UNIVERSITY OF
LEADING
THE WAY
WESTMINSTER

WestminsterResearch
http://www.westminster.ac.uk/research/westminsterresearch

Efficient replication of large volumes of data and maintaining data consistency by using P2P techniques in Desktop Grid

Bharat Gupta

Faculty of Science and Technology

This is an electronic version of a PhD thesis awarded by the University of Westminster. © The Author, 2014.

This is an exact reproduction of the paper copy held by the University of Westminster library.

The WestminsterResearch online digital archive at the University of Westminster aims to make the research output of the University available to a wider audience. Copyright and Moral Rights remain with the authors and/or copyright owners.
Users are permitted to download and/or print one copy for non-commercial private study or research. Further distribution and any use of material from within this archive for profit-making enterprises or for commercial gain is strictly forbidden.

Whilst further distribution of specific materials from within this archive is forbidden, you may freely distribute the URL of WestminsterResearch: (http://westminsterresearch.wmin.ac.uk/).

In case of abuse or copyright appearing without permission e-mail repository@westminster.ac.uk

Efficient replication of large volumes of data and maintaining data consistency by using P2P techniques in Desktop Grid

Bharat Gupta

A thesis submitted in partial fulfilment of the
requirements of the University of Westminster
for the degree of Doctor of Philosophy

September 2014

Abstract

Desktop Grid is increasing in popularity because of relatively very low cost and good performance in institutions. Data-intensive applications require data management in scientific experiments conducted by researchers and scientists in Desktop Grid-based Distributed Computing Infrastructure (DCI). Some of these data-intensive applications deal with large volumes of data. Several solutions for data-intensive applications have been proposed for Desktop Grid (DG) but they are not efficient in handling large volumes of data. Data management in this environment deals with data access and integration, maintaining basic properties of databases, architecture for querying data, etc. Data in data-intensive applications has to be replicated in multiple nodes for improving data availability and reducing response time. Peer-to-Peer (P2P) is a well established technique for handling large volumes of data and is widely used on the internet. Its environment is similar to the environment of DG. The performance of existing P2P-based solution dealing with generic architecture for replicating large volumes of data is not efficient in DG-based DCI. Therefore, there is a need for a generic architecture for replicating large volumes of data efficiently by using P2P in BOINC based Desktop Grid.

Present solutions for data-intensive applications mainly deal with read only data. New type of applications are emerging which deal large volumes of data and Read/Write of data. In emerging scientific experiments, some nodes of DG generate new snapshot of scientific data after regular intervals. This new snapshot of data is generated by updating some of the values of existing data fields. This updated data has to be synchronised in all DG nodes for maintaining data consistency. The performance of data management in DG can be improved by addressing efficient data replication and consistency. Therefore, there is need for algorithms which deal with data Read/Write consistency along with replication for large volumes of data in BOINC based Desktop Grid.

The research is to identify efficient solutions for data replication in handling large volumes of data and maintaining Read/Write data consistency using Peer-to-Peer techniques in BOINC based Desktop Grid. This thesis presents the solutions that have been carried out to complete the research.

List of Contents

List of Figures.....vii
List of Tables.....ix
Acknowledgements.....xi
Author's declaration.....xii
Abbreviations..... xiii

1. Introduction..... 1
1.1. Introduction..... 1
1.2. Research Overview..... 2
1.2.1. Data management Concepts and Challenges.....2
1.2.2. Research Objectives..... 3
1.3. Research Deliverables..... 4
1.3.1 Research Contributions.....4
1.4. Thesis Structure..... 4
1.5. Conclusion..... 6

2. Background Research and Related Work..... 7
2.1. Introduction..... 7
2.2. Data Management in Grids and Clouds..... 7
2.2.1. Service Grids..... 7
2.2.2. Desktop Grid Computing..... 10
2.2.3. Cloud Computing..... 11
2.3. Desktop Grid Computing Technologies..... 13
2.4. Existing DG Architecture at University of Westminster..... 15
2.5. Emerging case scenarios..... 16
2.6. Existing Data solutions in DG-based DCI 17
2.7. Applicability of Read/Write of data in Desktop Grid..... 20
2.8. General data solutions used for handling large volumes of data..... 23

2.9. Conclusion..... 24

3. Analysis and Design of Data access in DG-based DCI..... 25

3.1. Requirements for new data management solution..... 25

3.2. Analysis of data management in P2P-based solution for Desktop Grid Computing..... 25

3.2.1. P2P Architecture..... 26

3.2.2. Related Work in Data Replication and Consistency..... 27

3.2.3. Concurrency Control mechanisms..... 29

3.3. Finite State Model for new data management solution..... 31

3.4. Sequence diagrams for new data management solution..... 37

3.5. Conclusion..... 59

4. A Peer-to-Peer Architecture for handling large volumes of data in DG-based DCI.. 60

4.1. Proposed Desktop Grid Architecture..... 60

4.1.1. Coordinator architecture..... 61

4.1.2. Client architecture..... 63

4.1.3. Comparison of existing and proposed architecture..... 64

4.1.4. Suitability of the proposed architecture for R/W of data..... 65

4.1.5. Algorithms used in proposed architecture..... 68

4.2. Data Consistency Algorithms..... 69

4.2.1. Conflict Resolving Algorithm..... 69

4.2.2. Consistency Algorithm..... 71

4.3. Data Replication Algorithms..... 75

4.3.1. Replication performance measurement Algorithm..... 75

4.3.2. Replication performance improvement Algorithm..... 76

4.4. Relationship between Algorithms..... 81

4.5. Proposed Research Contributions..... 82

4.6. Conclusion..... 83

5. Experimental Testbed and Simulation Design..... 84

5.1. Experimental Testbed..... 84

5.1.1. Overview..... 84
5.1.2. Infrastructure..... 85
5.1.3. System Constraints..... 86
5.2. Simulation Design..... 87
5.2.1. Overview..... 87
5.2.2. Construction of Experiments..... 88
5.2.3. Parameters considered for Test Data..... 89
5.2.4. Structure of Test Data..... 90
5.2.5. Experiment Simulation..... 92
5.2.6. Representation of Results..... 93
5.2.7. Comparison Strategy..... 95
5.3. Conclusions..... 96

