the knowledge of the processing requirements for each job, before the actual execution begins. This algorithm prioritizes the tasks that require a higher processing time over those with lesser resource requirements. The motivation is to improve the throughput by enabling more precise and controllable load balancing.

As discussed earlier, the count of bathymetric packets within the raw data file is a good indicator of the processing requirements to convert to the final product. The stochastic estimation algorithm provides a fairly accurate estimate of the count of bathymetric packets in the data files. This metric can be used to determine the resources required for processing a particular file. To have greater control over the distribution of processing tasks across the compute nodes and improve job throughput, a modified version of the LJF was developed. Since the processing time of a set of tasks being executed in a parallel processing environment is dependent on the last task to complete, it is important to focus on the balancing of tasks across the compute nodes. Having the largest tasks scheduled earlier gives the algorithm a fine-grained control on balancing out the load inequalities amongst the compute nodes later in the process. The custom LJF algorithm prioritizes based on size of the tasks, with the tasks at the head of the task queue being the largest. This allows the load balancing algorithm to select the next largest task and assign it to a processing node, while trying to maintain a balanced compute system.

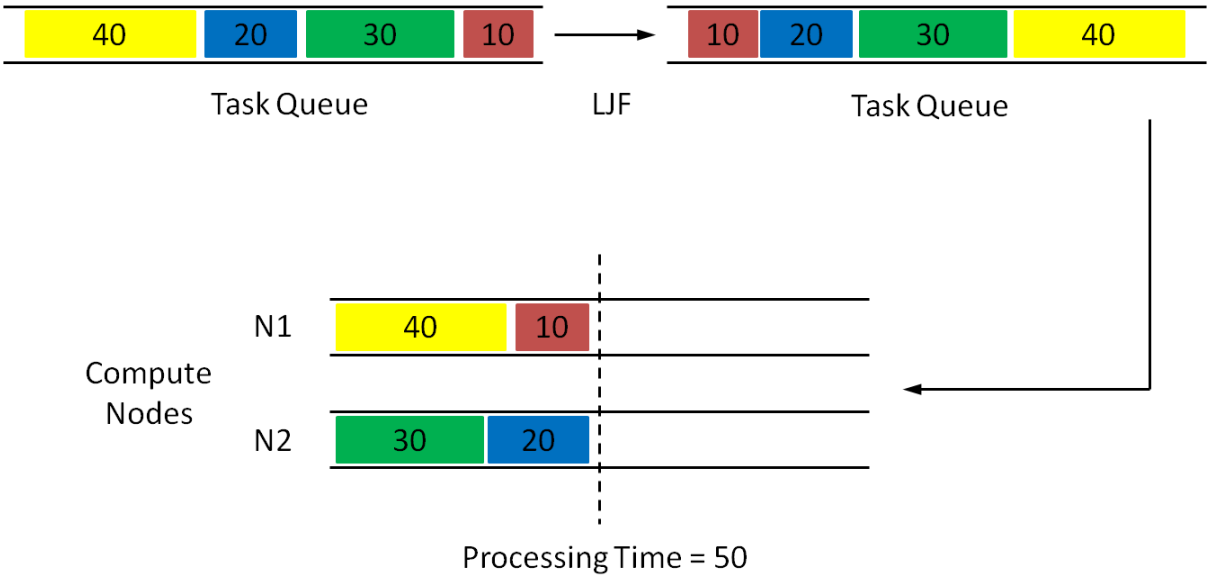

Figure 29: Working of LJF

Both FCFS and LJF focus on the balancing of the workload across the compute nodes in order to improve speedup and efficiency. However, in the parallel processing system access to the SAN is a point of contention that can significantly impact performance. The capacity of the SAN to handle multiple compute nodes simultaneously will determine the parallel IO that can take place between the storage and compute resources in the parallel processing system.

Raw bathymetric data stored on disks is accessed on two separate occasions during the operation of the parallel processing system. During the estimation phase, a few blocks of data are read from each file to determine the workload. This data access pattern holds the disk resource for a very small period of time and does not result in significant wait times for the compute nodes. The second access to the SAN is during the conversion process during which the entire raw data file is read into the compute node. This access pattern utilizes the disks for a significant period of time and, depending on the capabilities of the SAN, might cause contention issues for the compute nodes.

Since resource contention issues are caused when multiple components request access to the same resource at the same time, the delay between adjacent requests plays an important

role in determining the length of the queue at the shared resource. In this work, the length of the queue is determined by the number of compute nodes and by the difference in the data transfer time and conversion time of the raw bathymetric data files in a dataset. For a given number of compute nodes, the greater the conversion time, as compared to the transfer time, the lower the contention at the SAN. This is because a compute node can complete the data transfer within the window of conversion time of the other compute nodes. Conversely, longer transfer time coupled with short conversion time results in heavy contention.

For a given set of capabilities of the SAN, which consists of network bandwidth and simultaneous access support, there is a maximum possible speedup that can be achieved for a dataset. This speedup is possible when the cluster consists of infinite compute nodes. In this configuration, each processing task is assigned to a different compute node, and hence each compute node will access the SAN once during the processing of the dataset. In this case, the minimum possible processing time for the dataset will be sum of the time required to transfer the entire dataset (in the form of tasks) to the compute nodes and the conversion time of the last compute task. This dataset processing time denotes the maximum possible speedup for the given SAN capabilities.

Conversely, conversion of the raw files in the dataset using a single compute node yields the processing time for serial execution, which is the maximum processing time for the dataset. Since there are no overlaps in the transfer or processing of the raw files, the total processing time for the dataset set will be the sum of the transfer time and processing time for all the files in the dataset. Using the estimator, the total number of bathymetric packets in the dataset can be determined. This number can be used to estimate the processing time for the dataset. The total transfer time can also be determined through the speed of the link between the compute nodes and the SAN, and the size of the dataset.

As the number of compute nodes is increased, the speedup increases along with the wait time at the shared SAN resource. At a certain point, the contention begins to outweigh the benefits of adding more compute capacity, thus leading to a decrease in the efficiency of the parallel processing system. The aim of the final algorithm is to determine the maximum speedup possible for the input dataset and use this information to balance workload across a subset of the compute nodes, in a way that minimizes the contention at the SAN while attaining the highest possible performance. This algorithm is called the Contention-Reduction (CR) scheduling algorithm. Dividing the maximum possible processing time (from the serial processing case) by the minimum processing time (from the maximum speedup case) will yield the ideal number of nodes that can achieve maximum speedup with minimum contention. The processing system only distributes the conversion tasks among this ideal number of compute nodes, thereby minimizing contention at the SAN and delivering near-maximum speedup.

Figure 30 illustrates the working of the CR scheduling algorithm with two input datasets, DS-1 and DS-2. The estimator breaks down the datasets into their component tasks and records their metadata, for instance, the file size, estimate of the number of bathymetric packets, etc. The CR scheduling algorithm uses the system information and task information to determine the maximum speedup and efficiency combination for each dataset. Consider that, for DS-1, the optimal number is two compute nodes and hence, its three tasks are distributed among nodes N1 and N2, even though, initially, all four compute nodes have no pending tasks to process. Similarly assume that for DS-2, using the calculated compute node limit of five, its four tasks are distributed among the compute nodes with least pending processing workload.

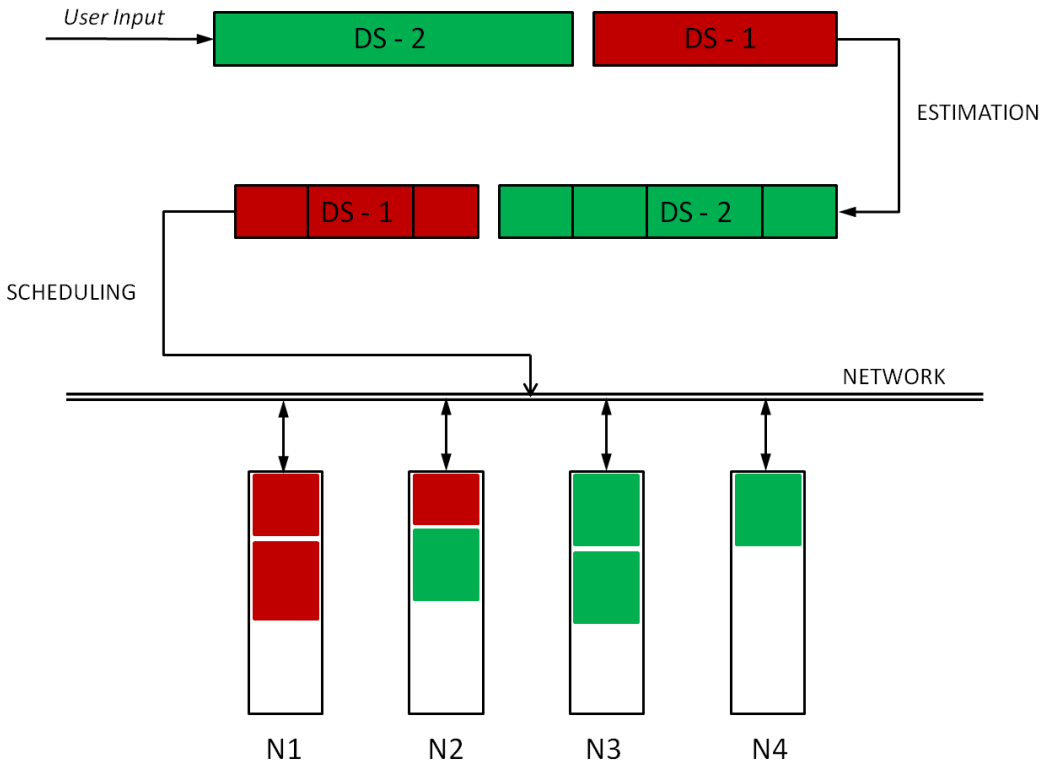

Figure 30: Working of CR Scheduling with Two Datasets

The following sections describe the experiments performed to exercise the previously mentioned algorithms. To select any one particular algorithm, a number of factors had to be considered. Since the aim of this thesis is to verify the feasibility of parallel processing, the metrics of speed-up and efficiency in the parallel environment as compared to sequential processing needed to be considered. It was also important to consider if the algorithm could cause any resource starvation.

4.4 Pre-experiment Calibration

Before the algorithms were tested to quantify their performance, there were a few parameters that needed to be specified and have their values established.

The main focus of this work was to measure the gains of using an array of compute nodes, processing a set of tasks in parallel, as compared to a linear model where tasks are processed one after another. The most important factor that defines the effectiveness of a

parallel processing setup is the speedup achieved. To measure speedup, the turnaround time needed to be measured. For this work, turnaround time is the time from when the job is submitted for processing until all the tasks in the job have been processed, which is the sum of the data transfer time and conversion (processing) time.

Raw data files are usually stored on Storage Area Network (SAN) equipment which is connected to the compute nodes through network links. The files would be transferred, over the network, to the compute nodes when they need to be processed. This means that there is delay between the time the compute node receives a command to process a particular bathymetric file and the time when the target file is cached locally and ready to be processed. This latency depends on the network bandwidth, the number of concurrent requests on the network and the size of the file being transferred.

