IOWA STATE UNIVERSITY
Digital Repository

Graduate Theses and Dissertations

Iowa State University Capstones, Theses and Dissertations

2016

Efficient computation and communication management for all-pairs interactions

Cory James Kleinheksel
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/etd

Part of the Computer Engineering Commons

Recommended Citation

Kleinheksel, Cory James, "Efficient computation and communication management for all-pairs interactions" (2016). *Graduate Theses and Dissertations*. 14969.
https://lib.dr.iastate.edu/etd/14969

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

المنارة للاستشارات

www.manaraa.com

Efficient computation and communication management for all-pairs interactions

by

Cory James Kleinheksel

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:

Arun K. Somani, Major Professor

Shashi Gadia

Manimaran Govindarasu

Suresh Kothari

Srikanta Tirthapura

Iowa State University

Ames, Iowa

2016

Copyright © Cory James Kleinheksel, 2016. All rights reserved.

المنارة للاستشارات

www.manaraa.com

DEDICATION

With all of my love, I dedicate this to my wife, parents, and sisters.

Thank you for all of the love, support, and encouragement!

나의 아내 은이, 아주 많이 사랑합니다.

하늘만큼 땅만큼 떡만큼!

www.manaraa.com

TABLE OF CONTENTS

LIST OF TABLES . . . vii

LIST OF FIGURES . . . ix

ACKNOWLEDGEMENTS . . . xiv

ABSTRACT . . . xv

CHAPTER 1 THE PROBLEM . . . 1

1.1 Communication Application . . . 3

1.2 Computation Applications . . . 4

1.2.1 Acceleration of applications . . . 4

1.2.2 Relaxing the all elements present requirement . . . 5

1.2.3 Scaling all-pairs algorithms . . . 6

1.3 Contributions . . . 7

1.3.1 Quorums . . . 7

1.3.2 Computation . . . 7

1.3.3 Communication . . . 7

CHAPTER 2 ALL-PAIRS COMPUTATIONS . . . 9

2.1 Database Joins . . . 9

2.1.1 Algorithm implementations . . . 10

2.1.2 Data structures . . . 12

2.1.3 Distributed database joins . . . 13

2.2 Spatial Databases . . . 15

2.3 N-Body Problems . . . 15

2.3.1 Approximation of forces . . . 16

2.3.2 Parallelizing the computation . . . 17

المنارة للاستشارات

www.manaraa.com

2.4 Metagenomics 22
2.4.1 Contributions from literature 22
2.4.2 Challenges 23
2.5 Gene Co-Expression Networks 23
2.5.1 PCIT introduced 24
2.5.2 Parallel PCIT algorithm using MPI 26
2.5.3 Parallel PCIT algorithm using OpenMP 30
CHAPTER 3 OPTICAL COMMUNICATION NETWORKS 36
3.1 Network Model 36
3.2 Light-Trails 37
3.3 Light-Trails, Cycle Routing, and Fault Tolerance 39
3.4 Quorums Sets for Routing 41
3.4.1 Point-to-point traffic 42
3.4.2 Multicast traffic 42
3.4.3 Broadcast traffic 43
3.4.4 Efficiency analysis 43
CHAPTER 4 ALL-PAIRS PROBLEM 44
4.1 General All-Pairs Problem Definition 44
4.2 Distributed All-Pairs Problem Definition 45
CHAPTER 5 QUORUMS AND CYCLIC QUORUMS 48
5.1 Defining Quorum Sets 49
5.2 Defining Cyclic Quorum Sets 50
5.3 All-Pairs Property for Quorum Sets 51
5.3.1 All-pairs property 52
5.3.2 Cyclic quorums have the all-pairs property 52
5.4 Redundant Cyclic Quorums Sets 57

المنارة للاستشارات
www.manaraa.com

CHAPTER 6 ALL-PAIRS APPLICATIONS IN COMPUTATION OPTIMIZATIONS . . . 62
6.1 Bioinformatics PCIT Application . . . 62
6.2 Test Setup . . . 62
6.3 Results . . . 64
6.3.1 Memory usage performance . . . 64
6.3.2 Runtime execution performance . . . 66
6.4 Adding Computation Management Logic . . . 70
6.4.1 Impact of cyclic quorum size . . . 70
6.4.2 Computation management logic . . . 73
6.4.3 Impacts of managing quorum set all-pairs computations . . . 77
CHAPTER 7 ALL-PAIRS APPLICATIONS IN FAULT TOLERANT OPTICAL COMMUNICATION OPTIMIZATIONS . . . 84
7.1 Fault Model . . . 84
7.2 Paired Cycle Fault Simulation . . . 85
7.3 Improving Fault Tolerance . . . 88
7.3.1 Additional cycle fault protection . . . 88
7.3.2 Modifying the cycle routing algorithm . . . 90
7.3.3 Redundant cyclic quorums sets . . . 91
7.4 Redundant Cyclic Quorums Set - Paired Cycle Network Analysis . . . 91
7.4.1 Fault-free operational analysis . . . 92
7.4.2 Fault tolerance operational analysis . . . 93
7.5 Redundant Cyclic Quorums Set - Single Cycle Network Analysis . . . 96
7.5.1 Fault-free operational analysis . . . 96
7.5.2 Fault-tolerant operational analysis . . . 99
7.6 Improving Single Cycle Routing Based on Redundant Cyclic Quorums . . . 102
7.6.1 Greedy cycle direction based on missing pairs . . . 103
7.6.2 Greedy missing pairs heuristic results . . . 106

المنارة للاستشارات
www.manaraa.com

CHAPTER 8 CONCLUSIONS 113

8.1 Quorums 113

8.2 Computation Application 114

8.3 Communication Application 114

BIBLIOGRAPHY 116

APPENDIX A OPTIMAL CYCLIC QUORUMS 125

APPENDIX B REDUNDANT CYCLIC QUORUMS 129

المنارة للاستشارات

www.manaraa.com

LIST OF TABLES

Table 6.1 Input Datasets Utilized in PCIT Experiments 63
Table 6.2 Memory Used Per Node (GB) 65
Table 6.3 Average Execution Runtimes (Seconds) 67
Table 6.4 Redundant Work Performed When Quorum Pairs Unmanaged 72
Table 6.5 Managed - Memory Used Per Node (GB) 79
Table 6.6 Managed - Average Execution Runtimes (Seconds) 80

Table 7.1 Paired quorum cycle fault simulation results 88
Table 7.2 Quad quorum cycle fault simulation results 89
Table 7.3 Mean links (95% CI) used by paired cycles based on redundant quorums sets 92
Table 7.4 Percent mean fault coverage (95% CI) of paired cycles using our redundant quorum solution experiencing a single link fault. 94
Table 7.5 Percent mean fault coverage (95% CI) of paired cycles using our redundant quorum solution experiencing two simultaneous link faults. 94
Table 7.6 Mean links used by single cycles compared to paired cycles using our redundant cyclic quorum solution (95% CI) 97
Table 7.7 Mean percent missing node pairs (95% CI) by single cycles using our redundant quorum solution 98
Table 7.8 Percent mean fault coverage (95% CI) of our single cycle, redundant quorum solution experiencing a single link fault. 99
Table 7.9 Percent mean fault coverage (95% CI) of our single cycle, redundant quorum solution experiencing two simultaneous link faults. 101

www.manaraa.com

Table 7.10 Comparing fault-free operation mean percent missing node pairs (95% CI) by single cycles using our redundant quorum solution and greedy cycle direction heuristic 107

Table 7.11 Greedy heuristic mean cycle direction flips while optimizing the redundant quorum single cycle solution (95% CI) 108

Table 7.12 Comparing percent mean fault coverage (95% CI) of our single cycle, redundant quorum solution and greedy cycle direction heuristic experiencing a single link fault. 109

Table 7.13 Comparing percent mean fault coverage (95% CI) of our single cycle, redundant quorum solution and greedy cycle direction heuristic experiencing two simultaneous link faults. 111

Table A.1 Optimal cyclic quorums for $N = 4$ to 111 125

Table B.1 Redundancy = 2, Cyclic quorums for $N = 4$ to 111 130

Table B.2 Redundancy = 3, Cyclic quorums for $N = 4$ to 101 133

المنارة للاستشارات

www.manaraa.com

LIST OF FIGURES

Figure 2.1 Driscoll et al. (2013) communication optimal n-body algorithm's data replication and distribution. Optimality achieved when $c = \sqrt{P}$, resulting in $\sqrt{P}$ teams, $\sqrt{P}$ replication rows, and each processor performing the pairing between two size-$\frac{N}{\sqrt{P}}$ arrays of elements. Note that their algorithm's communication steps are not depicted in this figure. 21

Figure 2.2 High-level summary of PCIT algorithm, gene expression correllation trio computation 24

Figure 2.3 Assigned rows ($index_{start} - index_{end}$) for approximately equal work distribution example for $P_{MPI} = 4$ 28

Figure 2.4 Example of a column-major, non-stride-1 memory access pattern . . . 31

Figure 2.5 Example of a column-major, stride-1 memory access pattern 31

Figure 3.1 Four nodes in a light-trail architecture 38

Figure 3.2 Example light-trail node structure 39

Figure 3.3 Cycle formed using the light-trail architecture 40

Figure 4.1 All-pairs of seven elements. 45

Figure 4.2 Driscoll et al. (2013) communication optimal n-body algorithm's data replication and distribution. Optimality achieved when $c = \sqrt{P}$, resulting in $\sqrt{P}$ teams, $\sqrt{P}$ replication rows, and each processor performing the pairing between two size-$\frac{N}{\sqrt{P}}$ arrays of elements. Note that their algorithm's communication steps are not depicted in this figure. 46

www.manaraa.com

Figure 5.1 A quorum set example with N elements and $P = 4$ processes. Set $\hat{D}$ divides the N elements into $P = 4$ datasets D_0 through D_3. Quorum set Q is then formed from sets (quorums) of these datasets, i.e., S_0 through S_3. 49

Figure 5.2 Defining a relaxed (P, k)-difference set. For a given set $A = \{a_0, \ldots, a_k\}$ and integer P, all integers $0, \ldots, (P-1)$ must be formed from the differences modulus P of integer pairs from set A. Figure (a) shows an invalid difference set corresponding to set $A = \{1, 2, 3\}$ and $P = 6$ because no pair of integer differences modulus 6 form $d \, mod \, 6 = 3$. Whereas Figure (b) with set $A = \{1, 2, 4\}$ and $P = 6$ is a valid difference set because all integer differences modulus 6 are formed, i.e., $0, \ldots, 5$ are all present. . 53

Figure 5.3 A cyclic quorum set example with 7 processes. On the left are the 7 quorums and on the right are all of the dataset pairings. The quorums and the corresponding pairs formed are colored. As Theorem 4 states, all pairs have been covered by a quorum set. 57

Figure 5.4 Defining a relaxed (P, k)-difference set for an $R = 2$ redundant cyclic quorum. For a given set $A = \{a_0, \ldots, a_k\}$ and integer P, all integers $0, \ldots, (P-1)$ must be formed twice from the differences modulus P of integer pairs from set A. Figure (a) shows an invalid difference set corresponding to $A = \{1, 2, 3, 4\}$ and $P = 7$ because integer differences modulus 7 formed $d \; mod \; 7 = 3$ and $d \; mod \; 7 = 4$ only once. Whereas Figure (b) with $A = \{1, 2, 3, 5\}$ and $P = 7$ is a valid difference set for an $R = 2$ redundant cyclic quorum because all integer differences modulus 7 were formed twice, i.e., $0, \ldots, 6$ are all present twice. 61

www.manaraa.com

Figure 6.1 Speedup of our cyclic quorum algorithm using (P) parallel nodes when compared to an optimized single node algorithm. This figure has near identical speedup curves for the computation of three smaller datasets that all fit within the available memory. The log-log scale shows that as additional node resources are added our algorithm scales to utilize the resources. 68

Figure 6.2 Speedup of our cyclic quorum algorithm using (P) parallel nodes when compared to an optimized single node algorithm. This figure has a speedup curve for a larger dataset that would have exceeded the single node memory resources, requiring the use of an alternate lower memory optimized algorithm. Here super-linear speedup is observed as our cyclic quorum algorithm is able to distribute the problem and process the input within the memory constraints. 69

Figure 6.3 A difference set example with 4 processes. This figure with $A = \{1, 2, 3\}$ and $P = 4$ is a valid relaxed difference set because all integer differences modulus 4 are formed, i.e., $0, \ldots, 3$ all occur one or more times. 74

Figure 6.4 A cyclic quorum set example with 4 processes to illustrate the cause and solution to redundant work. This figure has the corresponding cyclic quorum set on the left and the possible all-pairs formed for each on the right. The pairs and quorum are colored with useful work performed, while the uncolored pairs would be redundant work and should not be performed. 75

المنارة للاستشارات

www.manaraa.com

Figure 6.5 Comparing the speedup of our unmanaged cyclic quorum algorithm with that of our algorithm with additional computation management logic (U vs. M). Both were executed using (P) parallel nodes with speedups in reference to an optimized single node algorithm. Two input datasets are compared, one with $N = 39298$ rows which fit within available memory, and another with $N = 45265$ rows which exceeds a single node's memory resources, requiring the use of an alternate lower memory optimized algorithm. For non-Singer difference sets, the managed all-pairs computations were up to 30% faster than without the management logic. When P corresponded to a Singer difference set, the overhead of the management was typically less than 1%, which put the unmanaged speedup slightly ahead. 82

Figure 7.1 Example route fault tolerance using light-trails 85

Figure 7.2 Networks used for simulations: Figure (a) NSFNET, 14-Node/22-Link, Figure (b) ARPANET, 20-Node/31-Link, Figure (c) American Backbone [Tang et al. (2011)], 24-Node/43-Link, and Figure (d) Chinese Backbone [Tang et al. (2011)], 54-Node/103-Link. 86

Figure 7.3 Quad light-trails to provide additional cycle fault protection 89

Figure 7.4 Percent mean fault coverage of paired cycles using our redundant quorum solution experiencing two simultaneous link faults. 95

Figure 7.5 Percent mean fault coverage of our single cycle, redundant quorum solution experiencing a single link fault. 100

Figure 7.6 Percent mean fault coverage of our single cycle, redundant quorum solution experiencing two simultaneous link faults. For graph clarity and consistency NSFNET for $R = 2$ (Single) at 91.94% and for $R = 3$ (Single) at 93.37% mean fault coverage were not included in the graph. . . 102

Figure 7.7 Mean fault coverage (%) of our single cycle, redundant quorum solution and greedy cycle direction heuristic experiencing a single link fault. . . 110

www.manaraa.com

Figure 7.8 Mean fault coverage (%) of our single cycle, redundant quorum solution and greedy cycle direction heuristic experiencing two simultaneous link faults. For graph clarity and consistency NSFNET for $R = 2$ (Single) at 93.67% and for $R = 3$ (Single) at 94.88% mean fault coverage were not included in the graph. 112

للاستشارات المنارة
www.manaraa.com

ACKNOWLEDGEMENTS

I would like to thank my major professor, Dr. Arun K. Somani. You saw something special in me years ago when I sat through your computer architecture class. Thanks for working so hard over the years to teach me how to think and bringing that special something out of me. I am very proud of our work and will be forever grateful for the opportunity to study under you.

I would also like to thank all of my committee members, Drs. Shashi K. Gadia, Manimaran Govindarasu, Suresh C. Kothari, and Srikanta Tirthapura for all of the guidance, discussions, and helpful suggestions throughout my doctoral work and before I ever thought of starting down this path.

Dr. David Lastine, thank you for all of your thoughtful discussions and insights on optical network routing. Those conversations helped challenge my ideas and develop a clearer trail forward to build upon.

Ben Holland, thank you for more late night conversations and challenging problem discussions than I can count. I hope to return the favor some day.

Last, but certainly not least, thank you to an amazing family and so many great friends. I am truly blessed to have all of you in my life. The Ph.D. journey had its ups and downs, but you were there for me through it all. Your patience, encouragement, and support helped me get to the finish line.

المنارة للاستشارات
www.manaraa.com

ABSTRACT

Big data continues to grow in size for all sciences. New methods like those proposed are needed to further reduce memory footprints and distribute work equally across compute nodes both in local HPC systems and rented cluster resources in the cloud. Modern infrastructures have evolved to support these big data computations and that includes key pieces like our internet backbones and data center networks. Many optical networks face heterogeneous communication requests requiring topologies to be efficient and fault tolerant. The all-pairs problem requires all elements (computation datasets or communication nodes) to be paired with all other elements. These all-pairs problems occur in many research fields and have significant impacts, which has led to their continued interest.

We proposed using cyclic quorum sets to efficiently manage all-pairs computations. We proved these sets have an "all-pairs" property that allows for minimal data replication and for distributed, load balanced, and communication-less computation management. The quorums are $O\left(\frac{N}{\sqrt{P}}\right)$ in size, up to 50% smaller than dual $\frac{N}{\sqrt{P}}$ array implementations, and significantly smaller than solutions requiring all data. Scaling from 16 to 512 cores (1 to 32 compute nodes) and using real dataset inputs, application experiments demonstrated scalability with greater than 150x (super-linear) speedup and less than 1/4th the memory usage per process.

Cyclic quorum sets can provided benefits to more than just computations. The sets can also provide a guarantee that all pairs of optical nodes in a network can communicate. Our evaluation analyzed the fault tolerance of routing optical cycles based on cyclic quorum sets. With this method of topology construction, unicast and multicast communication requests do not need to be known or even modeled a priori. In the presence of network single-link faults, our simulated cycle routing had greater than 99% average fault coverage. Hence, even in the presence of a network fault, the optical networks could continue operation of nearly all node pair communications.

www.manaraa.com

Lastly, we proposed a generalized R redundant cyclic quorum set. These sets guarantee all pairs of nodes occur at least R times. When applied to routing cycles in optical networks, this technique provided almost fault-tolerant communications. More importantly, when applied using only single cycles rather than the standard paired cycles, the generalized R redundancy technique almost halved the necessary light-trail resources while maintaining the fault tolerance and dependability expected from cycle-based routing.

Problem Description

Big Data in recent years has become a focal point for science and commerce. As datasets grow larger, traditional methods and algorithms are challenged on whether they are able to truly scale. This has led to phrases like, "swimming in sensors, drowning in data."

Our work addresses some of the challenges facing a particular type of big data interaction. The interaction considered requires all elements in a set to interact with all other elements in the set.

The all-pairs interaction is a general computation or communication problem that occurs frequently and can be as simple as considering the shaking of hands by all attendees to a party. More formally there is set E_N, where there are N elements indexed 0 to $(N-1)$.

$$E_N = \{e_0, e_1, ..., e_{N-1}\}$$

The elements in this general formulation can be simple, single communication node or single item data structures, e.g., E_N could simply be all nodes in a network or be a large array of N values. Or, elements can be complex data structures with many fields / values. Fields are not restricted to a single data type either, as many big data problems can rely on heterogeneous datasets.

The all-pairs interaction considers all possible pairs of elements, $\binom{N}{2}$.

$$\{(e_0, e_1), (e_0, e_2), ..., (e_0, e_{N-1}), (e_1, e_2), (e_1, e_3), ..., (e_1, e_{N-1}), ..., (e_{N-2}, e_{N-1})\}$$

المنارة للاستشارات
www.manaraa.com

While the simple hand shake example could be considered a symmetric interaction.

$$e_i \leftrightarrow e_j, i < j$$

The all-pairs interaction can be more generally represented by two separate interactions to better represent the computational or communication complexity in those problems where the all-pairs operation is not commutative.

$$e_i \rightarrow e_j, i < j$$

$$e_i \leftarrow e_j, i < j$$

The computational complexity of this general algorithmic form is not daunting.

$$\binom{N}{2} = \frac{(N-1)\,N}{2} = O\left(N^2\right)$$

In fact, even for pair computations that do not have the commutative property, the complexity is unchanged. In general, polynomial $O\left(N^2\right)$ computations are considered highly computationally scalable.

When performing an all-pairs data interaction on the big data scale sizes, while the computational complexity theoretically is manageable, the data management becomes complex. The problem definition inherently requires access to the entire dataset, such that every data element can be paired and processed with every other data element in the set. When the datasets exceed a system's memory size, this presents a challenge, which our methods address.

Solution Approach

For efficiency and distributed control, it is common in distributed systems and algorithms to group nodes into intersecting sets referred to as quorum sets. Our management techniques rely on the established quorum set theories for their efficiencies and management. We then proved an "all-pairs" property of cyclic quorum sets, which is central to guaranteeing that all-pairs of elements (nodes or data) are able to interact in the system.

The all-pairs data computation problem requires all data elements to be paired with all other data elements. These all-pairs problems occur in many science fields, which has led to

المنارة للاستشارات
www.manaraa.com

their continued interest. Our research addresses the memory and computation time challenges of the general all-pairs big data interaction computations through the use of memory efficient computation management techniques. Proposed were methods using distributed computing to share the computational workload. Although the problem definition requires every data element to have access to and interact with the entire dataset, our cyclic quorum set techniques relax this restriction in distributed systems. This computation management is used to reduce memory resource requirements per node and enable big data scalability. Implementation evaluation of a large bioinformatics application demonstrated scalability on real datasets with linear and at times super-linear speedups. Reductions in memory requirements per node allowed for processing larger datasets that would not have been feasible on a single node either due to memory or time requirements.

Similar cyclic quorum set techniques were used to address efficient and fault tolerant communication routing challenges in optical networking. Cycle-based optical network routing, whether using SONET rings or p-cycles, provide the sufficient reliability in networks. Light-trails forming a cycle in the network allow broadcasts within a cycle to be used for efficient multicast communications. Using the proven "all-pairs" property of cyclic quorum sets, we could guarantee all pairs of nodes will occur in one or more quorums, so efficient, arbitrary unicast communication can occur between any two nodes. Efficient broadcasts to all network nodes are possible by a node broadcasting to all quorum cycles to which it belongs ($O\left(\sqrt{N}\right)$.) We analyzed node pair communications in networks, specifically, the fault tolerance aspects of using cyclic quorum sets to route cycles. Observed was better than 99% average single fault coverage and some node pair communications were protected by more than one cycle.

Exploiting this redundant node pair protections revealed even greater resource efficiencies. Common cycle routing techniques will use pairs of cycles to achieve both routing and fault-tolerance, which uses substantial resources and creates the potential for underutilization. Instead, when we intentionally designed cyclic quorum sets with R redundant pairs of nodes and utilized the R redundancy within the quorum cycles to replace the pair of cycles with just a single cycle, we saw network resource usage almost halved. Our analysis of several networks showed $R = 2$ redundant single cycles had 96.60 - 99.37% single link fault coverage, while re-

المنارة للاستشارات
www.manaraa.com

ducing resource usage by 42.9 - 47.18% on average. Increasing redundancy to $R = 3$ redundant cycles maintained a 93.23 - 99.34% average fault coverage even with two simultaneous link faults and used 38.85 - 42.39% fewer resources on average.

www.manaraa.com

CHAPTER 1 THE PROBLEM

Big data is having impacts across virtually all research and business fields. Many fields are seeing orders of magnitude increases in data generated. Algorithms have the difficult challenge of keeping up with this pace. Infrastructure like our internet backbones and data centers continue to evolve, but meet challenges on all sides; resources, bandwidth, and fault tolerance are all major constraints. This has led to phrases like, "swimming in sensors, drowning in data" [Drew (2010)]. As a solution, some have turned to the cloud to fulfill their various computing needs. They are able to rent virtually unlimited storage and compute power to meet their needs.

Our work addresses some of the challenges facing a particular type of big data interaction. The interaction considered requires all elements (nodes or data) to interact with all other elements in the set. This interaction can generally be referred to as an "all-pairs" interaction.

All-pairs problems are so common that in elementary schools and introductory computer science courses they may be taught as a popular "handshake" problem [Hedegaard (2016)]. The problem goes something like this: P people attend a party and a popular greeting is to shake hands, how many handshakes take place? After discussion and manipulation the answer of $\binom{P}{2} = \frac{P(P-1)}{2}$ is derived.

More formally there is set E_N, where there are N elements indexed 0 to $(N-1)$.

$$E_N = \{e_0, e_1, ..., e_{N-1}\} \tag{1.1}$$

The elements can be simple like a single communication node or single item data structure, e.g., E_N could simply be all nodes in a network or a large array of N values. Or elements can be complex structures with many fields. Fields are not restricted to a single data type either, as many big data problems can rely on heterogeneous datasets. An example of a data element with a complex structure may be a Tweet [Twitter (2016)]:

المنارة للاستشارات
www.manaraa.com

```
{
"coordinates": null,
"favorited": false,
"created_at": "Fri Mar 25 13:45:15 +0000 2016",
"truncated": false,
...
}
```


The all-pairs interaction considers all possible pairs of elements, $\binom{N}{2}$.

$$\left\{\left(e_0, e_1\right), \left(e_0, e_2\right), ..., \left(e_0, e_{N-1}\right), \left(e_1, e_2\right), \left(e_1, e_3\right), ..., \left(e_1, e_{N-1}\right), ..., \left(e_{N-2}, e_{N-1}\right)\right\} \tag{1.2}$$

While the simple hand shake example could be considered a symmetric interaction.

$$e_i \leftrightarrow e_j, i < j \tag{1.3}$$

The all-pairs interaction can be more generally represented by two separate interactions to better represent the computational or communication complexity in those problems where the all-pairs operation is not commutative.

$$e_i \rightarrow e_j, i < j \tag{1.4}$$

$$e_i \leftarrow e_j, i < j \tag{1.5}$$

The computational complexity of this general algorithmic form is not daunting.

$$\binom{N}{2} = \frac{N\left(N-1\right)}{2} = O\left(N^2\right) \tag{1.6}$$

In fact, even for pair computations that do not have the commutative property, the complexity is unchanged. In general, polynomial $O\left(N^2\right)$ computations are considered highly computationally scalable.

When performing an all-pairs data interaction on the big data scale sizes, while the computational complexity theoretically is manageable, the data management becomes complex. The problem definition inherently requires access to the entire dataset, such that every data element

المنارة للاستشارات
www.manaraa.com

can be paired and processed with every other data element in the set. When the datasets exceed a system's memory size, this presents a challenge, which our methods address.

Our research addresses the memory and computation time challenges of the general all-pairs big data interaction computations through the use of memory efficient computation management techniques. Distributed computing was used to share the computational workload. Even though the problem definition requires every data element to interact with the entire dataset, our techniques relax this restriction in this distributed environment so not all of those interactions must occur in the same compute node. This computation management is used to reduce memory resource requirements per node and enable big data scalability.

We begin this chapter with a brief introduction to various example applications of the all-pairs interaction. Then the remainder of this chapter will include a short itemized list of contributions made in our work.

1.1 Communication Application

Internet backbones and data centers have become key pieces of national infrastructure rising almost to the level of roads, bridges, electricity, and water. Optical networks in these settings are depended upon for high speed communications in distributed algorithms, as much as they are needed for the arbitrary point-to-point communications. Failures within a network are to be expected and can happen as much as every couple days. Protecting against these optical network faults is critical and there are many different approaches depending on the network needs and individual circumstances.

For efficiency and distributed control, it is common in distributed systems and algorithms to group nodes into intersecting sets referred to as quorum sets. Quorums sets for cycle-based routing to efficiently support arbitrary point-to-point and multi-point optical communication were first proposed in Somani and Lastine (2014). Cycles are created using quorums of nodes. Within a cycle, multicasts to all nodes in that cycle are possible. The proof that cyclic quorums have an "all-pairs" property guarantees unicast capabilities in a network using the proposed routing technique. Exploiting the same properties efficient broadcasts can be achieved with $O(\sqrt{N})$ multicasts.

المنارة للاستشارات
www.manaraa.com

Our research analyzed and evaluated novel enhancement to the fault tolerance of optical network cycle routing using optimal cyclic quorums [Kleinheksel and Somani (2015b)]. Observing communication pairs being protected by more than one cycle, we then applied established quorum set theory and exploited the redundancies to form suitable R redundant quorums for our optical network routing to improve resource efficiency and maintain a high degree of fault tolerance [Kleinheksel and Somani (2015c,a)]. This was a novel method to deliver the almost fault-tolerant capabilities in an optical network, while significantly reducing the resource utilization when compared to the state-of-art techniques.

1.2 Computation Applications

All-pairs or "handshake" problem occur frequently in the computation applications. In databases this manifests as a self-join without a join condition, forcing all tuples to interact with all other tuples. In physics, the n-body problem predicts the position and motion of N bodies by calculating the total forces every body has on every other body. In biometrics applications, a similarity matrix can be formed using a set of images compared with itself using facial recognition [Phillips et al. (2005)]. In metagenomics, finding a protein's likeness to every other protein is a crucial part of forming the complex graphs used in protein clustering, which has led to new discoveries of protein functions [Chapman and Kalyanaraman (2011)].

1.2.1 Acceleration of applications

Accelerating the execution of many of these important applications has been done using multicore CPUs, FPGAs, GPUs, Intel's many-core MIC, and distributed clusters. In Moretti et al. (2010) the authors provide a generalized framework to solve these all-pair classification of algorithms and show performance improvements for biometrics and data mining applications in a distributed system, e.g., cloud. A different approach was taken for a bioinformatics application seeking to reconstruct gene co-expression networks. The PCIT algorithm in Reverter and Chan (2008) was chosen to identify significant gene correlations. This method was optimized for Intel's multicore Xeon and many-core MIC by Koesterke et al. (2013, 2014).

المنارة للاستشارات
www.manaraa.com

The generalized framework in Moretti et al. (2010) showed that efficiently distributing all of the input data to all of the nodes prior to beginning execution resulted in faster turnaround times than reading from the disk on demand. A seemingly obvious result, but certainly highlights that every element interacting with every other element leads to a natural result of having all elements present in memory to maximize execution performance. The optimization of the PCIT algorithm in Koesterke et al. (2013) experienced having more data than memory available and created a second optimization strategy with longer runtimes, but had a minimal memory usage footprint to accommodate.

1.2.2 Relaxing the all elements present requirement

N-body problems have a natural all-pairs decomposition called atom-decomposition [Plimpton (1995)] that is based on equal distribution of N element responsibilities to P parallel processes. To address load imbalances and the need to communicate all data to all processes, the authors proposed a method to perform force-decomposition which still requires input data replication, but reduced it to 2 arrays of size $\frac{N}{\sqrt{P}}$ elements per process. The authors in Driscoll et al. (2013) showed that data replication in the system can be variable (c); and when $c = \sqrt{P}$, a lower bound on communication is achieved. When $c = 1$, their solution behaved similar to atom-decomposition, although requiring only 2 arrays of $\frac{N}{P}$ elements per process. When $c = \sqrt{P}$, their solution behaved similar to force-decomposition and required 2 arrays of size $\frac{N}{\sqrt{P}}$ elements per process.

Minimizing the amount of data replication in a distributed system, while maintaining efficient all-pairs algorithm operation, is a recurring theme in this classification of algorithms. Quorum systems are commonly used for coordination and mutual exclusion in distributed systems [Chao and Wang (2010); Luk and Wong (1997)]. Quorum's decentralized approach and slow quorum growth rate compared to the system size are two of the reasons that make them a good tool in managing replicated data [Kumar and Agarwal (2011)]. In 1985, quorums of size $O\left(\sqrt{P}\right)$ were proven using finite projective planes in Maekawa (1985). Relaxed difference sets later were used to create size $O\left(\sqrt{P}\right)$ cyclic quorum sets in Luk and Wong (1997).

المنارة للاستشارات
www.manaraa.com

1.2.3 Scaling all-pairs algorithms

The bioinformatics field notably has seen an increase in data. Led by several advances in science including Next Generation Sequencing (NGS) technology, the use of data to drive biological and medical discoveries has become a prominent research method. This has not come without its struggles as the size of data and computation time can easily eclipse local resources. Hence, scaling algorithms to larger datasets and for utilizing more resources has been a reoccurring theme in bioinformatics. The authors in Chae et al. (2013) surveyed publicly available bio and health cloud systems. Importance was given to graphical user interfaces, connecting to existing cloud datasets, security, and providing tools capable of running in a cloud environment. This was where their BioVLab excelled. Since then, Chae et al. (2014) have continued to add additional analysis tools to their Amazon cloud application.

Our research utilized the slow quorum growth rate compared to number of processes to scale all-pairs algorithms. Given that cyclic quorum sets have an "all-pairs" property [Kleinheksel and Somani (2016)], our solution only requires a single array of size $O\left(\frac{N}{\sqrt{P}}\right)$ elements per process. This being significantly less than current solutions that require all N elements per process and up to 50% improvement over those that have used replication techniques to reduce memory requirements to two arrays of size $\frac{N}{\sqrt{P}}$ elements per process.

For processes $P = 4, \ldots, 111$, our work uses the optimal cyclic quorums from Luk and Wong (1997) (see Appendix A for a reproduced and verified cyclic quorum listing). These cyclic quorums are optimal in memory and computation for all Singer difference sets [Colbourn (2010)] and near-optimal for all others. For the non-optimal difference sets, we developed a decentralized, load balanced, and communication-less management technique to identify and avoid all redundant computations in non-Singer difference sets. This adds greater flexibility for users to utilize all of their local or cloud HPC resources efficiently.

المنارة للاستشارات
www.manaraa.com

1.3 Contributions

1.3.1 Quorums

- Proof that cyclic quorums sets have an "all-pairs" property (Section 5.3.2)
- Verified optimal correctness of previously published cyclic quorums sets for $N = 1, \ldots, 111$ [Luk and Wong (1997)] (Appendix A)
- Definition of R redundant cyclic quorums (Section 5.4)
- Found $R = 2$ and $R = 3$ redundant cyclic quorums sets (Appendix B)

1.3.2 Computation

- New application of cyclical quorums to facilitate scaling all-pairs data interactions in distributed computing (Chapter 6)
- Cyclic quorums have a provable lower bound in size. This property is used to limit all-pairs data replication leading to decreased memory usage per node and ability to scale to larger problem sizes (Section 6.3)
- Cyclic quorums load balance all-pairs computations and scale well (Section 6.3)
- A distributed, communication-less management technique for efficient all-pairs computations (Section 6.4)

1.3.3 Communication

- Forming all-pairs of communication nodes in optical networks using pairs of light-trail cycles based on optimal cyclic quorums sets provides greater than 99% fault tolerance single network link faults (Section 7.2)
- Increasing the number of paired light-trails improves average single fault coverage to greater than 99.9% (Section 7.3)

المنارة للاستشارات
www.manaraa.com

- New application of R redundant cyclic quorums to facilitate all-pairs of node communications in optical networks (Sections 7.4 and 7.5)

- Increasing the R redundancy of the quorums sets, while using paired light-trail cycles improves the average single fault coverage to greater than 99.8% with less than 23% increase in resource usage (Section 7.4)

- Using R redundant quorums sets with single cycle routing rather than paired cycles can significantly reduce link resource usage, while single and two simultaneous fault coverage is only slightly reduced (Section 7.5)

- Fault-free and fault conditions can be improved in the R redundant quorum, single cycle routing by controlling the direction of the cycle routes in the implementation (Section 7.6.1)

المنارة للاستشارات

www.manaraa.com

CHAPTER 2 ALL-PAIRS COMPUTATIONS

This all-pairs or "handshake" problem manifests in many common computation problems. In this chapter, we present a non-exhaustive list of several example applications to provide background and context to the more general all-pairs problems our research addresses. We specifically highlight existing methods of reducing a specific application's runtime or computational complexity.

2.1 Database Joins

In regards to big data and data interactions it is obvious to begin by considering the decades of contributions made to relational databases.

In relational databases, one of the foundation operations is the join. Mathematically this can be considered a Cartesian product of two relations. Then the join query would apply a select operation on the result tuples, this is called the join condition.

When considering the self-join, a relation is joined with itself. Without a join condition this would give rise to an all-to-all (all-pairs) tuple interaction. Otherwise, join conditions provide opportunities for optimization.

Considering the self-join definition there are N such tuples in a relation E_N, each having a set of attributes. To form the Cartesian product of this relation, every specific tuple e_i must interact with all other tuples in the relation E_N.

This fundamental operation has been studied and researched for decades. There is not a single problem or approach that has been successful in all circumstances either. This is primarily due to the join condition being inherently data dependent, so different datasets result in different behavior.

www.manaraa.com

Algorithm 1 Nested loops algorithm

```
1: for each tuple r in relation R do
2:     for each tuple s in relation S do
3:         if join condition on r, s is true then
4:             add to result set
5:         end if
6:     end for
7: end for
```

Algorithm 2 Sort and Merge algorithm

```
 1: Sort relation R
 2: Sort relation S
 3: for each tuple r in relation R do
 4:     while tuple s in relation S is less than r do
 5:         Read next tuple s in relation S
 6:     end while
 7:     if join condition on r, s is true then
 8:         add to result set
 9:     end if
10: end for
```

2.1.1 Algorithm implementations

From a straight forward approach, form the Cartesian product via nested loops and evaluate the join condition (Algorithm 1). It is clear that forming the Cartesian product first will result in $O(NM)$ work or $O(N^2)$ work for self-join.