6. Experiments Results..... 97
6.1. Introduction..... 97
6.2. Experiments Results..... 97
6.2.1. Overview..... 97
6.2.2. Data Consistency Experiments.....100
6.2.2.1. Overview.....100
6.2.2.2. Scenario and Test Cases.....100
6.2.2.3. Scenario Analysis..... 106
6.2.3. Data Replication Experiments..... 106
6.2.3.1. Overview..... 106
6.2.3.2. Scenario and Test Cases..... 107
6.2.3.2.1. Scenario DR1: Replication Performance Measurement.....107
6.2.3.2.1.1.DR1 Scenario Analysis.....122
6.2.3.2.2. Scenario DR2: Replication Planning Strategies.....123
6.2.3.2.2.1.Data replication Scenario DR2 analysis.....132
6.2.4. P2P-based Architecture Experiments..... 132
6.2.4.1. Overview..... 132
6.2.4.2. Scenario..... 133

6.2.4.3. Test Cases..... 133
6.2.4.4. Scenario Analysis..... 140
6.3. Post-mortem Analysis..... 140
6.4. Conclusion..... 141

7. Conclusions..... 142
7.1. Summary..... 142
7.2. Knowledge Contributions..... 143
7.3. Future work..... 145

Appendix.....146
Bibliography..... 154

List of Figures

2.1 University of Westminster's DG architecture..... 16
2.2 Attic Workflow..... 19
3.1 Topology of the P2P systems..... 26
3.2: Proposed DG Architecture.....32
3.3 Finite state model for coordinator..... 33
3.4 Finite state model for client..... 35
3.5 Sequence diagram for case scenario 1..... 39
3.6 Sequence diagram for case scenario 2..... 41
3.7 Sequence diagram for case scenario 3..... 43
3.8 Sequence diagram for case scenario 4..... 45
3.9 Sequence diagram for case scenario 5..... 48
3.10 Sequence diagram for case scenario 6..... 51
3.11 Sequence diagram for case scenario 7.1..... 54
3.12 Sequence diagram for case scenario 7.2.....56
3.13 Sequence diagram for case scenario 7.3..... 58
4.1 Proposed detailed DG architecture..... 61
4.2 Proposed P2P Coordinator architecture..... 62
4.3 Proposed P2P Client Architecture..... 64
4.4 Suitability of proposed architecture in emerging applications..... 66
4.5 Relative order of execution of algorithms..... 81
4.6 Dependency diagram between four algorithms..... 82
5.1 SimGrid architecture..... 86
5.2. Simulation process.....87
5.3 Proposed DG architecture..... 88
5.4 Simulation Architecture..... 88
5.5 Generic architecture for an experiment..... 92
5.6 Graph of execution time for varying data size..... 95
6.1 Architecture diagram for scenario DC1.....102

6.2 Execution trend for scenario DC1.....103
6.3 Architecture diagram for scenario DC2.....105
6.4 Execution trend for scenario DC2.....106
6.5 Architecture diagram for scenario DR1..... 108
6.6 Execution trend for test case DR1TC1.....110
6.7 Execution trend for test case DR1TC2.....112
6.8 Execution trend for test case DR1TC3..... 113
6.9 Execution trend for test case DR1TC4..... 114
6.10 Execution trend for test case DR1TC5..... 116
6.11 Execution trend for test case DR1TC6..... 117
6.12 Execution trend for test case DR1TC7..... 119
6.13 Execution trend for test case DR1TC8..... 120
6.14 Execution trend for test case DR1TC9..... 122
6.15 Architecture diagram for DR2TC1 test case..... 124
6.16 Architecture diagram for DR2TC2 test case..... 127
6.17 Spanning tree for replication for DR2TC2E1 execution..... 128
6.18 Spanning tree for replication for DR2TC2E2 execution..... 129
6.19 Architecture diagram for DR2TC3 test case..... 130
6.20 Spanning tree path generated by DR2TC3E1 execution..... 131
6.21 Spanning tree path generated by DR2TC3E2 execution..... 132
6.22 Comparative analysis of test cases PDG1TC1 and EDG1TC1..... 136
6.23 Comparative analysis of execution time for test cases PDG2TC1 and EDG2TC1..... 139
6.24 Comparative analysis of message exchanged for test cases PDG2TC1 and EDG2TC1.... 140