To simulate the file transfer latency, the lab setup of CCOM/JHC at UNH was used as a reference. This lab has an array of blade servers that are connected to a SAN device, over iSCSI. The bandwidth measured between them was 67 MB/s. This value was used for simulating network bandwidth between the compute nodes and the SAN. Another variable that affects data transfer latency is the number of simultaneous users that can be serviced by the SAN. In the field, this number would vary depending on the type of SAN being employed. For the purpose of these experiments, it was decided that single host access, at 67 MB/s, would represent the worst case scenario for data transfer. Conversely, the best case scenario involved a SAN device that could service infinite hosts, each with infinite bandwidth. This allowed the speedup to be measured in two distinct modes and suggest that the performance of the proposed system will fall between the two cases outlined, for a typical setup.

The processing time of a task is another factor that affects the speedup in the proposed system. As mentioned in the previous sections, each data packet in a raw bathymetric data

file undergoes the same set of transformations to the final product. This packet processing time is more or less similar for all bathymetric data packets, regardless of their size. The implementation of the code that performs the conversion of raw bathymetric data to the finished product is beyond the scope of this work. Hence the conversion was simulated in code to obtain the processing time requirements for each raw bathymetric data file (task). The serial version of the conversion algorithm, provided by CARIS, was used to determine the processing time requirements for raw bathymetric data. It is expected that the conversion algorithm that would be used as part of the complete parallel processing system would be very similar because each compute node performs a serial conversion, independent of the other compute nodes. The datasets DS1 and DS2 were run through a serial version of the conversion algorithm and it was found that the algorithm was able to convert one bathymetric packet in 9.677 milliseconds. This value, in combination with the estimate of the number of bathymetric packets, was used to simulate the processing time of each raw bathymetric data file.

To test the performance of each algorithm, datasets DS1 and DS2 were used. As mentioned earlier, these datasets differ vastly in the size of the individual raw data files, and the density and the total number of bathymetric packets within each data file. For instance, the large sizes of files in DS1 would result in larger data transfer times but shorter processing times due to lower density of bathymetric packets. On the other hand, files in DS2 would incur a lighter data transfer penalty because of their relatively smaller sizes. The files in DS2 also varied in the number of bathymetric packets and this would stress the load balancer module to maintain equilibrium in processing workload amongst the compute nodes.

4.5 Scheduling and Load Balancing Experiments

Each of the algorithms was measured based on a number of factors. Speedup, as described earlier, defines the gain in the throughput or turnaround time obtained by employing an increasing number of parallel processing resources. The speedup (S_p), for p processors, is defined as the ratio of the job completion time (T_s) for serial execution, to the job completion time (T_p) when using p processors.

$$S_p = \frac{T_s}{T_p}$$

Speedup is hampered by increased complexity in the parallel processing system through increased communication between the various components, making data available to the appropriate compute node, etc. For this work, these overheads include scheduling and load balancing algorithms, state and metadata gathering and maintenance for various system components, and data transfer latency. The algorithms proposed here aimed to minimize complexity, wherever possible, and thereby increase speedup.

Associated with the metric of speedup is efficiency, which is a measure of the effectiveness of adding an increasing number of processors to aid in the computational tasks. Efficiency (E_p) of a parallel processing system consisting of p processors is defined as the ratio of the speedup (S_p) attained through parallel processing, to the number of processors (p).

$$E_p = \frac{S_p}{p}$$

The ideal speedup would be linear which corresponds to an efficiency of 1. However, due to multiple overheads in a parallel processing system, speedup is almost never linear, and hence efficiency is very rarely ideal.

The balance of workload across the compute nodes is another metric that was used to measure the performance of the algorithms. Since the completion time of a job is dependent on the last task to be completed, it is important that the compute nodes have approximately

the same number of packets to process. A perfect balance of bathymetric packets is not possible because the smallest unit by which packets can be distributed is a file. An optimal balance would enable maximum throughput and utilization of the compute resources.

For each of the following algorithms, experiments were performed for up to 12 compute nodes and information regarding job processing time and node utilization was collected through SQLite tables, which were responsible for maintaining system metrics.

4.5.1 First Come, First Served

For this algorithm, tasks are processed in the order they are submitted to the processing system. Each task has the same priority and is allocated resources after the preceding tasks have been assigned to specific compute nodes to be processed.

Under infinite network bandwidth conditions, the performance of the FCFS scheduling algorithm for the best case scenario is very close to ideal, in terms of speedup and efficiency, as the number of compute nodes is increased. In the worst case scenario, there were some minor gains in speedup as the number of compute nodes was increased to 3. However, increasing the number of compute nodes beyond 3 led to little or no gains in speedup. Consequently, the efficiency fell as compute nodes were added to the processing system. (Figure 31, Figure 32)

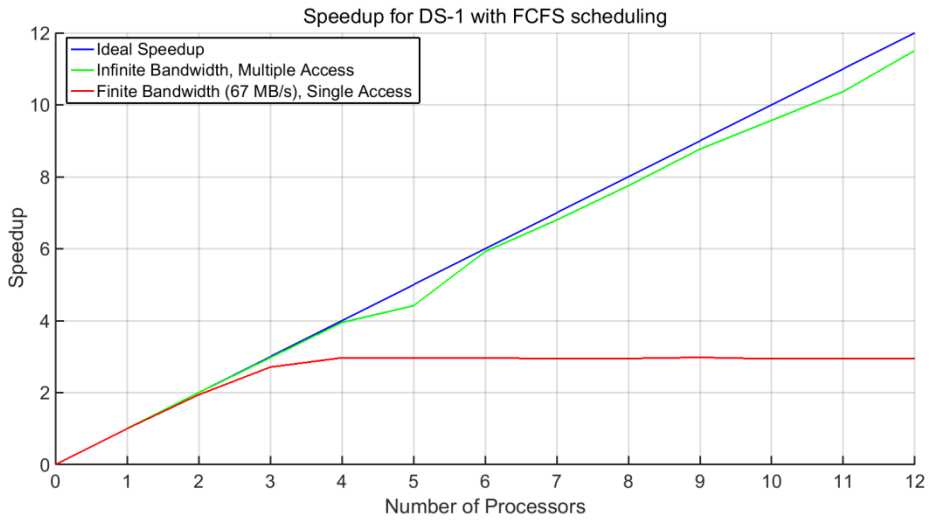

Figure 31: Speedup for DS-1 with FCFS Scheduling

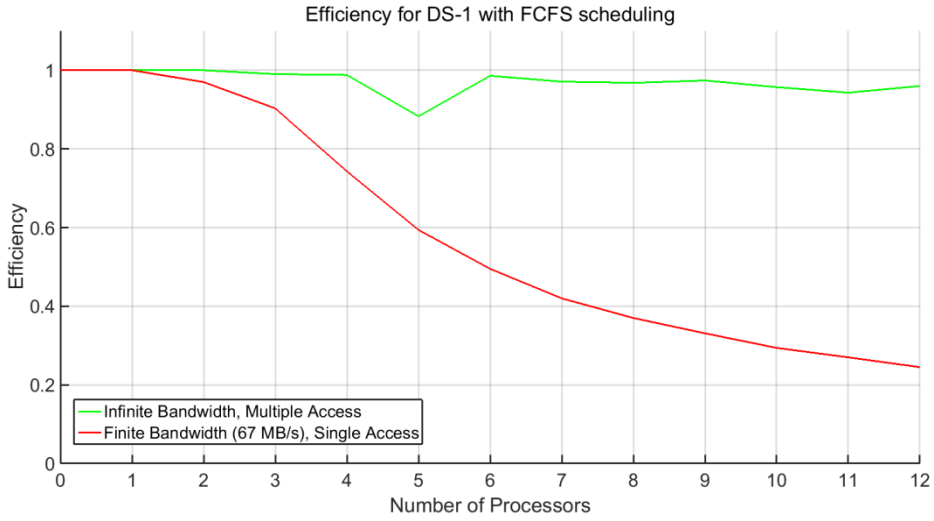

Figure 32: Efficiency for DS-1 with FCFS Scheduling

The percentage deviation of the count of bathymetric packets depicts the deviation of the worst node as compared to the ideal distribution of bathymetric packets across all the nodes. A perfectly balanced distribution is not possible because the distribution of the data packets is constrained by the files in which they are contained. For DS-1, owing to the uniformity of the data files, the order in which files are processed does not significantly impact the distribution of bathymetric packets and the resultant deviation is low. (Figure 33)

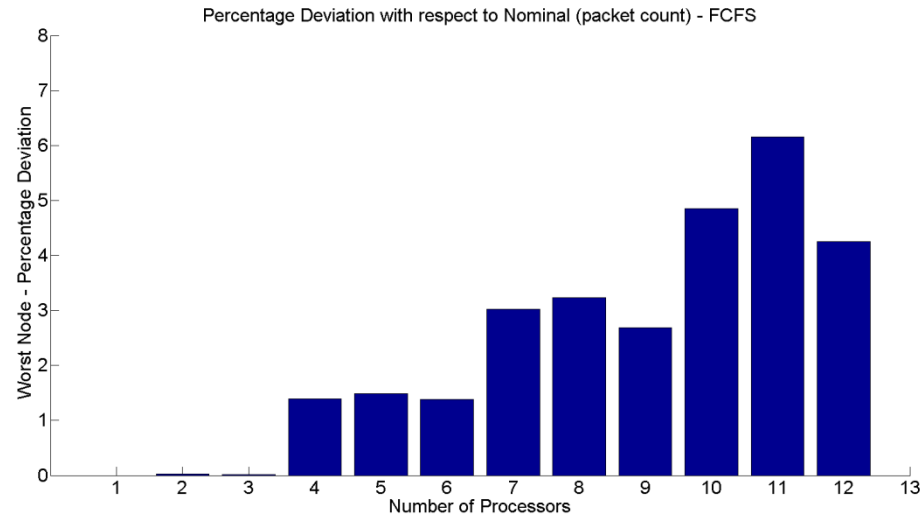

Figure 33: Deviation of Bathymetric Packets with FCFS (DS-1)

For DS-2, the speedup increased as compute nodes were added to the processing system, for both best and worst case scenarios (Figure 34,Figure 35).