When the join condition is more often false than true, then the problem begins to deviate from the all-pairs interaction problem. Here optimizations try to prevent unnecessary pairs that will fail the join condition from ever being formed.

Sort and merge (Algorithm 2) is a common algorithm for optimizing a merge. There is a preprocessing step where input relations are sorted based on the join condition attributes. Once sorted, each relation need only be iterated once in order to produce the result set. Hence reducing the work to perform a self join to $O(NlogN + N) = O(NlogN)$. If the input relations are already sorted on the join attributes, then the work is even further reduced to $O(N)$.

Hash join (Algorithm 3) is another way to optimize by reducing or eliminating unnecessary join condition operations. Here one relation will be scanned and the join condition attributes

المنارة للاستشارات
www.manaraa.com

Algorithm 3 Hash join algorithm

1: hash all tuple join conditions in relation R to form a hash table
2: **for** each tuple s in relation S **do**
3: **if** hash of join condition on s matches a hash in the table **then**
4: confirm join condition is met, then add to result set
5: **end if**
6: **end for**

are hashed. When scanning the second relation, only the tuples that have hash matches to the first relation will be processed.

2.1.1.1 Contributions from literature

Ehnasri and Navathe (1989) introduced a more I/O efficient nested loop algorithm that processes blocks of tuples at a time rather than individually. This more efficiently accesses the disk and memory.

Blasgen and Eswaran (1977) showed that if not much information is known about the input relations in terms of selectivities and the join attribute is not the index of the relation, then sorting and merging to form the join is the most efficient.

Bratbergsengen (1984) showed the efficiency of using the hash join method in eliminating unnecessary comparisons. This method can be very efficient because both relations only have to be scanned once. Although hash collisions reduce the efficiency, the non-equijoins present a challenge, considering most hash functions do not produce ordered hashes.

Han et al. (2012) introduced PI-join to address some algorithm inefficiencies of joins that have to be performed out of disk due to large data sizes combined with limited memory available. Cache is considered and tuples are fetched in a orderly fashion. Their multiple partitioning steps in their execution order resulted in significant performance gains both in time and data volumes compared to other traditional methods.

Ordonez and García-García (2010) evaluated the algorithm optimization performance for sort and merge and hash joins specifically when dealing with null and invalid keys. Hash joins were found to process nulls and invalids better than sort and merge. Also, when there is a significant number of invalid keys, creating a temporary relation with only tuples with valid keys can be more efficient when performing joins.

المنارة للاستشارات
www.manaraa.com

Algorithm 4 Join index algorithm

1: in order, scan the join indexes to find join condition matches
2: **for** each match m **do**
3: retreive corresponding tuples in relations R and S, then join and add to result set
4: **end for**

2.1.1.2 Challenges

From the fundamental perspective the join operation in databases has an all-pairs interaction component. This comes through when forming the Cartesian product.

Optimizing the algorithm such that the full Cartesian product does not have to be formed would be ideal. Hence other algorithms and enhancements on those algorithms have been proposed over the decades. However, the selectivity of the join condition can have an impact on the efficiency of the algorithm. In the worst case, the results are an all-pairs interaction of the data.

Estimating the selectivity of a join condition can be difficult because it is inherently based on the current data values in the relation. This is a motivating factor for use of data structures to pick up where algorithms begin to have limitations.

2.1.2 Data structures

One of the challenges of the algorithms was that entire relations were typically scanned to identify potential join condition matches. This was even more a challenge when the relation may not have been sorted by the join condition attributes.

Join indexes are sorted relations with the join condition attributes and a pointer to the corresponding tuple in the relation that the index is over. Hence these relations have the minimum data needed to evaluate the join condition match and they are sorted. Where selectivity is reasonable, this (Algorithm 4) will reduce the I/O required to perform the join query.

B-trees further improved join indexes by storing the indexes as a tree data structure. Hence the scanning of join indexes could still be done in order, but skipping directly to the matching indexes became a logarithmic function rather than linear one.

المنارة للاستشارات
www.manaraa.com

2.1.2.1 Contributions from literature

Valduriez (1987) introduced join indexes as a relation to identify result tuples in the original relations. This allowed for the identification of result tuples without having to scan the entire relations.

McCreight (1972) introduced B-trees as an effective method to store indexes. This allowed for logarithmic search, insertions, and deletions. Where the join index enabled skipping over unneeded relation tuples, this allowed skipping over many of the indexes as well when the join condition was not going to be a match.

2.1.2.2 Challenges

Join indexes must be updated as their corresponding relations are updated. This incurs some overhead, which must be justified. Similarly, the storage space does not come for free, so under certain circumstances the indexes can be almost as large as the relations they represent.

2.1.3 Distributed database joins

When databases are distributed, there are several factors introduced [Mishra and Eich (1992)]. Data partitioning results in data distributed across multiple systems. Sometimes the partitioning can also include data replication for both query optimization and fault tolerance. Depending on data location, the choice of where to process a query may be important.

Joins, when not all of the data is present at the local machine, are a challenge. Communicating all parts of a relation to where the query is takes time over a network and there simply may not be memory to hold the data. Performance could also be improved if multiple processors were working in parallel.

Semijoin method (Algorithm 5) can be used. Only the minimum necessary attributes from one system are transfered to the other to perform the join. The join will find only the tuples that would have matched the join condition in the uniprocessor algorithm, then send those back to be joined with the corresponding matches.

المنارة للاستشارات
www.manaraa.com

Algorithm 5 Semijoin algorithm

1: project only the join condition attributes of relation R
2: transfer projection to system with relation S
3: join projection and relation S
4: transfer projection join results to system with relation R
5: join projection join results and relation R

2.1.3.1 Contributions from literature

Valduriez (1982) introduced semijoins for distributed databases. This reduced the amount of data needed to be transferred between two systems in order for a join to be performed when not all data was present locally.

Kitsuregawa et al. (1983) introduced GRACE hash join method which could be used with multiple processors. Hashes were placed in buckets, which could be assigned to different processors, reducing the computation time to $O\left(\left(N+M\right)/P\right)$, where P is the number of processors.

Kruliš and Yaghob (2011) introduced a hash join for GPU architectures. While not distributed, it does have aspects of multiple processors dividing the same work. Value distribution is calculated to form even sized hash buckets, when processing every tuple in each bucket receives its own thread.

Groppe (2011) developed a parallel SPARQL engine for large Semantic Web databases. Parallel joins using a distribution thread, merge join over partitioned inputs, and parallel computation of operands are all discussed. Different algorithms and data situations deserve different processing orders. They found that when the results are large enough then parallel processing is beneficial, however parallel overheads can outweigh performance gains if the results are small.

2.1.3.2 Challenges

Masuyama et al. (1987) showed that as local networks increase in speed in relation to disk access, there are times where processing a semijoin projection locally prior to sending data to neighboring system may be inefficient.

المنارة للاستشارات
www.manaraa.com

Even with hash buckets distributed to multiple processors, the data values cannot necessarily be predicted also potentially leading to some hash buckets being considerably larger than others. This is where other partitioning techniques were introduced [Mishra and Eich (1992)].

Groppe and Groppe (2011) found that when the results are large enough, then parallel processing is beneficial. However, parallel overheads can outweigh performance gains if the results are small.

2.2 Spatial Databases

Spatial databases are unique in that tuples are related by their spatial relationship to one another.

Tauheed et al. (2013) introduced FLAT for neuroscientists. Indexes are stored in a tree structure with data at the leafs. Spatially close tuples form sets and are stored with one another. Then pointers are added to neighboring sets. They were able to demonstrate far superior read performance compared to existing techniques.

Tauheed et al. (2013) also introduced SCOUT for neuroscientists. Past tuple sets accessed during queries are summarized and after learning from a few queries, SCOUT can predict which sets will likely need to be accessed and prefetch them to memory. This increases hit rate and speeds up queries.

Tauheed et al. (2013) also introduced TOUCH. In-memory joins are a challenge because memory is a limited resource. They found that space-oriented joins caused far too many data items to be replicated and created the potential for duplicate work. Rather they used a data-oriented join followed by a spatial-filter to eliminate joins that spatially do not match the join condition.

2.3 N-Body Problems

The classical N-body problem considers moving celestial objects and the gravitational forces exerted on each other. The gravitational force between each object pair is estimated for a point in time. Once all forces on an object were found, they could be considered together to find

للاستشارات المنارة
www.manaraa.com

the total estimated force on each object. Using the current position, trajectory, and estimated total force, the future position and trajectory for each object is estimated. This process can be iterative using a problem specific time step interval.

The modern problem definition (sometimes referred to as many-body problem) is more general and considers objects of any size and any interaction operation. This definition still allows for the celestial objects with gravitational forces, but also broadens to consider applications with microscopic elements interacting sometimes with multiple forces in play. There are many such applications from molecular dynamics to computational chemistry to nuclear physics.

Considering this problem definition there are N such bodies each represented with multiple physical or virtual properties in a complex data structure. To find the interaction force applied on a specific element e_i, it must interact with all other elements in the set E_N.

2.3.1 Approximation of forces

The nature of many N-body applications is that elements closest to element e_i have the greatest force impacting that element. Then elements outside an application specific radius have less impact on the element. Using this application detail enables applications to effectively reduce computational complexity by either ignoring elements outside the radius or approximate them without precisely computing each interaction.

2.3.1.1 Contributions from literature

Lienhart et al. (2002) used a smoothing kernel of radius parameter h. Elements outside of radius h were ignored and contributed a small amount of error to the individual time steps. Using this application detail their application reduced $O\left(N^2\right)$ complexity all-to-all interaction to $O\left(CN\right)$ complexity all-to-few interaction, where C is significantly smaller number (≈50).

Ishiyama et al. (2012) and Yokota and Barba (2012) both used tree algorithms to approximate forces. The various tree algorithms have a finer granularity near the element where the forces are strongest and then progressively have coarser and coarser granularity approximations as the elements are further away. Their tree codes reduced the complexities to $O\left(NlogN\right)$ and $O\left(N\right)$, respectively.

المنارة للاستشارات
www.manaraa.com

2.3.1.2 Challenges

The problem definition still requires an all-pairs interaction. However, specific applications may have properties enabling the dataset interaction to be reduced without substantial increases in error. So long as the error is tolerable, then approximating the interaction may be acceptable.

2.3.2 Parallelizing the computation

The N-body problem has been a popular target for hardware acceleration. Many of the computations are embarrassingly parallel, which allows for execution speed to be defined by the amount of resources available, so long as the entire dataset is available to compute. When multiple iterations are required or when the dataset is not all locally available, particular attention to communication and load balancing costs may be important particularly with algorithms approximating forces rather than computing exact. Exact all-to-all interactions (or all-pairs) algorithms benefit from knowing exactly how many computations will occur and can optimize in a number of dimensions: memory, communication, data locality, etc. Approximations often require additional information from the dataset and can change from iteration to iteration, making it a challenge to distribute the work evenly while simultaneously being conscience of the overheads incurred to provide changes to that distribution each iteration.

Connolly et al. (2013) argued that for the Cosmic Frontier, Astrophysics, and Cosmology the growth of data volumes in the coming decades will be a serious challenge. This demands continuing evolution of existing codes and development of new algorithms to handle this growth. One of those critical applications is N-body simulations.

2.3.2.1 Contributions from literature

Lienhart et al. (2002) used an FPGA implementation for their application. Although their design had a much slower clock speed than their CPU, the FPGA allowed for 60 simultaneous floating-point operations (3.9 Gflops) and eliminated memory bottlenecks. At the time a general workstation was achieving only 100 Mflops due to the fewer floating-point units available and memory access bottlenecks.

المنارة للاستشارات
www.manaraa.com

Algorithm 6 Atom-Decomposition

1: All-to-all communicate current N element positions
2: Compute local element neighbors
3: Compute forces due to neighbors
4: Update N/P local elements

Arora et al. (2009) exploited multicore platforms for a similar goal to maximize the number of floating-point operations executed simultaneously. Multicore CPUs, the cell processor, and GPUs were all targeted on a single machine.

Chinchilla et al. (2004) proposed a GPU implementation of the all-to-all (all-pairs) N-body gravitational simulation and showed how to optimize for low-bandwidth situations. Van Meel et al. (2008) also used GPUs to solve N-body simulations. Their work was two algorithms, one an all-pairs interaction, and the other a short-range interaction, both in molecular dynamics. Moore et al. (2008) modeled gravitational forces using an all-to-all (all pair-wise) N-body algorithm implemented in CUDA.

Madsen and Filinski (2013) proposed a new streaming language for data-parallel executions. One of their targeted applications was an all-to-all (all-pairs) N-body simulation. It performed better than the CPU implementation, but still lagged the optimized GPU implementation.

Ishiyama et al. (2012) used a HPC machine to scale to a trillion bodies. This implementation used MPI/OpenMP hybrid to target 10's of thousands of processors. Yokota and Barba (2012) similarly used a HPC machine, but their's targeted 100's of GPUs. These and others like them are important to demonstrate scaling the N-body problem across multiple machines and not requiring the entire E_N dataset to be available at each node.

Plimpton (1995) described three distributed parallel approximation algorithms for molecular dynamics computations. Each have positives and negatives, which motivates understanding specific application scenarios in order to choose the most appropriate algorithm.

1. **Atom-Decomposition** (Algorithm 6) - Each processor is assigned N/P elements. All global atom positions are collected. Compute which neighbor elements are within the threshold distance of each element. Compute forces due to all neighbors. Update the N/P elements assigned.

المنارة للاستشارات

www.manaraa.com

Algorithm 7 Force-Decomposition

1: Collect $N/\sqrt{P}$ element positions in assigned row
2: Collect $N/\sqrt{P}$ element positions in assigned column
3: Compute which row-column element interactions are neighbors
4: Compute forces due to neighbors
5: Reduce row-wise global forces occurring on N/P local elements
6: Update local elements

Algorithm 8 Spatial-Decomposition

1: Exchange element positions with neighboring boxes
2: Compute all neighbors of local box elements
3: Compute forces due to neighbors
4: Update local box elements
5: Reassign elements that left local box

2. **Force-Decomposition** (Algorithm 7) - Processors are arranged in a 2D grid. Each processor is assigned an $N/\sqrt{P}$ x $N/\sqrt{P}$ block of forces out of a global grid of N x N forces formed by an all-to-all element interaction. Collect the $N/\sqrt{P}$ element positions for the assigned rows and columns. Compute which elements in the block of assigned forces are neighbors. Compute the assigned forces corresponding to those neighbors. Each processor still owns N/P elements as was the case in Atom-Decomposition. Reduce force calculations performed by other processors in the same row of the global grid (only $\sqrt{P}$ processors participating,) such that forces are reduced at the processor owning the element. Update the N/P elements assigned.

3. **Spatial-Decomposition** (Algorithm 8) - Processors are arranged spatially in a 3D grid. Each processor is assigned a box and all elements inside the box. Only elements in neighboring boxes (i.e. 26 neighbor boxes) may interact with local elements, so collect any of those elements which will be interacting. Compute which elements in the box are neighbors and include elements from neighboring boxes that will be interacting as well. Compute the forces due to those neighbors. Update the elements with the assigned box. As elements leave a box they are reassigned to the box they enter.

Driscoll et al. (2013) proposed a new variation on data replication with the goal of reducing communication through a blending of Atom-Decomposition algorithm (no replication) and

المنارة للاستشارات
www.manaraa.com

Force-Decomposition algorithm ($\sqrt{P}$ replication, 2 $N/\sqrt{P}$ elements per processor), both by Plimpton (1995). The authors proved communication optimality given a c replication factor.

The all-pairs interaction algorithm divides P processors into P/c teams (columns), where c is the replication factor (rows.) The N elements are divided across the P/c teams with the processors in row 0 being the team leaders. The leaders broadcast the elements to the members in their team. All team members make a copy of the elements prior to performing an initial shift and force update calculation. Then P/c^2 shift and update calculation steps are performed. After all shift steps, all element pairs have been formed and a reduction step within the team will combine all force updates for a particular iteration.

When $c = 1$, the algorithm is similar to Atom-Decomposition. When $c = \sqrt{P}$ like in Figure 2.1, the algorithm is similar to Force-Decomposition. In general, larger replication led to better performance due to the reduction in communication. Exception to this was when the shift costs were reduced or almost eliminated, but the increase in team members made the reduction step more expensive.

Driscoll et al. (2013) also proposed an approximate version of their all-to-all N-Body algorithm using a cutoff distance. The elements are distributed to teams based on spatial-decomposition, hence a team's neighboring teams are spatially close and farther teams are not as close. The shift procedure was modified such that communication only happened within the cutoff distance M. When a force update moves an element into a neighbor's space, the element is re-assigned.

2.3.2.2 Challenges

Load balancing an approximation algorithm is an example of something that can be a challenge. If unbalanced, regardless the number of resources, the performance could suffer more than having fewer resources. If distribution of work is by element, then there are situations where some elements may have more elements near them than others, hence more force calculations to precisely compute and fewer to approximate. If distribution of work by spatial location, similarly some processors may be responsible for more elements than others. However, distributing of work by force calculations can also suffer under certain circumstances. To

المنارة للاستشارات
www.manaraa.com

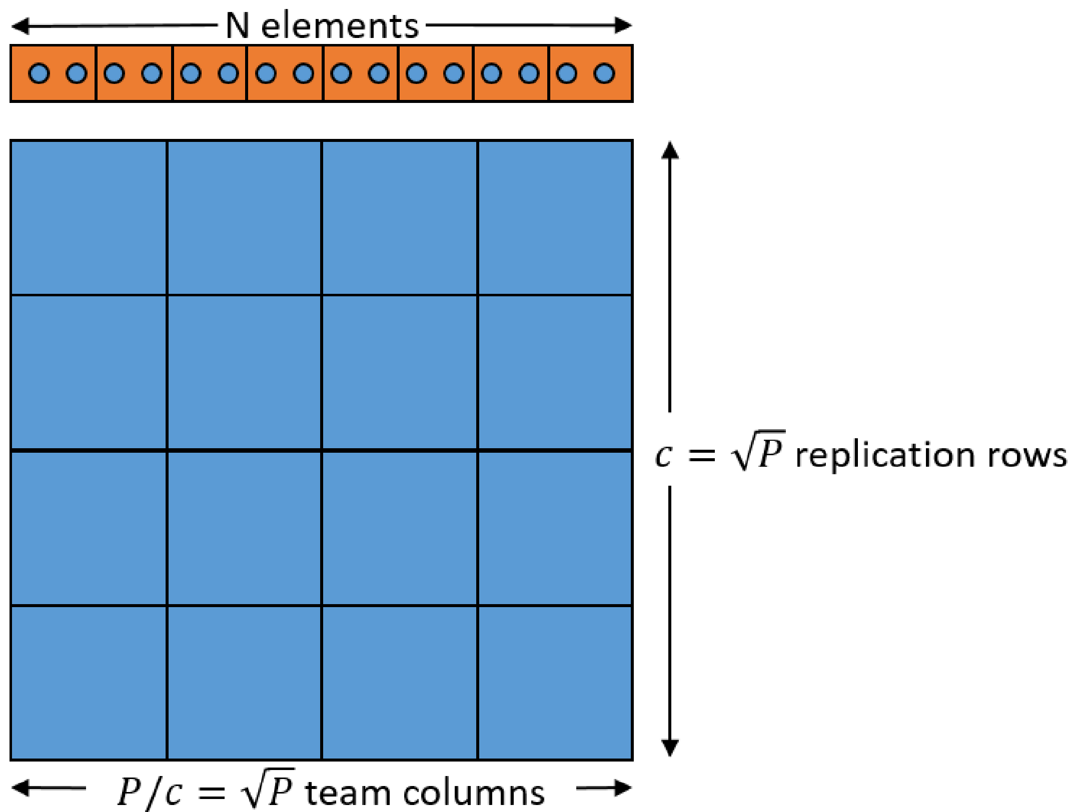


Figure 2.1: Driscoll et al. (2013) communication optimal n-body algorithm's data replication and distribution. Optimality achieved when $c = \sqrt{P}$, resulting in $\sqrt{P}$ teams, $\sqrt{P}$ replication rows, and each processor performing the pairing between two size-$\frac{N}{\sqrt{P}}$ arrays of elements. Note that their algorithm's communication steps are not depicted in this figure.

maintain balance, each distribution method requires a bit of extra work potentially eating into the parallel efficiency.

Both all-pairs interaction and approximation algorithms require at least some degree of global knowledge forcing communication and sharing of parallel compute data. Managing the amount of data that needs to be shared and when it gets shared, both impact the amount of communication overhead incurred due to parallelization.

Driscoll et al. (2013) faced a few challenges that many parallel N-Body all-pairs and approximate algorithms face. They showed that communication contributed significantly to the overall runtime of N-Body algorithms. Replication helped to reduce these costs, but they experienced exceptions to more is always better.

المنارة للاستشارات
www.manaraa.com

In the all-pairs algorithm, the reduction of force updates began to be significant once other communication costs were reduced and in one case performing a reduce communication caused an increase in comparison to a smaller replication factor. Their method also was unable to take advantage of Newton's third-law, which Plimpton (1995) used where it was beneficial. It is unclear if it could have been beneficial in this case. They also maintained 2 Nc/P elements in addition to the memory to track force updates, which is 2 $N/\sqrt{P}$ when $c = \sqrt{P}$ and still required a communication shift in addition to the element broadcast.

In the approximated cutoff algorithm, there are fewer computations performed and more forces are approximated due to the cutoff. Any communication that takes place becomes more significant to the overall runtime. Load balancing with spatial distributions is a challenge as described by Plimpton (1995) and observed by Driscoll et al. (2013). Assuming uniform spatial distribution cannot always be guaranteed and even the edges of the simulation will experience imbalance simply due to having fewer neighbors than those in the center of the simulation. This imbalance causes larger communication delays as lightly loaded processors have to wait for heavier, center processors. Re-assigning elements as they move through the simulation area also incurs overheads.

2.4 Metagenomics

Metagenomics looks at systems of organisms. Sometimes 1000's of organisms are working together to perform functions and identifying those functions along with the proteins and genes that express those function is very valuable. Examples of such system environments could be soil, saliva, human digestion, etc.

2.4.1 Contributions from literature

Chapman and Kalyanaraman (2011), Rytsareva and Kalyanaraman (2012), and Wu and Kalyanaraman (2013) are all performing protein clustering to try to group like proteins and potentially identify functions of proteins that had not been identified before. Their approach approximates every proteins likeness to every other protein and begins to form graph edges.

المنارة للاستشارات

www.manaraa.com

Using OpenMP, MapReduce, and GPUs, respectively, they accelerate the graph clustering aspect of the algorithm.

2.4.2 Challenges

Graph clustering is still a challenging problem and heuristics like those presented are trying to approximate the best fits.

Additionally, the protein-to-protein likeness is an expensive task, so methods are used to approximate this and reduce its complexity from all-pairs to K, few-to-few interactions. If an exact likeness measurement is needed, then the brute force all-pairs is the only option.

2.5 Gene Co-Expression Networks

From large DNA strands segments attributed to genes can be identified using data analysis and simple pattern techniques. Identification is not enough though. What do these genes do? What are their purpose?

Gene expression is the next key to connecting biological function to specific genes. Discovering these connections can have significant impact on biology and health. Still there remain genes with undefined functional roles. This is where identifying networks of potentially coordinating genes may connect genes with known functions.

Gene expression experiments have been improving over the last decade, but still often generate noisy data. This motivated creating methods to utilize the data despite the noise.

Multiple recent bioinformatics research publications utilize or develop gene co-expression network construction algorithms in their data analysis workflows. Fortes et al. (2012) used the PCIT algorithm in their succession of analyses identifying significant gene correlations in their heifer first service conception dataset. Gibson et al. (2013) introduced RMTGeneNet analyzing the functional robustness of the Random Matrix Theory (RMT) approach to gene co-expression network construction. Ficklin and Feltus (2013) used the RMTGeneNet package to generate their co-expression networks for their rice analysis. Despite the quality networks generated by PCIT, they found the tool could not be adapted to their processing workflow where RMTGeneNet could.

المنارة للاستشارات

www.manaraa.com

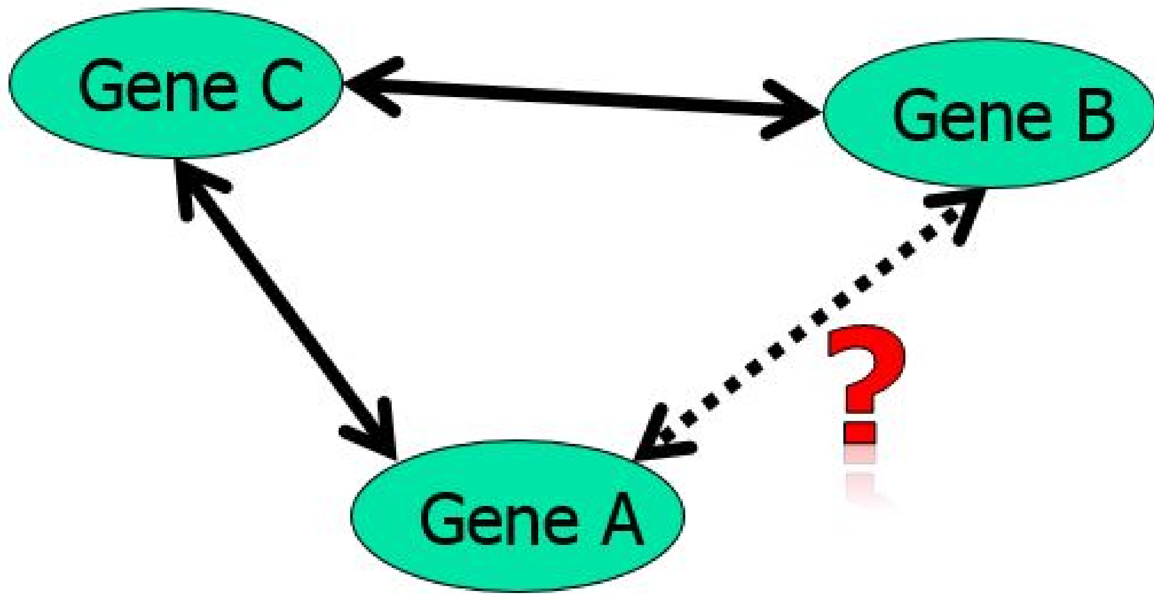


Figure 2.2: High-level summary of PCIT algorithm, gene expression correllation trio computation

While there are multiple techniques, the following subsections will look closer at the evolution of the partial correlation coefficients combined with an information theory approach (PCIT).

2.5.1 PCIT introduced

The partial correlation coefficients combined with an information theory approach (PCIT) algorithm was introduced by Reverter and Chan (2008). The algorithm can be used for gene co-expression network reconstruction and help to identify novel biological regulators. The technique processes N genes by building an $O\left(N^2\right)$ matrix and using a guilt-by-association heuristic to sequentially analyze node triplet partial correlations identifying whether a gene expression correlation is or is not meaningful (Figure 2.2).

It is not as simple as considering:

$$if\,(AC\,correlation\,exists)\,\&\,(BC\,correlation\,exists)\,then\,AB \tag{2.1}$$

Rather, the gene expression correlation between genes A and B is only considered significant

المنارة للاستشارات
www.manaraa.com

Algorithm 9 Original 2008 PCIT algorithm

```
1: for A ← 1, N − 2 do
2:     for B ← A + 1, N − 1 do
3:         for C ← B + 1, N do
             Given C, is correlation between A and B significant?
4:         end for
5:     end for
6: end for
```

once it can be shown that it is still significant after the contribution of gene C has been statistically removed.

First step is the partial correlations. Using a $O\left(N^2\right)$ matrix of all gene expression correlations, where r_{AB} is the gene expression correlation between genes A and B, the partial correlations are found using the following:

$$r_{AB,C} = \frac{r_{AB} - r_{AC}r_{BC}}{\sqrt{\left(1 - r_{AC}^2\right)\left(1 - r_{BC}^2\right)}} \tag{2.2}$$

$$r_{AC,B} = \frac{r_{AC} - r_{AB}r_{BC}}{\sqrt{\left(1 - r_{AB}^2\right)\left(1 - r_{BC}^2\right)}} \tag{2.3}$$

$$r_{BC,A} = \frac{r_{BC} - r_{AC}r_{AB}}{\sqrt{\left(1 - r_{AC}^2\right)\left(1 - r_{AB}^2\right)}} \tag{2.4}$$

Using the partial correlations, the contribution of C to the gene expression correlation of genes A and B. If the following is true then the correlation between genes A and B is not significant:

$$\left(|r_{AB}| \leq |\varepsilon r_{AC}|\right) \& \left(|r_{AB}| \leq |\varepsilon r_{BC}|\right) \tag{2.5}$$

$$where\, \varepsilon = \frac{1}{3}\left(\frac{r_{AB,C}}{r_{AB}} + \frac{r_{AC,B}}{r_{AC}} + \frac{r_{BC,A}}{r_{BC}}\right) \tag{2.6}$$

This calculation is performed for all values of A, B, and C (Algorithm 9).

2.5.1.1 Contributions of Reverter and Chan (2008)

The PCIT algorithm improved upon existing gene co-expression analysis by removing arbitrary global thresholding techniques and replacing it with a data driven, local thresholding technique. Each gene trio had a separate threshold calculated, which enabled better tolerance

المنارة للاستشارات
www.manaraa.com

to the noise in the data. This was demonstrated by their ability to significantly reduce the number of false positives compared to the existing techniques (although important to bioinformatics, these alternate techniques will not be explored in this survey which only indirectly considers bioinformatics as one of many applications.)

2.5.1.2 Challenges

As can be seen in Algorithm 9, a $O\left(N^3\right)$ computational complexity will increase dramatically in time as N scales. For small $N = 1000$, computations are already approximately 1 Billion. A factor of 10 increase in N, translates to 1000x increase in computations. This has dramatic impact on how quickly data can be processed, and with the rate of growth in bioinformatics data generation, this becomes a major bottleneck. This challenge is partially overcome by an algorithm in section 2.5.2.

As the number of genes, N, scales so does the memory required. If there are N genes, then there are $O\left(N^2\right)$ correlations and supporting data stored in memory. When $N = 1000$, storing a few megabytes of data can be handled without issue. Scale N by 10 or 100 and consider that double precision can be valuable in some applications of the algorithm, and what was once a few megabytes can quickly become almost 150 gigabytes. 150GBs is still manageable given CyEnce's fat (1TB) node, but it definitely makes this approach less than scalable for many without such resources. Even Amazon's AWS cloud providing high performance computing clusters for rent by-the-hour can be limiting. For HPC applications like the PCIT algorithm, they offer their latest generation of compute-optimized "C4" instances [Amazon (2016b)]. These do have Intel Xeon E5 v3 processors; but even with the largest instance ("c4.8xlarge"), they only have 36 virtual CPUs and 60GB of memory [Amazon (2016a)]. This challenge is partially overcome by an algorithm in section 2.5.3.

2.5.2 Parallel PCIT algorithm using MPI

Watson-Haigh et al. (2010) introduced a scalable MPI version of the PCIT algorithm in 2010, which could parallelize the computations across a cluster of computers whether local or

www.manaraa.com

Algorithm 10 DefineTasks

```
1: procedure DefineTasks(N, P_MPI)
2:     work_cumsum[0] ← 0
3:     for j ← 1, N do
4:         work_j ← (N−j)(N−j−1)/2
5:         work_cumsum[j] ← work_cumsum[j − 1] + work_j
6:     end for
7:     work_IDEAL ← N! / (3!(N−3)! P_MPI)
8:     for proc_ID ← 0, P_MPI − 1 do
9:         j ← 1
10:        while work_IDEAL * proc_ID ≥ work_cumsum[j] do
11:            j ← j + 1
12:        end while
13:        index_start[proc_ID] ← j
14:    end for
15:    index_end[P_MPI − 1] ← N
16:    for proc_ID ← P_MPI − 2, 0 do
17:        index_end[proc_ID] ← index_start[proc_ID + 1] − 1
18:    end for
19: end procedure
```

cloud. The parallel implementation was introduced as an R package making it more accessible to those already using the R environment for other aspects of their scientific workflow.

This was an important contribution to address the long execution time of the original sequential algorithm. The execution time could be hours or days for even medium sized networks (5,000 - 10,000 genes), when problem sizes could easily reach 47,000 genes or more a parallel implementation was critical.

Their MPI method was an extension of the original 2008 algorithm. This method started P_{MPI} processes in parallel. Each would allocate and compute an $O\left(N^2\right)$ matrix of gene correlations, although only the upper or the lower triangle would be required for computation.

The authors presented a simple method to distribute the $\binom{N}{3}$ work approximately equally to all P_{MPI} processes. This had a theoretical speedup of P_{MPI}, i.e. execution time of $O\left(\frac{N^3}{P_{MPI}}\right)$, where P_{MPI} is the number of MPI processes.

The root(0) MPI process distributes the workload using Algorithm 10. The PCIT algorithm performs $\binom{N}{3}$ total calculations; however for ease of coding though, the work was viewed and

المنارة للاستشارات
www.manaraa.com

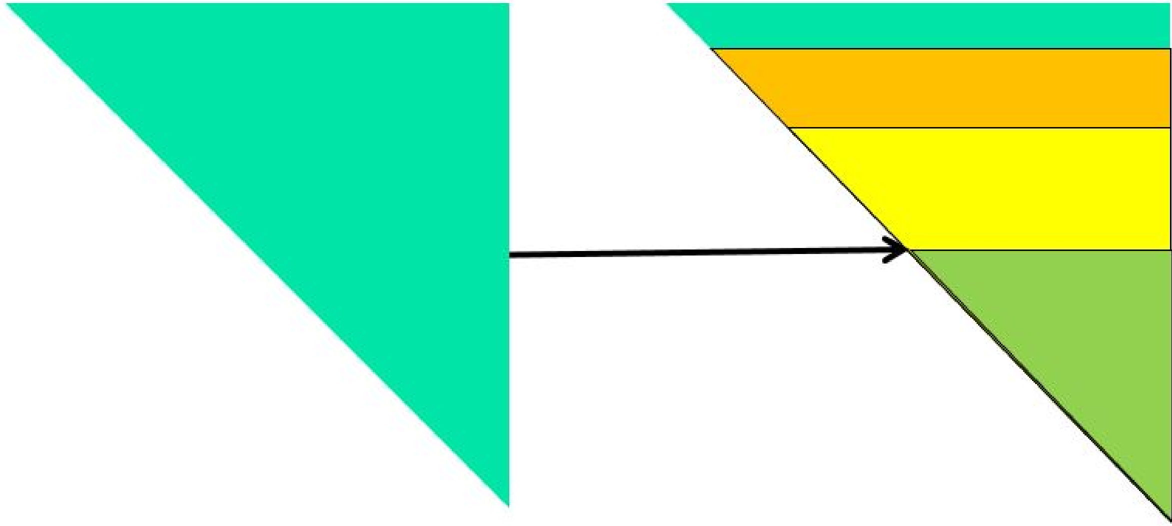


Figure 2.3: Assigned rows ($index_{start} - index_{end}$) for approximately equal work distribution example for $P_{MPI} = 4$

divided by rows (i.e., genes), effectively reducing the work division to a problem of $\binom{N}{2}$ row(gene) pairings divided across P_{MPI} processes.

Algorithm 10, Lines 2-6 calculated the work required to process each row(gene) and the cumulative work up to that row. Row j is paired with all rows greater than j for computational $\binom{N-j}{2}$ work (Line 4). Ideally every process will get $\frac{\binom{N}{3}}{P_{MPI}}$ work (Line 7). Each MPI process is assigned a starting work index and an ending work index by the root(0) process (Lines 8-18). This approximately equally load balances the work distribution such that each of the P_{MPI} processes would perform close to the ideal amount of work. A graphical representation of work division for $P_{MPI} = 4$ processors is showing in Figure 2.3.

The decision to divide work by rows made for a small sacrifice in workload balance across processors, although graphically in Figure 2.3 the imbalance appears significant. The appearance of imbalance is due to the 3 nested For-Loops in Algorithm 9. Smaller starting indexes for gene A For-Loop correspondingly will cause many more inner B and C For-Loop iterations. Algorithm 10 accounts for this (Lines 2-6) and distributes approximately equal amounts of total work (Lines 8-18.)

المنارة للاستشارات
www.manaraa.com

Each of the P_{MPI} processes calculates a result $O(N^2)$ matrix using the PCIT algorithm for their subset of assigned rows $(index_{start} - index_{end})$. Lastly the result matrices from all P_{MPI} are sent to the root(0) MPI process for merging into a single $O(N^2)$ result matrix.

2.5.2.1 Contributions of Watson-Haigh et al. (2010)

Their algorithm has the potential to dramatically improve the execution time. The underlying computation is embarrassingly parallel and hence for the computation could scale near linearly. The more resources available, the greater P_{MPI}, which translated into the better execution times.