List of Tables

2.1 Technology comparison matrix..... 15
3.1 Comparative analysis of the data replication research..... 28
3.2 State transition table for FSM of coordinator..... 34
3.3 State transition table for FSM of a client..... 36
3.4 Execution order of case scenario 1..... 40
3.5 Execution order of case scenario 2..... 42
3.6 Execution order of case scenario 3..... 44
3.7 Execution order of case scenario 4..... 47
3.8 Execution order of case scenario 5..... 50
3.9 Execution order of case scenario 6..... 53
3.10 Execution order of case scenario 7.1..... 55
3.11 Execution order of case scenario 7.2..... 57
3.12 Execution order of case scenario 7.3..... 59
4.1 Comparison of the components present in the proposed architecture..... 64
4.2 Comparison of Attic and proposed architecture..... 65
5.1 Comparison of existing simulators..... 85
6.1 Classification of experiments performed..... 98
6.2 Parameters value considered for the experimentation..... 99
6.3 Data consistency experiments scenarios.....100
6.4 Scenario DC1 test cases.....101
6.5 DC1 test case executions..... 102
6.6 Scenario DC2 test cases.....104
6.7 Test case DC2 executions.....105
6.8 Data replication experiments scenarios..... 107
6.9 DR1 Scenario test cases..... 109
6.10 Test case DR1TC1 executions..... 110
6.11 Test case DR1TC2 executions..... 111
6.12 Test case DR1TC3 executions..... 113

6.13 Test case DR1TC4 executions..... 114
6.14 Test case DR1TC5 executions..... 115
6.15 Test case DR1TC6 executions..... 117
6.16 Test case DR1TC7 executions..... 118
6.17 Test case DR1TC8 executions..... 120
6.18 Test case DR1TC9 executions..... 121
6.19 Ranking of experimentation input parameters..... 123
6.20 Scenario DR2 test cases..... 124
6.21 DR2TC1 Test case executions..... 125
6.22 Adjacency data matrix format of 5 nodes for Algorithm-4..... 125
6.23 Frequent access paths generated for DR2TC1 test case executions..... 126
6.24 DR2TC2 Test case executions..... 127
6.25 DR2TC2 Test case executions outcomes..... 127
6.26 DR2TC3 Test case executions..... 130
6.27 Test case DR2TC3 executions outcomes.....131
6.28 Comparison of proposed and existing architecture experiments scenarios..... 133
6.29 PDG1 scenario test case..... 134
6.30 Test case PDG1TC1 executions..... 134
6.31 EDG1 scenario test cases..... 135
6.32 Test case EDG1TC1 executions..... 135
6.33 PDG2 scenario test case..... 137
6.34 Test case PDG2TC1 executions..... 137
6.35 EDG2 scenario test case..... 137
6.36 Test case EDG2TC1 executions..... 138

Acknowledgements

I wish to express heartfelt gratitude towards my research supervisors: Prof. Stephen Winter, Prof. Gabor Terstyanszky and Prof. Peter Kacsuk for their encouragement, guidance and constant support. Their intellectual advices had always upgraded me academically & their physical presence had been mentoring my way of life. The challenges offered by them helped me to strive more & finally led towards the completion of this research work.

I would like to thank the scholarship committee of University of Westminster for granting me the scholarship that made it possible for me to undergo the doctoral study.

I would like to thank my friends, colleagues and CPC staff for supporting me throughout my research work.

Last but not the least my special thanks goes to my caring parents, adorable wife Namita & my lovely kids Kashish & Ishaan who permitted me to let go abroad for four long years in-order to accomplish my doctoral studies.

Lastly, I appreciate the support from my family to complete my research. Their encouragement helped me to achieve this success.

Bharat Gupta
London, United Kingdom,
September, 2014

Author declaration

I declare that all the material contained in this thesis is my own work. The work has not been submitted for any other degree or award at this university or at other place.

Bharat Gupta

Abbreviations

2PC	Two Phase Protocol
3G	Generic Grid-Grid Bridge
ADICS	A Data Intensive Cycle Sharing
CC	Cloud computing
DCI	Distributed Computing Infrastructure
DDBJ	DNA Data Bank of Japan
DG	Desktop Grid
EDGE	Enabling Desktop Grid for e-science
EDGI	European DG Initiative
ENA	European Nucleotide Archive
GRASS	Grid application Supports Service
gUSE	Grid User Support Environment
HDFS	Hadoop Distributed File System
IDGF	International Desktop Grid Federation
NTFS	New Technology File System
P2P	Peer-to-Peer
RC	Research Contribution
SG	Service Grid
WU	Work Unit

Chapter 1

Introduction

1.1. Introduction

A Distributed Computing Infrastructure (DCI) is a collection of heterogeneous computational resources. Examples of DCI are Service Grids, Desktop Grids, Clouds, Clusters, etc. A Service Grid [62] is a hardware and software infrastructure that provided dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. Service Grid (SG) provides users access to resources, abstracts the services of multiple providers along with high performance. A Desktop Grid (DG) [5] refers to the aggregation of heterogeneous, dynamic, volatile, non-dedicated, de-centralized, commodity personal computers (PCs) connected through a network and running (mostly) Microsoft Windows operating system. DG is increasing in popularity because of relatively very low cost as compared to SGs. It provides high performance when most of the resources are idle. Cloud Computing [54] is defined as a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". It offers more resources whenever there is a need of extra resources for computation.

As the commodity hardware underlying DG-based DCI gets cheaper and more powerful, managing sustainable DCI-based services in a cost effective way is increasingly a key challenge. DG-based DCI are mainly focusing on compute-intense applications while data management is a secondary concern. DG-based DCI provides computational requirements to research scientists where data-intensive applications will require data management. This is being achieved by expressing the requirements of scientists using complex, highly concurrent automated workflows running in DCI.