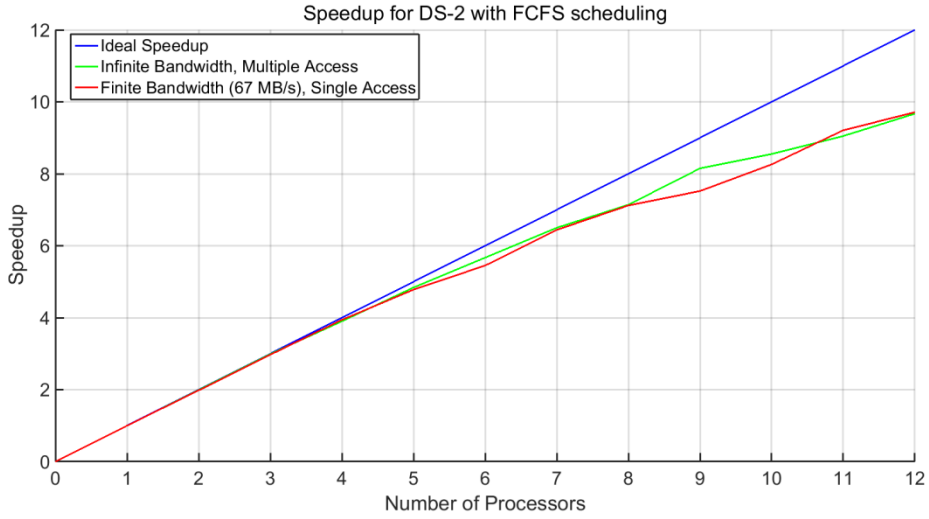

Figure 34: Speedup for DS-2 with FCFS Scheduling

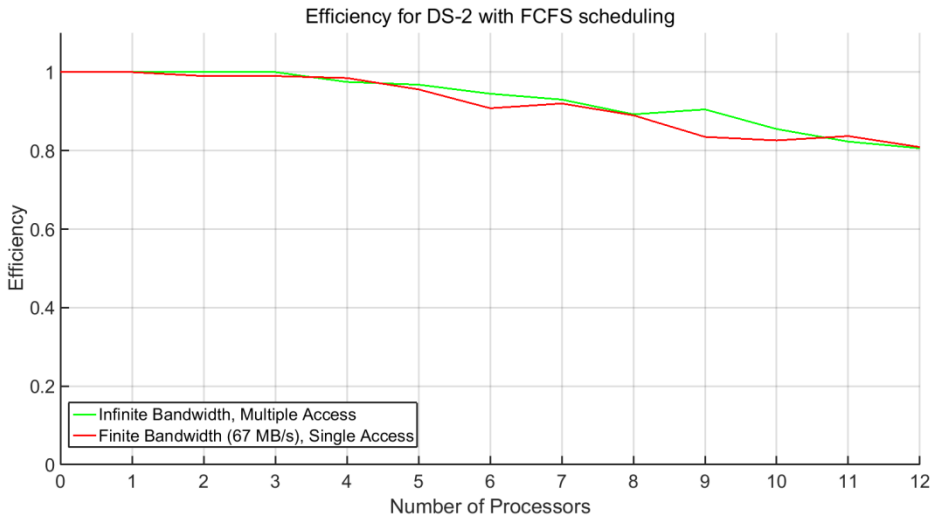

Figure 35: Efficiency for DS-2 with FCFS Scheduling

Speedup and efficiency derive from the balance of workload across the compute nodes. For DS-2, increasing the number of compute nodes also increased the deviation of the count of bathymetric packets at the worst node (Figure 36). Hence, the gain in speedup was reduced as the count of compute nodes was raised from 4 to 12. Efficiency followed a similar trend (Figure 34, Figure 35).

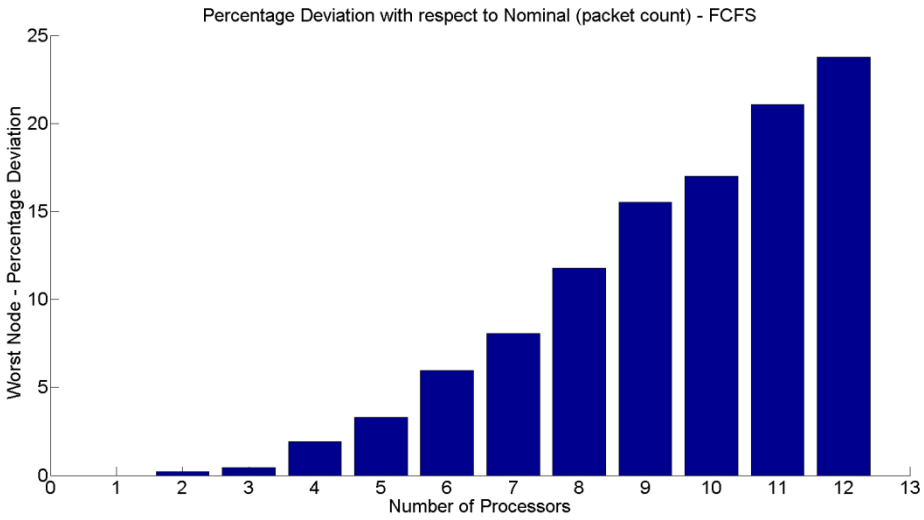

Figure 36: Deviation of Bathymetric Packets with FCFS (DS-2)

FCFS produces different gains for DS-1 and DS-2. For DS-1, there is a wide performance gap between the best and worst case network scenarios. The performance in the

limited bandwidth case remains low as the number of nodes is increased, even though the deviation of bathymetric packets in the worst node remains relatively low. This indicates that throughput for DS-1 is not limited by CPU capacity but suffers due to the bottleneck in the network path. This bottleneck is a result of the characteristics of DS-1 and the causes are further analyzed in the observations section of this chapter.

For DS-2, there are significant gains in throughput as the number of compute nodes are increased. However, the rate of these gains decreases as the processing capacity increases. The FCFS algorithm is very susceptible to the order in which tasks are submitted to the load balancing system. If the end of the stream of tasks consists of files with large variation in the number of bathymetric packets, it causes the nodes to become imbalanced. This causes increased skew from the ideal distribution of bathymetric packets between the nodes and results in lower throughput.

4.5.2 Longest Job First

The premise behind LJF is to schedule the tasks with the largest (and hence, longest) workloads first so that the smaller workloads can be used to balance the workload difference among the compute nodes efficiently. Therefore, this algorithm prioritizes the processing of the largest data files while relegating the smallest data files towards the end of the processing queue.

The performance of the LJF scheduling acting on DS-1 was very similar to that of the FCFS scheduling. The initial gain in speedup plateaus as the number of compute nodes is increased beyond 3 (Figure 37, Figure 38). Using LJF resulted in a more balanced distribution of bathymetric packets across the compute nodes, with the worst deviation reducing from about 6.5% to about 5% (Figure 39). As a result, there was a slight improvement in performance when using LJF scheduling as compared to FCFS scheduling.

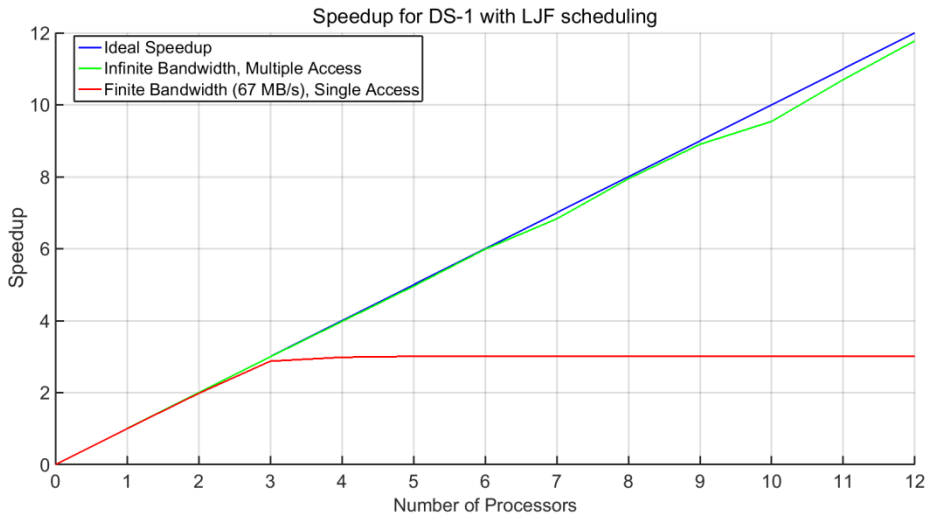

Figure 37: Speedup for DS-1 with LJF Scheduling

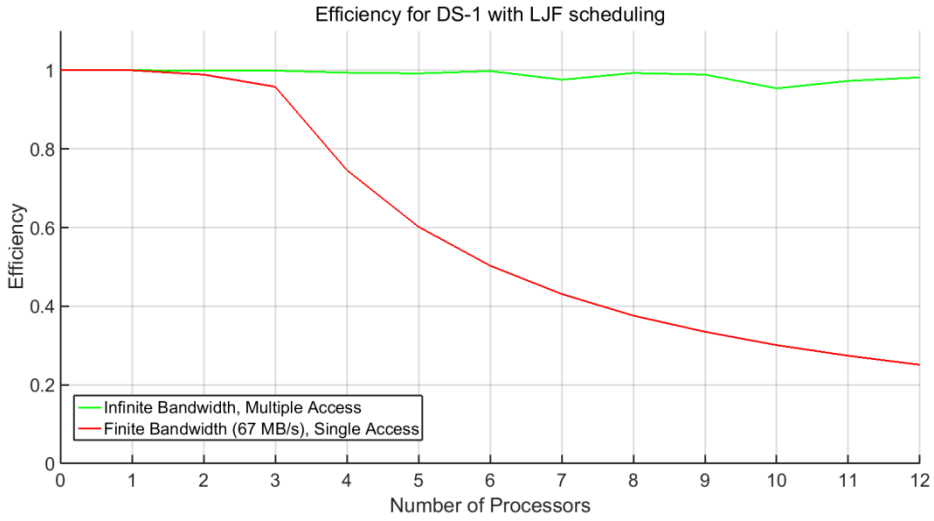

Figure 38: Efficiency for DS-1 with LJF Scheduling

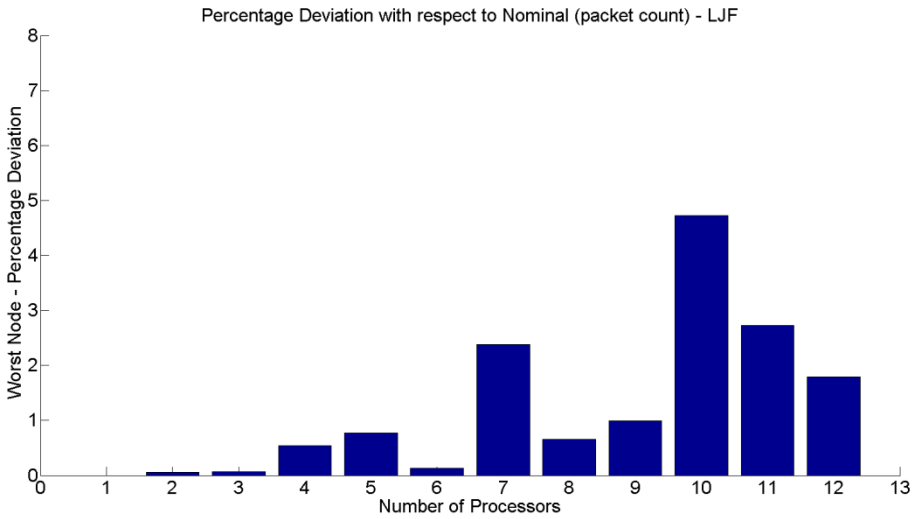

Figure 39: Deviation of Bathymetric Packets with LJF (DS-1)