Using the R platform enabled bioinformatics researchers already using that platform to easily deploy it in their workflow. This had the potential to expand the userbase of the algorithm.

Although this implementation used MPI, the embarrassingly parallel nature of the problem combined with providing all data to all MPI processes, meant that their implementation did not have the typical communication overheads experienced in other applications where the algorithm necessitates MPI communication to perform the computations. Had there been communication overheads during any of the computation loops then it would have been much harder to achieve their near linear computation scaling.

2.5.2.2 Challenges

The algorithm requires all data to be available in order to allow all rows(genes) to interact with all other rows(genes). Hence there was a natural requirement that each MPI process have a copy of the $O\left(N^2\right)$ gene correlation data.

This requirement appears to be the same memory challenge as the original algorithm [Reverter and Chan (2008)] bounding the maximum problem size at the memory size on the processing node. However, commonly multiple MPI processes execute on the same cluster node in the multi-core processor architectures. This dramatically increases memory demand to as much as $O\left(2P_{CORE}N^2\right)$, where P_{CORE} is the number of cores per node. Therefore the maximum problem size for multi-core architectures means making a choice between two options:

المنارة للاستشارات

www.manaraa.com

- one MPI process per core, resulting in division of the memory available equally to each of the processes for maximum execution speed
- one MPI process per node, resulting in the maximum memory available to each process in order to tackle those larger sized networks

Even medium sized networks that worked with the 2008 algorithm may run out of memory in the multi-core environment with as few as two cores per processing node, i.e. double the memory requirement per node. This forces the choice to sacrifice possible execution speed improvements from idle cores. Also, it could be considered an inefficient use of resources. Thus for large networks processed on clusters with multi-core architectures, the theoretical execution time was proportionately higher at $O\left(\frac{N^3}{P_{node}}\right)$, where P_{node} is the number of distributed processing nodes.

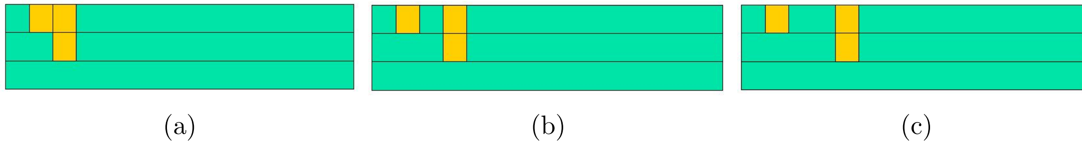
Conclusion is that while targeting execution time improvements was successful, the maximum problem size (i.e., scalability of problem size) remained the same and possibly even decreased. This memory complexity limits scalability to the larger problem sets.

2.5.3 Parallel PCIT algorithm using OpenMP

Koesterke et al. (2013, 2014) introduced dramatic algorithm updates reducing the computation complexity for the common case and optimized the parallel code further where possible. Their work was an extension of the original serial algorithm [Reverter and Chan (2008)]. This parallel version used OpenMP for a single node.

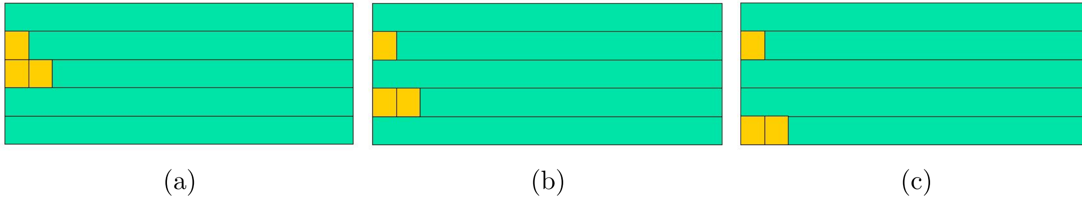
The first dramatic algorithm change was to enable stride-1 memory access, which produced as much as a 24x speedup. When bringing a value into the cache processing, the cache line that it belongs to also contains multiple neighboring array values. Figure 2.4 demonstrates a non-stride-1 memory access within a column-major memory layout. Although column 2 is reused in each time, columns 3, 4 and 5 are brought into the cache one after each other. Figure 2.5 demonstrates a stride-1 memory access of neighboring values in columns 1 and 2 within a column-major memory layout.

المنارة للاستشارات
www.manaraa.com



(a) (b) (c)

Figure 2.4: Example of a column-major, non-stride-1 memory access pattern



(a) (b) (c)

Figure 2.5: Example of a column-major, stride-1 memory access pattern

Fortran has column-wise memory layouts. When accessing a multiple dimensioned array the first index should be incremented in order to access a value in the neighboring memory address (stride-1). Whereas incrementing the second (or third, etc.) index, strides some N to a non-neighboring address.

The original Fortran code (Algorithm 11) had the inner loop incrementing the second index. The stride-1 code (Algorithm 12) reversed the loop variables such that the inner loop now increments the first index. As an aside, had the original code been implemented in C the row-major ordering of the memory would have had stride-1 memory access pattern and the new loop variables would not.

Algorithm 11 Before, not stride-1

1: **for** $A \leftarrow 1, N-2$ **do**
2: **for** $B \leftarrow A+1, N-1$ **do**
3: **for** $C \leftarrow B+1, N$ **do**
4: $r_{AB} \leftarrow c\,(A, B)$
5: $r_{AC} \leftarrow c\,(A, C)$
6: $r_{BC} \leftarrow c\,(B, C)$
...
7: **end for**
8: **end for**
9: **end for**

المنارة للاستشارات
www.manaraa.com

Algorithm 12 After, stride-1

```
1: for C ← 1, N − 2 do
2:     for B ← C + 1, N − 1 do
3:         for A ← B + 1, N do
4:             r_AB ← c(A, B)
5:             r_AC ← c(A, C)
6:             r_BC ← c(B, C)
         ...
7:         end for
8:     end for
9: end for
```

Computing divisions and square roots are costly operations. The original code had both of these operations:

$$r_{AB,C} = \frac{r_{AB} - r_{AC}r_{BC}}{\sqrt{\left(1 - r_{AC}^2\right)\left(1 - r_{BC}^2\right)}} \tag{2.7}$$

$$r_{AC,B} = \frac{r_{AC} - r_{AB}r_{BC}}{\sqrt{\left(1 - r_{AB}^2\right)\left(1 - r_{BC}^2\right)}} \tag{2.8}$$

$$r_{BC,A} = \frac{r_{BC} - r_{AC}r_{AB}}{\sqrt{\left(1 - r_{AC}^2\right)\left(1 - r_{AB}^2\right)}} \tag{2.9}$$

$$\varepsilon = \frac{1}{3}\left(\frac{r_{AB,C}}{r_{AB}} + \frac{r_{AC,B}}{r_{AC}} + \frac{r_{BC,A}}{r_{BC}}\right) \tag{2.10}$$

In addition, the same indexes may be computed multiple times across different loop iterations. So the updated code created two more $O\left(N^2\right)$ matrices to store precomputed values for these frequently computed and expensive operations.

$$ri_{AB} = \frac{1}{r_{AB}} \tag{2.11}$$

$$ris_{AB} = \sqrt{\frac{1}{\left(1 - r_{AB}^2\right)}} \tag{2.12}$$

This changed the loop computations to the cheaper multiplication operations.

$$r_{AB,C} = \left(r_{AB} - r_{AC}r_{BC}\right) * ris_{AC} * ris_{BC} \tag{2.13}$$

$$r_{AC,B} = \left(r_{AC} - r_{AB}r_{BC}\right) * ris_{AB} * ris_{BC} \tag{2.14}$$

$$r_{BC,A} = \left(r_{BC} - r_{AC}r_{AB}\right) * ris_{AC} * ris_{AB} \tag{2.15}$$

$$\varepsilon = one_third * \left(|r_{AB,C} * ri_{AB}| + |r_{AC,B} * ri_{AC}| + |r_{BC,A} * ri_{BC}|\right) \tag{2.16}$$

المنارة للاستشارات

www.manaraa.com

Algorithm 13 Algorithm with early exit

```
1: for C ← 1, N − 1 do
2:     for B ← C + 1, N do
3:         for A ← 1, N do
4:             if A == B||A == C then
5:                 continue to next A
6:             end if
       ...
7:             if (|r_BC| ≤ |εr_AB|) & (|r_BC| ≤ |εr_AC|) then
8:                 p_BC ← 0
9:                 break out of the inner loop
10:            end if
11:        end for
12:    end for
13: end for
```

The two previous changes were important, but this (Algorithm 13) was the key algorithm change that reduced the $\binom{N}{3} \rightarrow O\left(N^3\right)$ algorithm to a $O\left(N^{2+\epsilon}\right)$, where $0 \leq \epsilon \leq 1$. Line 3 uses both the upper and lower triangles of the $O\left(N^2\right)$ matrix of gene expression correlations, which also simplified the code (line 7) that determined non-significant correlations that used to be three separate conditional statements. The dramatic change came from recognizing that once a gene pair correlation was found to not be significant that it would not toggle and become significant again. This allowed for breaking early out of the inner For-Loop (Line 9.) If the pair is found to not be significant quickly, then the execution time could decrease by a factor as much as N in the best case.

Additionally, optimizations were done to vectorize the inner loop code to better utilize all processor resources. Because breaking early from a vectorized loop is not permitted, care was taken to chuck the loop processing. Flags were set at each vectorized loop iteration, and if the condition was met at the end of the chunk processing, then the inner loop was broken out of.

Parallel processing further decreased the computation execution time near linearly. The OpenMP framework uses multiple execution threads in a shared memory model. With the shared memory model only a single copy of the input, precalculated values, and result matrices $O(4N^2)$ is required.

المنارة للاستشارات
www.manaraa.com

Algorithm 14 Parallel algorithm with OpenMP

```
1: !$omp parallel do schedule(dynamic,10) &
2: !$omp shared(...)
3: !$omp private(...)
4: for C ← 1, N − 1 do
       ...
5: end for
```

Unlike the MPI method, this OpenMP method (Algorithm 14) does not require complicated work distribution methods. The loop processing is automatically broken into independent loop executions to execute in parallel across local lightweight threads. The framework can also handle dynamic work scheduling (Line 1) to ensure the $O(N^{2+\epsilon})$ work is distributed keeping all threads load balanced and finishing their work approximately at the same time.

This algorithm can only be executed on a single node and OpenMP uses the shared memory model, so it is not necessary to send or merge results in this method as all parallel threads are using the same address space.

Recognizing the scaling the problem size (i.e., larger N genes) can quickly grow larger than available memory, Koesterke et al. (2013) proposed an alternate algorithm with very low memory requirements as well. Rather than precomputing and storing three large $O\left(N^2\right)$ matrices for the correlations and precomputed expensive computations, the low-memory algorithm recalculates the gene expression correlations as needed. This is a trade-off. Of course this increased execution time of the computation; but when the problem size exceeds memory available, their low-memory algorithm is an option.

2.5.3.1 Contributions of Koesterke et al. (2013, 2014)

Their implementation sought to further reduce the execution time of the PCIT algorithm. The most dramatic of which was converting the original $O\left(N^3\right)$ algorithm to a $O\left(N^{2+\epsilon}\right)$ algorithm for a theoretical speed up of N. Several other contributions further added more speed up gains.

Precomputing expensive computations that were referenced multiple times, require two additional $O\left(N^2\right)$ matrices, but accelerated those areas of the code. This method required

المنارة للاستشارات
www.manaraa.com

$O(4N^2)$ memory for the input, precalculated values, and result data. This is only double from the original sequential algorithm so this method faces approximately the same problem size limitations, while delivering much improved execution time.

Also, writing the code to better use processor resources through vectorization helped. For their parallel implementation they chose a shared memory model with OpenMP, which avoided the multi-core memory scaling issue experienced using the MPI method before.

Their standard algorithm and the original algorithm both were limited in problem size scalability. Their low-memory algorithm is an application specific answer to that problem. This is important such that even with limited resources, a time trade-off can be made in order to even be able to compute the results.

2.5.3.2 Challenges

Their execution time improvement was dramatically successful; however, their algorithm required twice the memory as the original for the same problem. This effectively decreases the maximum problem size capability (i.e., scalability of problem size.) This memory complexity limits scalability to the larger problem sets.

Both the standard and alternate low-memory algorithm used the OpenMP framework, limiting their execution to a single node. Bioinformatics workflows commonly utilize cluster environments [Chae et al. (2013, 2014)] that more resources are available to scale memory and computation complexities further than this algorithm can do.

المنارة للاستشارات

www.manaraa.com

CHAPTER 3 OPTICAL COMMUNICATION NETWORKS

Fiber-optic lines make up the foundation of many networks across the globe. Failures within a network are to be expected and can happen as much as every couple days [Lastine et al. (2012)]. Protecting against these optical circuit faults is critical and there are many different approaches depending on the network needs and individual circumstances. SONET rings can be used to protect point-to-point and shared paths while enabling failure location. Using a pre-configured p-cycle backup [Grover and Shen (2003)], can also protect all node pair connections.

Knowing the unicast or multicast requests a priori is often not possible. This constraint makes protection against faults in those arbitrary communication paths a challenge. An efficient all node pairs protection scheme supporting both unicast and multicast communication is necessary.

For efficiency and distributed control, it is common in distributed systems and algorithms to group nodes into intersecting sets referred to as quorum sets. In Somani and Lastine (2014), it is shown that efficiency and distributed control can also be accomplished in optical network routing by applying the same established quorum set theory.

The next sections establish the foundation and context to our work to enhance optical network fault tolerance and efficiency.

3.1 Network Model

No two fiber-optic networks are the same. Some stretch hundreds of kilometers, while other networks are contained within buildings or rooms. Regardless of the physical environment, these optical circuits are depended upon for high-speed communications. Thus, it is important to extract the network's critical components that affect its ability to deliver reliable, arbitrary point-to-point and multi-point communications.

المنارة للاستشارات
www.manaraa.com

These fiber-optic networks consist of several transmitters and receivers interconnected by fiber-optic cables. As you might expect, transmitters and receivers are typically found together and generically called an optical node. The cables form the links (i.e., edges) between those nodes, which leads to a convenient model of a network in terms of a graph $G = (V, E)$. V are the set of nodes in the network and E are the set of edges.

Edge (e_i, e_j) is a fiber-optic link connecting nodes e_i and e_j in the network, where $e_i, e_j \in V$ and $(e_i, e_j) \in E$. It is a general assumption that the same set of optical wavelengths are available on all edges in E. The number of wavelengths available per optical fiber is dependent on the fiber-optic cables and the transmitter/receiver pairs.

3.2 Light-Trails

Lightpaths were a critical building block in the first optical communications, but required significant traffic engineering and aggregation to support point-to-point communication, or pay the penalty of low resource utilization on the fiber-optic link. Lightpaths cannot support multicast traffic. Light-trails were proposed in Gumaste and Chlamtac (2003); Chlamtac and Gumaste (2003) as a solution to the challenges facing lightpaths and could be built using commercial off-the-shelf technology. In the years since the introduction of light-trails, significant contributions have been made to enable adoption and advance the architecture [Lastine et al. (2012); Fang et al. (2004); Li et al. (2008); Zhang et al. (2011); Somani et al. (2011)].

Light-trails enable fast, dynamic creation of an unidirectional optical communication channel. This communication channel, unlike prior lightpaths, allows for channel receive and transmit access to all connected nodes, making them more suitable for IP-centric traffic [Fang et al. (2004)]. Point-to-point communications from an upstream node to a downstream node can be scheduled on the shared light-trail. Similarly, an upstream node can multicast to any number of downstream nodes.

A scheduling protocol is in place to avoid collisions within a light-trail and controls when nodes are able to transmit to downstream nodes. The scheduling is generally assumed to occur over a control channel, which may or may not be separate from the shared optical fiber that is being used for the light-trail.

المنارة للاستشارات
www.manaraa.com

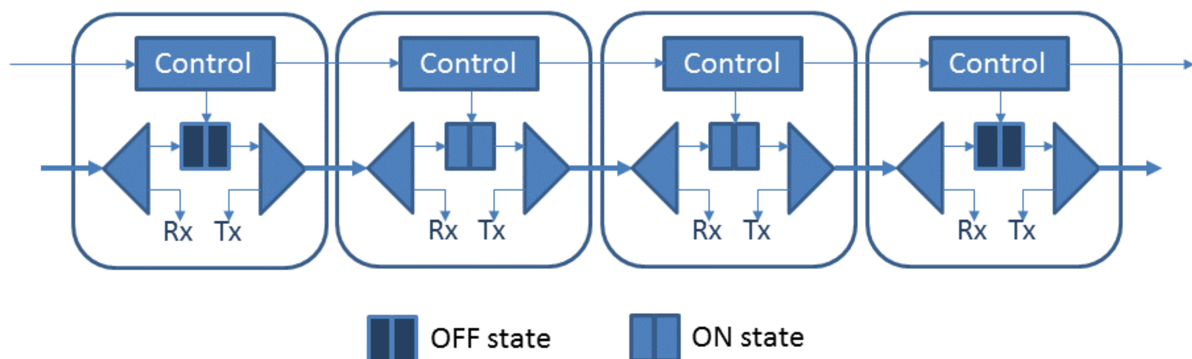


Figure 3.1: Four nodes in a light-trail architecture

An example four-node light-trail can be seen in Figure 3.1. Optical shutters allow for wavelength reuse within the network. Start and end nodes have their optical shutters in the *off* state, while intermediate nodes have their optical shutters in the *on* state. This effectively isolates an optical signal to a specific light-trail and allows for reuse of optical wavelength(s) elsewhere in the network.

Nodes can receive from the incoming signal while the signal is simultaneously continuing to downstream nodes, sometimes referred to as a drop and continue function. The node structure can be seen in Figure 3.2. DWDM fiber-optic networks use reconfigurable add/drop multiplexers (ROADMs) to demux incoming signals into separate wavelengths, then to mux the wavelengths before being output once again. Each wavelength can separately support the light-trail architecture, allowing multiple light-trails to share the same edge in the network. Next-generation ROADMs further increase add/drop and switching flexibility while reducing costs [Ji and Aono (2010)]. Early technology supported only a few wavelengths; however, the latest devices may support over 100 channels, hence allowing multiple light-trails to share the same edge in the network for a combined over 1-Terabits/s [Agrawal (2007)].

Light-trail communication is all optical and uses the same wavelength(s) from start to end node. Being all optical avoids any energy inefficiencies and time delays associated with unnecessary Optical-to-Electrical-to-Optical (O/E/O) conversions at intermediate hops. Transmissions within long haul networks, potentially passing through one or more nodes, used to be limited by the optical signal to noise ratio (OSNR). In recent years, several advancements and miti-

المنارة للاستشارات
www.manaraa.com

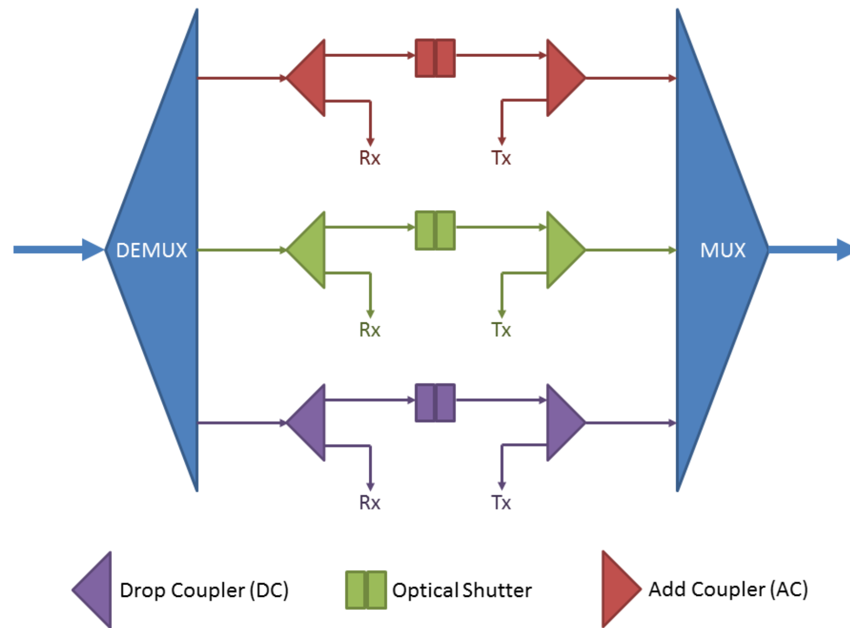


Figure 3.2: Example light-trail node structure

gating techniques have allowed for this limitation to be reduced, and in some cases, completely removed. One such advancement was the erbium- and ytterbium-doped optical fiber amplifier. These amplifiers compensate for signal losses and allow for signals to travel thousands of kilometers [Agrawal (2007)].

3.3 Light-Trails, Cycle Routing, and Fault Tolerance

Point-to-point and multi-point traffic requests have a set of nodes $C = \{e_i, ..., e_j\}$ that wish to communicate and need to be protected against network faults. Establishing a primary and backup multicast path from every node to every other node in C can be a waste of resources. Several methods protect the path or links along the route through an independently found tree or cycle. In this work, we utilize the light-trail architecture in the form of a cycle (Fig. 3.3). The bidirectional cycle will both route the multi-point request and protect it at the same time using fewer resources.

Figure 3.3 is simply a light-trail where the start and end node is the same node, referred to as the *hub node*. The hub node has its optical shutters in the *off* state, while intermediate

المنارة للاستشارات
www.manaraa.com

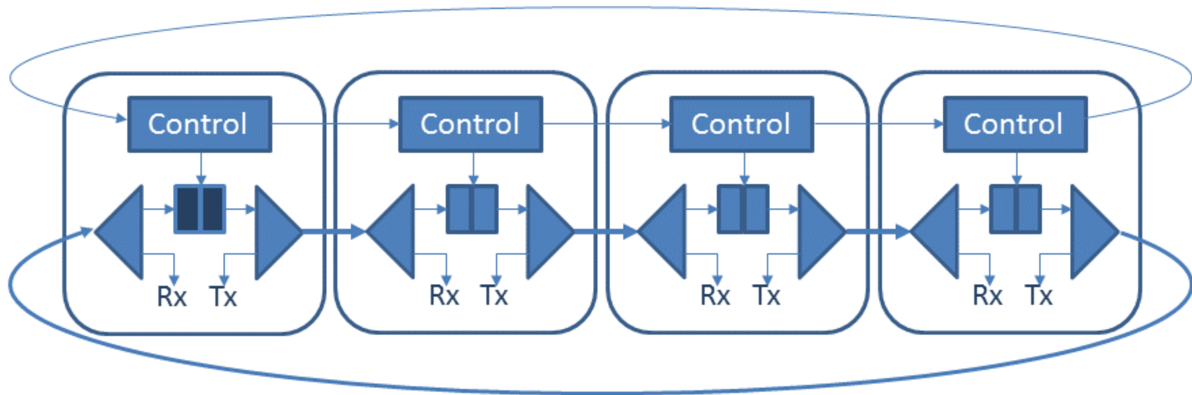


Figure 3.3: Cycle formed using the light-trail architecture

nodes have their optical shutters in the *on* state. The resources at each hub node can be utilized to allow all node pairs communication on the cycle using only one light-trail. Traffic from a node to nodes downstream requires a single transmission. Traffic from a node to an upstream node must undergo Optical-to-Electrical-to-Optical (O/E/O) conversion at the hub node and be transmitted on the light-trail a second time.

Alternately, we choose to set up a pair of light-trails, one in each direction. This enables upstream communications without the energy inefficiencies and time delays associated with O/E/O conversions. It also has fault tolerance properties.

Additionally, if traffic in the network is expected to be significantly different than equal traffic between all node pairs, multiple light-trails can be used to support those specific cycles expecting heavier traffic.

Failures within an optical network are to be expected. The generalization of p-cycle protection to allow for path and link protection was proposed by Grover and Shen (2003). P-cycle protection of unicast and multicast traffic networks requires preconfiguration, and the offline nature allows for the efficient cycles to be selected [Zhang and Zhong (2008); Zhang et al. (2008)]. Ramamurthy et al. (2003) examined both path-based and link-based protection schemes in WDM networks. The use of path-pair protection, link-based shared protection, spanning paths, and p-cycles to protect multicast sessions have all been proposed for WDM networks as well [Singhal et al. (2003); Qing and Ning (2005); Luo et al. (2006); Zhang and

www.manaraa.com

Zhong (2007); Feng et al. (2008)]. The Optimized Collapsed Rings (OCR) single link protection heuristic was developed to address the heterogeneous, part multicast / part unicast, nature of WDM traffic [Khalil et al. (2005)].

The multi-point cycle routing algorithm (MCRA) uses bidirectional cycles for fault tolerance and is capable of supporting SONET rings and p-cycles [Lastine et al. (2012)]. Although finding the smallest cycle supporting the multi-point communication is NP-Complete, the authors were able to show that their heuristic performed within 1.2 times of the optimal cycle size. ECBRA is a significant improvement of MCRA and outperforms the OCR heuristic [Somani et al. (2011)].

ECBRA heuristic balances optimality and speed, taking $O\left(|E|\,|C|^3\right)$ steps to find a close to optimal cycle. First, a modified breadth first search is performed on each node in set C of required communication nodes. The goal is to find a shortest path in G that also has the best ratio of nodes from set C versus total nodes on the path. The heuristic gives preference to paths with 2-degree C nodes as these 2-degree nodes are required to be a part of the cycle.

To complete the cycle, a path from the sink node returning to the source node must be found. No links may be used twice. If all nodes in C are in the cycle, then the cycle search is complete. Otherwise, a third step is required to add any missing nodes.

If needed, the final step iteratively removes edges from the cycle and inserts paths through missing nodes in C. Because insertion of the node can be cheaper by removing some links from the cycle rather than others, the optimal edge removal from the cycle and path replacement is computed for each missing node insertion.

3.4 Quorums Sets for Routing

Defining quorums and their relationship to all-pairs problems is in Chapter 5. However, it is important to establish quorums relationship to the foundation and context of our optical networking research. The network model $G = (V, E)$ that we are using was defined in Section 3.1. Quorums sets cover N entities, in this case $N = |V|$ optical network nodes. The number of entities, N, also defines the number of small subsets, i.e., quorums, that will be used in our solution.

المنارة للاستشارات
www.manaraa.com

Somani and Lastine (2014) used quorums for efficient point-to-point, multicast, and all-to-all traffic requests in optical networks. This is important because traffic in many optical networks is heterogeneous meaning the routing framework must be able to handle all types.

Point-to-point, multicast, and all-to-all traffic can be routed through an optical network with N cycles based on cyclic quorums. The following breaks down the handling of different traffic request types on the quorum supporting rings.

3.4.1 Point-to-point traffic

Somani and Lastine (2014) used cyclic quorums sets (see additional definitions in Chapter 5) as their basis for cycle routing to guarantee all possible node pairs occurred in at least one cycle (proven in Kleinheksel and Somani (2016)).

Lastine (2014) observed that node pairs may appear in multiple cycles and suggested using this to the network's advantage for load balancing. When new requests arrive and the source-destination node pair occur in multiple cycles, then the network could choose which cycle to service the request by selecting the light-trail cycle with the least load. Taking this one step further, Somani and Lastine (2014) proposed making cycles with cyclical quorums larger than the minimum required to further improve load balancing opportunities.

3.4.2 Multicast traffic

Optimally, if all multicast participants belong to the same cycle, then one cycle can be used. Realistically though, dynamic requests often will not be of this nature. Requests will span multiple quorums and/or be larger than a single quorum cycle. Hence in the worst case, no more than k cycles are required to efficiently route and protect traffic for each multicast traffic request (more discussion on this bound in the broadcast traffic discussion, Sect. 3.4.3).

For multicast cases requiring only a few cycles to communicate with all participants, then Li et al. (2008) presented an algorithm for light-trail ring networks that can be used. Using their techniques, a routing plan can be efficiently created.

Somani and Lastine (2014) proposed a multicast solution for requests known a priori and are smaller or equal in size to single cyclic quorum. In that case, the node ids of the participating

المنارة للاستشارات
www.manaraa.com

nodes could be mapped to a quorum guaranteeing that if a cycle route was found, their multicast traffic could also efficiently be routed. As the number of requests increase, the complexity of this process increases.

3.4.3 Broadcast traffic

Broadcast traffic is simply the worst case of a multicast traffic request. The upper bound of requiring no more than k cycles to route and protect broadcast traffic can be described as follows. In Chapter 5, any element will occur in at most k quorums. In those k quorums, all other entities must be present in order to form necessary point-to-point pairs as described before (Sect. 3.4.1 and proven by Kleinheksel and Somani (2016)). Hence, any optical node can communicate on all k quorum cycles that it is a member and reach all other optical nodes, thereby efficiently serving any dynamic broadcast request.

3.4.4 Efficiency analysis

Somani and Lastine (2014) compared the routing cycles based on cyclic quorum sets to that of more traditional point-to-point path-based connections. The traditional method used nearly double the number of resources and this was before considering any fault tolerance. Similar results were observed regarding hardware required at each optical node to support the traffic as well. This supports their claim that routing optical cycles based on cyclic quorums is efficient.

المنارة للاستشارات
www.manaraa.com

CHAPTER 4 ALL-PAIRS PROBLEM

The all-pairs problem (or "handshake" problem) occurs in many different fields and occurs in a broad classification of algorithms. On the surface the problem is very straight forward as shown in Figure 4.1. Given a set of elements (seven in our example), all elements are paired with all other elements. Notice that it is not necessary to explicitly form a (e_1, e_0) pair because the pair can be formed by the (e_0, e_1) pair already present.

4.1 General All-Pairs Problem Definition

The pseudocode for a general all-pairs algorithm would look like Algorithm 15. This could be applied to the pairing of any N elements. Those elements can be communication nodes in a network needing to exchange data with all other nodes. Equally likely, they can be a set of N data items needing to be paired and computed with all other data items.

Stated more formally:

$$\text{Set of } N \text{ elements } E_N = \{e_0, e_1, \ldots, e_{N-1}\} \tag{4.1}$$

$$Pair(e_i, e_j), \text{ where } 0 \leq i < N - 1 \text{ and } i < j < N \tag{4.2}$$

Equation 4.1 enumerates the N elements being paired, while Equation 4.2 performs all pairings resulting in $\binom{N}{2} = \frac{N(N-1)}{2}$ element pairings.

Algorithm 15 General all-pairs algorithms

```
1: Given: Array E_N
2: for i ← 0 to N − 2 do
3:     for j ← i + 1 to N − 1 do
4:         Perform pair (e_i, e_j)
5:     end for
6: end for
```

المنارة للاستشارات
www.manaraa.com

$$
\begin{array}{cccccc}
(e_0, e_1) & (e_0, e_2) & (e_0, e_3) & (e_0, e_4) & (e_0, e_5) & (e_0, e_6) \\
 & (e_1, e_2) & (e_1, e_3) & (e_1, e_4) & (e_1, e_5) & (e_1, e_6) \\
 & & (e_2, e_3) & (e_2, e_4) & (e_2, e_5) & (e_2, e_6) \\
 & & & (e_3, e_4) & (e_3, e_5) & (e_3, e_6) \\
 & & & & (e_4, e_5) & (e_4, e_6) \\
 & & & & & (e_5, e_6)
\end{array}
$$

Figure 4.1: All-pairs of seven elements.

4.2 Distributed All-Pairs Problem Definition

Applications for all-pairs computations can have very large input data and the time to compute a data element pairing can be significant. This motivates a need to use additional computation resources and hence, distribute the work across multiple cooperating processes. This might be in a local HPC system or in rented cluster resources in the cloud.

The distributed all-pairs problem distributes the $\binom{N}{2}$ element pairings across P processes. For this one can assume that $N \gg P$. Methods to perform this distribution of work and data vary, e.g., Watson-Haigh et al. (2010); Moretti et al. (2010); Plimpton (1995); Driscoll et al. (2013). Some implementations Watson-Haigh et al. (2010) give all of E_N to all processes and each process is responsible for a different portion of the $\binom{N}{2}$ element pairings. Other implementations have mechanisms to generate different subsets and then compute the global pairing of all of E_N by pairing individual subsets.

For example in Figure 4.2, Driscoll et al. Driscoll et al. (2013) distribute N data elements evenly across $\frac{P}{c}$ team columns, i.e., forming $\left|\hat{D}\right| = \frac{P}{c}$ datasets. To achieve the communication optimal lower bound, they showed that the replication factor should be $c = \sqrt{P}$. Each member in a team column receives a dataset copy of the same $\frac{N}{\sqrt{P}}$ data elements. Then, an additional $\frac{N}{\sqrt{P}}$ data element array is copied and shifted from within their respective replication row resulting

المنارة للاستشارات
www.manaraa.com

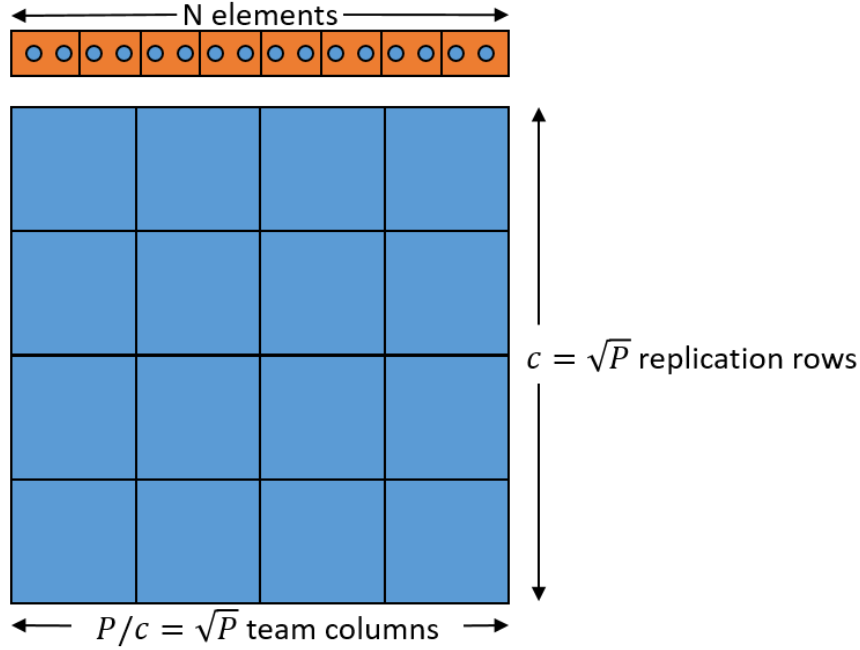


Figure 4.2: Driscoll et al. (2013) communication optimal n-body algorithm's data replication and distribution. Optimality achieved when $c = \sqrt{P}$, resulting in $\sqrt{P}$ teams, $\sqrt{P}$ replication rows, and each processor performing the pairing between two size-$\frac{N}{\sqrt{P}}$ arrays of elements. Note that their algorithm's communication steps are not depicted in this figure.

in all processors holding a pair of datasets (D_i, D_j). Their algorithm does this shift copy in such a way to generate all of the respective data pairings.

The general algorithm design approach for these distributed problems takes N data elements and groups them into smaller datasets, often a disjoint division by the number of processes P. However, Driscoll et al. (2013) for example used divisions by $\frac{P}{c}$. Equation 4.3 enumerates this set of P datasets. Datasets are a subset of the original dataset D (Eq. 4.4) and all elements from D must be present in at least one of the subsets (Eq. 4.5).

$$\text{Set of } P \text{ datasets } \hat{D} = \{D_0, D_1, \ldots, D_{P-1}\} \tag{4.3}$$

$$D_i \subseteq D,\ i \in 0, 1, \ldots, P-1 \tag{4.4}$$

$$D = \bigcup_{i=0}^{P-1} D_i \tag{4.5}$$

المنارة للاستشارات
www.manaraa.com

Equation 4.6 describes the global pairing of all datasets D_i and D_j. This is in contrast to Equation 4.2, which preforms the pairing of particular elements e_i and e_j. Additionally, in Equation 4.6 the range of index j is altered to allow for the pairing of D_i with itself. This is unnecessary in Equation 4.2 because elements in general would not need to be paired with themselves; however once placed in a subset, it is still necessary that elements within a subset be paired with others within the same subset and not simply with only other subsets.

$$Pair(D_i, D_j), \text{ where } 0 \leq i < P - 1 \text{ and } i \leq j < P \tag{4.6}$$

The global pairing of P datasets described must take place at some distributed process. In order for this to occur, there must be a process assigned to perform the pairing of (D_i, D_j) and that process must have datasets D_i and D_j available locally in order to carry out the pairing computation. To denote the set of datasets available to a particular process, we use set S_i, where i is the process's id. S_i is a subset of $\hat{D}$ (Eq. 4.7).

$$S_i \subseteq \hat{D},\ i \in 0, 1, \ldots, P - 1 \tag{4.7}$$

Just because a process has datasets in their set S_i does not mean they must compute all of the possible pairings though. Often there is a work assignment process for these distributed all-pairs algorithms. All or a subset of the datasets available can be paired to complete the distributed all-pairs problem. For example in Driscoll et al. (2013), at any one time a process would only have two datasets available so all (1) dataset pairings are computed. Whereas in Watson-Haigh et al. (2010), all of the data was available to each process, but processes were assigned only a subset of the possible pairings.