Some of the data-intensive experiments which includes advanced simulations and experimental analysis such as High-Energy Physics, astronomy, drug discovery, climate modelling requires processing of large volumes of data in terms of Terabytes or even more in Petabytes. These requirements are not only focusing on computation but also on data management aspects. The increased demand of data-intensive

applications from e-scientists has results in the form of new challenges for data management in DG-based DCI.

Some of the emerging applications [40] [41] [42] [43] require new features like Read/Write of data for maintaining data consistency between different nodes. This type of feature is not currently supported [18] by the existing data solutions in DG environment. The emerging applications deal with large volume of data [44]. There is a need to address data management in dealing with large volumes of data and maintaining R/W data consistency.

The aim of the research is to identify optimal architecture which handles large volumes of data efficiently and maintains data consistency due to Read/Write of data.

The purpose of this chapter is to introduce the framework for the research. It consists of different sections: research overview which includes data management concepts and research objectives, research deliverables, and structure of the thesis. The research overview section describes the objectives of this research. The data management concepts and challenges describe the key terms used throughout this thesis and consecutively describe the research challenges. The research deliverables section lists the proposed research contributions of the research. The thesis structure section describes the contents of this thesis.

1.2. Research Overview

DG-based DCI environment is volatile, dynamic and heterogeneous in nature. Handling large volumes of data in reliable way in such environment is very difficult to achieve as compared to the other stable media such as Clusters, SGs. In addition, recently there has been a growth in the use of DG-based DCI.

1.2.1. Data management Concepts and Challenges

Applications deal with large volumes of data in scientific experiments conducted by researchers and scientists in DG-based Distributed Computing Infrastructure. Several solutions [33] [34] [35] [36] for data-intensive applications have been proposed for DG-based DCI but they are not efficient in handling large volumes of data. Data management in this environment deals with data access and integration, maintaining basic properties of databases, architecture for querying data, etc. Data in data-intensive applications in DG-based DCI has to be replicated in multiple nodes for improving data availability and

reducing response time. Peer-to-Peer (P2P) is a well established technique for handling large volumes of data due to its data distribution policy. It distributed large volumes of data efficiently to large number of nodes simultaneously without causing bottleneck in the network. It is widely used on the internet. Its environment is similar to the environment of DG-based DCI. The performance of existing P2P-based solution [16] dealing with generic architecture for replicating large volumes of data is not efficient due to issues like data transfer only between coordinator and nodes, support of R/W data, etc. Therefore, there is a need for a generic architecture for replicating large volumes of data efficiently by using P2P in DG-based DCI.

The performance of applications dealing with large volumes of data can be improved by addressing efficient data replication and also by maintaining the consistency of data. Data consistency deals with a consistent view of data for each node when the data is updated by one of the nodes. Replication and consistency are implicitly related to each other. In the case of read only data, only replication is considered while in the case of data update, consistency is also considered. Replication is directly dependent upon maintaining consistency for updated data. Data consistency depends upon the synchronisation of the new/updated data within different nodes of DG-based DCI.

Present solutions for data-intensive applications [16] [37] [64] [72] mainly deal with read only data in DG-based DCI. In emerging scientific experiments [40], some nodes of DG generate new snapshot of scientific data after regular intervals. This new snapshot of data is generated by updating some of the values of existing data fields. This updated data has to be synchronised in all DG nodes for maintaining data consistency. So, there is a need for algorithms which deal with data Read/Write consistency along with replication for large volumes of data in DG-based DCI.

The research is to identify efficient solutions for data replication and consistency for large volumes of data by using Peer-to-Peer techniques in the DG-based DCI.

1.2.2. Research Objectives

The primary objective of the research is to introduce a novel architecture for handling large volumes of data in DG-based DCI. Existing data solutions [33] [34] [35] [36] do not have efficient performance in handling large volumes of data in DG-based DCI. Existing DG-based data solution [16] for handling large volumes of data is based on P2P techniques. Peer-to-Peer [57] systems have been used in the past for dealing with large

volumes of data efficiently in a large geographically distributed environment. P2P-based architecture is a promising solution for handling large volumes of data in DG-based DCI. So, a novel architecture should be developed by using P2P-based approach for handling large volumes of data in DG-based DCI.

The second objective in support of primary objective is to develop new algorithms to improve data replication performance in heterogeneous, dynamic and volatile environment of DG-based DCI.

The third objective in support of primary objective is to develop new concurrency control techniques for handling R/W data consistency due to multiple clients in DG based DCI. Distributed computing environment consists of large number of nodes which may modify the same data simultaneously.

The research, therefore, focuses on investigating and improving the solutions in future for the above mentioned three broad research objectives in finding efficient data replication and consistency strategies for handling large volumes of data in DG-based DCI.

1.3. Research Deliverables

1.3.1. Research Contributions

The proposed research contributions for the research in handling large volumes and maintaining data consistency in DG-based DCI are as follows:

RC1: This proposed research contribution is based on primary objective. It deals with identifying an optimal P2P-based architecture for handling large volume of data in DG-based DCI.

RC2: This proposed research contribution is based on secondary objective. It deals with first identifying and then developing optimal algorithms for handling data replication for the proposed P2P-based architecture.

RC3: The proposed research contribution is based on third objective. It deals with first identifying and then developing concurrency control techniques for the proposed P2P-based architecture.