For DS-2, LJF scheduling resulted in very significant improvements. The main aim of LJF was to solve the problem of load balancing amongst the nodes, by minimizing the deviation in the number of bathymetric packets processed by the compute nodes. As seen in Figure 40, the worst percentage deviation for LJF was around 0.3% as compared to almost 25% for FCFS. A more balanced system resulted in near ideal speedup and efficiency in the unlimited bandwidth, multiple access (best case) scenario. For exclusive access, the performance was slightly lower owing to the resource contention as a result of exclusive access to the SAN. However, this performance was higher than that observed during the FCFS (Figure 41, Figure 42)

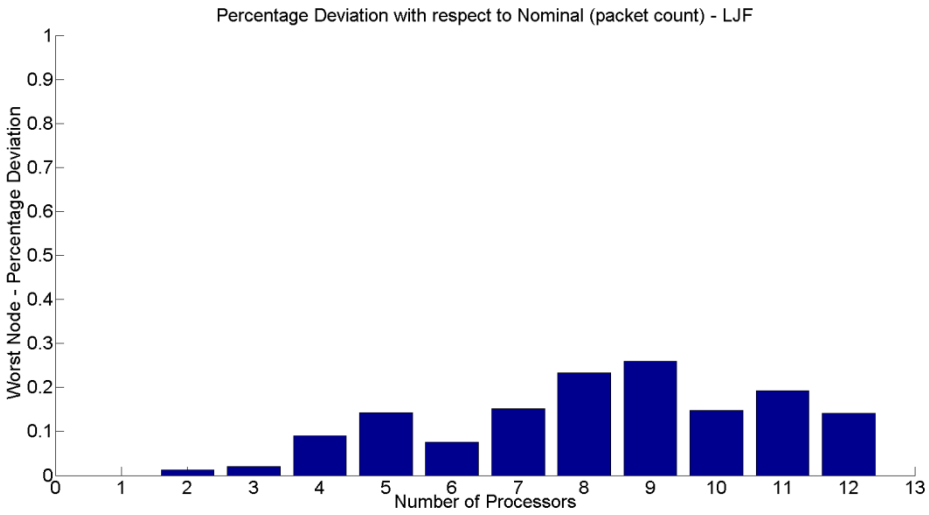

Figure 40: Deviation of Bathymetric Packets with LJF (DS-2)

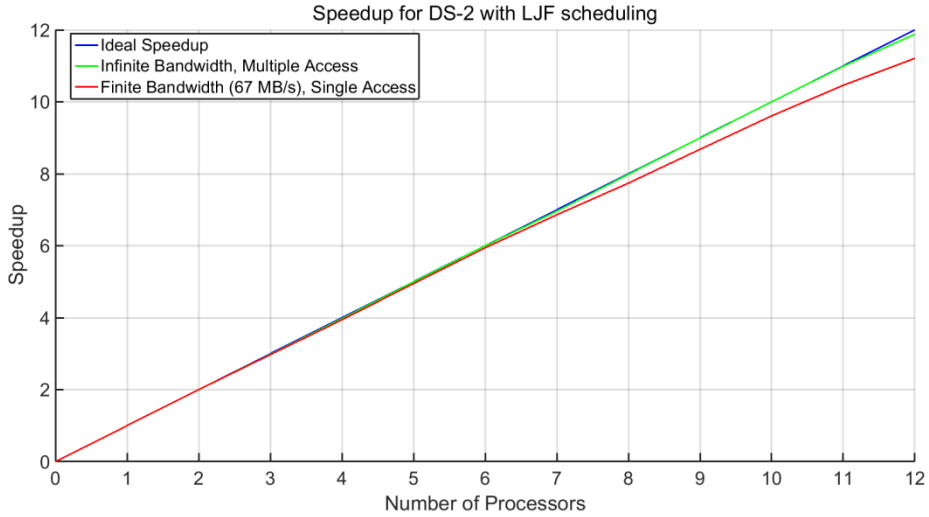

Figure 41: Speedup for DS-2 with LJF Scheduling

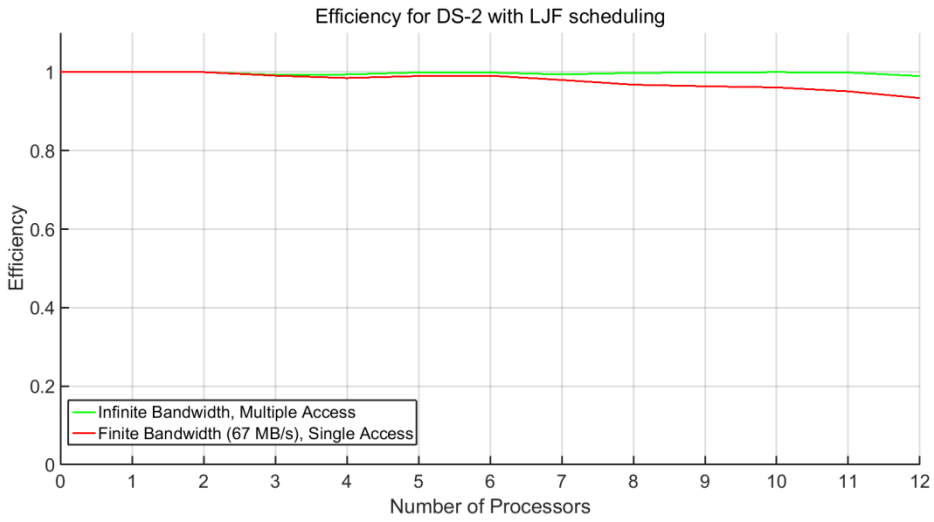

Figure 42: Efficiency for DS-2 with LJF Scheduling

LJF scheduling enables a finer granularity for load balancing as compared to FCFS scheduling. As a result, there was a very significant reduction in the deviation of bathymetric packets in the compute nodes when processing DS-2. This translated to better speedup and efficiency for processing. Furthermore, LJF was successful in improving the balance of distribution of bathymetric packets for DS-1. However, LJF scheduling does not address the core issue of performance problems for this dataset. Network congestion remained a bottleneck for the processing of DS-1 and hence performance gains, through better node balancing, were low.

4.5.3 Contention Reduction Scheduling

Both FCFS and LJF scheduling exhibit poor performance in the limited bandwidth and exclusive ownership scenario for the SAN. During this scenario, there is contention among the compute nodes for obtaining access to the storage resource, which results in a considerable period of time being spent on the wait queue for each task. The properties of the input dataset determine the severity of this resource contention. A detailed analysis is provided in the observations section of this chapter.

The CR algorithm aims to reduce the contention at the SAN by reducing the number of compute nodes that compete for this shared resource. The algorithm first finds the optimal number of compute nodes for maximum speedup for a dataset, and limits the distribution of the tasks of this dataset to that subset of compute nodes. The distribution of tasks on this subset of compute nodes is done using the LJF scheduling, to allow for workload balancing. By avoiding ineffective distribution of tasks across all available compute nodes, it improves the efficiency of the parallel processing system.

For testing the performance of the CR algorithm, the compute cluster was given the input datasets DS-1 and DS-2 and was run under the condition where the SAN bandwidth was limited, with exclusive access. This is the worst case scenario under which the datasets were tested. The speedup and efficiency was compared to the performance of LJF scheduling, under the same input datasets and SAN constraints.

Figure 43 and Figure 44 demonstrates the performance improvements due to the CR scheduling, as compared to LJF scheduling. For up to three nodes, the speedup is similar for both scheduling algorithms, but as the number of compute nodes increases beyond three, the divergence in their performance becomes more pronounced. For 12 nodes, a speedup of 7.89 was achieved with an efficiency of 0.66 using CR scheduling, as compared to a speedup of 6.17 and efficiency of 0.51 for LJF scheduling.

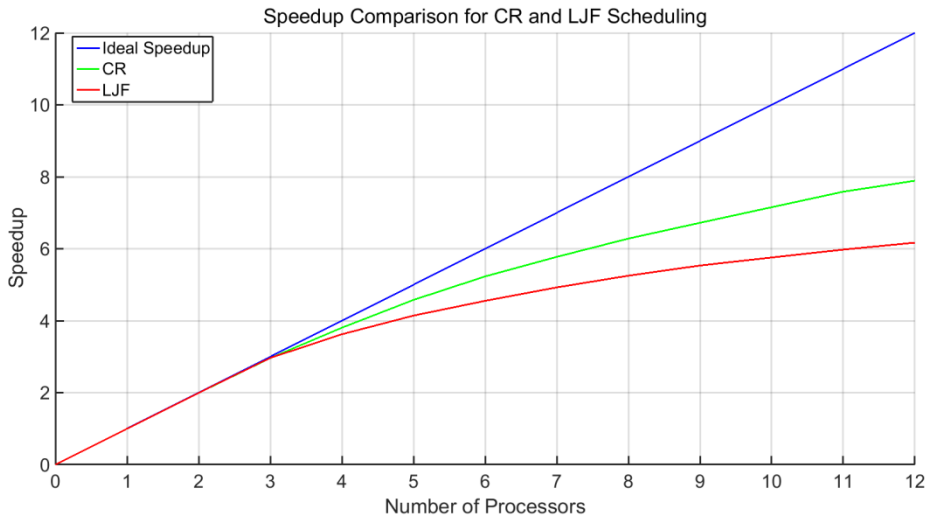

Figure 43: Speedup Comparison for CR and LJF Scheduling

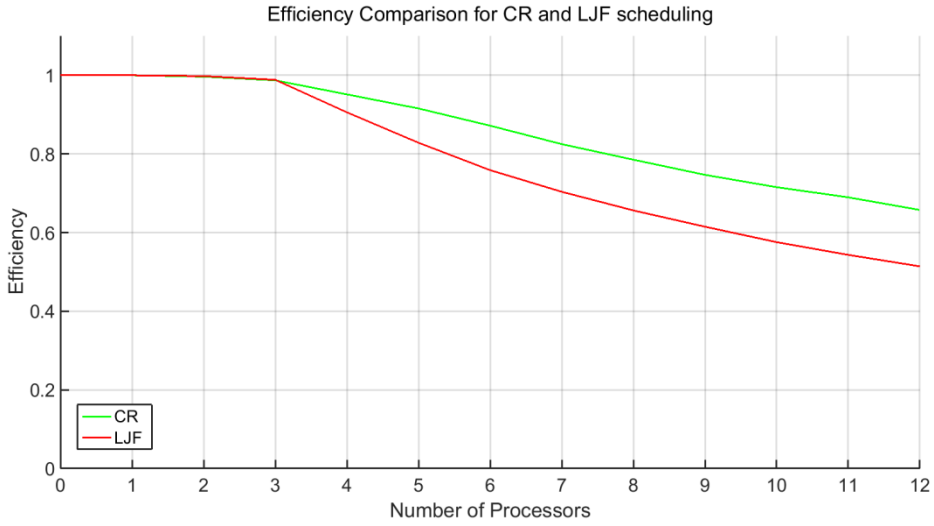

Figure 44: Efficiency Comparison for CR and LJF Scheduling

4.5.4 Observations

The FCFS scheduling algorithm was the easiest to implement and incurs the least overhead during execution. The lack of prioritization means that all the tasks submitted to the processing system will eventually complete, thereby avoiding starvation. However, the performance of FCFS scheduling is dependent on the order of tasks/jobs that are submitted to it. This was the main reason for the disparity between the observed performance of this algorithm on DS-1 and DS-2.