In the next chapter, we introduce the quorum sets that we use our all-pairs solutions.

www.manaraa.com

CHAPTER 5 QUORUMS AND CYCLIC QUORUMS

In distributed communication and algorithms, coordination, mutual exclusion, data replication and consensus implementations have grouped P processes or nodes into sets called quorums [Kumar and Agarwal (2011)]. This organization can minimize communications in operations like negotiating access to a global resource or reaching a joint, distributed decision.

A quorum set minimally has the property that all quorums must intersect. Specifically for distributed implementations, it is also desirable that each quorum have equal work and equal responsibility within the quorum set. Not every grouping of nodes into sets (quorums) will result in having these three properties, nor will the quorum sizes be minimal. Maekawa (1985) proved the lower bound on the size of a quorum set with these three properties. Cyclic quorum sets have these properties and we use them for efficient all-pairs communication, computation, and data replication management.

In the next sections we define what a quorum set is. We will be applying this to management of datasets in our computation applications, which look like distributed all-pairs problems (Section 4.2) and also applying this to management of optical cycle routing in our communication applications, which look like general all-pairs problems (Section 4.1). The general all-pairs problem pairs all elements in E_N and the distributed all-pairs problem pairs all subsets in $\hat{D}$. For the purpose of quorum definition, these sets are interchangeable. However for clarity, rather than expressing using both element and dataset representations, we will use the $\hat{D}$ dataset representation throughout.

المنارة للاستشارات
www.manaraa.com

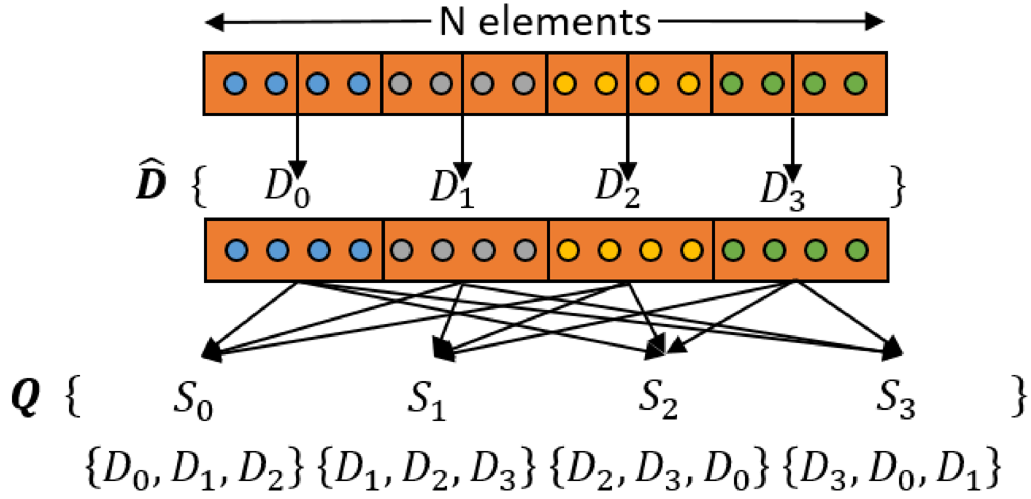


Figure 5.1: A quorum set example with N elements and $P = 4$ processes. Set $\hat{D}$ divides the N elements into $P = 4$ datasets D_0 through D_3. Quorum set Q is then formed from sets (quorums) of these datasets, i.e., S_0 through S_3.

5.1 Defining Quorum Sets

Set Q is a set of subsets (Eq. 5.1). Set Q is a quorum set, when Q's subsets covers all of $\hat{D}$ (Eq. 5.2) and all subsets also have non-empty intersections (Eq. 5.3).

$$Q = \{S_0, \ldots, S_{P-1}\} \tag{5.1}$$

$$\bigcup_{i=0}^{P-1} S_i = \{D_0, \ldots, D_{P-1}\} = \hat{D} \tag{5.2}$$

$$S_i \cap S_j \neq \emptyset, \; \forall \, i, j \in 0, 1, \ldots, P-1 \tag{5.3}$$

Figure 5.1 provides a visual representation of a quorum set for four processes. $\hat{D}$ is a set of four datasets, each containing a portion of the N elements. A quorum per process, S_0 through S_3, make up the quorum set Q. Each quorum can be seen to be a subset of the larger $\hat{D}$ and that all datasets in $\hat{D}$ are also present in Q. Additionally, each S_i set shares (intersects) at least one D_i dataset with the other S_j sets. For example, S_0 shares dataset D_0 with both S_2 and S_3 and shares dataset D_1 with both S_1 and S_3.

المنارة للاستشارات
www.manaraa.com

Additionally, it is desirable that each quorum S_i in the quorum set be of equal size (Eq. 5.4), such that there is equal work and it is desirable that each dataset D_i be contained in the same number of quorums (Eq. 5.5), such that there is equal responsibility. Both of these properties can be seen by inspection in Figure 5.1. All S_0 through S_3 contain the same $k = 3$ number of datasets (i.e., equal work) and counting the occurrences of each of the datasets D_0 through D_3 reveals that each occur $k = 3$ times in Q.

$$|S_i| = k, \; \forall \, i \in 0, 1, \ldots, P-1 \tag{5.4}$$

$$D_i \text{ is contained in } k \; S_j\text{'s}, \; \forall \, i \in 0, 1, \ldots, P-1 \tag{5.5}$$

Not every grouping of nodes into sets (quorums) will result in having these three properties, nor will the quorum sizes be minimal. The lower bounds for the maximum individual quorum size (i.e., $|S_i|$) in a minimum set is k, where Equation 5.6 holds and $(k-1)$ is a power of a prime, proved through equivalence to finding a finite projective plane [Maekawa (1985)]. Solving for k in Equation 5.6 results in a quorum size of $k \approx \sqrt{P}$ datasets.

$$P \leq k(k-1) + 1 \tag{5.6}$$

5.2 Defining Cyclic Quorum Sets

Cyclic quorum sets are based on cyclic block design and cyclic difference sets. However, searching for optimal sets requires an exhaustive search [Luk and Wong (1997)]. Cyclic quorum sets are unique in that once the first quorum (Eq. 5.7) is defined the remaining quorums in the set can be generated via incrementing the indices (modulus to keep indices within bounds is not shown in Equation 5.8 for conciseness).

$$S_0 = \{D_0, \ldots, D_j\} \tag{5.7}$$

$$S_i = \{D_{0+i}, \ldots, D_{j+i}\}, \; \forall \, i \in 0, 1, \ldots, P-1 \tag{5.8}$$

المنارة للاستشارات
www.manaraa.com

For simplicity, assume $D_0 \in S_0$ without loss of generality (any one-to-one re-mapping of indices can result in this assumption). This cyclic property can be seen in Figure 5.1 on Page 49. The datasets contained in S_1 all have indices one greater than the datasets found in S_0. Similarly, S_2's dataset indices are two greater than those in S_0 and rather than containing the $D_{2+2} = D_4$ dataset, which does not exist in figure, the modulus with $P = 4$ results in S_2 containing $D_{(2+2)mod4} = D_0$ dataset.

For our work, we used the $P = 4, \ldots, 111$ optimal cyclic quorums from Luk and Wong (1997) (see Appendix A for a reproduced and verified cyclic quorum listing). In the next section, we define and show a proof [Kleinheksel and Somani (2016)] that cyclic quorum sets have an all-pairs property that makes them ideal for all-pairs problems.

5.3 All-Pairs Property for Quorum Sets

Cyclical quorums were introduced in the previous section as having a small size ($|S_i| = k \approx \sqrt{P}$) and equal work/responsibility properties. However, it is not immediately evident how these small, equable cyclic quorum sets can support the pairing of all $\{D_0, \ldots, D_{P-1}\}$ datasets to solve all-pairs problems.

As all-pairs algorithms scale using multiple processes and distribute the $\binom{N}{2}$ pairs, it remains necessary that all pairs of elements are present in at least one process's memory in order to efficiently perform the pairing. Parallels with how this applies to the all-pairs computations may be evident; however, restating this to better clarify the connection with all-pairs communications may be needed. Networks require all nodes to be able to communicate, i.e., $\binom{N}{2}$ pairs. As networks grow it can become impractical to have every node connected directly to every other node. Rather it may be more realistic that nodes will cooperate to pass messages through multiple hops from source to destination. If we establish a set of such routings to execute all possible node pair communications, then here too we would need to make sure that all node pairs are present in at least one such routing.

In this section we define the all-pairs property for quorum sets and provide a proof that cyclical quorum sets satisfy this property.

www.manaraa.com

5.3.1 All-pairs property

In Section 4.2, we described how methods of distributing the work and data vary, e.g., Watson-Haigh et al. (2010); Moretti et al. (2010); Plimpton (1995); Driscoll et al. (2013). These all shared the same two basic components:

1. Processes each have a subset of the global data (Eq. 4.7)

2. Across the processes all dataset pairs are formed (Eq. 4.6)

Some implementations [Watson-Haigh et al. (2010)] gave all of D to all processes guaranteeing all element pairs could be formed. Other implementations have mechanisms to generate data subsets and permute them in a defined, predictable way to perform the global element pairing. Figure 4.2 on Page 46 showed a high level example of the distribution developed by Driscoll et al. (2013), where N data elements were evenly distributed and a shift copy permutation was used to generate all of the respective dataset pairings.

In order to define the all-pairs property for a distributed system, we first assign each process i a set S_i of datasets as previously stated in Equation 4.7. Then the following constraint must hold for the all-pairs property to be satisfied:

$$\exists S_i \ni (D_j, D_k) \ \forall \, j, k \in 0, 1, \ldots, P-1, \ \text{ where } S_i \in Q \tag{5.9}$$

Equation 5.9 states that for every pairing of datasets in $\hat{D}$ there exists at least one process's set S_i that contains the pair. Distributed systems with this all-pairs property can be used to satisfy Equation 4.6 from Page 47, that defined the work necessary to compute the distributed all-pairs problem.

5.3.2 Cyclic quorums have the all-pairs property

Our all-pairs methods utilize cyclic quorums to satisfy the all-pairs property with minimal node resources and communication. Each process i is assigned a quorum S_i of datasets. Definition 1 and Theorems 2 and 3 by Luk and Wong (1997) establish the relationship between cyclic quorum sets and relaxed difference sets. We use this relationship as part of our proof in Theorem 4 that cyclic quorums satisfy the all-pairs property [Kleinheksel and Somani (2016)].

المنارة للاستشارات
www.manaraa.com

$a_i - a_j = d$	1	2	3
1	$1-1=0$	$2-1=1$	$3-1=2$
2	$1-2=-1$	$2-2=0$	$3-2=1$
3	$1-3=-2$	$2-3=-1$	$3-3=0$

$d \bmod 6$	1	2	3
1	0	1	2
2	5	0	1
3	4	5	0

(a)

$a_i - a_j = d$	1	2	4
1	$1-1=0$	$2-1=1$	$4-1=3$
2	$1-2=-1$	$2-2=0$	$4-2=2$
4	$1-4=-3$	$2-4=-2$	$4-4=0$

$d \bmod 6$	1	2	4
1	0	1	3
2	5	0	2
4	3	4	0

(b)

Figure 5.2: Defining a relaxed (P, k)-difference set. For a given set $A = \{a_0, \ldots, a_k\}$ and integer P, all integers $0, \ldots, (P-1)$ must be formed from the differences modulus P of integer pairs from set A. Figure (a) shows an invalid difference set corresponding to set $A = \{1, 2, 3\}$ and $P = 6$ because no pair of integer differences modulus 6 form $d \, mod \, 6 = 3$. Whereas Figure (b) with set $A = \{1, 2, 4\}$ and $P = 6$ is a valid difference set because all integer differences modulus 6 are formed, i.e., $0, \ldots, 5$ are all present.

Definition 1. *Set $A = \{a_0, \ldots, a_k\}$ modulus P, $a_i \in 0, \ldots, P-1$ is a relaxed (P, k)-difference set if for every $d \neq 0$ modulus P, $\exists\,(a_i, a_j)$, $a_i, a_j \in A$ such that $a_i - a_j = d$ modulus P.*

Definition 1 [Luk and Wong (1997)] defines a relaxed difference set as a set of integers whose values are greater than or equal 0 and less than P. It has a restriction that every integer from 0 to (P-1) must also be able to be formed from the difference of some pair of integers in the set (using modulus when necessary). Figure 5.2 illustrates this definition through two examples. Figure 5.2 (a) forms all of the differences for $A = \{1, 2, 3\}$ on the top and then performs the modulus $P = 6$ on the bottom. This is not a valid difference set because $d \, mod \, 6 = 3$ is missing. Figure 5.2 (b), however, is a valid difference set because the differences for $A = \{1, 2, 4\}$ modulus $P = 6$ form all integers $0, \ldots, (P-1)$.

Theorem 2. *The cyclic quorum set Q defined by set $S_i = \{a_0 + i, \ldots, a_k + i\}$ modulus P, $i \in 0, \ldots, P-1$ is a relaxed (P, k)-difference set $A = \{a_0, \ldots, a_k\}$ modulus P, $a_i \in 0, \ldots, (P-1)$.*

The intuition for Theorem 2 [Luk and Wong (1997)] relies on the quorum set's intersection property, $S_i \cap S_j \neq \emptyset$, $\forall\, i, j$ (Eq. 5.3).

المنارة للاستشارات
www.manaraa.com

Proof. By contradiction, assume that set A was not a relaxed difference set, then there would be value $d \neq 0$ modulus P that no difference $(a_i - a_j)$ modulus P, $a_i, a_j \in A$ equaled.

Given that every quorum intersects in the set Q (Eq. 5.3), there must be a shared item in S_0 and S_d, where $d \in 0, \ldots, (P-1)$. Equation 5.10 assumes the shared item is at indices i and j, respectively, hence the shared item $S_{0,i}$ and $S_{d,j}$ are differenced on the left-hand side.

Using the cyclic quorum set definition, the values for the items are substituted on the right side. Equation 5.11 uses the quorum intersection to simplify the left side to 0 before rebalancing to show that the assumption that there was no $d = (a_i - a_j)$ modulus P is false, hence set A is a relaxed difference set.

$$S_{0,i} - S_{d,j} = (a_i + 0) - (a_j + d) \text{ modulus } P \tag{5.10}$$

$$a_i - a_j = d \text{ modulus } P \tag{5.11}$$

□

Theorem 3. *The relaxed (P, k)-difference set $A = \{a_0, \ldots, a_k\}$ modulus P is a cyclic quorum set Q defined by set $S_i = \{a_0 + i, \ldots, a_k + i\}$ modulus P, $i \in 0, \ldots, P-1$ and $a_i \in 0, \ldots, (P-1)$.*

The intuition for Theorem 3 [Luk and Wong (1997)] again relies on the quorum set's intersection property, $S_i \cap S_j \neq \emptyset$, $\forall\, i, j$ (Eq. 5.3).

Proof. By contradiction, assume that there were quorums S_x and S_y that did not intersect, i.e., they violated Equation 5.3 and hence set Q was not a quorum set.

Quorums S_x and S_y both have elements at indices i and j, respectively, where $i, j \in 0, \ldots, k$. Differencing these two elements results in Equation 5.12, where on the right the values for the elements are substituted using the cyclic quorum set definition. To show that $S_{x,i} = S_{y,j}$ for some combination of indices i and j, we set the left side to zero and rebalance for Equation 5.13. We are given that A is a relaxed difference set, so all differences $d mod P$ can be made from elements in A. Hence, there must be some combination of indices i and j, where $i, j \in 0, \ldots, k$ that result in $a_i - a_j = y - x \, mod \, P = d \, mod \, P$. This result confirms that $S_{x,i} = S_{y,j}$ for some

المنارة للاستشارات
www.manaraa.com

i and j and shows that our assumption that S_x and S_y that did not intersect was false, hence set Q is a quorum set.

$$S_{x,i} - S_{y,j} = (a_i + x) - (a_j + y) \text{ modulus } P \tag{5.12}$$

$$a_i - a_j = y - x \text{ modulus } P \tag{5.13}$$

□

Theorem 4. *The cyclic quorum set Q defined by set $S_i = \{a_0 + i, \ldots, a_k + i\}$ modulus P, $i \in 0, \ldots, P-1$ satisfies the all-pairs property (Section 5.3)*

Theorem 4 [Kleinheksel and Somani (2016)] uses the relationship between all differences occurring in a difference set (Definition 1) and that cyclic quorums are based on difference sets (Theorems 2 and 3) to prove that cyclic quorums satisfy the all-pairs property.

Proof. By contradiction, assume that the all-pairs property is not satisfied. Then there must be a pair of integers (a_x, a_y), $a_x, a_y \in 0, \ldots, P-1$ that are not present together in any quorum $S_i \in Q$.

Integer pair (a_x, a_y) have the following differences:

$$(a_x - a_y) \text{ modulus } P \text{ and } (a_y - a_x) \text{ modulus } P \tag{5.14}$$

From Theorem 2, we know that $A = \{a_0, \ldots, a_k\}$ is a relaxed difference set and that all differences $d \neq 0$ modulus P exist. So, the differences formed by (a_x, a_y) are also formed at least once from the difference set A. Assume that integers (a_i, a_j), $a_i, a_j \in A$ form those specific differences:

$$(a_i - a_j) \text{ modulus } P \text{ and } (a_j - a_i) \text{ modulus } P \tag{5.15}$$

Using the cyclic quorum set definition and distributive property of modular arithmetic, Equation 5.16 shows all S_i cyclic quorums form the same differences. So the differences in

www.manaraa.com

Equation 5.14, formed by the integer pair (a_x, a_y) not present together in any quorum, are formed in every cyclic quorum S_i.

$$\begin{aligned}(a_i - a_j) \bmod P &= (a_i + m) - (a_j + m) \bmod P \\ &= S_{m,i} - S_{m,j} \bmod P \textit{ and} \\ (a_j - a_i) \bmod P &= (a_j + m) - (a_i + m) \bmod P \\ &= S_{m,j} - S_{m,i} \bmod P, \\ \forall m \in 0, \ldots, P-1\end{aligned} \tag{5.16}$$

Equation 5.16 reveals that all cyclic quorums have the missing differences in Equation 5.14. The cyclic $a_i + m$ modulus P definition guarantees there is an m that $a_i + m$ modulus P equals a_x and the difference with a_j will still hold such that $a_j + m$ modulus P equals a_y too (Eq. 5.17).

$$\begin{aligned}a_x &= (a_i + m) \text{ modulus } P = S_{m,i} \\ a_y &= (a_j + m) \text{ modulus } P = S_{m,j}\end{aligned} \tag{5.17}$$

Equation 5.17 show that integers (a_x, a_y), $a_x, a_y \in 0, \ldots, P-1$ are present together in quorum S_m defined by difference set A and integer m modulus P. This contradicts the assumption that the pairs are not present together, hence cyclic quorum sets do satisfy the all-pairs property. □

Figure 5.3 is a visualization of cyclic quorum sets having the all-pairs property. The quorums on the left are colored in various shades and the corresponding dataset pairs that can be formed are colored that same shade on the right. All dataset pairs on the right are covered. Also note that all quorums performed the same amount of work and were able to cover the same number of pairs, so the data distribution results in load balanced work.

To follow Theorem 4 at a high level with Figure 5.3, one can pick any two numbers in $0, \ldots, 6$, e.g., 2 and 6. Making the assumption 2 and 6 are not present together would result in the all-pairs property not being satisfied. Notice that quorum S_0 has datasets with indices corresponding to the difference set $A = \{0, 1, 3\}$. The difference of the two chosen numbers (modulus 7 as necessary), e.g., $2 - 6 = -4 \, mod \, 7 = 3$, is then used to find the pair of numbers

المنارة للاستشارات
www.manaraa.com

Quorums						
$S_0 = \{D_0, D_1, D_3\}$	(D_0, D_1)	(D_0, D_2)	(D_0, D_3)	(D_0, D_4)	(D_0, D_5)	(D_0, D_6)
$S_1 = \{D_1, D_2, D_4\}$		(D_1, D_2)	(D_1, D_3)	(D_1, D_4)	(D_1, D_5)	(D_1, D_6)
$S_2 = \{D_2, D_3, D_5\}$			(D_2, D_3)	(D_2, D_4)	(D_2, D_5)	(D_2, D_6)
$S_3 = \{D_3, D_4, D_6\}$				(D_3, D_4)	(D_3, D_5)	(D_3, D_6)
$S_4 = \{D_4, D_5, D_0\}$					(D_4, D_5)	(D_4, D_6)
$S_5 = \{D_5, D_6, D_1\}$						(D_5, D_6)
$S_6 = \{D_6, D_0, D_2\}$						

Figure 5.3: A cyclic quorum set example with 7 processes. On the left are the 7 quorums and on the right are all of the dataset pairings. The quorums and the corresponding pairs formed are colored. As Theorem 4 states, all pairs have been covered by a quorum set.

in $A = \{0, 1, 3\}$ that when differenced and modulus 7 result in the same difference, e.g., $3 - 0 = 3 \; mod \; 7 = 3$. Lastly, beginning with that pair from A just found, e.g., 3 and 0, one can begin incrementing both numbers (modulus 7 as necessary) until finding the original pair of numbers assumed not to be present together, e.g., 2 and 6. Finding the pair of numbers in the same quorum set proves the assumption was false and the all-pairs property holds.

5.4 Redundant Cyclic Quorums Sets

In this section we define and generalize R redundant cyclic quorums sets. Cyclic quorum sets were proven to have an "all-pairs" property in the previous section. In our communication applications, the quorum-based cycle routing solution used these cyclic quorums to form a set of communication cycles which were shown to be almost fault tolerant in fiber optic networks [Somani and Lastine (2014); Kleinheksel and Somani (2015b)].

As defined in Section 5.1, there are P quorums in a quorums set (Q) and each quorum has k datasets. In analysis of networking capabilities, we are interested in whether every node can communicate with every other node (all-pairs). Equation 5.18 considers the number of pairs within a quorum, e.g., the pairs made between k nodes communicating with $(k-1)$ other nodes in a single quorum. Equation 5.19 considers the total pairs formed by all P quorums in the set.

المنارة للاستشارات
www.manaraa.com

For convenience we set M to be the total pairs for a given network with P nodes and k optimal quorum size.

$$\frac{k(k-1)}{2} = O(k^2) \tag{5.18}$$

$$P\frac{k(k-1)}{2} = O(Pk^2) \tag{5.19}$$

$$M = O(Pk^2) \tag{5.20}$$

When the quorum size, k, is minimal or larger, every pair of nodes (e_i, e_j) occurs together within a quorum in the set at least once (proven in Sect. 5.3.2). Optical networking, however, requires all directional point-to-point pairs to exist, i.e., both pairs (e_i, e_j) and (e_j, e_i). Previously this had been addressed by pairing each cycle with the same cycle and its direction reversed.

It was observed by Lastine (2014) that the quorum-based cycle routing solution had some node pairings occurring together in multiple cycles and it was proposed that these could be used for load balancing. As an alternative to that option, Kleinheksel and Somani (2015c) added a requirement that every pair (e_i, e_j) would occur together within at least two quorums rather than just one. Exploiting the natural occurrence of redundant pairs is an attempt to eliminate the need for paired cycles, thus moving the redundancy from the paired cycles and putting the redundancy in the quorums themselves.

The number of quorums in the solution remained the same P, hence to create the additional pairs the quorum size had to be enlarged to $\hat{k}$. Equation 5.21 calculates the number of node pairs in quorums of size $\hat{k}$. Equation 5.22 is our requirement that the total number of pairs have doubled from the original total pairs, M. Finally, Equation 5.23 solves for size $\hat{k}$ in relation to optimal k.

$$P\frac{\hat{k}(\hat{k}-1)}{2} = O(P\hat{k}^2) \tag{5.21}$$

$$O(P\hat{k}^2) = 2M \tag{5.22}$$

$$\hat{k} \approx \sqrt{2}k \tag{5.23}$$

المنارة للاستشارات
www.manaraa.com

This result is powerful. The number of node pairs doubled (2x), but the size of k only increased by a factor of $\sqrt{2}$. Using this reduced growth rate to our advantage, many node pairs can be created without substantially increasing the resources used. Additionally, this growth rate is far slower than simply duplicating a cycle, hence significantly challenging the need for paired cycles and opening the door for considerable resource savings.

Still there are applications that may benefit from improved fault tolerance, therefore we further generalize this approach for a generic desired R redundant factor to offer an opportunity to enhance the fault tolerance of our quorum-based cycle routing solution [Kleinheksel and Somani (2015a)]. Equation 5.24 balances the enlarged quorum size $\hat{k}$ solution against the known R times the optimal solution. Equation 5.25 solves for $\hat{k}$ in terms of known k.

$$O(P\hat{k}^2) = RM \tag{5.24}$$

$$\hat{k} \approx \sqrt{R}k \tag{5.25}$$

Equation 5.25 defines a quorum size, $\hat{k}$, for a given R in terms of the original optimal k value. To express minimum $\hat{k}$ in terms of just R redundancy and P nodes, we can revisit the optimal cyclic quorum set size, Equation 5.6. Using the same structure we can put minimum necessary redundant all-pairs count on the left side and the total quorum pair count on the right (Eq. 5.26). Simplifying the like terms, we arrive at Equation 5.27 and finally after rebalancing Equation 5.28.

$$R\frac{P(P-1)}{2} \leq P\frac{\hat{k}(\hat{k}-1)}{2} \tag{5.26}$$

$$R(P-1) \leq \hat{k}(\hat{k}-1) \tag{5.27}$$

$$P \leq \frac{\hat{k}(\hat{k}-1)}{R} + 1 \tag{5.28}$$

Although we are defining R redundant pairs, observe the definition of a cyclic quorum still holds (Sect. 5.2). Definition 1 and Theorems 2, 3, and 4 go unchanged because we are still using cyclic quorums. The only observable change is an increase in quorum size to accommodate that there are more pairings required by definition now. To further emphasize this, consider the two

المنارة للاستشارات
www.manaraa.com

examples in Figure 5.4 for $R = 2$ and $P = 7$. Figure 5.4 (a) forms all of the differences for $A = \{1, 2, 3, 4\}$ on the top and then performs the modulus $P = 7$ on the bottom. This is not a valid difference set for $R = 2$ redundant cyclic quorum because $d\,mod\,7 = 3$ and $d\,mod\,7 = 4$ only occur once, when theorem 4 would lead us to conclude these differences must occur twice in order to form $R = 2$ pairs. Figure 5.4 (b), however, is a valid difference set because the differences for $A = \{1, 2, 3, 5\}$ modulus $P = 7$ form all integers $0, \ldots, (P - 1)$ twice.

To the best of our knowledge, no efficient algorithm is known to find quorums of minimum size, particularly with the additional requirement that entity pairs appear a minimum R times within the quorums set solution. Luk and Wong (1997) used a brute force search to find optimal cyclic quorums for $N = 4 \ldots 111$. Using our generalized result from Eq. 5.25, we too used a brute force search beginning with the smallest possible quorum size for a given number of nodes P and a given desired redundancy factor R (see Appendix B for the redundant cyclic quorum listing).

The resulting redundant quorums were utilized in Chapter 7, as we analyzed and enhanced the efficiency and fault tolerance of quorum-based cycle routing in optical networking.

المنارة للاستشارات

www.manaraa.com

$a_i - a_j = d$	1	2	3	4
1	$1-1=0$	$2-1=1$	$3-1=2$	$4-1=3$
2	$1-2=-1$	$2-2=0$	$3-2=1$	$4-2=2$
3	$1-3=-2$	$2-3=-1$	$3-3=0$	$4-3=1$
4	$1-4=-3$	$2-4=-2$	$3-4=-1$	$4-4=0$

$d \bmod 7$	1	2	3	4
1	0	1	2	3
2	6	0	1	2
3	5	6	0	1
4	4	5	6	0

(a)

$a_i - a_j = d$	1	2	3	5
1	$1-1=0$	$2-1=1$	$3-1=2$	$5-1=4$
2	$1-2=-1$	$2-2=0$	$3-2=1$	$5-2=3$
3	$1-3=-2$	$2-3=-1$	$3-3=0$	$5-3=2$
5	$1-5=-4$	$2-5=-3$	$3-5=-2$	$5-5=0$

$d \bmod 7$	1	2	3	5
1	0	1	2	4
2	6	0	1	3
3	5	6	0	2
5	3	4	5	0

(b)

Figure 5.4: Defining a relaxed (P, k)-difference set for an $R = 2$ redundant cyclic quorum. For a given set $A = \{a_0, \ldots, a_k\}$ and integer P, all integers $0, \ldots, (P-1)$ must be formed twice from the differences modulus P of integer pairs from set A. Figure (a) shows an invalid difference set corresponding to $A = \{1, 2, 3, 4\}$ and $P = 7$ because integer differences modulus 7 formed $d \, mod \, 7 = 3$ and $d \, mod \, 7 = 4$ only once. Whereas Figure (b) with $A = \{1, 2, 3, 5\}$ and $P = 7$ is a valid difference set for an $R = 2$ redundant cyclic quorum because all integer differences modulus 7 were formed twice, i.e., $0, \ldots, 6$ are all present twice.

المنارة للاستشارات

www.manaraa.com

CHAPTER 6 ALL-PAIRS APPLICATIONS IN COMPUTATION OPTIMIZATIONS

To evaluate the performance of our cyclic quorum set method, we modified an existing all-pairs application [Koesterke et al. (2013)] to scale to larger datasets and at the same time be able to utilize more resources. The algorithm implemented the distributed all-pairs problem defined in Equation 4.6 using the cyclical quorum sets defined in Section 5.2.

6.1 Bioinformatics PCIT Application

The partial correlation coefficients combined with an information theory approach (PCIT) algorithm was introduced by Reverter and Chan (2008). The algorithm can be used as component to the work flow for gene co-expression network reconstruction and for helping to identify novel biological regulators. The technique processes N genes (and candidate genes) by building an $O\left(N^2\right)$ matrix and using a guilt-by-association heuristic to analyze gene pair partial correlations to identify using purely data whether a gene expression correlation is or is not meaningful. More background information on gene co-expression networks and the PCIT algorithm can be found in Section 2.5.

6.2 Test Setup

Virtually all research fields have seen a dramatic increase in data collection and processing over the past decade. The field of bioinformatics is no stranger to having to scale their algorithms to larger datasets by utilizing more resources. There, as in many research fields, scientists are turning to cloud systems to meet their needs [Chae et al. (2013, 2014)]. These cloud delivered systems are continuing to advance and can provide graphical user interfaces, connecting to existing cloud datasets, security, and providing tools capable of running in a cloud environment.

المنارة للاستشارات
www.manaraa.com

Table 6.1: Input Datasets Utilized in PCIT Experiments

Type	Rows	Columns
***Cattle**	27364	5
Simulated	33331	5
Simulated	39298	5
***Mice**	45265	5
Simulated	51232	1893
***Rice**	57194	1893
Simulated	63166	1893
Simulated	69133	1893
Simulated	75000	1893

Amazon's AWS cloud is one of the most common providers. They can provide high performance computing clusters for rent by-the-hour eliminating the hurdles of high upfront capital costs and ongoing maintenance. For HPC applications like the PCIT algorithm or any other all-pairs problem, they offer their latest generation of compute-optimized "C4" instances [Amazon (2016b)]. These have Intel Xeon E5 v3 processors with the largest instance ("c4.8xlarge") having 36 virtual CPUs and 60GB of memory [Amazon (2016a)].

We conducted our application experiments using Cyence, an HPC system at Iowa State University. Every node has dual Intel Xeon E5 8-core processors and 128GB of memory. Our executions ranged from 16 to 512 cores (1 to 32 nodes). In order to better model how our application would execute in a similar HPC environment to that of Amazon's, we restricted our application's memory usage to only 60GB per node.

Three real and six simulated input datasets were used in our testing. Table 6.1 marks the real datasets with an asterisk. Simulated datasets were generated for a particular number of rows and columns using a technique published by Reverter and Chan (2008). Number of input rows (N) is the primary determinate of the processing complexity of all-pairs algorithms, hence our input sizes were varied with increasing number of gene (and gene candidate) rows. The number of input columns for the simulated datasets were chosen to match that of similar real input datasets. Input columns correspond to the number of test subject conditions, e.g., the number of cattle, mice, or rice test samples.

www.manaraa.com

The single node PCIT algorithm from Koesterke et al. (2013) was run with 16 OpenMP threads on a node by itself. The quorum implementations ran with 4 to 32 nodes (64 to 512 cores) with one MPI process and 16 OpenMP threads per node. The number of HPC nodes was varied to demonstrate interesting strengths and weaknesses in the choice of number of parallel nodes selected. Common choices for number of parallel nodes are powers of 2, i.e., 4, 8, 16, and 32 nodes. However, cyclic quorum sets based on Singer difference sets [Colbourn (2010)] can have an advantage (more discussion on this in Section 6.4), so 7, 13, and 31 nodes are also tested.

6.3 Results

For each pair of dataset input and number of HPC nodes, we executed 30 runs of the application in order to establish confidence in the runtime measurements. There is a column for every dataset input and a row for the number of nodes used. The memory used per node is in Table 6.2 and the observed average execution runtimes with 95% confidence intervals is in Table 6.3.

6.3.1 Memory usage performance

The single node instance in Table 6.2 is the PCIT algorithm from Koesterke et al. (2013) for comparison. As additional nodes are used, our cyclic quorums implementation requires fewer memory resources per node to execute the same dataset. This is because not all of the input, intermediate, or result data are stored at a single node. Instead, the data is distributed in a predictable way using the cyclic quorum design described in Section 5.2 on Page 50. This resulted in up to an 80% reduction in memory requirements per node. This was a critical result for our larger real and simulated input datasets where memory used per node ran into the upper 60GB limit for the single node tests and some of our parallel tests as well. These instances have been marked with bold in Table 6.2.

While generally increasing the number of nodes will reduce the memory used, there are some local minimums that initially may not be expected by a user of our cyclic quorums algorithm. For example, choosing 7 nodes instead of 8 actually results in a lower memory usage (57% vs.

المنارة للاستشارات
www.manaraa.com

Table 6.2: Memory Used Per Node (GB)

#Nodes	*27364	33331	39298	*45265	51232	*57194	63166	69133	75000
1	25.113	37.257	51.789	**68.708**	**88.736**	**110.495**	**134.680**	**161.233**	**189.669**
4	18.831	27.939	38.837	51.526	**66.725**	**83.064**	**101.225**	**121.159**	**142.506**
7	10.762	15.967	22.193	29.445	38.439	47.812	58.224	**69.651**	**81.888**
8	12.555	18.627	25.893	34.352	44.724	55.647	**67.782**	**81.099**	**95.357**
13	7.727	11.463	15.934	21.140	27.801	34.554	42.054	50.282	59.091
16	7.848	11.643	16.185	21.471	28.224	35.083	42.698	51.054	59.996
31	4.862	7.213	10.025	13.302	17.760	22.035	26.787	31.994	37.563
32	5.495	8.151	11.331	15.033	19.974	24.802	30.156	36.032	42.316

المنارة للاستشارات
www.manaraa.com

only 50% reduction), which is a bit counter intuitive upon first inspection. In the $N = 63166$ row test, this resulted in the 8-node execution exceeding the 60GB memory limit, while with one fewer nodes, the 7-node test stayed under the limit. This can be explained by cyclic quorum sets based on Singer difference sets [Colbourn (2010)] can have some advantages like requiring fewer datasets (D_i) to still guarantee the all-pairs property (Section 5.3). So 7, 13, and 31 nodes reduce memory usage per node by 57%, 69%, and 80% respectively, while if the number of nodes were increased by a small amount to 8, 16, or 32 the reduction is slightly less at 50%, 68%, and 77%. More discussion on this topic is in Section 6.4.

6.3.2 Runtime execution performance

Similar to the analysis of the memory, in Table 6.3 the single node instance used the PCIT algorithm from Koesterke et al. (2013). Our cyclic quorums implementation benefits from the additional parallel processing nodes. The all-pairs computation work is equally distributed and the resulting average runtimes on the same datasets decrease significantly. The average runtimes observed over 30 executions are in Table 6.3 with their 95% confidence intervals. As described in our test setup (Section 6.2), we used a generous 60GB memory limit that even current datasets pushed up against (Table 6.2), and if trends continue, future datasets will push even further beyond. Where this limit was exceeded we did not collect execution runtime data, except in one single node instance which is marked with bold. Here the single node algorithm had to revert to a lower memory alternative, and consequently, the runtime increased dramatically as a trade off.