1.4. Thesis Structure

The remaining of the thesis is organized as follows:

Chapter 2: Background Research and Related Work

This chapter deals with the background of the current research and solutions for the data management in DG-based DCI. It also discusses the suitability of general data solutions used for handling large volumes of data. It also discusses the research carried out in order to maintain data consistency in general P2P-based solutions.

Chapter 3: Analysis and Design of Data access in DG-based DCI

This chapter deals with analysis of the problem, its requirements and limitations. It deals with the requirements for the emerging applications use cases, its finite state model and sequence diagrams for the data management in DG-based DCI.

Chapter 4: Peer-to-Peer Architecture for handling large volumes of data in DG-based DCI This chapter deals with proposed architecture for handling large volumes of data in DG-based DCI. The working and components of the P2P coordinator and P2P clients is also discussed in this chapter. It also deals with the working of the proposed algorithms for handling large volumes in the proposed architecture. It then deals with the list of the research contributions made for the research.

Chapter 5: Experimental Testbed and Simulation Design

It discusses experimental testbed, infrastructure and its system constraints used for performing experiments. It also discusses the simulation design used for the conducting the experiments.

Chapter 6: Experiments Results

This chapter deals with a set of experiments to validate the research contributions for the proposed P2P-based architecture as discussed in chapter 4. The experiments outcomes provide the comparative performance analysis of the proposed and existing architecture.

Chapter 7: Conclusions

This chapter summarises and reflects on the research activities, highlights the contributions to knowledge produced, and its impact in the DG-based DCI. It also suggests additional research that can be conducted for the future work.

1.5. Conclusion

The objective of this chapter is to introduce the research framework. The primary and secondary objectives of the research have been identified in this chapter. The primary objective of the research is to identify optimal architecture for handling large volumes of data in DG-based DCI. The secondary objectives of the research are the development of algorithms and concurrency techniques for dealing with R/W of data.

Chapter 2

Background Research and Related Work

2.1. Introduction

The purpose of this chapter is to identify an optimal architecture for handling large volumes of data in DG-based DCI. The data solutions in Grid and Clouds are explored to find an optimal solution for DG-based DCI. The general database solutions are also explored to find an optimal solution. Rest of the chapter consists of 9 sections. Section 2.2 deals with existing data management solutions used in Grids and Clouds, section 2.3 deals with current Desktop Grid computing technologies, section 2.4 deals with existing DG architecture at University of Westminster, section 2.5 deals with new emerging case scenarios dealing with R/W of data, section 2.6 deals with existing data management solutions in Desktop Grid computing, section 2.7 deals with applicability of Read/Write of data due to emerging applications in Desktop Grid, section 2.8 deals with general data solutions used for handling large volumes of data, and section 2.9 concludes it.

2.2. Data Management in Grids and Clouds

This section deals with study of existing data management solutions used in SG, DG and cloud-based DCI.

2.2.1 Service Grids

The term "Grid" was coined in the mid 1990's to describe a collection of hardware and software infrastructure that provides access to high-end computational resources [7] [62]. Grid projects for e-scientists started in 1996, and resulted in many national and international grids. These are now known as Service Grids [63]. Service Grid (SG) in this context is a collection of distributed, generally dedicated clusters of fixed dimensions and location.

Computational scientific workflows allow the scientists to specify, and through workflow tools, execute large-scale complex e-science applications in an automated manner. Data access is a potential performance bottleneck. Performance degradation in data access and integration is due to factors that include driver interfaces, data coupling, client execution, execution control, protocol used, interface restrictions, file formats (structured, semi-structured, un-structured), data access (static, semi-static, dynamic), data transformation strategies, data staging area, etc.

Within the grid environment a number of data handling standards have evolved. PGI [76] focuses on the interoperation of the two most widely used file storage systems, SRM (Storage Resource Manager) [77] and SRB (Storage Resource Broker) [78]. SRM is a protocol for Grid access to mass storage systems while SRB is a solution for data management, including file movement, file replication and metadata management. SRB is widely utilized in Globus toolkit based Grids [79] such as the US TeraGrid [80] or the UK NGS [81].

Current grid computing environments provide secure access to remote data resources which are stored as flat-file data, relational data, etc. A Data Grid focuses on optimisation of data provisioning in a geographically distributed locations. Presently, Service Grids mainly focus on the computational perspective and there is not much concentration on the optimization of the data provisioning within the grid. E-scientists process large amount of data so they need a mechanism for dealing with the services of database management software in the grid. The enormous data in grids is complex to handle, and storage and analysis also becomes costly. Intra-workflow interoperation of grid data resources is one of the key areas of generic interoperation. Although some solutions exist for data access such as OGSA-DAI [64], GRelC [65] in service grid, the performance is not good.

OGSA-DAI [64] is a solution for distributed data access and management. It allows data resources e.g. relational, XML, files to be accessed via web services on the web or within grids. GRelC (Grid Relational Catalogue Project) [65] [66] [67] [68] is based on the Globus Toolkit and provides access to both relational and non-relational data resources. The AMGA metadata catalogue [69] was designed to provide access to metadata for files stored on the grid. It also provides simplified access to relational databases. Spitfire [75] provides access for grid applications to the relational databases for simple query requests. G-DSE (Grid Data Source Engine) [70, 71] has added the query manager as a new component for the query purpose in the gLite. The application interfaces of OGSA-DAI and GRelC are general but in the case of AMGA metadata catalogue, Spitfire and G-DSE are restricted, and specific to SQL. Mobius [64] software provides a set of tools and services to facilitate the management and sharing of the data and metadata in a grid environment. It complements the functionality provided by OGSA-DAI.