Figure 45 illustrates the clustering of files, in terms of the number of bathymetric packets contained in them. The files are ordered in the sequence determined by the operating system and hence are the same as received by the parallel processing system when using the FCFS algorithm. In this default ordering, the files in DS-1 show clustering of files that contain similar number of bathymetric packets. Additionally, each region of the dataset has a range of bathymetric packets within which most of the files fall. Finally, the tail end of the sequence of the raw data files is roughly uniform, in terms of bathymetric packet count.

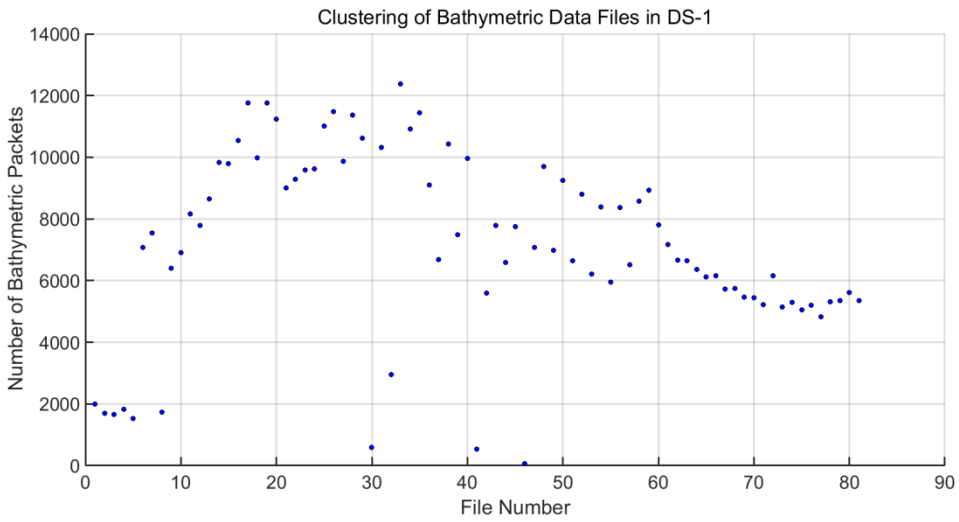

Figure 45: Clustering of Bathymetric Data Files in DS-1

As a result, the FCFS algorithm is able to balance the raw data files fairly well across the compute nodes, leading to low deviation from the expected distribution of bathymetric packets (Figure 33). In the case where the SAN can be simultaneously accessed by multiple nodes, through the network pipe with unlimited bandwidth, the clustering and uniformity of the raw data files do not pose any issues. However in the opposite case, where bandwidth is limited and SAN access is exclusive, this uniformity hinders the performance of the system. The raw data files submitted to the compute nodes require roughly the same time for conversion. As the nodes complete processing the files, they issue a request to the SAN for the transfer of next file to be processed, at approximately the same time. These data access

requests result in resource contention. This contention becomes worse as the number of compute nodes is increased and more nodes get stuck in the queue, waiting for disk access. This is the cause of the stark contrast between the performances in the best and worst case scenarios for DS-1.

Figure 46 illustrates the variation in number of the bathymetric packets in DS-2. The default ordering of the files shows the absence of clustering, as compared to DS-1, and a high disparity in the count of bathymetric packets of adjacent files. The behavior is seen throughout the stream of the data files, with some of the largest files coming towards the tail end.

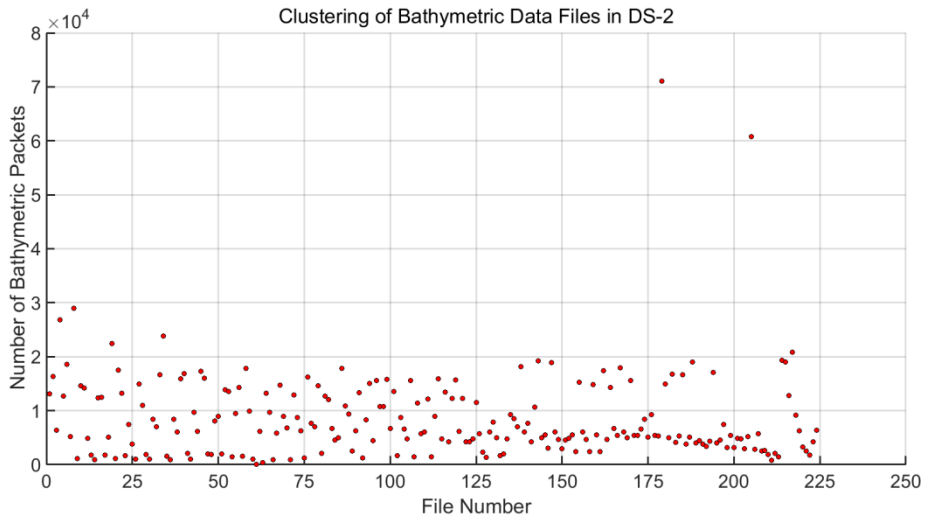

Figure 46: Clustering of Bathymetric Data Files in DS-2

The disparate nature of individual files in DS-2 and their ordering helped to circumvent any resource contention that might have occurred in the exclusive access scenario, as was observed in the experiment on DS-1. However, the factors of file diversity and randomness in workload resulted in a sub-optimal distribution of workload across an increasing number of compute nodes. The presence of some larger sized files towards the end of the dataset skewed the balance of data packets on the compute nodes, which became more

prominent as the number of compute nodes increased. This results in diminishing efficiency as the processing capacity is increased.

For FCFS, a stream of a mix of small and large tasks aids in the load balancing amongst the compute nodes. On the other hand, even a small variation in this scenario, for instance, having a few large files towards the end of the stream can skew the balance of workload on the compute nodes, thereby resulting in lower throughput.

To overcome the issue of balancing of bathymetric packets across the compute nodes, LJF scheduling was chosen. This algorithm sorts the stream of files to be processed in descending order of their number of bathymetric packets. Figure 47 shows the streams of files from both datasets, DS-1 and DS-2, after LJF scheduling has sorted them.

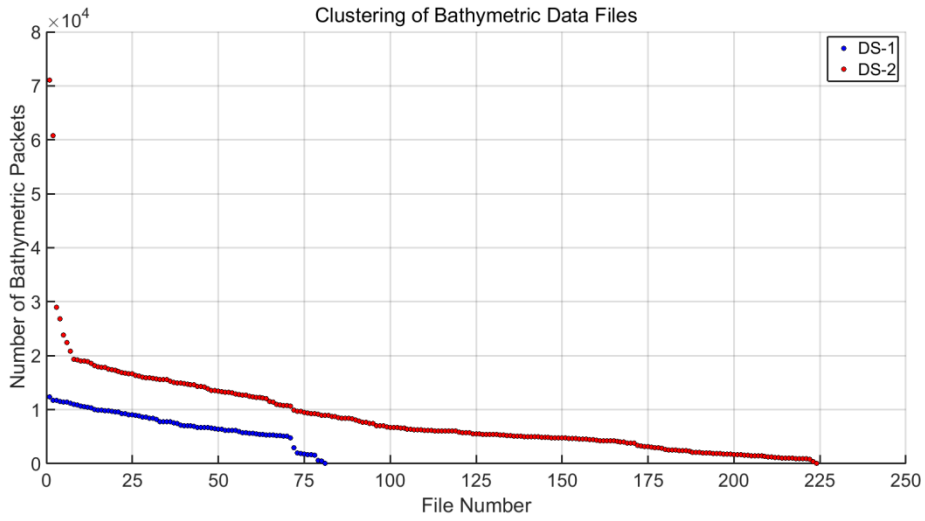

Figure 47: Clustering of files in DS-1 and DS-2

The effect of LJF scheduling was more prominent in DS-2 as compared to DS-1, as the range of bathymetric packets in DS-2 is significantly larger. The default OS-sorted stream for DS-2 had files with high disparity of number of bathymetric packets, adjacent to each other, and large files were located towards the end of the stream. With LJF scheduling, the prioritization of the largest tasks in the stream of incoming tasks allows the load balancer to have a finer granularity of control on the distribution of processing load. This allows for a

balanced distribution of bathymetric packets across the compute nodes. The final result is significant improvements in speedup and efficiency.

Running LJF scheduling on DS-1 produced a more balanced distribution of files in the data stream as compared to FCFS scheduling. However, the network congestion issue, in the exclusive SAN access scenario, that causes performance degradation for DS-1 was not addressed by LJF scheduling. Hence, the gains in performance were very small.

When access to the SAN is exclusive, then for every dataset there is a limit to maximum speedup that can be achieved. In an ideal scenario of a parallel processing cluster with infinite compute resources, the only wait time incurred is when the compute nodes wait to have exclusive access to the SAN. This means that each node will be able to transfer data from the SAN when the previous node has completed its data request. Consequently, the final node will begin processing when the total data in the dataset has been transferred to the compute nodes. Hence, the smallest total conversion time possible for the dataset will be the sum of the time required to transfer all the data to the compute nodes and the processing time for the last node. This total conversion time denotes the smallest possible processing time and therefore the maximum possible speedup that can be achieved for the dataset, without overlapping I/O.

For the infinite compute resource scenario, the complete data in DS-1, which is 93.01 GB, can be transferred in 1421.59 s, over iSCSI, operating at 67 MB/s. The speedup achieved, as compared to serial processing, would be 2.98. Similarily, for DS-2, the dataset of 22.57 GB can be transferred in 344.94 s over the same network using iSCSI. This results in a speedup of 27.88. These speedups for DS-1 and DS-2 would be the maximum achievable within the constraints of exclusive access to the SAN.

In the case of a finite number of nodes, the time spent by the compute nodes waiting for access to storage decreases the efficiency of the parallel processing system. The greater

the difference between the transfer time and conversion time, with the transfer time being larger, the lesser the contention for exclusive access to the SAN. This is because the large conversion time for a particular node will overlap with the data transfer time for other nodes. When the conversion is complete, the probability is high for the network channel to be free or to have a reduced wait-time, as all or most of the other nodes are in the data conversion stage. Hence, increasing the number of compute nodes does not always result in increased throughput, but might result in decreased efficiency.

As Figure 48 illustrates, there are significant differences between the data transfer times and processing times, for the files in DS-1 and DS-2. For files in DS-2, in general, the processing times are an order of magnitude greater than the transfer times. However, for files in DS-1, the processing times are a small multiple of the transfer times.