The speedup (or scalability) of our algorithm is calculated as how many times faster our algorithm runs given additional processing nodes (P) with respect to the optimized single node algorithm, i.e., $SpeedUp = \frac{Time_{single}}{Time_{parallel}(P)}$. Figure 6.1 shows the speedup curves for three smaller datasets all of which fit within the available memory constraints. The curves are nearly identical for the datasets, hence the increased execution speed is independent of the dataset and can be attributed to our algorithm's ability to distribute the data and all-pairs computation work. The trend of the curves on the log-log scale is close to linear demonstrating the ability to utilize additional resources to arrive at results in a shorter amount of time efficiently. The

المنارة للاستشارات
www.manaraa.com

Table 6.3: Average Execution Runtimes (Seconds)

#Nodes	*27364	33331	39298	*45265	51232	*57194	63166	69133	75000
1	119.1 ± 2.9	162.5 ± 0.1	245.5 ± 0.1	**2312.5 ± 0.2**	-	-	-	-	-
4	52.1 ± 0.0	73.3 ± 0.1	112.0 ± 0.5	170.6 ± 0.1	-	-	-	-	-
7	17.9 ± 0.0	25.5 ± 0.0	38.2 ± 0.0	60.5 ± 0.0	317.5 ± 0.9	1209.8 ± 2.0	490.8 ± 2.3	-	-
8	24.8 ± 0.0	35.3 ± 0.0	53.2 ± 0.0	83.1 ± 0.0	364.2 ± 1.2	1631.4 ± 2.2	-	-	-
13	10.1 ± 0.0	14.4 ± 0.0	21.3 ± 0.0	32.1 ± 0.0	246.3 ± 2.4	784.8 ± 1.5	372.8 ± 2.1	446.1 ± 1.2	522.8 ± 2.9
16	10.6 ± 0.0	15.2 ± 0.0	22.5 ± 0.0	35.1 ± 0.0	251.2 ± 2.3	827.0 ± 2.0	377.9 ± 2.1	454.2 ± 1.2	530.5 ± 2.0
31	4.6 ± 0.0	6.4 ± 0.0	9.9 ± 0.1	14.7 ± 0.0	178.6 ± 1.2	407.8 ± 2.5	250.1 ± 1.8	306.4 ± 1.1	361.1 ± 1.5
32	5.8 ± 0.0	8.3 ± 0.0	12.2 ± 0.0	18.9 ± 0.0	197.9 ± 1.2	516.4 ± 1.2	284.6 ± 0.7	337.4 ± 2.1	394.6 ± 2.2

المنارة للاستشارات
www.manaraa.com

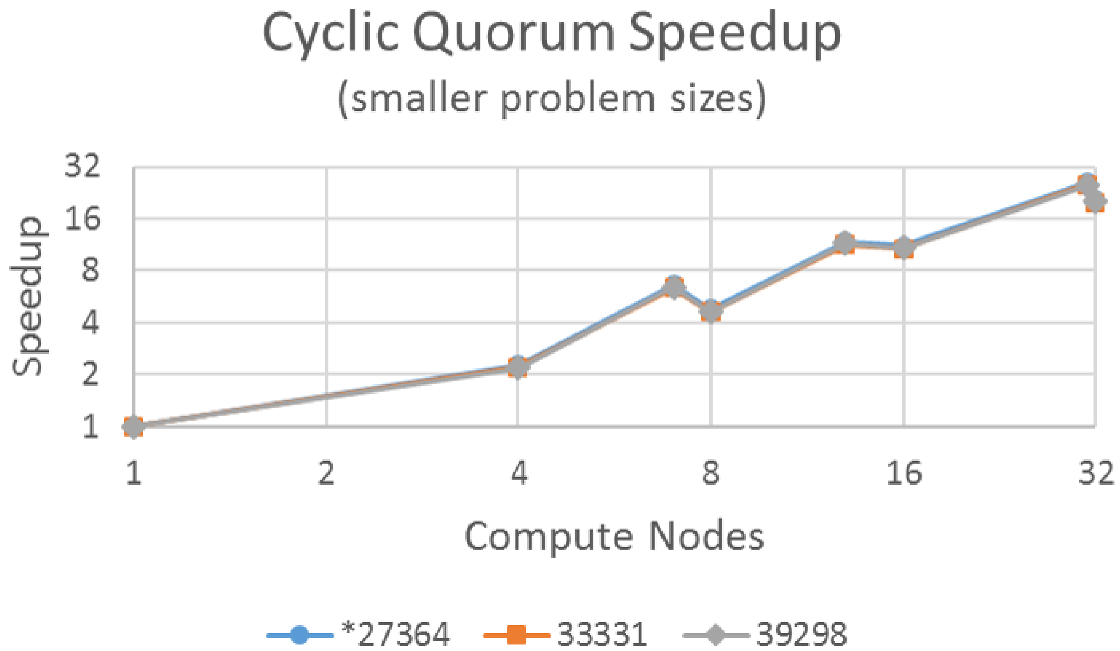


Figure 6.1: Speedup of our cyclic quorum algorithm using (P) parallel nodes when compared to an optimized single node algorithm. This figure has near identical speedup curves for the computation of three smaller datasets that all fit within the available memory. The log-log scale shows that as additional node resources are added our algorithm scales to utilize the resources.

speedup is not ideally linear however, as seen by the speedup value falling slightly short of the number of nodes used. As expected, this indicates there are some overheads when parallelizing across multiple nodes that was not present in the optimized single node algorithm. Additionally, the curves are not smooth with the increasing nodes used, i.e., speedup of $P = 7, 13,$ and 31 exceeds that of $P = 8, 16,$ and 32. This can be explained by cyclic quorum sets based on Singer difference sets [Colbourn (2010)] can have some advantages like requiring fewer datasets (D_i) to guarantee the all-pairs property (Section 5.3), hence there are fewer datasets at each node being paired with each other meaning less work to be performed. This is an interesting result and creates the question of whether additional computation management could improve the performance of quorum sets not based on Singer difference sets, which is the topic in Section 6.4.

المنارة للاستشارات
www.manaraa.com

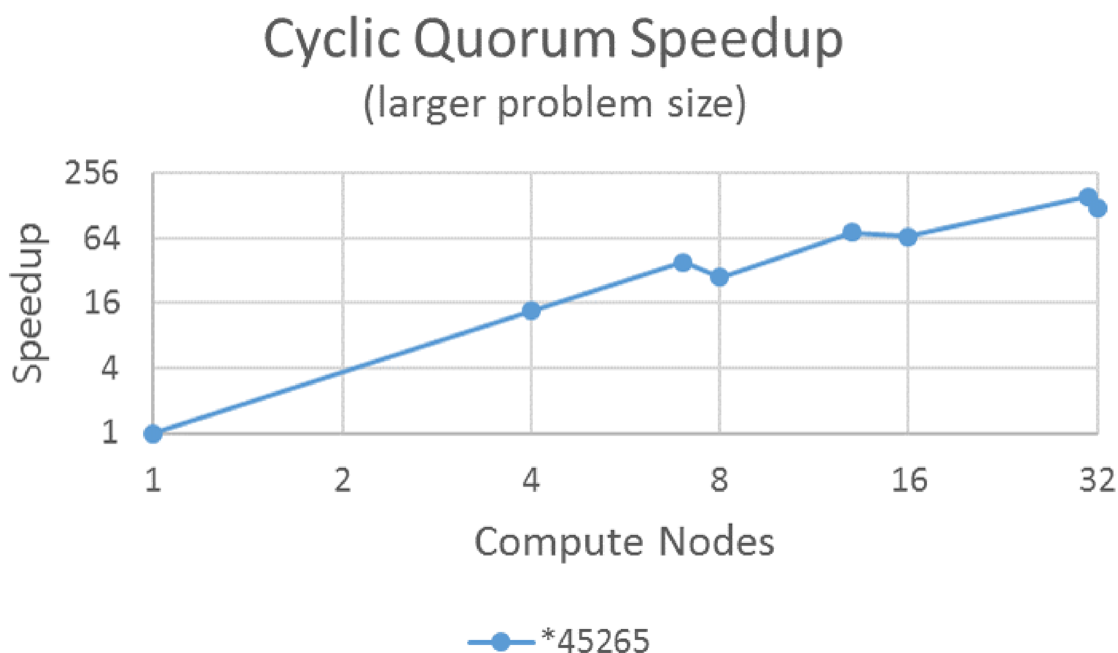


Figure 6.2: Speedup of our cyclic quorum algorithm using (P) parallel nodes when compared to an optimized single node algorithm. This figure has a speedup curve for a larger dataset that would have exceeded the single node memory resources, requiring the use of an alternate lower memory optimized algorithm. Here super-linear speedup is observed as our cyclic quorum algorithm is able to distribute the problem and process the input within the memory constraints.

Without the distributed cyclic quorums set solution, the time to compute the ever increasing dataset sizes is prohibitive. To further illustrate this point, Figure 6.2 shows the speedup curve for the $N = 45265$ row dataset. The trend of the curve on the log-log scale is close to linear, again demonstrating our ability to scale to arrive at results in a shorter amount of time efficiently. However, unlike Figure 6.1, the speedup in Figure 6.2 is super-linear as seen by the speedup value for some inputs being more than 5x the number of nodes used. This is a common phenomenon where distributed parallel algorithms are able to more efficiently process a larger problem than their single node equivalents. Meaning that when renting the processing resources from the cloud or utilizing resources on a local HPC system, not only will our solution compute the all-pairs results faster, it will also do so more cheaply by using as much as 5x fewer node compute hours to complete the computation.

المنارة للاستشارات
www.manaraa.com

Lastly, again the speedup curve in Figure 6.2 is not smooth just like that of Figure 6.1. This is an interesting result highlighting some of the advantages of Singer difference sets. The question of whether additional computation management could improve the performance of quorum sets not based on Singer difference sets is addressed in the next section, Section 6.4.

6.4 Adding Computation Management Logic

In the prior section, Section 6.3, it was observed that our cyclic quorums algorithm scaled well using less memory per node and achieving linear and at times super-linear speedups. However, the speedup when additional nodes were added was not as smooth of a curve as anticipated in Figures 6.1 and 6.2 and similarly observed in Table 6.3 with executions $P = 8, 16,$ and 32 under performing expectations.

In this section, we discuss how the size of the cyclic quorum impacts these results, as well as providing additional computation management logic to compliment the implicit data management techniques that cyclic quorum sets provides. Lastly, we implement and test our management logic.

6.4.1 Impact of cyclic quorum size

Common choices for number of parallel nodes are powers of 2, i.e., 4, 8, 16, and 32 nodes. These choices were shown to not perform as well as lesser obvious choices of 7, 13, and 31 nodes. And in the prior section, Section 6.3, we attributed this to the advantages of cyclic quorum sets based on Singer difference sets [Colbourn (2010)]. Here, we expand more on this topic by looking more closely at the size of the optimal cyclic quorum sets used for processes $P = 4, \ldots, 111$ from Luk and Wong (1997) (see Appendix A for a reproduced and verified cyclic quorum listing).

Table 6.4 summarizes the characteristics of our test parameter P's impact on the all-pairs computations performed, i.e., without additional management logic, redundant work will be performed. Column 1 is the number of P nodes. $|S_i| = k$ comes from Equations 5.6 and 5.5. There is a minimum size that the cyclic quorum can have, while still being valid and therefore having the all-pairs property (Section 5.3). We verified that the optimal cyclic quorums found

المنارة للاستشارات
www.manaraa.com

from Luk and Wong (1997) were in fact the minimum and column 2 contains these quorum sizes. One of the attractive features is the slow quorum growth rate, $O\left(\sqrt{P}\right)$, compared to number of P processes. This means that the size of the quorum set, i.e., $|S_i| = k$, will grow far slower than the increase in number of P nodes being used.

The distributed all-pairs problem was defined in Section 4.2 on Page 45. Additionally, Equation 4.6 described the required global pairing of all D_i for $i \in 0, \ldots, P-1$. What this amounts to is $\binom{P}{2} = \frac{P(P-1))}{2}$ datasets paired with each other and then the same P datasets paired with themselves for a total ideal pairing performed being:

$$\frac{P\,(P-1)}{2} + P = \frac{P\,(P+1)}{2} \tag{6.1}$$

This value is calculated for each P-nodes in Table 6.4 and put in column 3, Ideal Pairs. The Unmanaged Pairs column is the number of pairs computed in our cyclic quorums algorithm without any additional management logic. Each node has k datasets available in memory, where the specific datasets available are defined by Equation 5.8. They can perform $\binom{k}{2} = \frac{k(k-1)}{2}$ pairings and 1 additional pairing of dataset D_{0+i} with itself. When this is performed across all nodes the total unmanaged pairings is:

$$P\left(\frac{k\,(k-1)}{2} + 1\right) \tag{6.2}$$

This value is calculated for each P-nodes in Table 6.4 and put in column 4, Unmanaged Pairs.

The Redundant Pairs, column 5, of Table 6.4 is the difference between the ideal number of global pairs computed in the distributed system and the number of pairs computed globally when the cyclic quorum set algorithm goes unmanaged. The Redundant Percentage is the ratio of the unmanaged pairs computed that is redundant. On average over $P = 4, \ldots, 111$ the amount of redundant pairs is not insignificant at 19.7%. This really motivates the need for adding computation management logic to our cyclic quorums set solution.

When looking at Table 6.4, there are several entries that have 0 redundant pairs. These instances are where the cyclic quorum is formed from a Singer difference set. The distributed

المنارة للاستشارات
www.manaraa.com

Table 6.4: Redundant Work Performed When Quorum Pairs Unmanaged

#Nodes	$\mathbf{\lvert S_i \rvert = k}$	**Ideal Pairs**	**Unmanaged Pairs**	**Redundant Pairs**	**Redundant Percentage**
4	3	10	16	6	37.5%
7	3	28	28	0	0.0%
8	4	36	56	20	35.7%
13	4	91	91	0	0.0%
16	5	136	176	40	22.7%
31	6	496	496	0	0.0%
32	7	528	704	176	25.0%
		Average Redundant Percentage for #Nodes $= 4, \ldots, 111$			**19.7%**

المنارة للاستشارات

www.manaraa.com

global pairings across all of the processing nodes in these instances perfectly cover the necessary pairs without any overlap. An example of a 7-node cyclic quorum covering all pairs can be seen in Figure 5.3 on Page 57. Singer difference sets for $P < 100$ are $P = 7, 13, 21, 31, 57, 73,$ and 91.

6.4.2 Computation management logic

The previous section identified the issue as distributed nodes performing redundant work and provided a way to account for how much wasted work actually exists in the solution. Now the challenge is to locate the source and fix it.

6.4.2.1 Identifying redundancy

Observe that in order to perform redundant all-pairs work two or more nodes must share (intersect) two or more of the same datasets. Section 5.3 defined the all-pairs property for quorums and has the corresponding proof that cyclic quorums have the all-pairs property. Notice the key to forming pairs is rooted in the difference set for the cyclic quorum. It is this difference set that defines which datasets are in each quorum, hence we return to Definition 1 (Relaxed (P, k)-difference set).

In Figure 6.3, every $d \neq 0$ modulus $P = 4$ is present, which is required for $A = \{1, 2, 3\}$ to be a difference set (and cyclic quorum). The key observation is that there are instances that differences modulus P occur more than once. This is the source of all of the redundant work for $P = 4$. To see why reoccurring differences lead to duplicate pairs, consider for some difference set A:

$$a_i - a_j \; mod \, P = a_x - a_y \; mod \, P \tag{6.3}$$

Using the definition of cyclic quorums these differences would correspond to pairs for some quorum S_l and S_m:

$$(a_i + l, a_j + l) \mod P \tag{6.4}$$

$$(a_x + m, a_y + m) \mod P \tag{6.5}$$

www.manaraa.com

$a_i - a_j = d$	1	2	3
1	$1-1=0$	$2-1=1$	$3-1=2$
2	$1-2=-1$	$2-2=0$	$3-2=1$
3	$1-3=-2$	$2-3=-1$	$3-3=0$

d mod 4	1	2	3
1	0	1	2
2	3	0	1
3	2	3	0

Figure 6.3: A difference set example with 4 processes. This figure with $A = \{1, 2, 3\}$ and $P = 4$ is a valid relaxed difference set because all integer differences modulus 4 are formed, i.e., $0, \ldots, 3$ all occur one or more times.

Knowing that both pairs correspond to equal differences modulus P, then we can add k modulus P to the first difference to obtain the second difference:

$$(a_i + k) - (a_j + k) \; mod \; P = a_x - a_y \; mod \; P \tag{6.6}$$

This gives way to creating a new pair equivalence

$$(a_i + k + l, a_j + k + l) \mod P = (a_x + m, a_y + m) \mod P \tag{6.7}$$

And now we can define a new quorum index $\hat{l} = l + k \mod P$.

$$\left(a_i + \hat{l}, a_j + \hat{l}\right) \mod P = (a_x + m, a_y + m) \mod P \tag{6.8}$$

The result is two quorums $\hat{l}$ and m with a redundant pair all because there was a reoccurring difference in their shared difference set definition A.

6.4.2.2 Removing the redundancy

With the source of the redundancy identified, ideally we would just remove it. However, we cannot change the difference set to eliminate the redundancy without violating the definition. Instead, we take the simple approach above to identify any redundancy. When one is found, we must decide how to handle it differently than the other unique differences.

المنارة للاستشارات
www.manaraa.com

$S_0 = \{D_0, D_1, D_2\}$	(D_0, D_1)	(D_0, D_2)	(D_1, D_2)
$S_1 = \{D_1, D_2, D_3\}$	(D_1, D_2)	(D_1, D_3)	(D_2, D_3)
$S_2 = \{D_2, D_3, D_0\}$	(D_2, D_3)	(D_0, D_2)	(D_0, D_3)
$S_3 = \{D_3, D_0, D_1\}$	(D_0, D_3)	(D_1, D_3)	(D_0, D_1)

Figure 6.4: A cyclic quorum set example with 4 processes to illustrate the cause and solution to redundant work. This figure has the corresponding cyclic quorum set on the left and the possible all-pairs formed for each on the right. The pairs and quorum are colored with useful work performed, while the uncolored pairs would be redundant work and should not be performed.

Algorithm 16 Enumerating pairs for each quorum

```
1: Given: Array A
2: for x ← 0 to length(A) − 2 do
3:     for y ← x + 1 to length(A) − 1 do
4:         Pair (D_{a_x+i}, D_{a_y+i})
5:     end for
6: end for
```

To get an idea what this different handling would look like consider Figure 6.4. All of the cyclic quorums are listed on the left for the difference set shown in Figure 6.3. On the right the $\binom{k}{2}$ pairs for each quorum are enumerated using Algorithm 16.

This enumeration pattern uses Equation 5.8 and gives a consistent order to forming the pairs for all quorums S_i, $i \in 0, \ldots, P-1$. The first pair for each quorum is based on differences $a_0 - a_1$ and $a_1 - a_0$. These differences have not been computed yet, thus their quorum dataset pairs are not redundant work, so we color them with the corresponding quorums' color in Figure 6.4. This is important to observe because the differences

$$a_0 - a_1 \, mod \, 4 = 1 - 2 \, mod \, 4 = 3, \text{ and} \tag{6.9}$$

$$a_1 - a_0 \, mod \, 4 = 2 - 1 \, mod \, 4 = 1 \tag{6.10}$$

occur multiple times in Figure 6.3. The repetition of differences was identified as being the source of redundant work. However, the work is not redundant yet, so the work must be performed.

للاستشارات المنارة
www.manaraa.com

Algorithm 17 Management Logic(A,P,i)

```
1:  DC[d] ← False, d ∈ 0,...,P − 1
2:  for x ← 0 to length(A) − 2 do
3:      for y ← x + 1 to length(A) − 1 do
4:          d_xy ← (A[x] − A[y]) modulus P
5:          d_yx ← (A[y] − A[x]) modulus P
6:          if DC[d_xy] ≠ True and
              DC[d_yx] ≠ True then
7:              DC[d_xy] ← True
8:              DC[d_yx] ← True
9:              if d_xy ≠ d_yx then
10:                 Compute Pair (D_{A[x]+i}, D_{A[y]+i})
11:             else if i < P/2 then
12:                 Compute Pair (D_{A[x]+i}, D_{A[y]+i})
13:             end if
14:         end if
15:     end for
16: end for
```

The second pair for each quorum is based on differences $a_0 - a_2$ and $a_2 - a_0$. These pairs have not been computed yet either, but on closer inspection the differences

$$a_0 - a_2 \, mod \, 4 = 1 - 3 \, mod \, 4 = 2, \text{ and} \tag{6.11}$$

$$a_2 - a_0 \, mod \, 4 = 3 - 1 \, mod \, 4 = 2 \tag{6.12}$$

are actually the same difference. This redundancy with itself occurs when $P \, mod \, 2 = 0$, i.e., the number of nodes is even. If all nodes were to compute this pair, redundant work will occur. Rather, we arbitrarily decide that the first half of the nodes corresponding with quorums S_i, $0 \leq i < \frac{P}{2}$ should do the work, so we color them with the corresponding quorums' color. We leave the would be redundant pairs uncolored to indicate that they are not computed.

The last quorum dataset pair is based on differences:

$$a_1 - a_2 \, mod \, 4 = 2 - 3 \, mod \, 4 = 3, \text{ and} \tag{6.13}$$

$$a_2 - a_1 \, mod \, 4 = 3 - 2 \, mod \, 4 = 1 \tag{6.14}$$

However, these differences have already been computed and hence are redundant and are not colored in Figure 6.4.

www.manaraa.com

We took a simple approach described above and in Figure 6.4 to prevent redundancy at the time of computation, since it cannot be eliminated from the difference set. This computation management logic is more formally stated by Algorithm 17. Firstly, computing differences multiple times is the source of redundancy. Line 1 is an array of booleans to track which differences have been computed and which have not. Lines 2 and 3 enumerate all $\binom{k}{2}$ difference pairs. The prevention of redundancy logic is primarily in the if statement on Line 6, which verifies whether a difference has been computed or not. For those differences that have not been computed, most will go on to compute the quorum dataset pairings on Line 10. However, there is a special case for when differences form redundancies with themselves, i.e., $d_{xy} = d_{yx}$. This only occurs when the number of P nodes is even; and in this case, only half of the nodes need to compute their quorum dataset pairs (Line 11).

6.4.3 Impacts of managing quorum set all-pairs computations

Table 6.4 on Page 72 illustrated that there was room for substantial improvement in all-pairs computations using cyclic quorum sets not based on Singer difference sets. The table suggests on average 19.7% of the work would be redundant if something were not done to avoid it.

We could implement a central manager that assigns work and arbitrates which node performs which computation pairs, but that adds another layer of complexity and overhead, potentially negating any benefits. A decentralized communication approach without a leader would appear to be a good fit, considering those applications are what quorum sets are regularly used for. However, that approach would also add an unnecessary layer of complexity and some overheads.

Rather, Algorithm 17 can be implemented in a distributed system without any communication requirement. All decisions are made locally and with very little overhead other than keeping track of the Differences Computed (DC) array.

To test out this computation management logic, we used the same application and same test setup as described in Sections 6.1 and 6.2 on Page 62. Once again we restricted our application's memory usage to only 60GB per node to model how our application would execute in a similar

المنارة للاستشارات
www.manaraa.com

cloud HPC enivironment to that of Amazon's "c4.8xlarge" instance cluster [Amazon (2016a)]. The three real and six simulated input datasets described in Table 6.1 were used again as well.

We modified our prior cyclic quorum to add the computation management logic. For each pair of dataset input and number of HPC nodes, we executed 30 runs of the application in order to establish confidence in the runtime measurements. There is a column for every dataset input and a row for the number of nodes used. The memory used per node is in Table 6.5 and the observed average execution runtimes with 95% confidence intervals is in Table 6.6.

6.4.3.1 Memory usage performance

The single node instance in Table 6.5 is the PCIT algorithm from Koesterke et al. (2013) for comparison. Our cyclic quorums implementation with the new management logic has a substantially similar memory footprint to the implementation without. This was expected considering that the redundancies were not able to be removed from the difference sets, i.e., the data continues to be distributed in a predictable way using the cyclic quorum design described in Section 5.2 on Page 50. Also, we only added a small amount of overhead to keep track of which differences had, and which ones had not yet, been computed.

In Table 6.5, we continue to see that in general as additional nodes are used, fewer memory resources per node are required to execute the same dataset. This again results in up to an 80% reduction in memory requirements per node. This was a critical result for our larger real and simulated input datasets where memory used per node ran into the upper 60GB limit for the single node tests and some of our parallel tests as well. These instances have been marked with bold.

The advantages of using Singer difference sets [Colbourn (2010)] are still present in the memory usage data for the same reason. 7, 13, and 31 nodes reduce memory usage per node by 57%, 69%, and 80%, respectively, while if the number of nodes were increased by a small amount to 8, 16, or 32 the reduction is slightly less at 50%, 68%, and 77%. Generally increasing the number of nodes will reduce the memory used. However, these results illustrate that there are some local minimums that initially may not be expected by a user of our cyclic quorums algorithm.

المنارة للاستشارات
www.manaraa.com

Table 6.5: Managed - Memory Used Per Node (GB)

#Nodes	*27364	33331	39298	*45265	51232	*57194	63166	69133	75000
1	25.113	37.257	51.789	**68.708**	**88.736**	**110.495**	**134.680**	**161.233**	**189.669**
4	18.831	27.938	38.837	51.526	**66.725**	**83.065**	**101.222**	**121.159**	**142.506**
7	10.762	15.967	22.193	29.445	38.439	47.812	58.226	**69.651**	**81.886**
8	12.555	18.627	25.893	34.352	44.724	55.647	**67.780**	**81.099**	**95.357**
13	7.727	11.463	15.934	21.140	27.801	34.554	42.054	50.282	59.091
16	7.848	11.643	16.185	21.471	28.224	35.083	42.698	51.054	59.996
31	4.862	7.213	10.025	13.302	17.760	22.035	26.787	31.994	37.566
32	5.495	8.151	11.331	15.033	19.974	24.802	30.156	36.032	42.316

المنارة للاستشارات
www.manaraa.com

Table 6.6: Managed - Average Execution Runtimes (Seconds)

#Nodes	*27364	33331	39298	*45265	51232	*57194	63166	69133	75000
1	119.1 ± 2.9	162.5 ± 0.1	245.5 ± 0.1	**2312.5 ± 0.2**	-	-	-	-	-
4	38.4 ± 0.1	54.0 ± 0.1	82.4 ± 0.2	124.0 ± 0.0	-	-	-	-	-
7	18.0 ± 0.0	25.4 ± 0.1	38.3 ± 0.1	60.4 ± 0.1	317.6 ± 0.9	1213.4 ± 1.6	491.9 ± 2.3	-	-
8	18.0 ± 0.0	25.7 ± 0.1	38.6 ± 0.1	57.9 ± 0.1	341.5 ± 1.0	1175.0 ± 2.8	-	-	-
13	10.1 ± 0.0	14.4 ± 0.0	21.5 ± 0.1	32.3 ± 0.0	244.1 ± 1.8	788.3 ± 1.9	371.9 ± 2.2	446.5 ± 1.3	519.4 ± 2.5
16	9.0 ± 0.0	12.9 ± 0.0	19.0 ± 0.1	29.0 ± 0.0	244.5 ± 2.1	711.2 ± 2.3	364.3 ± 1.8	439.3 ± 1.2	509.6 ± 2.0
31	4.8 ± 0.0	6.6 ± 0.0	10.0 ± 0.0	14.9 ± 0.1	178.8 ± 1.3	415.4 ± 1.8	249.9 ± 2.2	307.1 ± 1.1	361.4 ± 1.2
32	4.8 ± 0.0	6.9 ± 0.0	10.1 ± 0.0	15.4 ± 0.0	193.8 ± 1.1	430.8 ± 1.2	277.8 ± 1.2	328.4 ± 1.5	381.4 ± 1.7

المنارة للاستشارات
www.manaraa.com

6.4.3.2 Runtime execution performance

The average runtimes observed over 30 executions are in Table 6.6 with their 95% confidence intervals. As described in our test setup (Section 6.2), we used a generous 60GB memory limit that even current datasets pushed up against (Table 6.5). Where this limit was exceeded we did not collect execution runtime data, except in one single node instance which is marked with bold. Here, the single node algorithm had to revert to a lower memory alternative and consequently the runtime increased dramatically as a trade off.

Figure 6.5 is a comparison of speedup between our unmanaged cyclic quorum set implementation of the PCIT algorithm from Section 6.3 and a modified version that has additional computation management logic. The unmanaged implementation is denoted with a "U" and managed implementation with an "M". Speedup curves for two datasets are shown in the figure. A smaller $N = 39298$ rows dataset which fits within the available memory constraints, and a larger $N = 45265$ rows dataset which exceeded the memory constraints of a single node.

The trend of the $N = 39298$ curves on the log-log scale are close to linear demonstrating both implementations' abilities to scale with increasing node resources. The managed cyclic quorum implementation's speedup was up to 27% faster for this input dataset than without the management logic. However for Singer difference sets [Colbourn (2010)], which require fewer datasets (D_i) to guarantee the all-pairs property (Section 5.3), our cyclic quorum implementation without management was already efficient without any redundant work. When the management logic was added, the speedup decreased by up to 1% for this input dataset due to the overheads introduced.

It is worth noting that the managed "M-39298" and "M-*45265" curves are considerably more smooth with increasing nodes than their unmanaged curve partners. This is the impact of being able to apply the computation management logic to prevent redundant work. Every additional node is now contributing to complete useful work without any redundant work being performed. The unmanaged implementation is not as smooth because some choices of P nodes result in redundant work being performed, hence taking longer to complete decreasing the observed speedup.

المنارة للاستشارات
www.manaraa.com

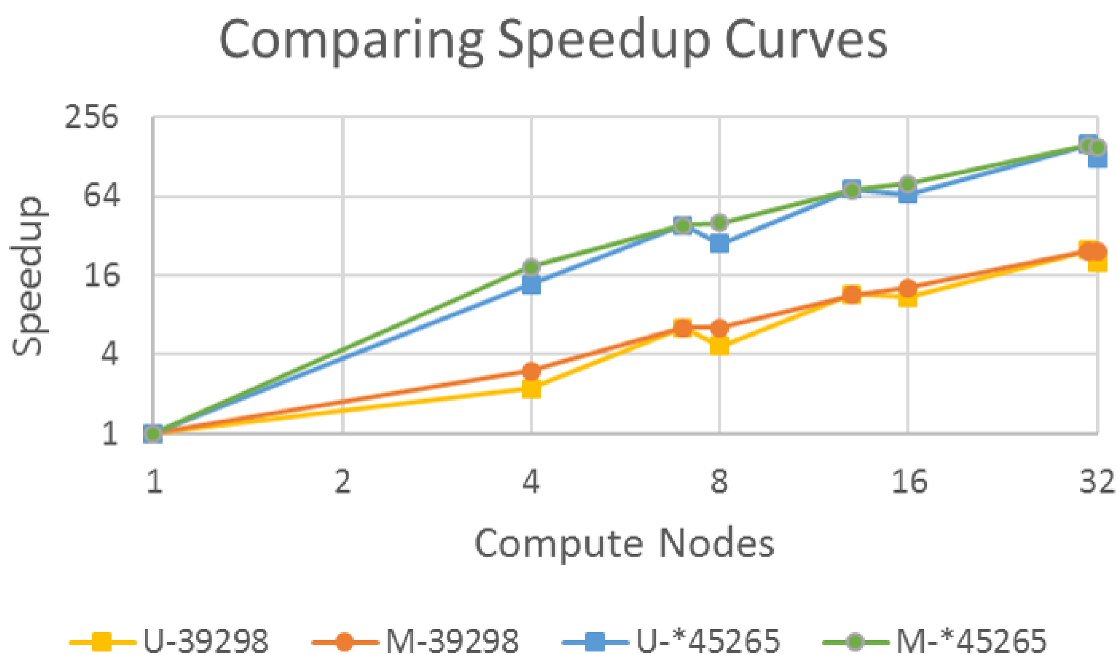


Figure 6.5: Comparing the speedup of our unmanaged cyclic quorum algorithm with that of our algorithm with additional computation management logic (U vs. M). Both were executed using (P) parallel nodes with speedups in reference to an optimized single node algorithm. Two input datasets are compared, one with $N = 39298$ rows which fit within available memory, and another with $N = 45265$ rows which exceeds a single node's memory resources, requiring the use of an alternate lower memory optimized algorithm. For non-Singer difference sets, the managed all-pairs computations were up to 30% faster than without the management logic. When P corresponded to a Singer difference set, the overhead of the management was typically less than 1%, which put the unmanaged speedup slightly ahead.

Lastly, the speedup curves corresponding to the $N = 45265$ dataset in Figure 6.5, and the runtime data in Table 6.6, work toward illustrating our distributed cyclic quorums set solution enables the scaling to larger dataset sizes that previously may have been time and resource prohibitive. The trend of the curves on the log-log scale are close to linear, again demonstrating both implementations' abilities to scale to arrive at results in a shorter amount of time efficiently. However unlike the $N = 39298$ curves, the speedup is super-linear as seen by the speedup value for some inputs being more than 5x the number of nodes used. When renting the processing resources from the cloud or utilizing resources on a local HPC system, this means not only will our solution compute the all-pairs for larger input datasets results

المنارة للاستشارات
www.manaraa.com

faster, it will also do so more cheaply by using as much as 5x fewer node compute hours to complete the computation.

www.manaraa.com

CHAPTER 7 ALL-PAIRS APPLICATIONS IN FAULT TOLERANT OPTICAL COMMUNICATION OPTIMIZATIONS

Chapter 3 has a background on optical network operations and architectures. It included recent work where the same cyclic quorums used in our computation application (Chapter 6) were shown by Somani and Lastine (2014) to efficiently support arbitrary point-to-point and multi-point communication for cycle-based routing in optical networks.

Optical networks are highly depended upon too. The fault tolerance aspect of these route designs are important (Sect. 3.3). In this chapter, we analyze the fault tolerance of optical networks when cyclic quorum sets are used as the basis for cycle routing. Here we utilize the properties of cyclic quorum sets to deliver resource efficient routing solutions, while maintaining fault tolerance in the network.

7.1 Fault Model

The fault model assumed for our work is the link (edge) failure. While a simple model, it does cover most real single fault scenarios.

The most direct fault to consider is the optical link fault. This occurs when a link is broken, like planned maintenance or the accidental severing during land excavation. Modeling link faults as a single edge failure is straightforward.

Each modeled node needs a pair of transmitters and receivers for each occurrence in a cycle. These pairs of devices can fail too. Short of a natural disaster, pairs will likely fail independently of one another. When a transmitter/receiver pair fails within a modeled node, the affect on the global network is similar to that link failing. Modeling as a single edge failure, while not an exact fault mapping, is an appropriate abstraction.

Cycle-based routing can protect against faults of this nature [Lastine et al. (2012); Somani et al. (2011)]. Commonly a paired cycle implementation is used to address this. When a link fails,

المنارة للاستشارات
www.manaraa.com

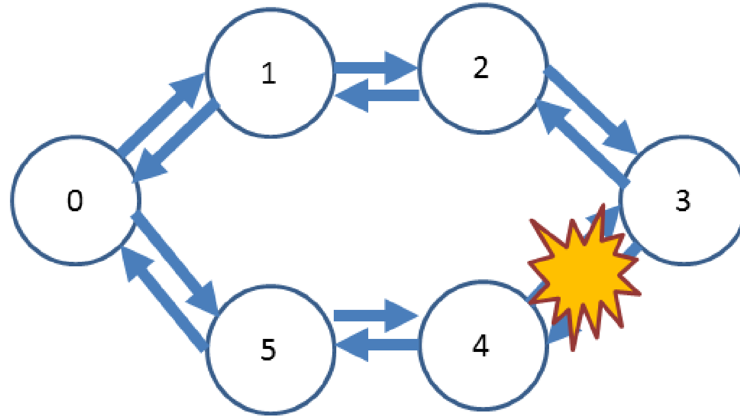


Figure 7.1: Example route fault tolerance using light-trails

both cycles break. In the Figure 7.1 example, hub node 3 would no longer have a downstream edge to node 4; however, upstream communication can be used to still reach node 4.

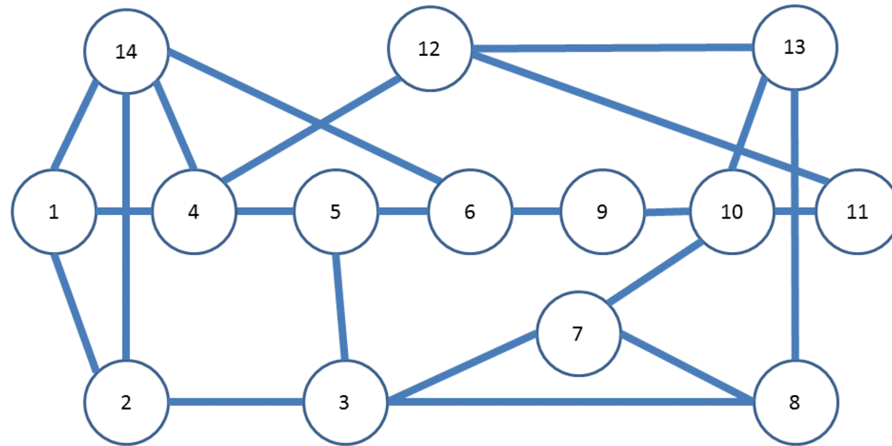
7.2 Paired Cycle Fault Simulation

Maintaining the ability to serve all dynamic point-to-point traffic requests despite fault is important. We examined the fault tolerance of the NSFNET, ARPANET, American backbone, Chinese backbone networks (Fig. 7.2). For each of these four networks, we created 1000 random numbering schemes for the nodes and found cycles to support the optimal cyclical quorums given in Luk and Wong (1997) (see Appendix A for a reproduced and verified cyclic quorum listing).