In OGSA-DAI [34], workflows are submitted by clients to OGSA-DAI web services. The user can query, integrate, update, or transform the data from different data resources. OGSA-DQP (Distributed Query Processing) is a component of OGSA-DAI that enables distributed queries over relational data resources exposed by OGSA-DAI servers. DQP allows the tables from multiple distributed relational databases to be queried using SQL.

Presently, the limitations that affect the performance [83] [86] of OGSA-DQP are mentioned below:

- The performance is degraded due to not pushing many SQL operations to the database.
- The excessive use of the memory is due to join implementations which store the data from one side of the join in memory.
- Not all complex nested queries can be handled.
- The existing optimizers are essentially heuristic and do not make use of a cost model as in the current commercial databases.

Kukla [84] has investigated how the P-GRADE portal [88] can be extended with data access and manipulation capabilities via OGSA-DAI. OGSA-DAI portlets have been connected to the P-GRADE portal that provides a graphical user interface for database browsing and manipulation capabilities. An improvement in the performance of data transfer has been reported by using the csv (comma separated value) file format as compared to WebRowSet [89] format used by OGSA-DAI for the larger queries.

Kiss et al. [85] describe the generic requirements for the interoperation of Grid data resources within computational workflows. OGSA-DAI has been used for getting the data from relational or XML databases for the computational workflows and combining these with more traditional file storage systems. The interoperation of grid data resources via workflow level integration has been achieved. The performance of data access relative to the current commercial databases was not reported.

Wang et al. [86] have done a performance analysis of the OGSA-DAI 3.0 software. They found that software performance is degraded in handling concurrent clients above a point in the tested environment. Above this point, the round trip time cost, the CPU and memory occupancy increases abruptly.

Xiang [87] has investigated running OGSA-DQP queries against Oracle and SQL server to access massive data of several in TBytes in magnitude. The query performance was poor on large queries due to fetching of all columns of tables. The client memory becomes exhausted when large numbers of rows are returned. These are some of the reasons reported for the performance degradation in the OGSA-DQP software.

OGSA-DAI in SG-based DCI has degraded performance due to factors like fetching of all columns of tables instead of specific one for larger queries, not doing automatic data conversions for the integration, not able to handle concurrent execution of jobs above a critical point, etc. Also structure of SG-based DCI is static in nature. It is not able to handle massive data operations efficiently [87]. So, it is not desirable to adapt the existing solution like OGSA-DAI for data management in DG-based DCI. There is a need of

addressing the above mentioned degraded performance for new data management solutions in DG-based DCI.

2.2.2 Desktop Grid Computing

A Desktop Grid Computing is composed of individual computers that join together to provide an aggregate computing resource. It is a collection of heterogeneous, dynamic, volatile, non-dedicated, personal computers connected through a network. A Desktop Grid [109] in an organisation uses its existing desktop computers to handle its long running computational tasks. DG's are increasing in popularity because of relatively very low cost and good performance in organisations/institutions. Desktop Grid computing addresses the potential of harvesting the idle computing resources of desktop PCs. It provides good performance when most of the resources are idle i.e. not utilised by users. But DG has some limitations due to finite number of resources. Some of the advantages of the DG are utilization of existing resources, low cost and maintenance, non-dedicated distributed systems, etc.

Urbah et al. [90] have connected the EGEE (Enabling Grids for E-science) service grids to the BOINC and XtremWeb Desktop Grids by using 3G Bridge (Generic Grid to Grid). By using 3G Bridge, different jobs can be submitted, scheduled and executed across the SG and DG-based DCI.

Kacsuk et al. [17] in the SZTAKI Desktop grid has improved the features of existing DG's by providing a flexible, versatile and scalable interconnection of different BOINC projects and execution of parameter sweep applications from a generic, high level user interface. The University of Westminster Desktop Grid [91] which is based on SZTAKI solution has been running successfully for the last few years.

The BitDew [72] framework provides a programmable environment for data management and distribution services in Desktop Grids by using multi-protocols file transfers. A 3-layered architecture has been proposed which includes API, Services and Back-ends as layers. For the data distribution, a range of different protocols like Bit-torrent, HTTP, FTP, etc have been used. The architecture is dependent upon a specific set of metadata for the data management operations. The programmer has to tag each data before it can be used for the data management/transfer purpose. A single centralized server has been used for all data services. The remote data storage has been provided between DG and Amazon S3 storage services [73].

Some existing solutions like Attics [16] are only focusing on data management in DG-based DCI. These solutions also have not efficiently addressed handling of large volumes

of data in DG-based DCI. Next section deals with studying data solutions in clouds in order to find the potential solution for the research.

2.2.3 Cloud Computing

NIST defines Cloud Computing [54] as: "a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". Some of the advantages of cloud compared to SG are elastic infrastructure, reduced cost, increased storage, etc.

The analyst firm Gartner [92] has predicted that Cloud Computing (CC) will be the top IT area in the coming years. The UK government is building the G-Cloud [93], which will be a huge private cloud for public sector organizations with the aim of cutting millions of pounds from state IT spending. By using the cloud computing, the resources are not only optimized but costs are saved, giving an opportunity for the small organizations to exploit the benefits cheaply. In today's world, the smaller organization can also use the required resources, computation, services, etc external to their organization by using cloud computing. Cloud computing is rapidly emerging as an alternate to the existing systems.