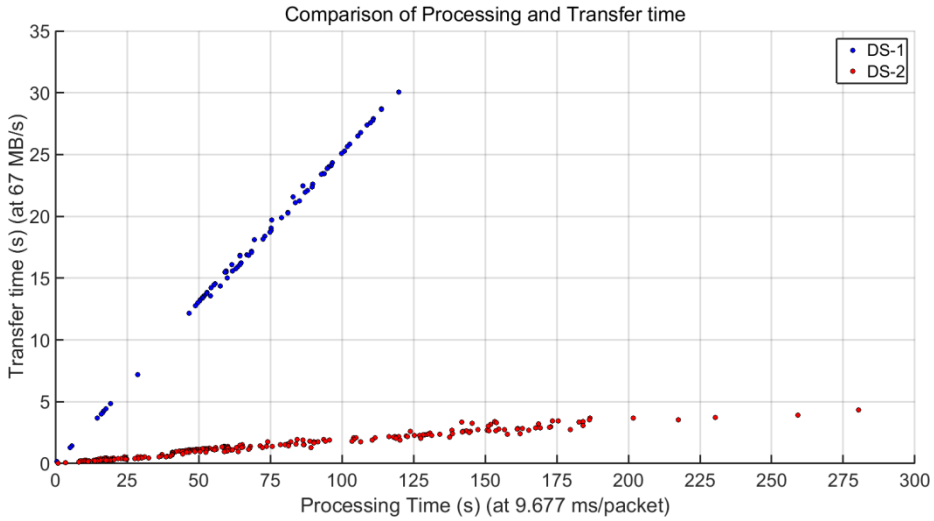

Figure 48: Processing and Transfer times for DS-1 and DS-2

As a result, the speedup limit for DS-1 is reached when the number of compute nodes is three (Figure 37). Increasing the number beyond three does not produce any improvements in speedup, as a greater number of nodes compete for access to the SAN. This increases the compute node wait time, and hence reduces the efficiency of the parallel processing system. This is the reason for the plateau for speedup, and decrease in efficiency observed when

processing DS-1 with more than three compute nodes. Owing to the intrinsic characteristics of DS-2, increasing the number of compute nodes improves the speedup as the small transfer times and large processing times minimize the contention for storage access. Therefore, for up to twelve compute nodes, a higher compute capacity translates to higher speedup while maintaining high efficiency.

From the performance of FCFS and LJF, it was clear that increasing the compute capacity of the parallel processing cluster does not necessarily increase the speedup for every dataset. The CR scheduling algorithm is designed to determine the unique characteristics of each dataset and adjust the scheduling of its tasks accordingly. This algorithm ensures that maximum speedup is obtained while maintaining high efficiency. This becomes more important when the parallel processing system handles multiple datasets and can, hence, avoid one particular dataset reducing the speedup and efficiency for all other datasets.

As mentioned earlier, the minimum processing time for DS-1 is 1421.59 seconds, for a cluster with infinite compute nodes. The CR algorithm uses this information and computes the number of nodes required to achieve this speedup, which is three compute nodes. This result is verified through experiments with LJF where the maximum speedup and efficiency combination for DS-1 was achieved with three compute nodes. The CR algorithm uses LJF scheduling to distribute the processing tasks among three compute nodes, irrespective of the number of available compute nodes. For DS-2, the calculated limited on compute nodes is 28 and hence, tasks could be assigned across all available compute nodes. For each experiment, DS-1 and DS-2 were processed in order, while increasing the number the number of compute nodes from three to twelve. These experiments were performed for LJF and CR scheduling. Since the distribution of tasks for DS-1 is only limited beyond three computes nodes, the performance of both algorithms was very similar as the compute nodes was increased from one to three. However, beyond three compute nodes, the speedup and efficiency advantage

for the CR algorithm began to increase. The conversions of tasks in DS-1 are limited to three compute nodes and avoid ineffective distribution to all available nodes. This prevents the extra contention for access to the SAN and improves the system efficiency. The effect of this reduced contention becomes more pronounced as the number of compute nodes is increased.

4.6 Conclusion

The effectiveness of a scheduling algorithm used for bathymetric data conversion in the parallel processing cluster depends on its ability to adapt to the variations in the input datasets. The datasets, DS-1 and DS-2, used in the work vary significantly in a variety of factors such as the data packet density, relation between transfer and conversion times, file sizes, etc. The factors play an important role in the how the various resources of the parallel processing system are utilized, during processing of these datasets. The difference in performance for DS-1 and DS-2 in FCFS signified the effect of the dataset variations on the simple scheduling algorithm. The OS-specified ordering of tasks causes the nodes to be imbalanced in FCFS scheduling. The balancing of computational tasks across the compute nodes is improved by the LJF algorithm, thereby increasing the speedup and efficiency. However, due to nature of DS-1 the gains were very limited. CR scheduling reduces the speedup and efficiency loss by avoiding the ineffective distribution of processing tasks across all available compute nodes. This improves the efficiency and speedup, especially in cases when the parallel processing system is handling multiple input datasets.

CHAPTER 5

ROBUSTNESS

One of the key drivers of this research was enabling raw bathymetric data processing on board survey vessels. As discussed earlier, the capability of processing raw data in near-real time provides many advantages as compared to traditional processing methods. It allows for analysis of the bathymetric data as it is being collected and calibration of acoustic devices according to quality specifications. As these activities are performed at or near the survey area, the economic benefits can be significant. This near-real time processing capability enables identification of new areas of interest, should they arise through the data analysis. Adding the parallel processing paradigm to this system allows for increased on-demand scalability, which means capacity can be added and removed with minimal effort. Shorter processing turnaround times allow for increased efficiency for funding agencies and faster access to data for the scientific and user community.

Designing a computational system intended to be used on board survey vessels presented some unique challenges. During an expedition, a survey vessel has limited computational and human resources. Due to the remoteness of survey locations and limited technical expertise on board the survey vessel, the reliability of these computational resources should be very high. As a result, when designing the parallel processing system as part of this work, it was important that the architecture be able to tolerate errors in a fail-safe manner. This meant that the system had to detect any faults, automatically perform steps to mitigate

the impacts of the faults, and perform steps to rectify the issues while continuing processing operations, possibly in a degraded mode. If manual intervention is required, the system should notify the administrator who would then perform necessary actions to rectify the error and bring the system back to its full operational capacity. Also, it is important that performing these recovery and maintenance actions does not require a dedicated individual, with specialized technical expertise, on board the survey vessel.

Taking into consideration the limited space and resources on board a survey vessel and to minimize the installation costs, the system was designed so as to consist of Commercial-Off-The-Shelf (COTS) components which would not require custom apparatus and exclusive rack space. The use of COTS equipment for processing needs allows for the sharing of compute resources (during idle cycles) as required and easy replacement of damaged or non-functioning parts, thereby reducing the operational and maintenance costs.

The software of the processing system needed to be designed in such a way that it could tolerate any issues arising from hardware and software malfunctions. Various points of failure were identified as part of the robustness analysis. The following subsections describe each failure scenario, its severity, and solutions that would ensure high availability of the parallel processing system.

5.1 Compute Node Failure

In the parallel processing system, the compute nodes are responsible for conversion of raw bathymetric data to the finished product. On a system installed on the survey vessel, these would most likely consist of blade servers or other forms of hosts that could support the bathymetric transformation package. These compute resources will be visible and accessible to the head node, which would submit processing tasks to them. The issue arises when one or more of the compute nodes go offline, due to a hardware or software malfunction. The

parallel processing system would need to handle such a situation, due to which the overall compute capacity is reduced and the processing tasks assigned to the failed node are, now, without a compute resource.

One solution for error recovery is for the head node to attempt a power-cycle at the affected node. If this action fails, error recovery would require manual intervention, which would involve repairing or replacing the affected compute node. After the error is rectified, the administrator of the processing system would need to set the availability of the compute node in the head node database. The compute node will then be added to the pool of available compute resources available for performing bathymetric processing, which completes the error recovery procedure.

This implementation of the parallel processing system includes a watchdog thread that is responsible for monitoring the health of each compute node in the system. During the fault identification and recovery process, to ensure high availability, the parallel processing system will recognize the malfunctioning compute node through the watchdog thread. Each node periodically updates it status in the node table, through a message, indicating to the system its availability (up/down). If the node table indicates an error status with any of the compute nodes, the watchdog thread would reassign all the pending processing tasks from the faulted node to the remaining available compute nodes. This redistribution of workload is done in accordance to the scheduling algorithm being used by the parallel processing system. This allows the system to be available and continue its processing operations, albeit at a lower capacity, until the fault is rectified.

The following simulation experiment demonstrates the working of the fault tolerant capabilities of the parallel processing system when a compute node failure occurs. A parallel processing system with five compute nodes is used to process a dataset (DS-1 and DS-2). Compute node 3 is set to offline state in the midst of the execution. The graphs for each

dataset show how the system responds to this event and rebalances the processing load across the remaining compute nodes.

Figures 49-52 illustrate the behavior of the system, for DS-1, at different points in time during the dataset processing. In each plot, the line represents the ideal distribution of bathymetric packets when all five nodes are operational and the bars represent the distribution of bathymetric packets in the compute node failure case. Figures 53-56 represent the results for DS-2.