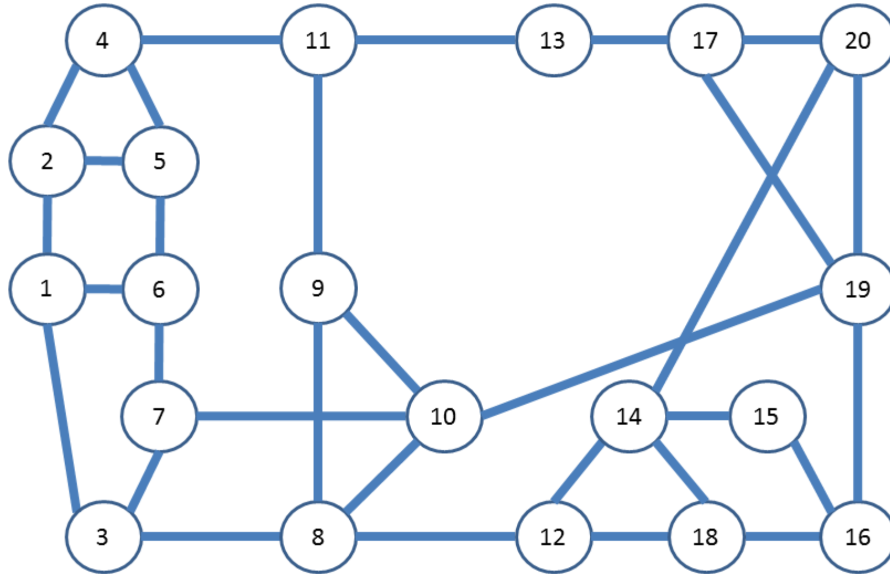
To model the fault, we simulate the failure of each edge, $(e_i, e_j) \in E$, in the network model, $G = (V, E)$. We then examine the network's ability to serve all potential point-to-point requests by counting pairs of nodes that would be able to communicate and conversely those pairs that are unable to communicate. In the non-fault case, all nodes can unidirectionally communicate with all other nodes for a total of $|V|\,(|V| - 1)$ pairs.

The results in Table 7.1 show acceptable fault tolerance performance. Out of the 1000 tests per network and simulation of the failure of each edge in the networks, the mean number of missing communication pairs per edge failure case was less than 3 (95% CI) for all networks. Therefore, any link fault in the network will typically be recoverable and few point-to-point connections will experience connection loss when using quorums set-based cycle routing.

المنارة للاستشارات
www.manaraa.com



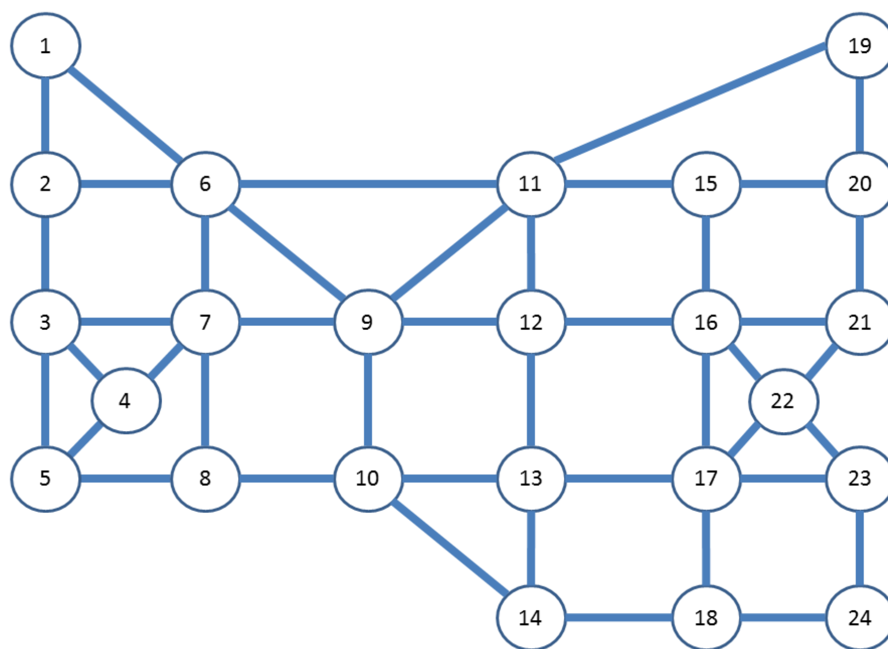
(a)



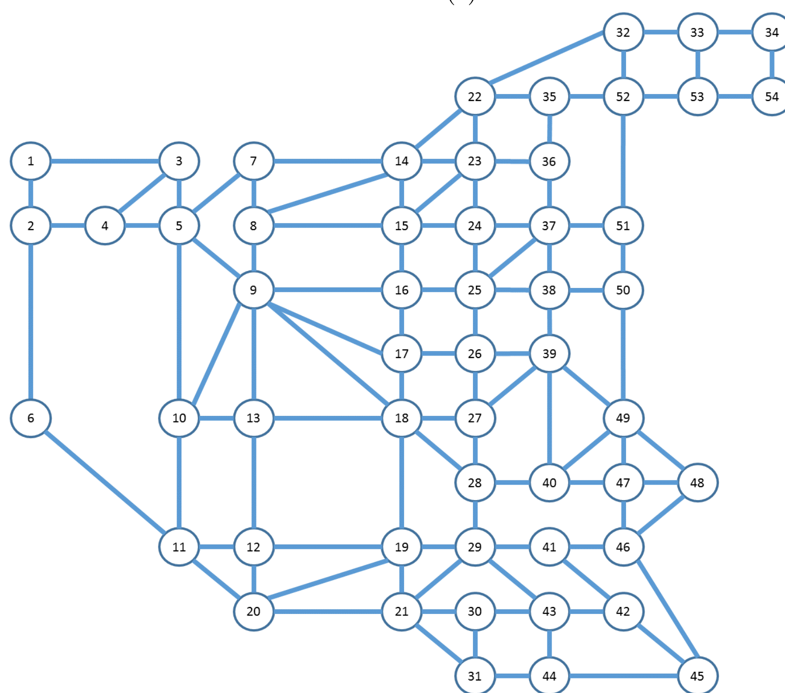
(b)

Figure 7.2: Networks used for simulations: Figure (a) NSFNET, 14-Node/22-Link, Figure (b) ARPANET, 20-Node/31-Link, Figure (c) American Backbone [Tang et al. (2011)], 24-Node/43-Link, and Figure (d) Chinese Backbone [Tang et al. (2011)], 54-Node/103-Link.

www.manaraa.com



(c)



(d)

Figure 7.2: (Continued)

Networks used for simulations: Figure (a) NSFNET, 14-Node/22-Link, Figure (b) ARPANET, 20-Node/31-Link, Figure (c) American Backbone [Tang et al. (2011)], 24-Node/43-Link, and Figure (d) Chinese Backbone [Tang et al. (2011)], 54-Node/103-Link.

www.manaraa.com

Table 7.1: Paired quorum cycle fault simulation results

Network	Nodes	Total Pairs	Missing Pairs High	Missing Pairs Mean (95% CI)	Missing Pairs Low	Fault Coverage (%)
NSFNET	14	182	12	0.93644 ± 0.02070	0	99.485
ARPANET	20	380	16	0.76051 ± 0.01715	0	99.800
American	24	552	26	2.05273 ± 0.02812	0	99.628
Chinese	54	2862	56	2.77809 ± 0.02400	0	99.903

To calculate fault coverage in this scenario, we calculated the mean connected pairs divided by the total pairs.

$$1 - \frac{Mean\ Missing\ Pairs}{Total\ Pairs} \tag{7.1}$$

The results of our simulation showed that greater than 99% average fault coverage can be expected.

7.3 Improving Fault Tolerance

Many of the simulated networks typically performed well, but ideally we would like to see the fewer missing pairs per fault case. A reduction in the highest missing pairs observed is also important.

7.3.1 Additional cycle fault protection

Recall from Figure 3.3 on Page 40 that hub nodes in light-trail cycles have their optical shutters in the *off* state. When a fault occurs like in Figure 7.1, it is possible that the hub node is 0 and the optical shutters could prevent a necessary communication path between 3 and 4.

We experimented with adding an additional pair of cycles to form quad cycles. The pair has its hub node directly across from the original pair's hub node, i.e., at position $\left\lfloor \frac{CycleLength}{2} \right\rfloor$. In the Figure 7.3 example, hub nodes 0 (inner blue light-trail) and 3 (outer red light-trail) are across from one another. Node 3 still does not have a downstream edge to node 4 on either the inner or outer cycle, but there does exist an upstream path on the outer cycle to node 4.

www.manaraa.com

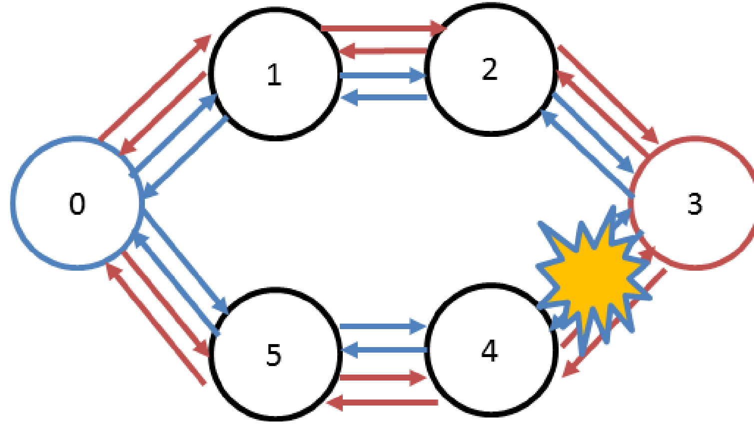


Figure 7.3: Quad light-trails to provide additional cycle fault protection

Table 7.2: Quad quorum cycle fault simulation results

Network	Nodes	Total Pairs	Missing Pairs High	Missing Pairs Mean (95% CI)	Missing Pairs Low	Fault Coverage (%)
NSFNET	14	182	6	0.09130 ± 0.00574	0	99.950
ARPANET	20	380	6	0.08710 ± 0.00496	0	99.977
American	24	552	12	0.28661 ± 0.00833	0	99.948
Chinese	54	2862	20	0.52939 ± 0.00768	0	99.982

The results in Table 7.2 show that most of the networks had the mean number of missing pairs improved by an order of magnitude (95% CI). Any edge fault in the quad cycle network will typically be recoverable and fewer point-to-point connections will experience connection loss than with only the paired cycles. This translates into higher fault coverage values as well. Similarly, the edge faults that generated the highest missing pairs observed in the simulated networks decreased by 50% or more when the additional pair of cycles was added.

Adding the additional pairs of cycles and still having missing communication pairs may have been a bit surprising. While the additional cycles significantly helped in fault reductions, the underlying missing pairs come from the intermediate nodes common to both cycles. Depending on the failing edge, these nodes may not have a downstream or upstream path on any of the cycles to nodes on the opposite half of the cycle.

المنارة للاستشارات
www.manaraa.com

7.3.2 Modifying the cycle routing algorithm

Rather than addressing the missing point-to-point communication pairs with additional cycles, we could try to address the underlying cause of the missing pairs by changing the cycles themselves. Here, we briefly outline additional algorithm steps that could be added to the optical network cycle routing algorithm, ECBRA [Somani et al. (2011)].

In Section 3.4.1, every point-to-point request was supported by at least one quorum cycle. Being opportunistic, even if one cycle is experiencing a fault, there may be another quorum cycle that also is supporting that point-to-point request. Hence, quantifying missing pairs and attempting to compensate should only occur once all cycles for a graph's quorums set are found.

To enumerate the missing pairs, simulate the failure of each edge and examine the network's ability to serve all potential point-to-point requests by counting pairs of nodes that would be able to communicate and conversely those pairs that are unable to communicate. Given results in Section 7.2 the mean number of missing pairs is expected to be less than 3 per fault edge (95% CI). Form a *Missing-Pair* tuple $t =< \{e_s, e_d\}, (e_i, e_j) >$, where e_s is the source and e_d is the destination of the missing communications pair and (e_i, e_j) is the edge whose failure was being simulated.

Every pair missing can be isolated back to a cycle responsible for the pair (Sect. 3.4.1). For each tuple t, remove the faulty edge (e_i, e_j) in the responsible cycle for pair $\{e_s, e_d\}$. To complete the cycle, find the shortest path between the two disconnected nodes, such that cycle edges are not reused and avoiding the use of (e_i, e_j). Once all Missing-Pair tuples have been processed, repeat the enumeration and removal of missing pairs once again. Repetition is required to confirm that in the process of addressing one Missing-Pair tuple that another new missing pair was not added.

There is a possibility that a cycle may have to use edge (e_i, e_j). An example would be a node with only two edges and the node being a member of the quorum for that cycle. This is a known limitation and is a challenge in general for establishing appropriate fault tolerance for communications to/from this node, as well as, through this node serving others. This limitation is discussed further in Somani et al. (2011).

المنارة للاستشارات
www.manaraa.com

7.3.3 Redundant cyclic quorums sets

In Section 7.2, ECBRA was used to route each of the quorums-based cycles. It was shown that the quorums set approach provided fault tolerance and Lastine (2014) showed that this technique required far fewer links to accomplish the routing of all-to-all traffic, when compared to using point-to-point connections.

As an unintended benefit, some quorums sets resulted in node pairs occurring in more than one quorums-based cycle. It was these occurrences of node pairs multiple times that improved the fault tolerance performance. This also motivated us to examine whether redundant node pairs could be generated intentionally as described in Section 5.4.

Predictably redundant pairs can improve the dependability of the optical network, by guaranteeing even if a cycle failed, all node pairs for point-to-point and multi-point communications could still be present in a second cycle within the network. Alternatively, this could be used to eliminate the paired cycle implementations and use only a single cycle, significantly reducing the required amount of network resources while still maintaining a similar level of fault-tolerance. Kleinheksel and Somani (2015c,a) examined both of these approaches and the results are presented in Sections 7.4 and 7.5.

7.4 Redundant Cyclic Quorums Set - Paired Cycle Network Analysis

We begin by examining our proposed expansion of redundant quorums by comparing apples-to-apples using the paired cycle routing in prior art. We used four common networks (Fig. 7.2 on Page 86) and an implementation of the ECBRA heuristic [Somani et al. (2011)] to perform the cycle routing.

ECBRA is sensitive to node and edge numbering that a total of 100 variations on the inputs were considered, each being a one-to-one mapping with the respective network. For simulation of prior art, we used the $N = 4, \ldots, 111$ optimal cyclic quorums from Luk and Wong (1997) (see Appendix A for a reproduced and verified cyclic quorum listing). Redundant cyclic quorums for $R = 2$ and $R = 3$ were found using the techniques described in Section 5.4 (see Appendix B for the redundant cyclic quorum listing).

المنارة للاستشارات
www.manaraa.com

Table 7.3: Mean links (95% CI) used by paired cycles based on redundant quorums sets

Network	R = 1	R = 2	R = 3
NSFNET	249.32 ± 1.37	270.82 ± 1.20	290.10 ± 1.49
ARPANET	511.90 ± 1.87	539.26 ± 1.53	589.50 ± 1.92
American	641.38 ± 2.10	720.06 ± 1.96	753.72 ± 1.53
Chinese	2673.30 ± 7.11	3054.64 ± 6.99	3271.40 ± 5.98

7.4.1 Fault-free operational analysis

It is expected that a majority of the time the optical network will be operating without faults. It is important that the resource utilization during this period be analyzed.

The metric we use to measure resource utilization is the number of links used in a solution. Comparing network-to-network is not particularly insightful, but comparing multiple solutions for a particular network is. The more links that a set of quorum cycles use, the fewer (wavelength) resources that can be assigned to each link. Additionally each logical link represents a required physical transmitter and receiver, hence capital costs.

Table 7.3 shows that applying redundancy within the quorums and using paired cycles will lead to an increase in mean network links used (95% confidence intervals). $R = 1$, column two, is the standard, no redundant pairs, implementation seen in Somani and Lastine (2014); Kleinheksel and Somani (2015b). $R = 2$ and $R = 3$ have twice and three times redundant pairs present in their quorum solutions, respectively. Despite having that added redundancy, the resource usage, columns 3 and 4, only increased 5.34-14.26% and 15.16-22.37%, respectively, across the networks.

This resource usage result shows that applying our redundant quorums set technique to paired cycle solutions available today will not pose too significant of a resource burden. Next, we consider the fault case for paired cycles using our redundant quorum cycle solution and show the increase in resources is being utilized to improve fault recoveries without any optical cycle reconfigurations.

www.manaraa.com

7.4.2 Fault tolerance operational analysis

Optical networks are highly depended upon. The fault-tolerance aspect of this route design is critical. Maintaining the ability to serve all dynamic point-to-point traffic requests despite faults is important.

We assume fiber link failure(s) as described in Section 7.1. Modeling as a single edge failure, while not an exact fault mapping, is an appropriate abstraction.

7.4.2.1 Single fault case

To model the fault, we simulate the failure of each used edge, $(e_i, e_j) \in E$, in the network model, $G = (V, E)$. It is possible for some networks and their corresponding cycle routes to not utilize one or more network links. The edges not used in a particular solution are not considered in our simulation because if they were included, they would bias the results with zero missing pairs data points.

We then examine the network's ability to serve all potential point-to-point requests by counting pairs of nodes that would be able to communicate and conversely those pairs that are unable to communicate. The results are then reported as fault coverage, i.e., total pairs able to communicate as a percentage of total point-to-point pairs. 100% would be perfect coverage, whereas 0% would be no fault coverage at all.

Our simulation results showed our redundant quorum-based cycle technique had 99.85 - 99.98% and 99.95 - 99.99% percent mean fault coverage, $R = 2$ and $R = 3$ respectively, in the four networks tested. In Table 7.4, we compare the state-of-art paired cycle approach with our redundant technique also with paired cycles. With single edge failures, the paired cycles had a mean missing communication pair rate of less than 3 pairs or less than 0.53% across all networks (95% CI). Hence in column two, it can be seen that the fault coverage is greater than 99.47%. Our redundant quorum cycles technique, columns 3 and 4, had a mean missing pair rate (95% CI) of less than 0.46 and 0.26 respectively across all networks, which is reflected in fault coverages greater than 99.85%.

Depending on the network, the difference between a 99.47% and 99.99% fault coverage could be significant. Being able to achieve that with only the moderate overheads examined in the

المنارة للاستشارات
www.manaraa.com

Table 7.4: Percent mean fault coverage (95% CI) of paired cycles using our redundant quorum solution experiencing a single link fault.

Network	**R = 1 (%)**	**R = 2 (%)**	**R = 3 (%)**
NSFNET	99.47 ± 0.04	99.85 ± 0.02	99.99 ± 0.00
ARPANET	99.81 ± 0.01	99.93 ± 0.01	99.98 ± 0.00
American	99.60 ± 0.02	99.92 ± 0.01	99.95 ± 0.00
Chinese	99.90 ± 0.00	99.98 ± 0.00	99.99 ± 0.00

Table 7.5: Percent mean fault coverage (95% CI) of paired cycles using our redundant quorum solution experiencing two simultaneous link faults.

Network	**R = 1 (%)**	**R = 2 (%)**	**R = 3 (%)**
NSFNET	97.61 ± 0.03	98.71 ± 0.03	99.05 ± 0.03
ARPANET	98.71 ± 0.02	99.19 ± 0.01	99.50 ± 0.01
American	98.63 ± 0.01	99.44 ± 0.01	99.62 ± 0.01
Chinese	99.69 ± 0.00	99.91 ± 0.00	99.95 ± 0.00

previous section is just one of the benefits of the redundant quorums set technique. Being able to dial in on the fault coverage desired using single ($R = 1$), double ($R = 2$), or triple ($R = 3$) redundancy also adds to the flexibility.

7.4.2.2 Two fault case

A significantly more complex model considers two faults simultaneously. All possible two edge failure combinations in the network are simulated. Once again, it is possible for some networks and their corresponding cycle routes to not utilize one or more network links, so edges not used in a particular solution are not considered in our simulation to avoid biasing the results with zero missing pairs data points. We then examine the network's ability to serve all potential point-to-point requests by counting pairs of nodes that would be able to communicate and conversely those pairs that are unable to communicate.

Our simulation results showed our redundant quorum-based cycle technique had 98.71 - 99.91% and 99.05 - 99.95% fault coverage, $R = 2$ and $R = 3$ respectively, in the four networks tested (Table 7.5). In Figure 7.4, we compare the state-of-art paired cycle approach with our redundant technique also with paired cycles. With two edge failures, the paired cycles had a

www.manaraa.com

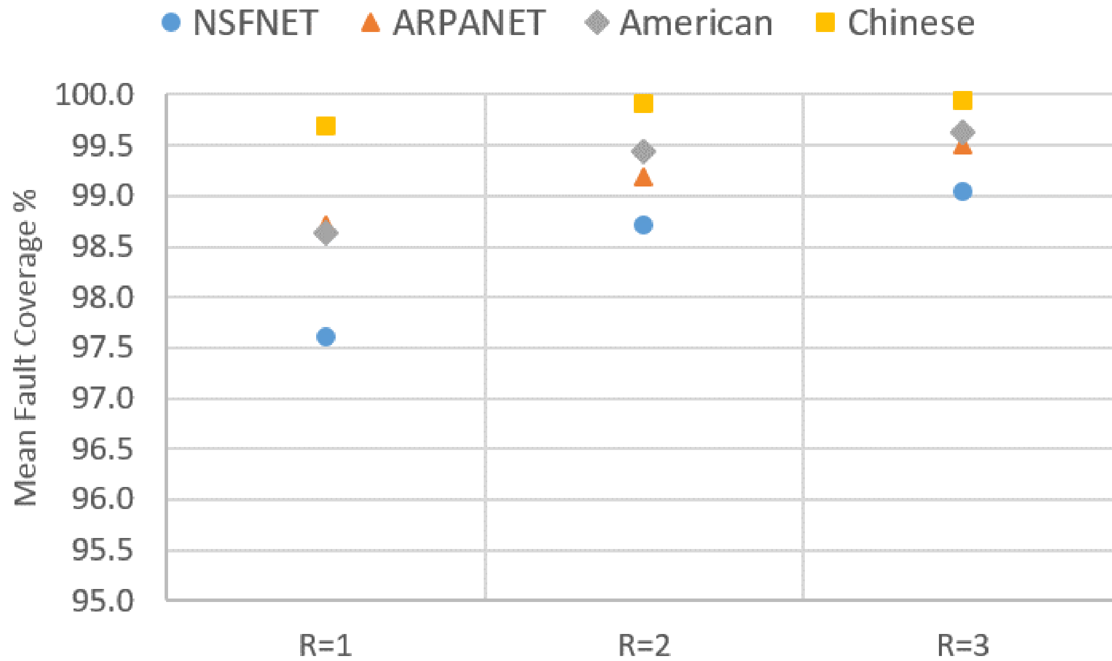


Figure 7.4: Percent mean fault coverage of paired cycles using our redundant quorum solution experiencing two simultaneous link faults.

mean missing communication pair rate of 8.89 pairs (2.39%) or less across all networks (95% Confidence Interval). Hence in the $R = 1$ column, it can be seen that the fault coverage is 97.61% or more. Our redundant quorums set technique, $R = 2$ and $R = 3$, had an average missing pair rate (95% CI) of 3.09 and 2.09 or less respectively across all networks, which is reflected in fault coverages of 98.71% or more.

An interesting aspect of the data is that the benefit of additional quorum redundancy in paired cycle solutions is network dependent. All networks had improved fault coverage with additional quorums set redundancy (Fig. 7.4); however, the NSFNET network had larger increases than others. Additionally, all networks increasing to $R = 3$ had diminishing returns compared to the increases seen when moving from $R = 1$ to $R = 2$.

المنارة للاستشارات
www.manaraa.com

7.5 Redundant Cyclic Quorums Set - Single Cycle Network Analysis

In the previous analysis using paired cycle routing (Section 7.4), our generalization of R redundant quorums sets had moderate increases in resource usage and showed improvements to fault coverage. This section uses a similar experiment setup with the same four common networks (Fig. 7.2 on Page 86) and an implementation of the ECBRA heuristic [Somani et al. (2011)] to perform the cycle routing. However in contrast, we are examining using additional cyclic quorum redundancy with just a single cycle compared to paired cycles used in the previous section and prior art.

The additional redundancy is used to distribute node communication pairs across different cycles, e.g., 0→1 and 0←1 would not occur within the same cycle. Recall that optical light-trail cycles are unidirectional, so prior art would satisfied both 0→1 and 0←1 with a pair of cycles. One cycle would form the forward communication 0→1, while the second cycle would form the backward communication 0←1. By distributing the communication pair responsibility throughout the network, we may only need a single cycle, thus significantly reducing the required amount of network resources. We examine this solution and its ability to deliver the needed fault tolerance.

7.5.1 Fault-free operational analysis

The more links that a set of quorum cycles uses, the fewer (wavelength) resources that can be assigned to each link. Additionally, each logical link represents a required physical transmitter and receiver, hence capital costs.

Table 7.6 shows significant 38.81 - 42.42% resource reduction when using $R = 3$ redundancy in quorums over the more traditional, prior art methods of simply using paired cycles. Using $R = 2$ gives even better resource reduction, as shown in Table 7.6. This reduction represents the potential for lower capital costs in terms of physical transmitters and receivers needed and the potential for more (wavelength) resource availability within the network. The paired cycles results with a 95% confidence interval (CI) for $R = 1$ in Table 7.3 is repeated in column two of Table 7.6 for comparison to the single cycle, increased quorum redundancy technique. Our

المنارة للاستشارات
www.manaraa.com

Table 7.6: Mean links used by single cycles compared to paired cycles using our redundant cyclic quorum solution (95% CI)

	R = 1 (Paired)	**R = 2 (Single)**		**R = 3 (Single)**	
Network	**Links**	**Links**	**Reduction (%)**	**Links**	**Reduction (%)**
NSFNET	249.32 ± 1.37	135.41 ± 0.60	-45.69	145.05 ± 0.74	-41.82
ARPANET	511.90 ± 1.87	269.63 ± 0.76	-47.33	294.75 ± 0.96	-42.42
American	641.38 ± 2.10	360.03 ± 0.98	-43.87	376.86 ± 0.77	-41.24
Chinese	2673.30 ± 7.11	1527.32 ± 3.50	-42.87	1635.70 ± 2.99	-38.81

المنارة للاستشارات
www.manaraa.com

Table 7.7: Mean percent missing node pairs (95% CI) by single cycles using our redundant quorum solution

Network	**R = 1 (Paired) (%)**	**R = 2 (Single) (%)**	**R = 3 (Single) (%)**
NSFNET	0.00 ± 0.00	0.95 ± 0.15	0.04 ± 0.03
ARPANET	0.00 ± 0.00	0.36 ± 0.07	0.13 ± 0.04
American	0.00 ± 0.00	0.49 ± 0.07	0.21 ± 0.04
Chinese	0.00 ± 0.00	0.27 ± 0.03	0.09 ± 0.01

redundant quorum technique uses far fewer links (shown in columns 3 - 6.) $R = 2$ comes close to halving the necessary resources, whereas $R = 3$ is slightly larger at approximately 60% of the paired $R = 1$ paired cycle solution.

Previously paired light-trails were used to form all of the point-to-point communication node pairs with minimum sized quorum cycles. Kleinheksel and Somani (2015c,a) consider utilizing intentionally formed redundant node pairs within the quorum routing to reduce the resources used as a potential trade-off to network performance. We analyze the impact of increasing the redundancy within quorums to $R = 3$ and its impact of keeping resource utilization low. To measure this cost, the missing node pairs metric is used.

Ideally, like the paired cycle case, there would be 0% missing, however single cycles do not have the benefit of both (e_i, e_j) and (e_j, e_i) pairs occurring in the same cycle. Table 7.7 shows two important results. First, the dramatic reduction in resource utilization came at a trade off of a few missing communication pairs. $R = 2$ missed 0.95% or fewer on average (95% CI), and $R = 3$ missed even fewer at 0.21% or less on average (95% CI). The paired cycles (column 2, Table 7.6) used significantly more resources and did not miss any pairs (column 2, Table 7.7). Secondly, compared to $R = 2$ single cycles, our redundant $R = 3$ cycles performs approximately 2+ times better every time. As seen in Table 7.6, this performance improvement came at a only a slightly higher cost, while still being significantly smaller than the state of art approach.

The cyclic quorums set method was proven to guarantee that all of the node pairs exist (Sections 5.3.2 and 5.4). It is the limitations of unidirectional optical light-trail cycles with its required one optical shutter in the *off* state per cycle that has caused the missing pairs and the potential need for additional compensation steps. Compensation is possible using an

المنارة للاستشارات
www.manaraa.com

Table 7.8: Percent mean fault coverage (95% CI) of our single cycle, redundant quorum solution experiencing a single link fault.

Network	**R = 1 (Paired) (%)**	**R = 2 (Single) (%)**	**R = 3 (Single) (%)**
NSFNET	99.47 ± 0.04	96.52 ± 0.09	97.81 ± 0.09
ARPANET	99.81 ± 0.01	98.25 ± 0.05	99.05 ± 0.04
American	99.60 ± 0.02	98.23 ± 0.04	98.90 ± 0.03
Chinese	99.90 ± 0.00	99.36 ± 0.01	99.71 ± 0.00

off-the-shelf solution of an additional routing step involving an Optical-to-Electrical-to-Optical (O/E/O) conversion and retransmission by a hub node. Even so, on average the $R = 2$ and $R = 3$ redundant quorums cycle solutions would require infrequent additional steps considering the missing pairs are less than 1% on average.

7.5.2 Fault-tolerant operational analysis

Using our generalized R quorum redundancy rather than cycle pairs can save significant resources; however, this cannot come at a significant determent to fault tolerance.

7.5.2.1 Single fault case

Again to model the fault, we simulate the failure of each used edge, $(e_i, e_j) \in E$, in the 100 node mappings of each network model, $G = (V, E)$. The edges not used in a particular mapping are ignored to prevent biasing the results with zero missing pairs data points. We then examine the network's ability to serve all potential point-to-point requests by counting pairs of nodes that would be able to communicate and conversely those pairs that are unable to communicate. The results are then reported as fault coverage, total pairs able to communicate as a percentage of total point-to-point pairs. 100% would be perfect coverage, whereas 0% would be no fault coverage at all.

Our simulation results (Table 7.8) showed our redundant quorum-based cycle technique had 96.52 - 99.36% and 97.81 - 99.71% fault coverages, $R = 2$ and $R = 3$ respectively, in the four networks tested. In Figure 7.5, we compare the state-of-art paired cycle approach with our quorum redundant technique with single cycles that uses significantly fewer resources. With

المنارة للاستشارات
www.manaraa.com

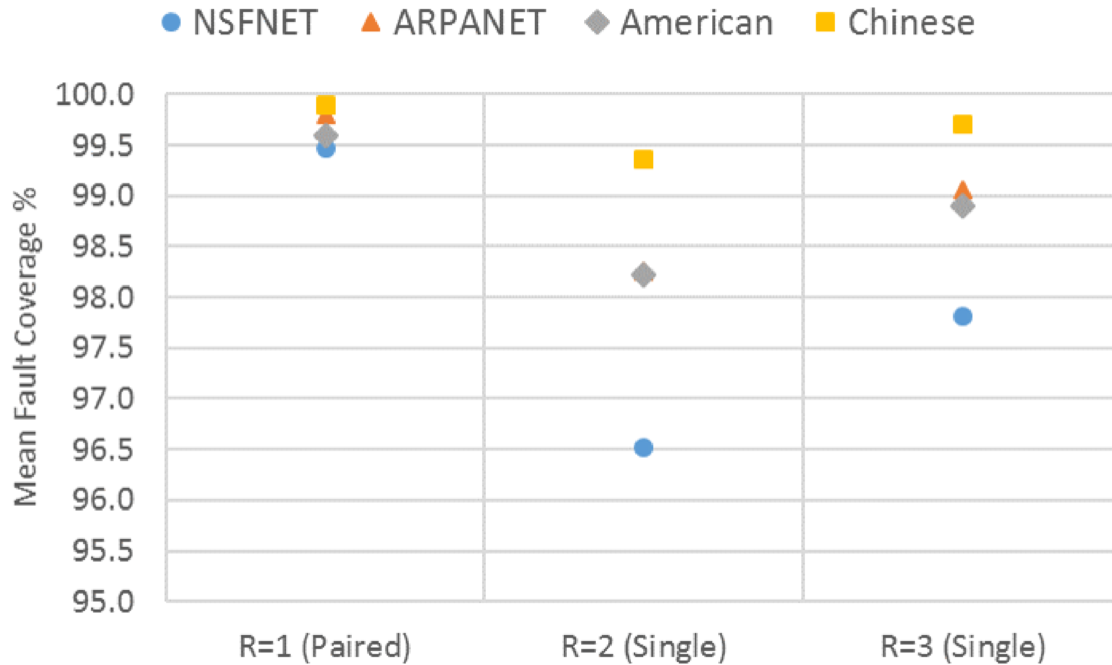


Figure 7.5: Percent mean fault coverage of our single cycle, redundant quorum solution experiencing a single link fault.

single link failures, the paired cycles had a mean missing communication pair rate of less than 3 pairs or less than 0.53% across all networks (95% CI). Hence the $R = 1$ (Paired) column shows mean fault coverage percentages is greater than 99.47% for all four networks. Our redundant quorum cycles technique, $R = 2$ and $R = 3$ (Single), could not reach that level of coverage, but did achieve an acceptable mean fault coverage rate (95% CI) of greater than 96.52 and 97.81%, respectively, across all networks.

While neither single cycle $R = 2$ or $R = 3$ could achieve the same level of fault coverage as the paired cycle solution, they did have missing pair rates better than 3.48 and 2.29%, respectively, while achieving significant resource savings. In networks where an additional approximately 40% of resources could better be utilized for communication rather than redundancy, the trade off of missing a relatively small percentage of communications during fault conditions may be considered tolerable.

المنارة للاستشارات
www.manaraa.com

Table 7.9: Percent mean fault coverage (95% CI) of our single cycle, redundant quorum solution experiencing two simultaneous link faults.

Network	R = 1 (Paired) (%)	R = 2 (Single) (%)	R = 3 (Single) (%)
NSFNET	97.61 ± 0.03	91.94 ± 0.05	93.37 ± 0.05
ARPANET	98.71 ± 0.02	95.17 ± 0.03	96.64 ± 0.03
American	98.63 ± 0.01	96.09 ± 0.02	97.20 ± 0.01
Chinese	99.69 ± 0.00	98.77 ± 0.00	99.33 ± 0.00

7.5.2.2 Two fault case

The more complex two fault model considers all possible two edge failure combinations in the simulated networks. Edges not utilized by a particular set of quorum cycle routings are ignored to prevent biasing of the results data with zero missing pairs data points. We then examine the network's ability to serve all potential point-to-point requests by counting pairs of nodes that would be able to communicate and conversely those pairs that are unable to communicate.

Our simulation results (Table 7.9) showed our redundant quorum-based cycle technique had 91.94 - 98.77% and 93.37 - 99.33% fault coverage, $R = 2$ and $R = 3$ respectively, in the four networks tested. In Figure 7.6, we compare the state-of-art paired cycle approach with our quorum redundant technique with single cycles that uses significantly fewer resources. With two edge failures, the paired cycles had a mean missing communication pair rate of less than 9 pairs or less than 2.39% across all networks (95% CI). Hence in the $R = 1$ (Paired) column, it can be seen that the fault coverage is greater than 97.61%. Our redundant quorums set technique, $R = 2$ and $R = 3$ (Single), could not reach that level for all networks. Overall the missing pair rate (95% CI) was less than 8.06 and 6.63%, respectively, across all networks, which is reflected in fault coverages greater than 91.94%.

It is worth pointing out again that the paired cycles solution uses more than 38% more resources on average (95% CI). The redundant quorums cycle technique using only single cycles performed at most 6% worse on average in terms of mean fault coverage on the rarer two simultaneous fault cases. This could be an acceptable trade off in many networks.