The essential features of the cloud are on-demand self-service, rapid elasticity, resource pooling, broad network access, measured service. By elasticity, we mean that a service can expand and contract on demand. The service models for the cloud can be classified as Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS).

In SaaS, the consumer has the capacity to use the provider's applications running on a cloud infrastructure. The applications can be accessed through a thin client interface such as a web browser. Some the examples of SaaS are Saleforce CRM, Google Mail. In PaaS, the consumer has the capacity to deploy its own business application or process on to the cloud infrastructure by using the programming languages and tools supported by the provider. Some examples of PaaS are Windows Azure [94], Google Application Engine [95], IBM Websphere Cloudburst [96], Force.com [97]. In IaaS, infrastructure resources such as computing, storage, networking, and operating systems, etc are offered as a service. The IaaS cloud computing has evolved from the utility grid concept. Some examples of IaaS are Amazon Web Services (AWS) [98], GoGrid [99], VMware vCloud [100], etc. Some of the open source cloud solution providers are Eucalyptus [101], OpenNebula [102], etc.

Cloud computing can be broadly classified depending upon deployment as Private, Public, Hybrid or Community computing. In Private cloud, the infrastructure is operated solely for an organization. In Public cloud, the infrastructure is made available to the public or an industry group and is owned by an organization selling cloud services. While in Community cloud, the cloud infrastructure is shared by several organizations and share same concerns for a specific community. On the other hand, Hybrid cloud is a composition of two or more clouds. Some public cloud computing are offered as a Pay-As-You-Go basis.

Abadi et al. [55] has discussed the limitations and opportunities of deploying data management issues on emerging cloud computing platforms such as Amazon Web Services. It is difficult to maintain basic properties of databases over large geographic distances due to high latency and network bandwidth bottlenecks. So, cloud structure is more suitable for tasks related to read-only data as compared to transactional databases.

Recently the grid and cloud infrastructure have started to merge in the DCI's. Accessing the cloud from the grid for the job submissions by using the 3-G Bridge has been reported [103]. This approach establishes connections between the cloud and grid environment.
EDGI [107] project has developed a DG-Cloud bridge middleware which is used to get additional resources for DG systems from cloud when the Quality of Service (QoS) requirements in DG can't be met from available resource. It has improved performance of Desktop Grid middleware to handle QoS requirements and SG-DG bridge middleware support to data-intensive applications.

Some pilot projects [108] in Cloud Computing at University of Westminster have addressed optimal scheduling of scientific application workflows for cloud-augmented grid infrastructures. The outcome of these projects indicates that the computational performance of the grid solutions has increased significantly.
Reynolds et al. [106] has reported that when number of tasks increased beyond a limit in DG then drop in performance is observed which can be handled by augmenting cloud resources to DG.

DG-based DCI is widely used as it is scalable and a cost effective solution. Hence, this environment is considered for the proposed research. The present standards of the databases do not define the generic architecture and performance metrics for data management in DG-based DCI. Some of the issues related to data are degraded

performance, architecture for querying data, data access and integration, etc. The effort of adapting existing data solutions like OGSA-DAI of SG-based DCI is not worthwhile due to degraded performance. It is necessary to investigate whether the same issues arise for handling large volumes of data (data size ≥ GB's) in DG-based DCI. There is need for addressing efficient data management for large volumes of data in DG-based DCI. Next section deals with the different technologies used in Desktop Grid-based DCI.

2.3. Desktop Grid Computing Technologies

Desktop Grid's are based on BOINC [6], XtremWeb [9], OurGrid [8], Condor [7], SZTAKI DG [10], etc. The University of Westminster Desktop Grid [15] which is based on SZTAKI solution has been running successfully for the last few years. SZTAKI Desktop Grid is based on BOINC technology.

BOINC [6] is an open source platform for Desktop Grid computing. The BOINC contains server and client software. Server software is used for creating volunteer computing projects. The client software periodically contacts the server to inform its availability, and in response receives a set of instructions for downloading and executing a job. Client uploads output files to the server when the job is completed, and requests more work. BOINC projects use a single centralized web server for data distribution. The BOINC client transfers files to and from data server using HTTP protocol. The server becomes bottleneck when tasks share the input files or due to limited bandwidth of server. This architecture is not be appropriate when dealing with large volumes of data due to server bottleneck and increase in data replication cost.

XtremWeb [9] is open source software to build a lightweight Desktop Grid by utilising the unused desktop computers. Its architecture consists of servers, workers, and clients. Its architecture is based on server/client architecture. It provides multi users, multi applications and cross domains deployments to run concurrently. Users of XtremWeb install clients on their desktop computers to interact with the infrastructure. Workers are installed on unused desktop computers for providing computing resources in XtremWeb infrastructure. Work units are provided with the URL's of input files. These are downloaded as a pre-processing step when a client job is launched. It uses a single centralized web server for data distribution. So, this architecture is not appropriate for dealing with large volumes of data.

OurGrid [8] is open source middleware which enables the creation of Peer-to-Peer computational grids. Grid participants provide the computing and storage resources to OurGrid infrastructure. It applies tit-for-tat policy for the resources allocation to the participants i.e. who have contributed most will also get the most. It provides a platform for parallel applications whose tasks are independent. The data size used for task execution is in order of few MB's. The focus of this technology is mainly on task distribution. So, this architecture is not appropriate for dealing with large volumes of data.