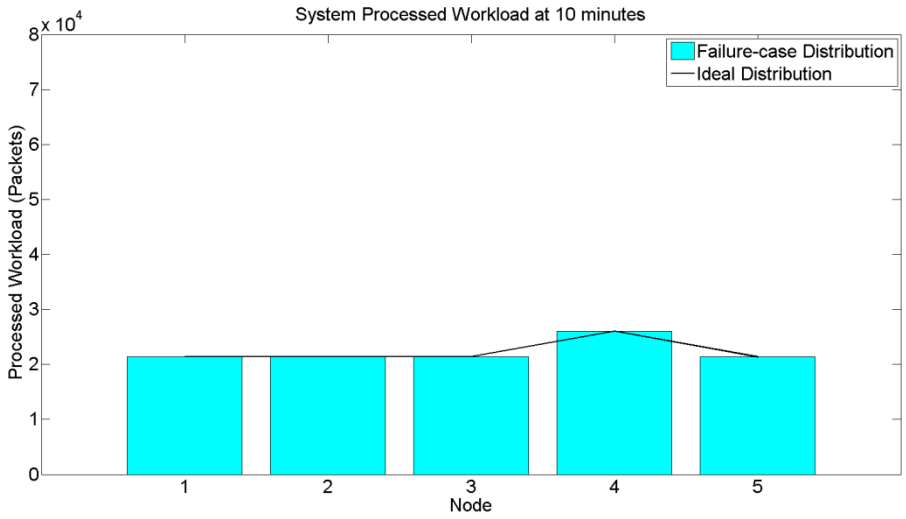

Figure 49: DS-1 Processed Workload at 10 minutes

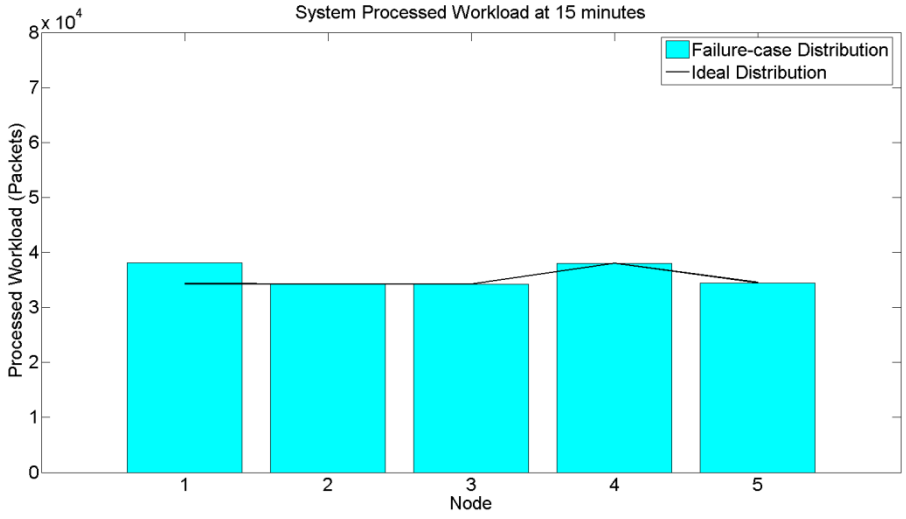

Figure 50: DS-1 Processed Workload at 15 minutes

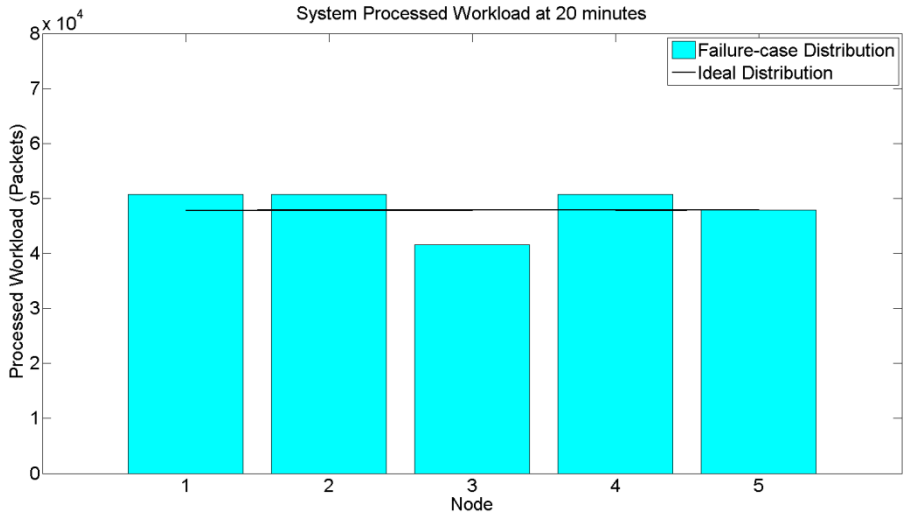

Figure 51: DS-1 Processed Workload at 20 minutes (Node 3 Fail)

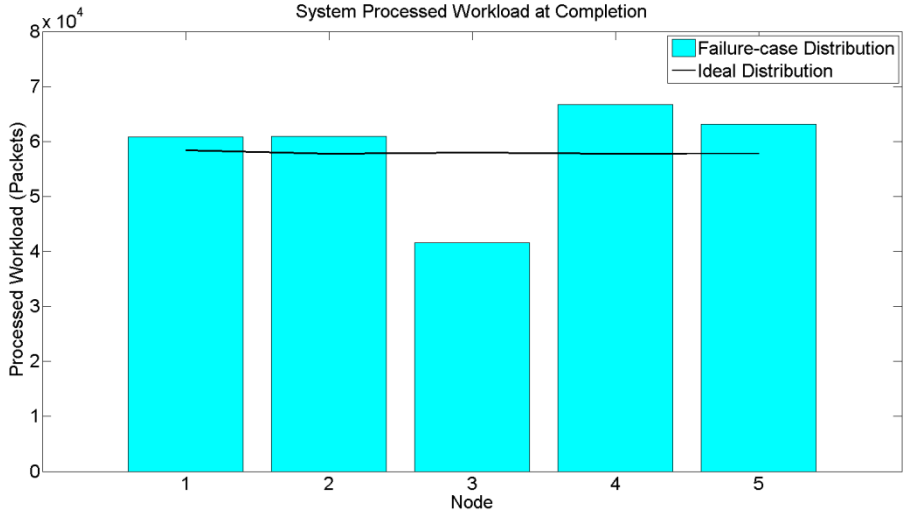

Figure 52: DS-1 Processed Workload at Completion

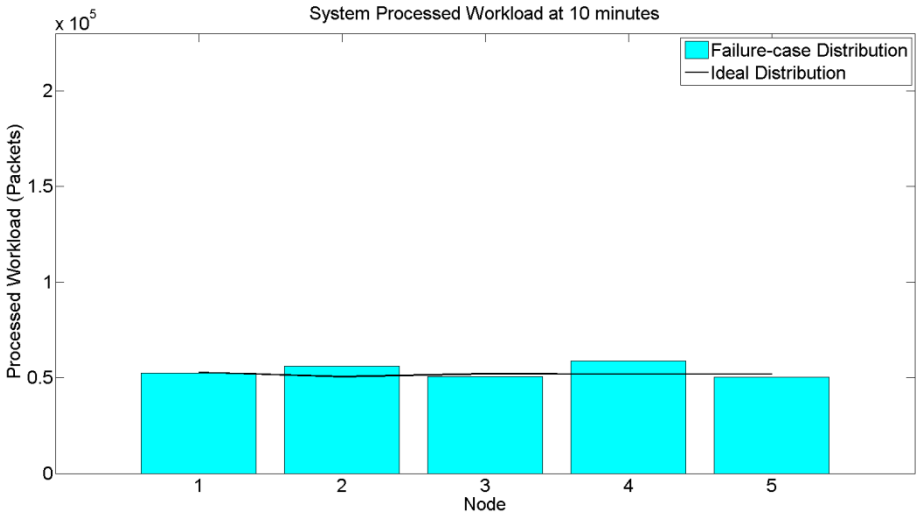

Figure 53: DS-2 Processed Workload at 10 minutes

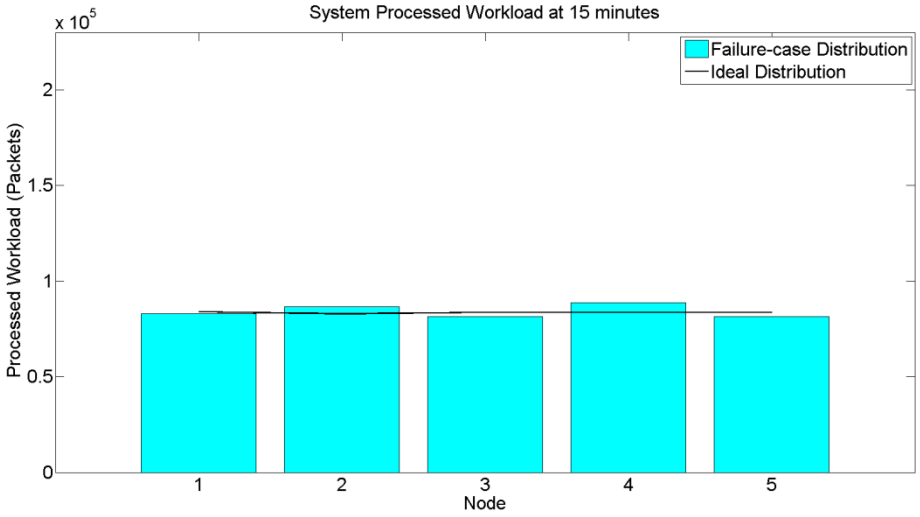

Figure 54: DS-2 Processed Workload at 15 minutes

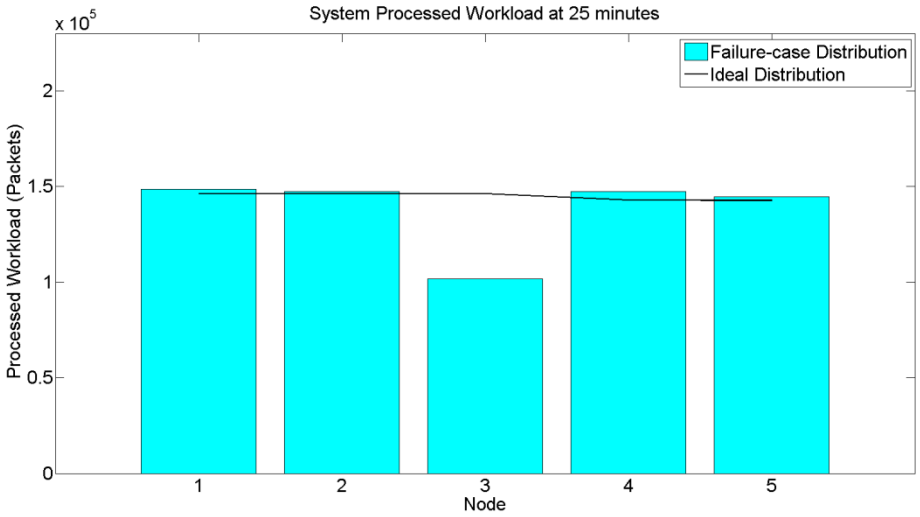

Figure 55: DS-2 Processed Workload at 25 minutes (Node 3 Fail)

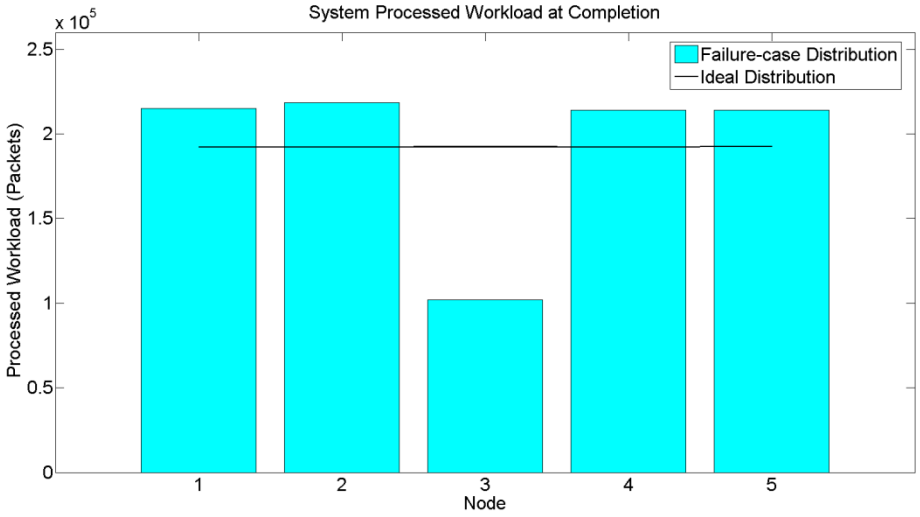

Figure 56: DS-2 Workload at Completion

For both the datasets, the processing system was able to move the pending workload (including files whose results have not been written back to the SAN) for the failed compute node to the available compute nodes. This action allows the processing system to be tolerant to compute node faults and remain operational while the fault is being rectified.

5.2 Head Node Failure

The head node houses the intelligence of the parallel processing system. It is responsible for accepting job requests from the user, estimating the workload contained in

each task, distributing the processing of these tasks across the compute nodes and, in a production system, collecting the results and notifying the user regarding job completion and any associated errors. Hence, the head node going offline hinders every aspect of the functioning of the parallel processing system.