المنارة للاستشارات
www.manaraa.com

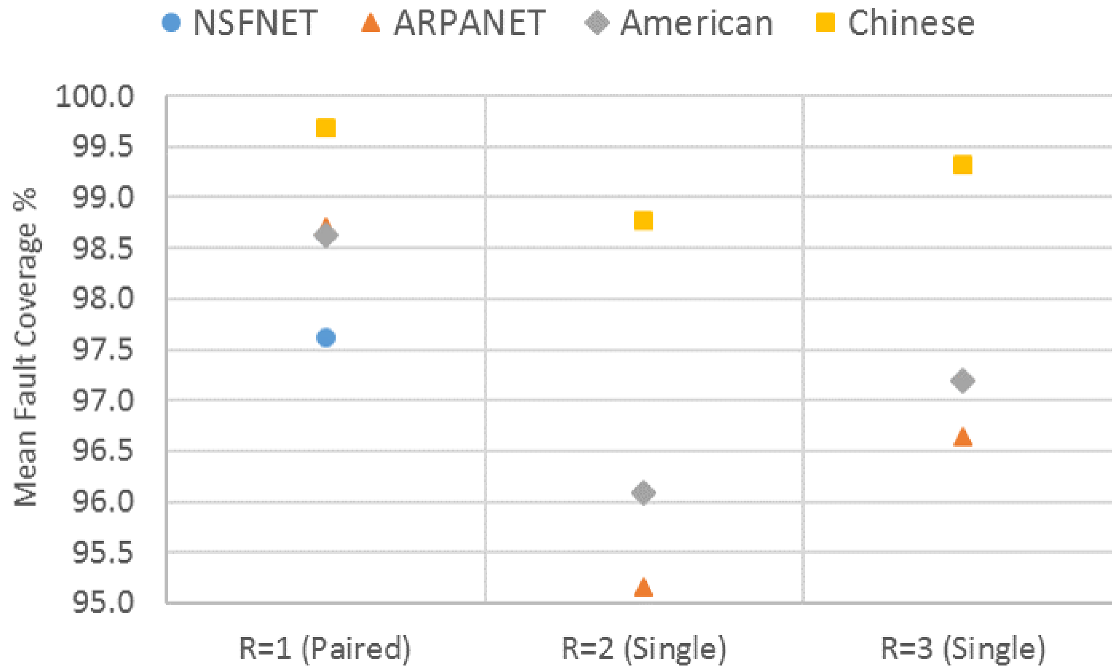


Figure 7.6: Percent mean fault coverage of our single cycle, redundant quorum solution experiencing two simultaneous link faults. For graph clarity and consistency NSFNET for $R = 2$ (Single) at 91.94% and for $R = 3$ (Single) at 93.37% mean fault coverage were not included in the graph.

Additionally, an interesting aspect of the data is that the results appear to have some network dependence. The networks with the larger number of nodes had better resiliency to faults than the smaller networks. This is likely a byproduct of our quorums set solution, where every cycle contains only a small, size k, subset of the total nodes. In all graphs, this translates into cycles that potentially can span the diameter of the graph. In larger graphs this could lead to more non-quorum nodes being passed through while forming the cycle, which could translate into more than the minimum calculated quorums set pairs (see Sections 5.1 and 5.4).

7.6 Improving Single Cycle Routing Based on Redundant Cyclic Quorums

In Section 7.5.1, we had an interesting observation that although the cyclic quorums set method was proven to guarantee that all of the node pairs exist (Sections 5.3.2 and 5.4), the implementation that utilized the quorums still had the limitation of unidirectional optical light-

المنارة للاستشارات
www.manaraa.com

trail cycles requiring one optical shutter in the *off* state per cycle. This led to missing pairs even in fault-free states of the network and likely contributed to at least some of missing pairs during fault scenarios.

When we used paired cycles, the routing order within a cycle did not matter too much. In any given pair of cycles, there was one in the forward direction and the other in the backwards. For example, if the forward cycle route formed the $0 \rightarrow 1$ communication route, then the backward cycle route would form the $0 \leftarrow 1$ communication route.

With a single, unidirectional cycle we were taking for granted that redundant cyclic quorums were guaranteeing two pairs of all node pairs. However, the route direction of the pairs is not enforced. Hence, without the backwards cycle from paired cycle, it turned out that the single cycle could still result in two forward $0 \rightarrow 1$ paths in the network's cycle route implementation rather than one of each ($0 \rightarrow 1$ and $0 \leftarrow 1$ communication routes).

Looking closer at the source of the problem, i.e., the unidirectional optical light-trail cycles, we developed an alternative heuristic solution to improve the performance. The cycles were routed using the ECBRA heuristic [Somani et al. (2011)] and the redundant cyclic quorums for $R = 2$ and $R = 3$ (see Appendix B for the redundant cyclic quorum listing). Once routed, we treated the first node in the corresponding cyclic quorum to be the hub node and then formed the cycle in the order provided by the ECBRA heuristic. By controlling the order to either be forward or backwards on a particular unidirectional optical light-trail cycle, we can influence which node pairs are formed.

7.6.1 Greedy cycle direction based on missing pairs

There are $|V| = P$ redundant cyclic quorums for a given network $G = (V, E)$. Each of these quorums has a corresponding cycle route and each route is either in the forward or backward direction. This results in $O\left(2^P\right)$ possible combinations of cycle directions for a network.

In this section, we describe our greedy algorithm to determine cycle direction and evaluate the improvement in network performance. Ultimately, by controlling the forwards or backwards direction of a cycle, we want to decrease (and eliminate) the number of missing pairs under fault-free conditions. While doing this, the network's fault tolerance is also anticipated to increase.

المنارة للاستشارات
www.manaraa.com

Algorithm 18 Initial Cycle Direction($Cycles$,V)

1: Pair Count $PC\left[e_i\right]\left[e_j\right] \leftarrow 0,\ \forall e_i, e_j \in V$
2: **for all** $c \in Cycles$ **do**
3: Count number of new pairs added to PC if Cycle $c.direction = Forward$
4: Count number of new pairs added to PC if Cycle $c.direction = Backward$
5: **if** Forward count $\geq$ Backward count **then**
6: $c.direction \leftarrow Forward$
7: Increment $PC\left[e_i\right]\left[e_j\right]$ for each forward direction pair from Cycle c
8: **else**
9: $c.direction \leftarrow Backward$
10: Increment $PC\left[e_i\right]\left[e_j\right]$ for each backward direction pair from Cycle c
11: **end if**
12: **end for**

Algorithm 18 greedily chooses each cycle's initial direction. All forward and backward node pairs need to be formed in the network. We keep track of how many of each pair have been form in variable PC on Line 1. We iterate through all of the routed cycles, one for each redundant quorum (Line 2). The cycle's direction is chosen by whichever direction will eliminate more missing pairs from PC. Then all of that direction's pairs (not just the missing ones) are added to PC (Line 7 or Line 10). The next cycle's direction is chosen the same way until all cycles have been assigned an initial direction.

There are $|V| = P$ redundant cyclic quorums for a given network $G = (V, E)$. Each of these quorums has a corresponding cycle route in $Cycles$ causing Algorithm 18's For-loop on Line 2 to execute $O(P)$ times. Both forward and backward new pair counts, as well as, the final incrementing of PC after a direction is chosen requires enumerating all possible pairs in a particular cycle. From Equation 5.28 in Section 5.4, each quorum (i.e., cycle) will be of size approximately $O\left(\sqrt{RP}\right)$. Forming all pairs is $\binom{N}{2} = O\left(N^2\right)$ operation, so $O\left(\sqrt{RP}^2\right) = O(RP)$. All combined this results in an $O(P * RP) = O\left(RP^2\right)$ runtime to find the initial cycle direction.

The order of cycle iteration has an impact on the final direction of all cycles. It is possible that a cycle processed later may add pairs to PC that a previous cycle had already contributed. This opens up the possibility that a cycle processed earlier may be able to change to a more favorable direction and further reduce the number of missing pairs. Because of this, we have a second greedy heuristic, Algorithm 19.

المنارة للاستشارات
www.manaraa.com

Algorithm 19 Greedy Update Cycle Direction($Cycles$,V,PC)

1: $Changed \leftarrow True$
2: **while** $Changed \neq False$ **do**
3: $Changed \leftarrow False$
4: **for all** $c \in Cycles$ **do**
5: Decrement $PC\,[e_i]\,[e_j]$ for each pair from Cycle c
6: Count number of new pairs added to PC if Cycle $c.direction = Forward$
7: Count number of new pairs added to PC if Cycle $c.direction = Backward$
8: **if** Forward count > Backward count **then**
9: Increment $PC\,[e_i]\,[e_j]$ for each forward direction pair from Cycle c
10: **if** $c.direction \neq Forward$ **then**
11: $c.direction \leftarrow Forward$
12: $Changed \leftarrow True$
13: **end if**
14: **else if** Backward count > Forward count **then**
15: Increment $PC\,[e_i]\,[e_j]$ for each backward direction pair from Cycle c
16: **if** $c.direction \neq Backward$ **then**
17: $c.direction \leftarrow Backward$
18: $Changed \leftarrow True$
19: **end if**
20: **else** ▷ Forward count = Backward count, default to no change
21: Increment $PC\,[e_i]\,[e_j]$ for each pair from Cycle c
22: **end if**
23: **end for**
24: **end while**

This algorithm modifies the direction of a single cycle at a time to improve the missing pairs count. It continues modifying cycle directions until no further single cycle direction flips result in improvements being made. Although this appears to be an infinite loop on the surface, its runtime is considerably faster than evaluating all $O\left(2^P\right)$ possible combinations of cycle directions for a network.

Algorithm 19 uses a variable *Changed* to determine if it should continue searching for a better combination of cycle directions (Lines 1-3). Line 4 iterates through all of the routed cycles. The pair count variable PC from the cycle direction initialization is utilized once again to account for the number of node pairs formed and also for how many pairs are missing. The cycle's direction is then chosen by whichever direction will eliminate more missing pairs from PC. Then all of that direction's pairs (not just the missing ones) are added to PC (Line 9 or Line 15). If this selection causes the cycle's direction to change, then the *Changed* variable is

المنارة للاستشارات
www.manaraa.com

set to $True$ so the While-loop knows to continue searching. If the backward and forward missing pair counts are the same, then the direction defaults to the direction the cycle is currently (Line 20). This process continues until an entire iteration of $Cycles$ results in no direction changes.

The outer While-loop continues until there are no direction changes. A direction change will only occur if it results in an improvement in the number of missing pairs. There are $|V| = P$ nodes and forming all pairs is $\binom{P}{2} = O\left(P^2\right)$ operation. So, even if every iteration of the While-loop only improved the number of missing pairs by 1, it would still only loop $O\left(P^2\right)$ times. The For-loop on Line 4 will execute $O\left(P\right)$ times as it iterates over all cycles. Both forward and backward new pair counts, as well as, the initial decrement and final incrementing of PC after a direction is chosen requires enumerating all possible pairs in a particular cycle, hence $O\left(RP\right)$, same as was calculated for Algorithm 18. All combined, this results in an $O\left(P^2 * P * RP\right) = O\left(RP^4\right)$ runtime to greedily update the cycle direction.

A 4th power is certainly not desired for most algorithms; but compared to $O\left(2^P\right)$, it scales well. For P greater than 16 this greedy approach is better than the brute force approach. And certainly for $P \leq 16$, the outer While-loop in Algorithm 19 will typically not behave in the worst case having to execute $O\left(P^2\right)$ times. In fact, given results from Section 7.5.1, on average less than 1% of the pairs were missing. Therefore, we can expect that the runtime behavior of the While-loop to be quite small for network inputs similar to those in our analysis.

7.6.2 Greedy missing pairs heuristic results

This section uses a similar experiment setup as Sections 7.4 and 7.5. We used the same four common networks (Fig. 7.2 on Page 86) and an implementation of the ECBRA heuristic [Somani et al. (2011)] to perform the cycle routing. We also are using just a single cycle based on our generalized R redundant quorums sets (see Appendix B for the redundant cyclic quorum listing).

7.6.2.1 Fault-free operational analysis

Section 7.5.1 showed significant resource usage reductions freeing up (wavelength) resource availability within the network and adding the potential for lower capital costs in terms of

www.manaraa.com

Table 7.10: Comparing fault-free operation mean percent missing node pairs (95% CI) by single cycles using our redundant quorum solution and greedy cycle direction heuristic

Network	R = 1 (Paired) (%)	R = 2 (Single) (%) Forward	Random	Greedy	Reduction
NSFNET	0.00 ± 0.00	0.95 ± 0.15	0.85 ± 0.12	0.02 ± 0.02	-98.27
ARPANET	0.00 ± 0.00	0.36 ± 0.07	0.31 ± 0.06	0.01 ± 0.01	-97.78
American	0.00 ± 0.00	0.49 ± 0.07	0.52 ± 0.07	0.02 ± 0.01	-95.56
Chinese	0.00 ± 0.00	0.27 ± 0.03	0.26 ± 0.02	0.01 ± 0.00	-94.98

Table 7.10: (Continued)

Comparing fault-free operation mean percent missing node pairs (95% CI) by single cycles using our redundant quorum solution and greedy cycle direction heuristic

Network	R = 1 (Paired) (%)	R = 3 (Single) (%) Forward	Random	Greedy	Reduction
NSFNET	0.00 ± 0.00	0.04 ± 0.03	0.85 ± 0.12	0.00 ± 0.00	-100
ARPANET	0.00 ± 0.00	0.13 ± 0.04	0.31 ± 0.06	0.00 ± 0.00	-100
American	0.00 ± 0.00	0.21 ± 0.04	0.52 ± 0.07	0.00 ± 0.00	-100
Chinese	0.00 ± 0.00	0.09 ± 0.01	0.26 ± 0.02	0.00 ± 0.00	-100

physical transmitters and receivers needed. The challenge, though, was this came at a cost. There were missing communication node pairs even in fault-free operation of the network.

Table 7.10 shows the significant improvements that our greedy heuristic had on this issue. An average reduction of greater than 94% of missing pairs for single cycle routing based on $R = 2$ redundant cyclic quorums. And a complete elimination of missing pairs for $R = 3$ redundancy.

Column 2 in Table 7.10 is the prior art paired cycle solution (Same results as are in Table 7.7). This requires at minimum 38% more link resources on average, but no communication node pairs are missing. Originally in Section 7.5, we used the cycle routing directly from the ECBRA heuristic and called this the forward direction. This produced the percent missing pairs in Column 3 (Same results as are in Table 7.7).

The emphasis on the greedy heuristic was that chosen direction of the cycle matters. To emphasize this point and to confirm that our approach is actually doing something intelligent, we also included a random cycle direction algorithm (Column 4). There we see the random

www.manaraa.com

Table 7.11: Greedy heuristic mean cycle direction flips while optimizing the redundant quorum single cycle solution (95% CI)

Network	**R = 2 (Single)**	**R = 3 (Single)**
NSFNET	0.31 ± 0.11	0.00 ± 0.00
ARPANET	0.24 ± 0.11	0.00 ± 0.00
American	0.50 ± 0.14	0.09 ± 0.06
Chinese	1.61 ± 0.25	0.18 ± 0.08

direction on average performs similar (and at times worse) than simply using the forward cycle directions. Columns 5 and 6 show the impact from our greedy heuristic. Nearly all of the missing pairs are removed on average from the $R = 2$ redundant quorum cycles and all of the missing pairs are eliminated from the $R = 3$ redundant quorum cycles. This makes our R redundant quorum, single cycle solution a significant improvement over prior art's paired cycle approach. All communication pairs are formed in the network and done so with significantly fewer resources.

In the prior section (Section 7.6.1), Algorithm 19 had a theoretical runtime of $O\left(RP^4\right)$, which could appear to be significant. One of the drivers of this was that there could potentially be up to $O\left(P^2\right)$ cycle direction changes. When we ran our heuristic while testing on the four networks in Figure 7.2, we did not find anywhere close to that many direction changes. Table 7.11 shows that on real networks, the number of flips is less than two on average. This means that the observable runtime of the greedy heuristic is much more manageable and closer to $O\left(RP^2\right)$, which is the same as Algorithm 18's greedy determination of the initial single cycle directions. Both runtimes are significantly better than the brute force alternative of $O\left(2^P\right)$.

7.6.2.2 Fault-tolerant operational analysis

Section 7.5.2 illustrated a trade off between the significant reduction to resource usage and maintaining the same level of fault tolerance that prior art offered. Using only single cycles had a small impact on fault tolerance of the optical network, which for some networks and applications may still be acceptable given the improved resource usage. With the greedy

المنارة للاستشارات
www.manaraa.com

Table 7.12: Comparing percent mean fault coverage (95% CI) of our single cycle, redundant quorum solution and greedy cycle direction heuristic experiencing a single link fault.

Network	**R = 1 (Paired) (%)**	**R = 2 (Single) (%)**		
		Forward	**Random**	**Greedy**
NSFNET	99.47 ± 0.04	96.52 ± 0.09	96.84 ± 0.08	97.92 ± 0.07
ARPANET	99.81 ± 0.01	98.25 ± 0.05	98.28 ± 0.05	98.79 ± 0.04
American	99.60 ± 0.02	98.23 ± 0.04	98.21 ± 0.04	98.92 ± 0.03
Chinese	99.90 ± 0.00	99.36 ± 0.01	99.38 ± 0.01	99.67 ± 0.01

Table 7.12: (Continued)

Comparing percent mean fault coverage (95% CI) of our single cycle, redundant quorum solution and greedy cycle direction heuristic experiencing a single link fault.

Network	**R = 1 (Paired) (%)**	**R = 3 (Single) (%)**		
		Forward	**Random**	**Greedy**
NSFNET	99.47 ± 0.04	97.81 ± 0.09	96.84 ± 0.08	98.59 ± 0.06
ARPANET	99.81 ± 0.01	99.05 ± 0.04	98.28 ± 0.05	99.42 ± 0.03
American	99.60 ± 0.02	98.90 ± 0.03	98.21 ± 0.04	99.34 ± 0.02
Chinese	99.90 ± 0.00	99.71 ± 0.00	99.38 ± 0.01	99.84 ± 0.00

heuristic addressing missing communication pair issues in the prior section, this section looks at the improvements to fault tolerance.

To model the fault(s), we simulate the failure of used edge(s), $(e_i, e_j) \in E$, in the 100 node mappings of each network model, $G = (V, E)$. The edges not used are ignored. We then examine the network's ability to serve all potential point-to-point requests. The results are then reported as fault coverage, total pairs able to communicate as a percentage of total point-to-point pairs. 100% would be perfect fault coverage, whereas 0% is no fault coverage at all.

Our single fault simulation results (Table 7.12) showed our redundant quorum-based cycle technique with the greedy cycle direction heuristic had 97.92 - 99.67% and 98.59 - 99.84% fault coverages, $R = 2$ and $R = 3$ respectively, in the four networks tested. This was a 31.16 - 48.74% and 35.48 - 44.85%, $R = 2$ and $R = 3$ respectively, improvement over the number of missing pairs when only using forward cycle directions (i.e., Section 7.5.2). Again, we also test the performance against random cycle direction choices (Column 4). This illustrates that

www.manaraa.com

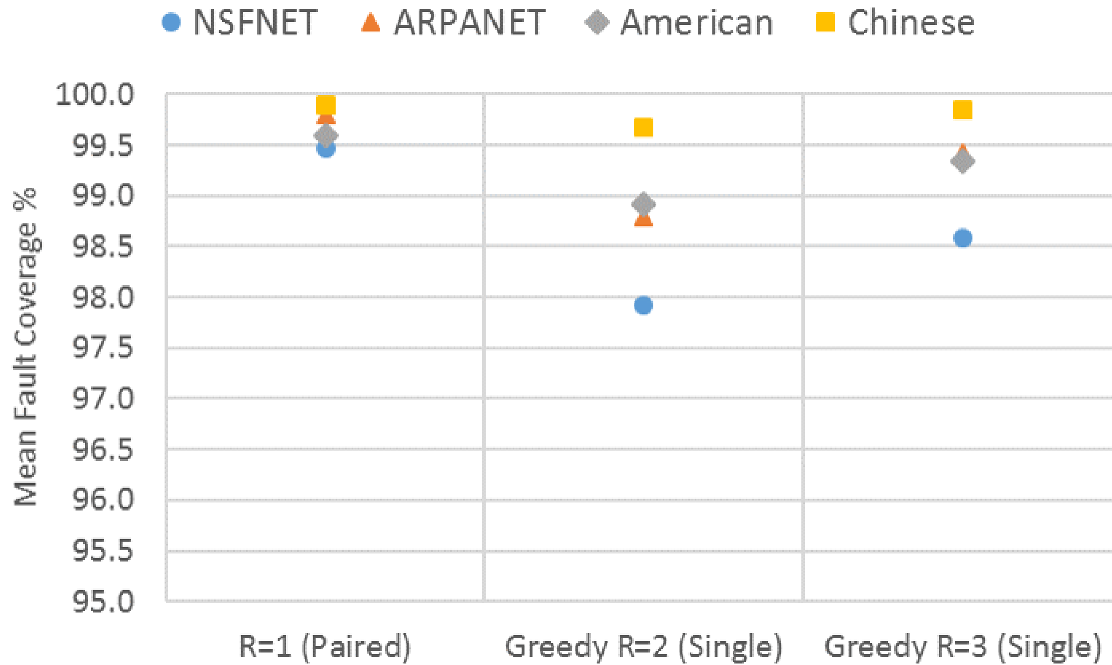


Figure 7.7: Mean fault coverage (%) of our single cycle, redundant quorum solution and greedy cycle direction heuristic experiencing a single link fault.

the greedy heuristic, while it was designed to eliminate missing pairs for fault-free network operation, still provides an intelligent improvement for fault conditions as well.

In Figure 7.7, we compare the state-of-art paired cycle approach with our greedy quorum redundant technique with single cycles that uses significantly fewer resources. With single link failures, the paired cycles had a mean missing communication pair rate of less than 3 pairs or less than 0.53% across all networks (95% CI). Hence the $R = 1$ (Paired) column shows mean fault coverage percentages is greater than 99.47% for all four networks. Our redundant quorum cycles technique with greedy heuristic, $R = 2$ and $R = 3$ (Single), reduced the number of missing pairs by greater than 30% when compared to without the heuristic (i.e., all forward cycles). Even with the improvement though, the fault tolerance still could not reach the level of coverage that the prior art's paired cycles. The Figure 7.7 does show a competitive mean fault coverage rate (95% CI) of greater than 97.92 and 98.59%, respectively, across all networks when the greedy cycle direction heuristic was used with our single quorum cycle solution.

المنارة للاستشارات
www.manaraa.com

Table 7.13: Comparing percent mean fault coverage (95% CI) of our single cycle, redundant quorum solution and greedy cycle direction heuristic experiencing two simultaneous link faults.

Network	**R = 1 (Paired) (%)**	**R = 2 (Single) (%)**		
		Forward	**Random**	**Greedy**
NSFNET	97.61 ± 0.03	91.94 ± 0.05	92.43 ± 0.05	93.67 ± 0.05
ARPANET	98.71 ± 0.02	95.17 ± 0.03	95.18 ± 0.03	95.89 ± 0.03
American	98.63 ± 0.01	96.09 ± 0.02	96.09 ± 0.02	96.99 ± 0.01
Chinese	99.69 ± 0.00	98.77 ± 0.00	98.78 ± 0.00	99.13 ± 0.00

Table 7.13: (Continued)

Comparing percent mean fault coverage (95% CI) of our single cycle, redundant quorum solution and greedy cycle direction heuristic experiencing two simultaneous link faults.

Network	**R = 1 (Paired) (%)**	**R = 3 (Single) (%)**		
		Forward	**Random**	**Greedy**
NSFNET	97.61 ± 0.03	93.37 ± 0.05	92.43 ± 0.05	94.88 ± 0.05
ARPANET	98.71 ± 0.02	96.64 ± 0.03	95.18 ± 0.03	97.33 ± 0.02
American	98.63 ± 0.01	97.20 ± 0.01	96.09 ± 0.02	97.83 ± 0.01
Chinese	99.69 ± 0.00	99.33 ± 0.00	98.78 ± 0.00	99.50 ± 0.00

Our two simultaneous link fault simulation results (Table 7.13) showed our redundant quorum-based cycle technique with the greedy cycle direction heuristic had 93.67 - 99.13% and 94.88 - 99.50% fault coverages, $R = 2$ and $R = 3$ respectively, in the four networks tested. This was a 14.98 - 29.61% and 20.66 - 26.14%, $R = 2$ and $R = 3$ respectively, improvement over the number of missing pairs when only using forward cycle directions (i.e., Section 7.5.2).

In Figure 7.8, we compare with the state-of-art paired cycle approach. With two simultaneous link failures, the paired cycles had a mean missing communication pair rate of less than 9 pairs or less than 2.39% across all networks (95% CI). Hence the $R = 1$ (Paired) column shows mean fault coverage percentages is greater than 97.61% for all four networks. Our redundant quorum cycles technique with greedy heuristic, $R = 2$ and $R = 3$ (Single), reduced the number of missing pairs by greater than 14% when compared to without the heuristic (i.e., all forward cycles). Even with the improvement though, the fault tolerance still could not reach the level of coverage that the prior art's paired cycles. The Figure 7.8 does show a competitive mean fault coverage rates (95% CI) and may be justified by the significant reductions in resource usage (Table 7.6).

المنارة للاستشارات
www.manaraa.com

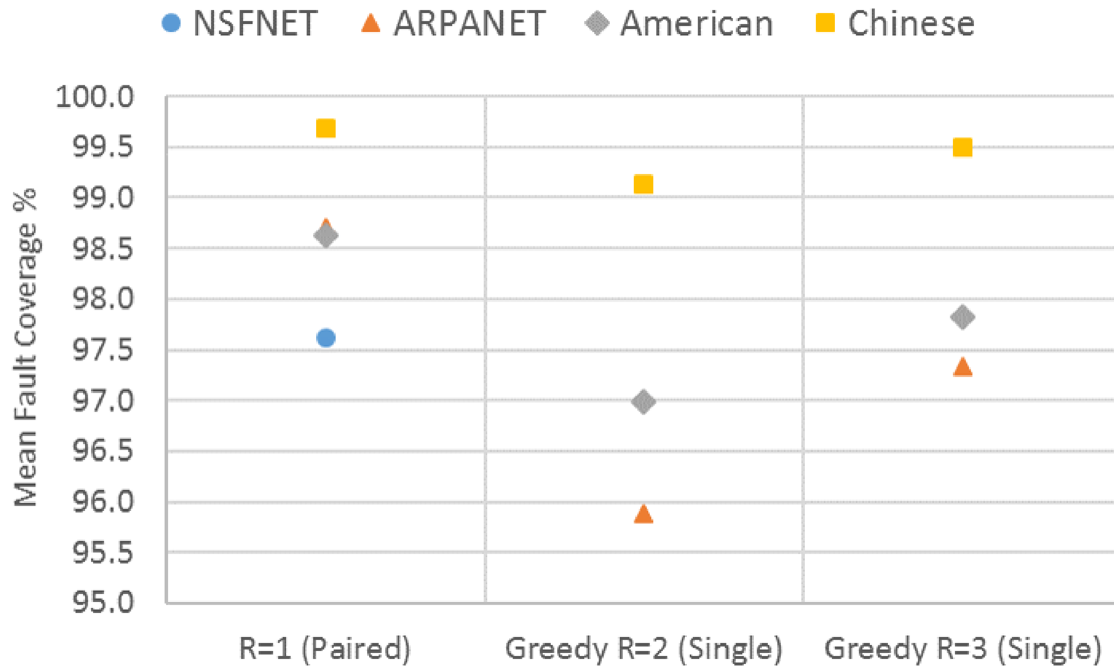


Figure 7.8: Mean fault coverage (%) of our single cycle, redundant quorum solution and greedy cycle direction heuristic experiencing two simultaneous link faults. For graph clarity and consistency NSFNET for $R = 2$ (Single) at 93.67% and for $R = 3$ (Single) at 94.88% mean fault coverage were not included in the graph.

المنارة للاستشارات
www.manaraa.com

CHAPTER 8 CONCLUSIONS

Our work addresses some of the challenges facing "all-pairs" interactions. These interactions require all elements (nodes or data) to interact with all other elements in the set. Challenges with growing data sizes impacts both computations and communications all-pairs applications. We analyzed applications in both areas and proposed solutions based on cyclic quorums sets.

8.1 Quorums

Quorums are typically seen in distributed communication and algorithms, coordination, mutual exclusion, data replication and consensus applications. We used their properties to address challenges in a different problem, all-pairs interactions.

For a given set of elements, numerous quorums sets can be found. Many of those sets may also satisfy desirable distributed properties defined by Maekawa (1985) and may further be able to facilitate the all-pairs interactions in the problems we addressed. Our work did not seek to enumerate all possible quorums set solutions for all-pairs interaction problems, but instead focused on systematic solutions that eloquently and efficiently managed the distributed problems.

To do this, we observed and proved that cyclic quorums have an "all-pairs" property [Kleinheksel and Somani (2016)], such that all pairs of elements are present together in at least one quorum. The cyclic quorums also had desirable properties like equal size and equal distribution of elements, such that work and responsibilities were equable.

Observing that there could be redundant pairs within the quorums set, we sought to exploit this natural occurrence. We defined R redundant cyclic quorums sets as having R redundant pairs of elements [Kleinheksel and Somani (2015c,a)]. The same difference sets that guaranteed all-pairs could also be used to guarantee R all-pairs too.

المنارة للاستشارات
www.manaraa.com

8.2 Computation Application

We used cyclic quorums sets to scale all-pairs computation problems. Cyclic quorum sets have an all-pairs property that allows for data replication to be minimal and leads to simple computation management as well [Kleinheksel and Somani (2016)]. The corresponding dataset quorums were $O\left(\frac{N}{\sqrt{(P)}}\right)$ in size, up to 50% smaller than the dual $\frac{N}{\sqrt{(P)}}$ array implementations, and significantly smaller than solutions requiring all data. The cyclic quorums "all-pairs" property led to a simple algorithm design with all of the data needed for pairing existing in a node's dataset quorum.

Our evaluation took a single node bioinformatics all-pairs implementation and demonstrated scalability with our cyclic quorum set methods on real and synthetic datasets. Average runtimes showed the ability to scale linearly with additional nodes added. On one real dataset, greater than 150x (super-linear) runtime speedup and 80% reduction in memory usage per node was observed using 31 nodes.

For non-Singer difference sets, we developed a decentralized, communication-less computation management technique to identify and avoid all redundant computations in the $\binom{k}{2}$ quorum dataset pairings. This showed a greater flexibility for users to efficiently utilize all of their local or cloud HPC resources. For some P nodes, the additional computation management logic increased the average runtime speedup by as much as 30%, while the overhead of the management was typically less than 1%.

8.3 Communication Application

We show efficiency, distributed control, and fault tolerance can also be accomplished in optical network routing by applying quorums set theory. Specifically, we used cyclic quorums sets to route light-trail cycles, such that all pairs of nodes occurred in one or more cycles.

We analyzed the fault tolerance of the cyclic quorums set routing approach [Kleinheksel and Somani (2015b)]. In the presence of network single link faults, greater than 99% average fault coverage enabled the continued operation of nearly all point-to-point communication requests in the network.

المنارة للاستشارات

www.manaraa.com

When we applied R redundant cyclic quorums to the state of the art paired cycle techniques, it guaranteed all network communication pairs appeared R times within a routed cycle set [Kleinheksel and Somani (2015a)]. The results showed optical networks achieving near fault-tolerance with 98.65 - 99.91% and 99.04 - 99.95% fault coverage rates respectively on the two simultaneous faults simulation. This improvement in fault tolerance performance for $R = 2$ and $R = 3$ came at the expense of a small additional resource cost. The paired cycle techniques on average used 5.63 - 14.18% and 15.21 - 22.29% more resources, respectively.

When applied to our single cycle routing technique, the R redundant quorums significantly reduced resource usage and maintained high fault tolerance capabilities [Kleinheksel and Somani (2015c,a)]. The single cycle technique had almost fault-tolerant cycles that used significantly fewer resources (42.91 - 47.19% and 38.85 - 42.39% fewer, respectively), while at the same time maintained a high degree of fault-tolerance with 92.01 - 98.77% and 93.23 - 99.34% fault coverage, respectively, on the two simultaneous faults simulation.

With the paired cycles in prior art, the direction of communication pairs were not a consideration because both directions were always present. When we switched to single cycles, the resource usage fell dramatically (greater than 38%), and there was no longer the guarantee that both directions of a communication pair existed. To address this limitation, we developed a greedy algorithm to determine whether a cycle should be routed in the forward or backward direction in order to eliminate the most missing pairs and achieve higher fault tolerance. Missing pairs were reduced by over 94% for $R = 2$ redundant quorum single cycles and eliminated completely for $R = 3$. Fault tolerance improved as well. Missing pairs during single link failures decreased by over 30%. Two simultaneous link failures saw a decrease in missing pairs by over 14%. This illustrated a trade off between significant resource usage reductions and the fault tolerance those resources provide.

www.manaraa.com

BIBLIOGRAPHY

Agrawal, G. P. (2007). *Nonlinear fiber optics*. Academic press.

Amazon (2016a). Amazon ec2 instance types. https://aws.amazon.com/ec2/instance-types/.

Amazon (2016b). Aws high performance computing. https://aws.amazon.com/hpc/.

Arora, N., Shringarpure, A., and Vuduc, R. W. (2009). Direct n-body kernels for multicore platforms. In *ICPP*, volume 9, pages 379–387.

Blasgen, M. W. and Eswaran, K. P. (1977). Storage and access in relational data bases. *IBM Systems Journal*, 16(4):363–377.

Bratbergsengen, K. (1984). Hashing methods and relational algebra operations. In *Proceedings of the 10th International Conference on Very Large Data Bases*, pages 323–333. Morgan Kaufmann Publishers Inc.

Chae, H., Jung, I., Lee, H., Marru, S., Lee, S.-W., and Kim, S. (2013). Bio and health informatics meets cloud: Biovlab as an example. *Health Information Science and Systems*, 1(1):6.

Chae, H., Rhee, S., Nephew, K. P., and Kim, S. (2014). Biovlab-mmia-ngs: microrna–mrna integrated analysis using high-throughput sequencing data. *Bioinformatics*, page btu614.

Chao, C.-M. and Wang, Y.-Z. (2010). A multiple rendezvous multichannel mac protocol for underwater sensor networks. In *Wireless Communications and Networking Conference (WCNC), 2010 IEEE*, pages 1–6. IEEE.

Chapman, T. and Kalyanaraman, A. (2011). An openmp algorithm and implementation for clustering biological graphs. In *Proceedings of the first workshop on Irregular applications: architectures and algorithm*, pages 3–10. ACM.

المنارة للاستشارات
www.manaraa.com

Chinchilla, F., Gamblin, T., Sommervoll, M., and Prins, J. F. (2004). Parallel n-body simulation using gpus. *Department of Computer Science, University of North Carolina at Chapel Hill, http://gamma. cs. unc. edu/GPGP, Technical Report TR04-032*.

Chlamtac, I. and Gumaste, A. (2003). Light-trails: A solution to ip centric communication in the optical domain. In *Quality of Service in Multiservice IP Networks*, pages 634–644. Springer.

Colbourn, C. J. (2010). *CRC handbook of combinatorial designs*. CRC press.

Connolly, A., Habib, S., Szalay, A., Borrill, J., Fuller, G., Gnedin, N., Heitmann, K., Jacobs, D., Lamb, D., Mezzacappa, T., et al. (2013). Snowmass computing frontier: Computing for the cosmic frontier, astrophysics, and cosmology. *arXiv preprint arXiv:1311.2841*.

Drew, C. (2010). Military is awash in data from drones. http://www.nytimes.com/2010/01/11/business/11drone.html.

Driscoll, M., Georganas, E., Koanantakool, P., Solomonik, E., and Yelick, K. (2013). A communication-optimal n-body algorithm for direct interactions. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 1075–1084. IEEE.

Ehnasri, R. and Navathe, S. B. (1989). Fundamentals of database systems. *The Benj amin/Cummings Publishing Company, Inc.*

Fang, J., He, W., and Somani, A. K. (2004). Optimal light trail design in wdm optical networks. In *Communications, 2004 IEEE International Conference on*, volume 3, pages 1699–1703. IEEE.

Feng, T., Ruan, L., and Zhang, W. (2008). Intelligent p-cycle protection for multicast sessions in wdm networks. In *Communications, 2008. ICC'08. IEEE International Conference on*, pages 5165–5169. IEEE.

Ficklin, S. P. and Feltus, F. A. (2013). A systems-genetics approach and data mining tool to assist in the discovery of genes underlying complex traits in oryza sativa. *PloS one*, 8(7):e68551.

المنارة للاستشارات
www.manaraa.com

Fortes, M., Snelling, W., Reverter, A., Nagaraj, S., Lehnert, S., Hawken, R., DeAtley, K., Peters, S., Silver, G., Rincon, G., et al. (2012). Gene network analyses of first service conception in brangus heifers: Use of genome and trait associations, hypothalamic-transcriptome information, and transcription factors. *Journal of Animal Science*, 90(9):2894–2906.