To recover the state of the parallel processing system, all in-memory data structures would be written to disk periodically, which in turn would be replicated over the network for additional redundancy. In the current implementation, the contents of the task queue are written to an SQL table at regular intervals of time. On system start, the head node is designed to query this table and transfer its contents to the in-memory task-queue. This allows the head node to recover state information and resume processing of the tasks that it was handling at the time of the failure. As the system expands to perform more functions, for instance collecting and assimilation of processing results, the associated data structures can also be backed up on to the disk.

Backup and recovery of the in-memory data structures can be used for error recovery and fault tolerance in the parallel processing system. There are numerous ways to recover from a head node failure, each depending on the severity of the fault. The production system can have a watchdog running on another host which would perform a hard power cycle of the head-node, which might solve some transient operating system or disk issues. Additionally, the system architecture can be designed to incorporate a redundant head node which will take over responsibilities of the primary head node if it goes offline. The auxiliary head node can retrieve the data from disk, construct its own data structures and resume operation of the parallel processing system. Another, simpler, solution would be to recover or replace the head node. This solution does not provide high availability but ensures error recovery. The implementation and analysis of a head-node watchdog or multi-head node parallel processing system is beyond the scope of this work.

5.3 Database Failure

In addition to the in-memory data structures, the SQLite database is another point of failure within the parallel processing system. The database consists of tables holding information regarding the user request, compute nodes, jobs, etc. The loss of the database would cause the head node to be unable to perform activities like accept and process user requests, schedule and balance existing tasks, monitor the compute node status, etc., thereby causing the parallel processing system to become unavailable.

The following list, while not exhaustive, describes the most frequent scenarios that could result in database corruption:

- Power failure on the disk hosting the database, rendering it inaccessible.
- Operating system glitch causing the database to become corrupt
- User-induced corruption, though highly unlikely, is still a possibility. This can be due to database deletion, table deletion or modification, etc. The probability of this scenario increases as the system becomes more complex to handle additional features.
- Media failure which could be caused by disk head or controller failure, resulting in the database becoming inaccessible.

To deal with corruption issues, SQLite provides a set of APIs that allows backing up of a live database to an auxiliary location, as chosen by the administrator. This backup process is minimally disruptive as the backup thread would only hold a read-lock on the primary database for short intervals of time [16]. This avoids a large performance penalty on the working of the parallel processing system. The location of the database backup depends on the type of fault tolerance required. If the system is designed to have a redundant head node, the backup database should be local to the redundant head node or in a network drive accessible to it. This ensures high availability of the system in case of head-node or database failure. For error recovery, the backup database should be located on a network drive that is

accessible to the administrator of the parallel processing system. Error recovery, though comparatively simpler to implement than high availability, requires more manual intervention to bring up the parallel processing system. The administrator would have to find the root cause of the database corruption, take necessary steps to prevent its reoccurrence or swap out faulty equipment and then copy over the backup database to the head node so that it can resume operations.

5.4 Conclusion

The study of robustness is critical because one of the goals of this work is to enable the deployment of the parallel processing system on board survey vessels and hence provide near-real time processing of raw bathymetric data. Limited computational and human resources mean that the system must be designed to ensure high availability and fault tolerance. The previous sections have identified the various points of failures within this proof-of-concept implementation of the parallel processing system. The failure of non-critical resources, like compute nodes, are handled by a watchdog thread which can reconfigure and rebalance the processing capacity of the system and perform basic error recovery actions, like hard or soft power cycles. Faults in system critical components like the head-node, its disks or database require a more complex mechanism to ensure availability and error recovery. Backing up state information and volatile data structures, and replication of this data would ensure the recovery of state in case of an anomaly. This kind of error recovery would involve manual intervention and would necessitate system downtime. Adding redundant head-nodes and databases (with synchronous database mirroring) would ensure high availability in this situation. A production parallel processing system would involve many more components and hence introduce new points of failure. However, the strategies for high availability and fault

tolerance would be the similar to the ones highlighted for the cases above and would revolve around the concepts of backup, replication and redundancy.

CHAPTER 6

CONCLUSION

This research has demonstrated the concept of parallel processing of bathymetric data, through a solution that is cost-effective, efficient and robust. This proof-of-concept implementation shows the feasibility of generating a highly accurate estimate of the computational requirements for processing individual raw bathymetric data files. Combining these estimates with other workload metadata allows a variety of algorithms to distribute the workload amongst the compute nodes, depending on requirements of speedup, efficiency and throughput. Robustness is incorporated into the design through critical data redundancy, fault tolerant design and error recovery mechanisms.

The work presented here opens up opportunities for a new parallel processing architecture, which is a departure from the constraints of the current, workstation-based serial implementation for processing bathymetric data. A cluster of network attached compute nodes can be created using COTS components, which could include high performance, low footprint blade servers or existing workstations. The stochastic estimation algorithm for XTF bathymetric data files provides a very fast (under 51 milliseconds) and highly accurate (an average percentage error of 0.09%) estimate of the computational workload required for processing the raw data, which in turn enables load balancing algorithms to distribute the workload across the available compute resources. The current implementation highlights the working of three such algorithms, each with unique characteristics and varied performance.

The First Come First Served (FCFS) scheduling implementation for this work allows for simplicity and minimum operational overhead but is heavily reliant on the input task pattern for effective load balancing. Longest Job First (LJF) scheduling prioritizes larger tasks, allowing for a more granular workload distribution, and thereby attempting to maximize speedup. Like FCFS scheduling, LJF scheduling is also affected by the dataset properties, the most prominent of which being the difference between the raw data file transfer time and its conversion time, which determines the severity of the contention on the SAN. Contention Reduction (CR) scheduling aims to determine the maximum speedup and efficiency combination for each input dataset and uses this information for a more intelligent distribution of processing workload, consisting of multiple datasets, across the available compute nodes. This avoids ineffective distribution of workload across compute nodes, thereby improving efficiency without sacrificing speedup gains. The system is also flexible enough to accommodate a new scheduling algorithm that fits the specific requirements of the installation, potentially combining efficiency, throughput and turnaround time at various levels.

The ability to work in a resource-limited environment like a survey vessel was one of the primary goals of this research. Hence, robustness was incorporated into the system design. The system can continue functioning, with degraded capacity, when a compute node experiences failure. It automatically and seamlessly transfers any pending processing tasks to the available compute nodes. In-memory data structures and the database are replicated to preserve and recover the system state in case of a head node or disk failure. Although not part of this proof-of-concept design, redundant head nodes, databases and disks could automatically take over in failure cases and prevent any downtime as the repairs to the faulty components are made. Also, installation and maintenance costs for this system would be low because it only requires COTS equipment and can also be run on existing hardware. The fault

tolerant features help this system run without manual intervention, in the face of potential issues. Even in the case of error recovery, the system can be brought back online by replacing the faulty components. Thus, maintaining a production-grade implementation of a parallel processing system would require minimal training and would not require a fulltime technician on board the survey vessel.

Using this proof-of-concept as a reference, a fully functional parallel processing system for bathymetric data can be implemented. This would allow the data processing capabilities to catch up to the tremendous advances in data collection technology, as witnessed over the past few years. It would realize the goal of a powerful, yet affordable, processing platform that enables near-real time processing of bathymetric data. This, in turn, would help to maintain quality of surveys, identifying new points of interest during the survey activity itself. Finally, getting the processed data out to the scientific community faster would accelerate the process of discovery and increase the return on investments for all stakeholders.

To fulfill the above goals, the next best step would be to investigate how the estimation algorithm developed as part of this work can be modified and applied to other formats used for collection of raw bathymetric data. Workload balancing can be improved through development of new scheduling algorithms that prioritize different properties such as speedup, efficiency, administrator-defined ordering, etc. The parallel processing system could dynamically switch between different scheduling algorithms based on system performance feedback or administrator preferences. Developing the capability of the system to enlist the processing capabilities of compute nodes, not primarily part of the compute cluster, will allow the system to deal with sudden spikes in processing demand or reduction in compute capacity. Zero downtime can be achieved by utilizing a combination of multiple head nodes, with synchronously replicated databases, that actively monitor each other’s status and take

over when failure is detected. The implementation of these capabilities will allow for the development of an efficient, robust system that would utilize the developments in computer technologies for a near-real time, on board processing system for bathymetric data.

REFERENCES

1. *Using Lead Lines to Collect Hydrographic Data*. National Oceanographic and Atmospheric Administration.
2. Calder, B.R. and L.A. Mayer, *Automatic processing of high-rate, high-density multibeam echosounder data.* Geochem, Geophys. Geosyst. (G3), 2003. **4**(6).
3. Mayer, L.A., *Frontiers in seafloor mapping and visualization.* Marine Geophysical Researches, 2006(27): p. 7–17.
4. Masetti, G., Calder, Brian R., *Remote identification of a shipwreck site from MBES backscatter.* Journal of Environmental Management, 2012. **111**: p. 44-52.
5. Calder, B.R., *Automatic statistical processing of multibeam echosounder data.* Int. Hydro. Review, 2003. **4**: p. 36-52.
6. Lever, J.A., S. Raffa, D. McCarren, A. Lewando, and M. Carron. *A concept for the Naval Oceanographic Office Survey Operations Center*. in *OCEANS '99 MTS/IEEE. Riding the Crest into the 21st Century*. 1999.
7. Raffa, S. and S. Douglas. *A broadband ship-to-shore communications network for the Naval Oceanographic Office*. in *OCEANS, 2001. MTS/IEEE Conference and Exhibition*. 2001.
8. Gathof, J.M. and J.B. Bassich. *Recent development in the Naval Oceanographic Office survey operation center*. in *OCEANS 2003. Proceedings*. 2003.
9. Hare, R., D.R. Peyton, and J. Conyon, *Remote Processing of Ship Based Hydrographic Multibeam Data.* Hydro 2011, 2011.
10. Peyton, D., J. Conyon, and D. Wardle, *New Concepts in Bathymetric Source Data Storage and Manipulation*. HYDRO2010, Rostock, Germany.
11. Doucet, M., M. Paton, B. Calder, and J. Beaudoin, *From Research to Reality; A GeoCoder Implementation's Past, Present, and Future*. U.S Hydro 2011 Conference, Tampa, Florida.
12. Sienicki, J., P. Agrawal, V. Agrawal, and M. Bushnell. *Superlinear speedup in multiprocessing environment*. in *Proceedings of the First International Workshop on Parallel Processing*. 1994.
13. Willebeek-LeMair, M.H. and A.P. Reeves, *Strategies for dynamic load balancing on highly parallel computers.* Parallel and Distributed Systems, IEEE Transactions on, 1993. **4**(9): p. 979-993.
14. Huh, E.-N. and L.R. Welch, *Adaptive resource management for dynamic distributed real-time applications.* J. Supercomput., 2006. **38**(2): p. 127-142.
15. *eXtended Triton Format (XTF) Manual (Rev -33)*. Triton Imaging Incorporated.
16. *SQLite Backup API*. 2014 [cited 2014 20 September]; Available from: http://www.sqlite.org/backup.html.