Gibson, S. M., Ficklin, S. P., Isaacson, S., Luo, F., Feltus, F. A., and Smith, M. C. (2013). Massive-scale gene co-expression network construction and robustness testing using random matrix theory. *PloS one*, 8(2):e55871.

Groppe, J. and Groppe, S. (2011). Accelerating large semantic web databases by parallel join computations of sparql queries. *ACM SIGAPP Applied Computing Review*, 11(4):60–70.

Groppe, S. (2011). *Data Management and Query Processing in Semantic Web Databases*. Springer.

Grover, W. D. and Shen, G. (2003). Extending the p-cycle concept to path-segment protection. In *Communications, 2003. ICC'03. IEEE International Conference on*, volume 2, pages 1314–1319. IEEE.

Gumaste, A. and Chlamtac, I. (2003). Light-trails: a novel conceptual framework for conducting optical communications. In *High Performance Switching and Routing, 2003, HPSR. Workshop on*, pages 251–256. IEEE.

Han, X., Li, J., and Yang, D. (2012). Pi-join: Efficiently processing join queries on massive data. *Knowledge and information systems*, 32(3):527–557.

Hedegaard, R. (2016). Handshake problem. http://mathworld.wolfram.com/HandshakeProblem.html.

Hudson, N. J., Reverter, A., and Dalrymple, B. P. (2009a). A differential wiring analysis of expression data correctly identifies the gene containing the causal mutation. *PLoS computational biology*, 5(5):e1000382.

للاستشارات المنارة

www.manaraa.com

Hudson, N. J., Reverter, A., Wang, Y., Greenwood, P. L., and Dalrymple, B. P. (2009b). Inferring the transcriptional landscape of bovine skeletal muscle by integrating co-expression networks. *PloS one*, 4(10):e7249.

Ishiyama, T., Nitadori, K., and Makino, J. (2012). 4.45 pflops astrophysical n-body simulation on k computer–the gravitational trillion-body problem. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–10. IEEE.

Ji, P. N. and Aono, Y. (2010). Colorless and directionless multi-degree reconfigurable optical add/drop multiplexers. In *Wireless and Optical Communications Conference (WOCC), 2010 19th Annual*, pages 1–5. IEEE.

Khalil, A., Hadjiantonis, A., Ellinas, G., and Ali, M. (2005). Pre-planned multicast protection approaches in wdm mesh networks. In *Optical Communication, 2005. ECOC 2005. 31st European Conference on*, pages 25–26. IET.

Kitsuregawa, M., Tanaka, H., and Moto-Oka, T. (1983). Application of hash to data base machine and its architecture. *New Generation Computing*, 1(1):63–74.

Kleinheksel, C. J. and Somani, A. K. (2015a). Enhancing fault tolerance capabilities in quorum-based cycle routing. In *Reliable Networks Design and Modeling (RNDM), 2015 7th International Workshop on*, pages 27–33, Munich, Germany. IEEE.

Kleinheksel, C. J. and Somani, A. K. (2015b). Optical quorum cycles for efficient communication. *Photonic Network Communications*, pages 1–10.

Kleinheksel, C. J. and Somani, A. K. (2015c). Resource efficient redundancy using quorum-based cycle routing in optical networks. In *Transparent Optical Networks (ICTON), 2015 17th International Conference on*, pages 1–4. IEEE.

Kleinheksel, C. J. and Somani, A. K. (2016). Scaling distributed all-pairs algorithms. In *Information Science and Applications (ICISA) 2016*, pages 247–257. Springer.

المنارة للاستشارات

www.manaraa.com

Koesterke, L., Koltes, J. E., Weeks, N. T., Milfeld, K., Vaughn, M. W., Reecy, J. M., and Stanzione, D. (2014). Discovery of biological networks using an optimized partial correlation coefficient with information theory algorithm on stampede's xeon and xeon phi processors. *Concurrency and Computation: Practice and Experience*, 26(13):2178–2190.

Koesterke, L., Milfeld, K., Vaughn, M. W., Stanzione, D., Koltes, J. E., Weeks, N. T., and Reecy, J. M. (2013). Optimizing the pcit algorithm on stampede's xeon and xeon phi processors for faster discovery of biological networks. In *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery*, XSEDE '13, pages 14:1–14:8, New York, NY, USA. ACM.

Kruliš, M. and Yaghob, J. (2011). Revision of relational joins for multi-core and many-core architectures. In *Proceedings of the Dateso 2011 Workshop on Databases, Texts, Specifications and Objects, Písek, Czech Republic*. Citeseer.

Kumar, V. and Agarwal, A. (2011). Multi-dimensional grid quorum consensus for high capacity and availability in a replica control protocol. *High Performance Architecture and Grid Computing*, pages 67–78.

Lastine, D. (2014). *Efficient communication using multiple cycles and multiple channels*. PhD thesis, Iowa State University.

Lastine, D., Sankaran, S., and Somani, A. K. (2012). A fault-tolerant multipoint cycle routing algorithm (mcra). In *Broadband Communications, Networks, and Systems*, pages 341–360. Springer.

Li, Y., Wang, J., Gumaste, A., Xu, Y., and Xu, Y. (2008). Multicast routing in light-trail wdm networks. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1–5. IEEE.

Lienhart, G., Kugel, A., and Manner, R. (2002). Using floating-point arithmetic on fpgas to accelerate scientific n-body simulations. In *Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on*, pages 182–191. IEEE.

للاستشارات

www.manaraa.com

Luk, W.-S. and Wong, T.-T. (1997). Two new quorum based algorithms for distributed mutual exclusion. In *Distributed Computing Systems, 1997., Proceedings of the 17th International Conference on*, pages 100–106. IEEE.

Luo, H., Yu, H., Li, L., and Wang, S. (2006). On protecting dynamic multicast sessions in survivable mesh wdm networks. In *Communications, 2006. ICC'06. IEEE International Conference on*, volume 2, pages 835–840. IEEE.

Madsen, F. M. and Filinski, A. (2013). Towards a streaming model for nested data parallelism. In *Proceedings of the 2nd ACM SIGPLAN workshop on Functional high-performance computing*, pages 13–24. ACM.

Maekawa, M. (1985). An algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems (TOCS)*, 3(2):145–159.

Mandagere, N., Zhou, P., Smith, M. A., and Uttamchandani, S. (2008). Demystifying data deduplication. In *Proceedings of the ACM/IFIP/USENIX Middleware'08 Conference Companion*, pages 12–17. ACM.

Masuyama, S., Ibaraki, T., Nishio, S., and Hasegawa, T. (1987). Shortest semijoin schedule for a local area distributed database system. *Software Engineering, IEEE Transactions on*, 13(5):602–606.

McCreight, E. (1972). Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–189.

Mishra, P. and Eich, M. H. (1992). Join processing in relational databases. *ACM Computing Surveys (CSUR)*, 24(1):63–113.

Moore, A. J., Quillen, A. C., and Edgar, R. G. (2008). Planet migration through a self-gravitating planetesimal disk. *arXiv preprint arXiv:0809.2855*.

Moretti, C., Bui, H., Hollingsworth, K., Rich, B., Flynn, P., and Thain, D. (2010). All-pairs: An abstraction for data-intensive computing on campus grids. *Parallel and Distributed Systems, IEEE Transactions on*, 21(1):33–46.

المنارة للاستشارات
www.manaraa.com

Ordonez, C. and García-García, J. (2010). Evaluating join performance on relational database systems. *JCSE*, 4(4):276–290.

Phillips, P. J., Flynn, P. J., Scruggs, T., Bowyer, K. W., Chang, J., Hoffman, K., Marques, J., Min, J., and Worek, W. (2005). Overview of the face recognition grand challenge. In *Computer vision and pattern recognition, 2005. CVPR 2005. IEEE computer society conference on*, volume 1, pages 947–954. IEEE.

Plimpton, S. (1995). Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics*, 117(1):1–19.

Qing, Y. and Ning, G. (2005). Protecting dynamic multicast sessions in optical wdm mesh networks. In *Information, Communications and Signal Processing, 2005 Fifth International Conference on*, pages 1187–1190. IEEE.

Ramamurthy, S., Sahasrabuddhe, L., and Mukherjee, B. (2003). Survivable wdm mesh networks. *Journal of Lightwave Technology*, 21(4):870.

Reverter, A. and Chan, E. K. (2008). Combining partial correlation and an information theory approach to the reversed engineering of gene co-expression networks. *Bioinformatics*, 24(21):2491–2497.

Rytsareva, I. and Kalyanaraman, A. (2012). An efficient mapreduce algorighm for parallelizing large-scale graph clustering.

Singhal, N. K., Sahasrabuddhe, L. H., and Mukherjee, B. (2003). Provisioning of survivable multicast sessions against single link failures in optical wdm mesh networks. *Journal of lightwave technology*, 21(11):2587.

Somani, A., Lastine, D., and Sankaran, S. (2011). Finding complex cycles through a set of nodes. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–5.

المنارة للاستشارات
www.manaraa.com

Somani, A. K. and Lastine, D. (2014). Optical paths supporting quorums for efficient communication. In *Optical Communications and Networks (ICOCN), 2014 13th International Conference on*. IEEE.

Tang, L., Huang, W., Razo, M., Sivasankaran, A., Tacca, M., and Fumagalli, A. (2011). Multicast tree computation in networks with multicast incapable nodes. In *High Performance Switching and Routing (HPSR), 2011 IEEE 12th International Conference on*, pages 95–100. IEEE.

Tauheed, F., Nobari, S., Biveinis, L., Heinis, T., and Ailamaki, A. (2013). Computational neuroscience breakthroughs through innovative data management. In *Advances in Databases and Information Systems*, pages 14–27. Springer.

Twitter (2016). Post statuses/update. https://dev.twitter.com/rest/reference/post/statuses/update.

Valduriez, P. (1982). Semi-join algorithms for distributed database machines. In *DDB*, pages 23–37.

Valduriez, P. (1987). Join indices. *ACM Transactions on Database Systems (TODS)*, 12(2):218–246.

Van Meel, J. A., Arnold, A., Frenkel, D., Portegies Zwart, S., and Belleman, R. G. (2008). Harvesting graphics power for md simulations. *Molecular Simulation*, 34(3):259–266.

Watson-Haigh, N. S., Kadarmideen, H. N., and Reverter, A. (2010). Pcit: an r package for weighted gene co-expression networks based on partial correlation and information theory approaches. *Bioinformatics*, 26(3):411–413.

Wetterstrand, K. (2013). Dna sequencing costs: Data from the nhgri genome sequencing program (gsp). www.genome.gov/sequencingcosts/.

Wu, C. and Kalyanaraman, A. (2013). Gpu-accelerated protein family identification for metagenomics. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 559–568. IEEE.

المنارة للاستشارات

www.manaraa.com

Yokota, R. and Barba, L. A. (2012). Hierarchical n-body simulations with autotuning for heterogeneous systems. *Computing in Science & Engineering*, 14(3):30–39.

Zhang, F. and Zhong, W.-D. (2007). Applying p-cycles in dynamic provisioning of survivable multicast sessions in optical wdm networks. In *Optical Fiber Communication Conference*, page JWA74. Optical Society of America.

Zhang, F. and Zhong, W.-D. (2008). Performance evaluation of p-cycle based protection methods for provisioning of dynamic multicast sessions in mesh wdm networks. *Photonic Network Communications*, 16(2):127–138.

Zhang, F., Zhong, W.-D., and Jin, Y. (2008). Optimizations of-cycle-based protection of optical multicast sessions. *Lightwave Technology, Journal of*, 26(19):3298–3306.

Zhang, W., Kandah, F., Wang, C., and Li, H. (2011). Dynamic light trail routing in wdm optical networks. *Photonic Network Communications*, 21(1):78–89.

المنارة للاستشارات

www.manaraa.com

APPENDIX A OPTIMAL CYCLIC QUORUMS

Optimal cyclic quorums are difficult to find and require an exhaustive search Maekawa (1985); Luk and Wong (1997). Once found, their benefits to all-pairs computations and communications has been shown in our papers as well as related work Somani and Lastine (2014); Kleinheksel and Somani (2015b, 2016).

To enable the use of optimal cyclic quorums and eliminate the need for others to perform the necessary exhaustive search to find the quorum sets, Table A.1 has been reproduced from known optimal cyclic quorums Luk and Wong (1997). We have verified that all of these are correct and are optimal cyclic quorums.

Table A.1: Optimal cyclic quorums for $N = 4$ to 111

N	$\mathbf{|S_i| = k}$	Optimal Cyclic Quorum					
4	3	1	2	3			
5	3	1	2	3			
6	3	1	2	4			
7	3	1	2	4			
8	4	1	2	3	5		
9	4	1	2	3	5		
10	4	1	2	3	6		
11	4	1	2	3	6		
12	4	1	2	4	8		
13	4	1	2	4	10		
14	5	1	2	3	4	8	
15	5	1	2	3	4	8	
16	5	1	2	3	6	9	
17	5	1	2	3	5	13	
18	5	1	2	3	6	12	
19	5	1	2	3	7	10	
20	6	1	2	3	4	7	11
21	5	1	2	5	15	17	
22	6	1	2	3	4	8	12
23	6	1	2	3	4	8	12

www.manaraa.com

Table A.1: (Continued)

Optimal cyclic quorums for $N = 4$ to 111

N	$\mathbf{\vert S_i \vert = k}$	**Optimal Cyclic Quorum**								
24	6	1	2	3	4	8	16			
25	6	1	2	3	4	9	13			
26	6	1	2	3	6	10	16			
27	6	1	2	3	6	14	23			
28	6	1	2	5	16	21	23			
29	7	1	2	3	4	5	10	15		
30	7	1	2	3	4	5	10	20		
31	6	1	2	4	9	13	19			
32	7	1	2	3	4	8	12	20		
33	7	1	2	3	4	7	17	28		
34	7	1	2	3	4	8	13	21		
35	7	1	2	3	4	9	13	22		
36	7	1	2	3	6	13	15	21		
37	7	1	2	3	5	11	16	23		
38	8	1	2	3	4	5	9	15	24	
39	7	1	2	3	5	14	19	34		
40	8	1	2	3	4	5	10	15	25	
41	8	1	2	3	4	5	10	16	26	
42	8	1	2	3	4	5	10	16	26	
43	8	1	2	3	4	5	11	16	27	
44	8	1	2	3	4	7	17	28	39	
45	8	1	2	3	4	6	13	19	27	
46	8	1	2	3	4	7	19	26	39	
47	8	1	2	3	4	6	17	23	41	
48	8	1	2	3	6	10	21	27	37	
49	8	1	2	3	6	25	34	37	45	
50	8	1	2	4	9	18	29	33	39	
51	8	1	2	3	6	12	19	31	39	
52	9	1	2	3	4	5	7	15	22	31
53	9	1	2	3	4	5	8	22	30	45
54	9	1	2	3	4	5	10	16	22	32
55	9	1	2	3	4	5	7	20	27	48
56	9	1	2	3	4	5	12	17	34	40
57	8	1	2	4	14	33	37	44	53	
58	9	1	2	3	4	8	22	34	38	51
59	9	1	2	3	4	7	14	22	36	45
60	9	1	2	3	5	10	16	26	31	43
61	9	1	2	3	4	8	16	26	37	46
62	9	1	2	3	5	11	33	40	47	52

المنارة للاستشارات
www.manaraa.com

Table A.1: (Continued)

Optimal cyclic quorums for $N = 4$ to 111

N	$\mathbf{\|S_i\| = k}$	**Optimal Cyclic Quorum**											
63	9	1	2	3	7	9	21	39	42	55			
64	9	1	2	3	6	15	17	35	43	60			
65	9	1	2	3	7	11	29	36	52	55			
66	10	1	2	3	4	5	6	14	20	40	47		
67	10	1	2	3	4	5	6	13	21	27	40		
68	10	1	2	3	4	5	11	17	22	39	46		
69	10	1	2	3	4	5	11	18	23	34	46		
70	10	1	2	3	4	5	10	21	36	50	63		
71	10	1	2	3	4	5	11	19	24	35	47		
72	10	1	2	3	4	7	12	19	32	38	52		
73	9	1	2	4	8	16	32	37	55	64			
74	10	1	2	3	4	8	29	31	44	58	66		
75	10	1	2	3	6	9	19	31	33	42	57		
76	10	1	2	3	7	10	26	36	47	59	64		
77	10	1	2	3	5	11	16	38	50	57	62		
78	10	1	2	3	8	14	17	34	52	56	71		
79	10	1	2	3	7	14	29	32	48	49	72		
80	11	1	2	3	4	5	6	11	24	41	57	72	
81	11	1	2	3	4	5	6	13	21	27	40	54	
82	11	1	2	3	4	5	6	13	21	27	41	54	
83	11	1	2	3	4	5	6	13	22	28	41	55	
84	11	1	2	3	4	5	8	19	27	47	55	76	
85	11	1	2	3	4	5	10	14	26	41	55	69	
86	11	1	2	3	4	5	12	18	25	30	49	55	
87	11	1	2	3	4	5	11	43	55	63	68	74	
88	11	1	2	3	4	6	12	25	30	37	44	74	
89	11	1	2	3	4	6	13	19	44	58	66	72	
90	11	1	2	3	4	7	34	47	55	68	75	82	
91	10	1	2	4	10	28	50	57	62	78	82		
92	11	1	2	3	5	41	51	52	60	65	72	78	
93	11	1	2	3	6	15	21	25	32	53	61	69	
94	12	1	2	3	4	5	6	7	15	24	31	47	62
95	11	1	2	3	6	9	18	29	40	54	64	83	
96	12	1	2	3	4	5	6	9	22	31	54	63	87
97	12	1	2	3	4	5	6	10	18	34	44	55	80
98	12	1	2	3	4	5	6	12	28	41	55	70	82
99	12	1	2	3	4	5	6	13	22	28	35	49	63
100	12	1	2	3	4	5	6	14	21	29	35	57	64
101	12	1	2	3	4	5	6	13	50	64	73	79	86

المنارة للاستشارات
www.manaraa.com

Table A.1: (Continued)

Optimal cyclic quorums for $N = 4$ to 111

N	$\mathbf{|S_i| = k}$	**Optimal Cyclic Quorum**											
102	12	1	2	3	4	5	7	14	29	35	43	51	86
103	12	1	2	3	4	5	8	39	54	63	78	86	94
104	12	1	2	3	4	5	10	20	33	47	58	73	85
105	12	1	2	3	4	5	11	16	37	40	62	67	90
106	12	1	2	3	4	6	49	54	70	77	83	90	98
107	12	1	2	3	4	6	21	28	36	43	49	59	99
108	12	1	2	3	4	8	13	21	35	42	50	58	86
109	12	1	2	3	4	8	16	40	50	59	84	90	95
110	12	1	2	3	7	18	26	40	44	47	53	81	101
111	12	1	2	3	6	13	28	37	39	45	53	66	94

المنارة للاستشارات
www.manaraa.com

APPENDIX B   REDUNDANT CYCLIC QUORUMS

Optimal redundant cyclic quorums, like their non-redundant siblings in Appendix A, are difficult to find and require an exhaustive search Maekawa (1985); Luk and Wong (1997). Once found, their benefits to all-pairs applications has been shown in our publications Kleinheksel and Somani (2015c,a).

To enable the use of redundant cyclic quorums and eliminate the need for others to perform an exhaustive search to find the quorum sets, we have included Tables B.1 and B.2. The quorum sets in Table B.1 guarantee a redundancy of 2 and those in Table B.2 guarantee a redundancy of 3.

To speed up the $\binom{N}{k}$ exhaustive search significantly, for some N and k inputs (marked with an asterisk in the tables) we began searching with one greater than the ideal k value. This increases the number of pairs formed for a given combination significantly and helped to arrive at a quorum set solution sometimes days faster, than if beginning from the ideal k size. To justify this heuristic technique consider that an increase from quorum size $k = 14$ to $k = 15$ is less than an 8% increase. Additionally, many of these quorums likely are the optimal redundant cyclic quorum sets given that in Table A.1 of the sets for $N = 75, \ldots, 111$, i.e., 37 quorum sets total, only 9 optimal cyclic quorum sets had the same size as their ideal k value. This means that greater than 75% of the time, the optimal cyclic quorum set size k for these N were one greater than their ideal size any ways. No quorum set redundant or non-redundant was more than one greater than their ideal size though.

المنارة للاستشارات
www.manaraa.com

Table B.1: Redundancy = 2, Cyclic quorums for $N = 4$ to 111

N	$\mathbf{\lvert S_i \rvert = k}$	Redundancy = 2, Cyclic Quorum										
4	3	1	2	3								
5	4	1	2	3	4							
6	4	1	2	3	4							
7	4	1	2	3	5							
8	5	1	2	3	4	5						
9	5	1	2	3	4	6						
10	5	1	2	3	4	7						
11	5	1	2	3	5	8						
12	6	1	2	3	4	5	8					
13	6	1	2	3	4	5	9					
14	6	1	2	3	4	6	10					
15	6	1	2	3	4	7	11					
16	7	1	2	3	4	5	6	11				
17	7	1	2	3	4	5	7	12				
18	7	1	2	3	4	5	8	13				
19	7	1	2	3	4	5	9	14				
20	7	1	2	3	4	7	11	16				
21	7	1	2	3	5	9	12	17				
22	8	1	2	3	4	5	6	10	16			
23	8	1	2	3	4	5	6	11	17			
24	8	1	2	3	4	5	8	11	16			
25	8	1	2	3	4	5	9	14	20			
26	8	1	2	3	4	6	11	15	21			
27	8	1	2	3	4	7	11	17	22			
28	9	1	2	3	4	5	6	9	15	22		
29	9	1	2	3	4	5	6	10	16	23		
30	9	1	2	3	4	5	6	11	17	24		
31	9	1	2	3	4	5	7	13	18	25		
32	9	1	2	3	4	5	8	13	20	26		
33	9	1	2	3	4	6	10	12	17	22		
34	9	1	2	3	5	8	16	18	26	30		
35	10	1	2	3	4	5	6	7	13	20	28	
36	10	1	2	3	4	5	6	8	15	21	29	
37	9	1	2	4	8	18	25	26	30	36		
38	10	1	2	3	4	5	6	13	14	19	20	
39	10	1	2	3	4	5	7	12	14	20	26	
40	10	1	2	3	4	6	10	15	16	23	26	
41	10	1	2	3	4	6	20	22	31	35	37	
42	10	1	2	3	4	9	10	29	30	33	34	
43	11	1	2	3	4	5	6	7	13	15	21	28

المنارة للاستشارات
www.manaraa.com

Table B.1: (Continued)

Redundancy = 2, Cyclic quorums for $N = 4$ to 111

N	$\mathbf{\lvert S_i \rvert = k}$	**Redundancy = 2, Cyclic Quorum**													
44	11	1	2	3	4	5	6	7	15	16	22	23			
45	11	1	2	3	4	5	6	8	14	16	23	30			
46	11	1	2	3	4	5	6	12	13	18	27	31			
47	11	1	2	3	4	5	7	11	16	23	24	34			
48	11	1	2	3	4	5	8	11	25	33	38	40			
49	11	1	2	3	4	6	12	14	18	23	30	37			
50	11	1	2	3	4	7	8	15	24	32	39	42			
51	11	1	2	3	4	6	15	22	30	37	43	47			
52	12	1	2	3	4	5	6	7	12	13	24	27	40		
53	12	1	2	3	4	5	6	7	13	17	24	33	41		
54	12	1	2	3	4	5	6	8	14	16	25	34	41		
55	12	1	2	3	4	5	6	11	12	25	29	41	45		
56	12	1	2	3	4	5	6	12	20	35	36	41	48		
57	12	1	2	3	4	5	7	23	34	36	44	46	53		
58	12	1	2	3	4	5	7	15	29	37	42	44	53		
59	12	1	2	3	4	6	8	20	22	28	29	50	52		
60	12	1	2	3	4	6	16	27	35	43	44	51	57		
61	12	1	2	3	4	7	17	22	31	39	45	52	56		
62	12	1	2	3	4	7	8	17	18	25	26	37	38		
63	12	1	2	3	7	10	11	22	36	42	47	49	52		
64	13	1	2	3	4	5	6	7	14	23	40	41	47	55	
65	13	1	2	3	4	5	6	8	14	34	44	45	52	54	
66	13	1	2	3	4	5	6	10	16	21	28	30	38	51	
67	13	1	2	3	4	5	6	12	32	39	47	51	56	57	
68	13	1	2	3	4	5	7	10	18	29	30	39	53	60	
69	13	1	2	3	4	5	9	21	33	38	49	51	63	64	
70	13	1	2	3	4	5	8	9	24	25	33	34	60	61	
71	13	1	2	3	4	6	13	22	23	28	30	34	36	59	
72	13	1	2	3	5	12	17	20	25	33	39	45	49	66	
73	14	1	2	3	4	5	6	7	10	17	25	35	37	54	63
74	14	1	2	3	4	5	6	7	13	16	21	32	34	39	56
75	14	1	2	3	4	5	6	7	13	21	32	35	42	55	64
76	14	1	2	3	4	5	6	9	10	19	22	33	34	44	59
77	14	1	2	3	4	5	6	9	16	26	32	41	50	59	67
78	14	1	2	3	4	5	6	9	10	27	28	37	38	67	68
79	14	1	2	3	4	5	6	15	25	26	32	33	38	41	66
80	14	1	2	3	4	5	8	17	27	35	45	52	62	68	73
81	14	1	2	3	4	5	9	17	26	36	42	55	62	68	73
82	14	1	2	3	4	7	25	34	38	49	50	64	66	76	78

المنارة للاستشارات
www.manaraa.com

Table B.1: (Continued)

Redundancy = 2, Cyclic quorums for $N = 4$ to 111

N	$\lvert \mathbf{S_i} \rvert = \mathbf{k}$	**Redundancy = 2, Cyclic Quorum**																
83	14	1	2	3	4	7	10	27	34	35	48	56	70	74	75			
84	14	1	2	3	4	8	23	24	29	32	33	40	50	73	74			
*85	*15	1	2	3	4	5	6	7	10	17	20	28	36	48	56	65		
*86	*15	1	2	3	4	5	6	7	10	11	30	31	41	42	74	75		
*87	*15	1	2	3	4	5	6	7	13	20	28	40	49	59	68	77		
*88	*15	1	2	3	4	5	6	10	13	20	27	29	40	42	48	60		
*89	*15	1	2	3	4	5	6	8	17	28	29	36	38	42	46	75		
*90	*15	1	2	3	4	5	6	10	13	16	28	42	45	58	72	75		
*91	*15	1	2	3	4	5	9	10	18	29	42	53	63	65	75	78		
*92	*15	1	2	3	4	5	9	10	20	22	32	40	46	55	69	80		
93	15	1	2	3	4	6	10	16	26	31	48	57	59	65	76	83		
94	15	1	2	3	4	7	11	17	20	31	36	48	56	63	74	75		
*95	*16	1	2	3	4	5	6	7	8	11	20	32	40	51	62	73	83	
*96	*16	1	2	3	4	5	6	7	8	14	22	31	44	54	65	75	85	
*97	*16	1	2	3	4	5	6	7	9	19	25	46	54	65	68	74	91	
*98	*16	1	2	3	4	5	6	7	11	18	19	28	29	39	49	58	68	
*99	*16	1	2	3	4	5	6	7	12	14	25	26	42	59	60	74	75	
*100	*16	1	2	3	4	5	6	7	12	19	33	38	41	58	64	82	91	
*101	*16	1	2	3	4	5	6	10	16	26	34	44	45	54	73	80	90	
*102	*16	1	2	3	4	5	6	14	17	23	41	48	49	55	78	83	88	
*103	*16	1	2	3	4	5	8	10	18	26	37	45	57	63	77	83	96	
*104	*16	1	2	3	4	5	8	13	22	24	37	52	64	68	74	81	82	
*105	*16	1	2	3	4	5	8	13	30	35	51	56	71	76	90	95	100	
*106	*16	1	2	3	4	8	11	19	23	34	36	46	50	59	64	70	88	
*107	*17	1	2	3	4	5	6	7	8	10	31	33	41	43	52	65	95	97
*108	*17	1	2	3	4	5	6	7	8	12	20	21	31	32	43	54	64	75
*109	*17	1	2	3	4	5	6	7	8	13	22	31	45	64	65	78	86	100
*110	*17	1	2	3	4	5	6	7	8	20	27	32	43	52	60	70	81	103
*111	*17	1	2	3	4	5	6	7	11	21	28	29	40	41	61	69	70	101

المنارة للاستشارات

www.manaraa.com

Table B.2: Redundancy = 3, Cyclic quorums for $N = 4$ to 101

N	$\|S_i\| = k$	Redundancy = 3, Cyclic Quorum											
4	4	1	2	3	4								
5	4	1	2	3	4								
6	5	1	2	3	4	5							
7	5	1	2	3	4	5							
8	6	1	2	3	4	5	6						
9	6	1	2	3	4	5	6						
10	6	1	2	3	4	6	7						
11	6	1	2	3	5	6	8						
12	7	1	2	3	4	5	7	8					
13	7	1	2	3	4	5	7	9					
14	7	1	2	3	5	9	10	12					
15	7	1	2	3	5	6	9	11					
16	8	1	2	3	4	5	6	9	11				
17	8	1	2	3	4	5	7	9	12				
18	8	1	2	3	4	6	7	11	13				
19	8	1	2	3	4	6	8	12	15				
20	9	1	2	3	4	5	6	8	11	14			
21	9	1	2	3	4	5	6	8	16	17			
22	9	1	2	3	4	5	7	11	13	18			
23	9	1	2	3	4	5	7	11	14	19			
24	9	1	2	3	4	7	12	15	19	21			
25	10	1	2	3	4	5	6	7	11	13	18		
26	10	1	2	3	4	5	6	8	13	15	21		
27	10	1	2	3	4	5	6	8	13	16	22		
28	10	1	2	3	4	5	8	11	15	19	24		
29	10	1	2	3	4	5	8	14	18	23	26		
30	11	1	2	3	4	5	6	7	9	15	17	24	
31	11	1	2	3	4	5	6	7	9	15	18	25	
32	11	1	2	3	4	5	6	8	13	17	21	27	
33	11	1	2	3	4	5	6	9	13	22	23	28	
34	11	1	2	3	4	5	7	10	12	22	24	28	
35	11	1	2	3	4	5	9	10	13	19	24	26	
36	11	1	2	3	5	7	13	14	15	20	23	34	
37	12	1	2	3	4	5	6	7	10	15	25	26	32
38	12	1	2	3	4	5	6	7	10	20	25	31	32
39	12	1	2	3	4	5	6	7	11	15	26	27	33
40	12	1	2	3	4	5	6	11	12	17	24	28	32
41	12	1	2	3	4	5	8	11	19	21	23	32	37
42	12	1	2	3	4	5	9	15	16	20	25	28	37
43	12	1	2	3	4	6	11	15	18	26	28	34	40

المنارة للاستشارات
www.manaraa.com

Table B.2: (Continued)

Redundancy = 3, Cyclic quorums for $N = 4$ to 101

N	$\|S_i\| = k$	Redundancy = 3, Cyclic Quorum																
44	13	1	2	3	4	5	6	7	11	15	26	32	33	39				
45	13	1	2	3	4	5	6	7	10	18	19	25	29	39				
46	13	1	2	3	4	5	6	8	13	17	23	25	31	38				
47	13	1	2	3	4	5	6	9	16	25	26	34	37	43				
48	13	1	2	3	4	5	8	9	15	19	20	28	29	37				
49	13	1	2	3	4	5	8	14	16	27	32	36	37	44				
50	14	1	2	3	4	5	6	7	8	13	16	23	26	32	40			
51	14	1	2	3	4	5	6	7	8	15	16	24	31	35	41			
52	14	1	2	3	4	5	6	7	9	15	19	26	28	35	43			
53	14	1	2	3	4	5	6	7	10	16	24	31	36	41	47			
54	14	1	2	3	4	5	6	7	13	20	23	28	34	43	48			
55	14	1	2	3	4	5	6	8	15	21	26	32	40	47	48			
56	14	1	2	3	4	5	7	14	19	21	27	31	36	42	50			
57	14	1	2	3	4	5	8	11	15	20	23	28	31	37	47			
58	14	1	2	3	5	6	19	24	25	35	39	46	51	52	54			
59	15	1	2	3	4	5	6	7	8	13	20	21	30	38	40	51		
60	15	1	2	3	4	5	6	7	9	13	20	31	40	41	49	50		
61	15	1	2	3	4	5	6	7	11	17	23	31	32	43	50	55		
62	15	1	2	3	4	5	6	7	12	24	26	33	39	49	52	57		
63	15	1	2	3	4	5	6	8	13	21	26	31	37	43	52	57		
64	15	1	2	3	4	5	6	10	14	18	24	25	31	36	41	50		
65	15	1	2	3	4	5	8	11	17	29	34	38	45	48	53	56		
66	15	1	2	3	5	6	8	13	19	28	34	38	47	48	55	56		
*67	*16	1	2	3	4	5	6	7	8	11	22	24	32	36	47	56	62	
*68	*16	1	2	3	4	5	6	7	8	12	22	27	33	40	49	57	61	
*69	*16	1	2	3	4	5	6	7	8	15	23	24	34	41	47	49	61	
*70	*16	1	2	3	4	5	6	7	12	15	20	32	38	47	52	54	61	
*71	*16	1	2	3	4	5	6	7	11	17	18	20	26	38	48	49	56	
72	16	1	2	3	4	5	6	8	11	17	26	34	35	42	53	61	63	
73	16	1	2	3	4	5	6	11	14	19	31	38	45	54	56	60	66	
74	16	1	2	3	4	5	9	20	31	38	43	46	52	53	58	62	66	
75	16	1	2	3	4	7	8	23	25	28	37	41	51	59	66	68	70	
*76	*17	1	2	3	4	5	6	7	8	9	17	27	36	44	48	55	62	68
*77	*17	1	2	3	4	5	6	7	8	12	19	28	30	40	48	51	65	70
*78	*17	1	2	3	4	5	6	7	8	13	16	26	40	48	52	56	63	65
*79	*17	1	2	3	4	5	6	7	10	13	21	35	42	46	55	58	65	68
*80	*17	1	2	3	4	5	6	7	13	16	34	41	44	50	58	63	66	70
*81	*17	1	2	3	4	5	6	10	16	29	36	46	47	54	62	63	68	74
82	17	1	2	3	4	5	6	11	12	20	24	30	40	47	55	60	61	72

المنارة للاستشارات
www.manaraa.com

Table B.2: (Continued)

Redundancy = 3, Cyclic quorums for $N = 4$ to 101

N	$\mathbf{\lvert S_i \rvert = k}$	**Redundancy = 3, Cyclic Quorum**																		
83	17	1	2	3	4	5	8	13	17	23	24	30	36	41	44	54	62	71		
*84	*18	1	2	3	4	5	6	7	8	9	14	17	28	43	52	56	60	68	70	
*85	*18	1	2	3	4	5	6	7	8	9	16	17	26	27	28	43	45	56	59	
*86	*18	1	2	3	4	5	6	7	8	10	17	27	37	38	46	50	64	72	75	
*87	*18	1	2	3	4	5	6	7	8	12	16	23	34	42	51	58	65	70	78	
*88	*18	1	2	3	4	5	6	7	8	15	23	24	35	42	50	56	66	67	80	
*89	*18	1	2	3	4	5	6	7	12	13	14	35	36	37	49	50	74	75	76	
*90	*18	1	2	3	4	5	6	7	13	15	22	28	30	35	40	51	54	61	75	
*91	*18	1	2	3	4	5	6	8	13	20	30	39	43	48	61	64	72	78	84	
*92	*19	1	2	3	4	5	6	7	8	9	10	17	26	29	37	47	54	60	69	83
*93	*19	1	2	3	4	5	6	7	8	9	11	19	23	35	51	60	62	71	73	81
*94	*19	1	2	3	4	5	6	7	8	9	12	21	24	34	48	57	60	68	71	81
*95	*19	1	2	3	4	5	6	7	8	9	15	23	24	36	37	46	47	48	73	84
*96	*19	1	2	3	4	5	6	7	8	11	23	27	35	38	44	49	59	72	84	90
*97	*19	1	2	3	4	5	6	7	8	11	16	28	31	35	47	53	64	69	85	90
*98	*19	1	2	3	4	5	6	7	9	16	18	31	33	40	41	47	51	59	80	84
*99	*19	1	2	3	4	5	6	7	8	16	23	24	32	45	51	57	65	68	75	91
*100	*19	1	2	3	4	5	6	7	12	23	28	37	42	46	51	75	80	88	92	95
*101	*19	1	2	3	4	5	6	7	11	12	13	34	35	36	47	48	49	86	87	88

المنارة للاستشارات
www.manaraa.com