Condor [7] manages pool of workstations and dedicated clusters to provide a distributed high throughput computing system. A Condor pool consists of single machine which serves as the central manager, and an arbitrary number of other machines that have joined the pool. The focus of this technology is mainly on distributed batch processing.

The comparison of the above technologies used in Desktop Grid is described below:

	BOINC	**XtremWeb**	**OurGrid**	**Condor**
Architecture	Client-Server	Client-Server	Peer-to-Peer	Central Broker
Application Management	Centralized	Centralized	Decentralized	Decentralized
Task Distribution (Pull: Client to Server) (Push: Server to Client)	Pull	Pull	Push	Push
Resource Providers can act as Resource Consumers	No	Yes	Yes	Yes
Support for Volunteer Desktop Grids	Yes	Yes	Yes	No
Application Development/ Porting Complexity	High / Medium	Low	Low	Medium
Deployment/ Administration Complexity	Medium / Low on client side	Low	Low	Medium
Focus on data management	No	No	No	No

File size used in MB	1-300	1-250	Few MB's	1-50
Support for large volumes of data without splitting in small chunks	No	No	No	No
Shared data storage	No	No	No	No
Internal communication between worker nodes for data	No	No	No	No

Table 2.1: Technology comparison matrix [18]

BOING technology [110] is widely supports many projects in desktop grid computing. Next section deals with BOINC-based DG architecture.

2.4. Existing DG Architecture at University of Westminster

BOINC [114] is the most popular framework for volunteer systems. It has proved to be successfully used in many projects. SZTAKI Desktop Grid [17] is based on BOINC. Even the architecture of Desktop Grid at University of Westminster is based on SZTAKI Desktop Grid. Hence Desktop Grid is selected for the research. The purpose of this section is to understand the working of DG based on BOINC technology. The end user submits the job to the DG server via the P-Grade portal using web browser. P-Grade communicates with gUSE (Grid User Support Environment) to submit jobs to gUSE DG submitter. The components of the SZTAKI DG are as follows:

- BOINC Client: It is an application responsible for downloading inputs/application files, starting applications, uploading files, etc. It communicates with BOINC server.
- BOINC Scheduler: It decides which task has to be assigned to BOINC client(s).
- Job database: It is the central storage for all jobs that is submitted to DG submitter.
- Queue Manager: It checks the contents of job database at regular intervals and sends this information to DCI-API master along with scheduling policy.
- DC-API Plugin: DC-API applications consist of two major components: a master application and one or more client applications. The master is responsible for dividing the global input data into smaller chunks and distributing these chunks in

the form of work units (WU). The master interprets and then combines the output generated by the work units in form of global output.

The existing architecture of Desktop Grid [15] at University of Westminster is as follows:

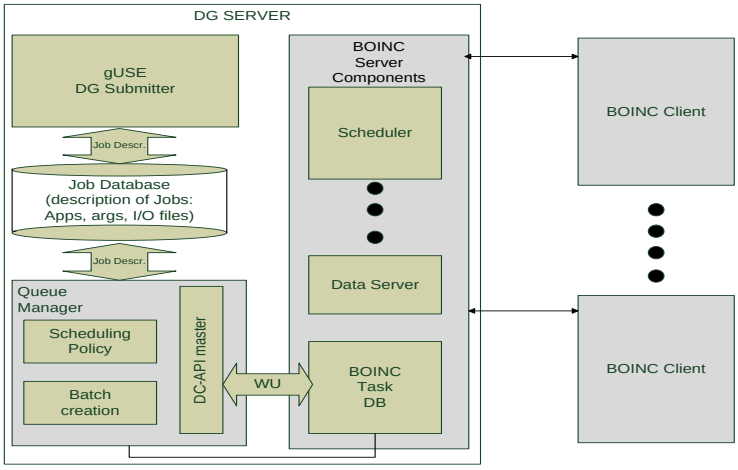

Figure 2.1: University of Westminster's DG [15] architecture

Presently DG supports the following requirements [18] for the application:

- Only master/worker or parameter sweep parallelisation.
- No shared data storage.
- No MPI or internal communication between worker nodes.
- Nodes can use the results of other nodes, but only through the server.
- Typically long running jobs with small or medium-sized (max. 100 MB per slave) inputs and outputs.

Next section deals with new requirements for emerging applications which is not supported by the existing DG architecture.

2.5. Emerging Applications Scenario

Most of the applications in physical and life sciences, especially biology and astronomy are data-intensive. Bioinformatics discipline consists of different databases which provide a different perspective on a collection of organisms, genes, proteins, diseases, etc. Scientists make these databases publicly downloadable, so that other scientists can copy the contents from databases and start doing their experiments. Meanwhile the original data sources continue to be edited. Some data providers publish weekly or monthly lists of data updates. These updates have to be synchronised with other existing data sources. These applications require handling of large volumes of data and support for R/W of data.

The International Nucleotide Sequence Database [40] Collaboration is a case scenario of emerging applications. The International Nucleotide Sequence Database [40] Collaboration is a joint effort to collect and disseminate databases containing DNA and RNA sequences. It collects nucleotide sequences data from researchers and issues the internationally recognized accession number to data submitters.

It collaboratively exchanges data between DNA Data Bank of Japan [42], GenBank USA [43], and European Nucleotide Archive UK [41] over 18 years. The data synchronization is maintained according to a number of guidelines published by an International Advisory Board. New and updated data on nucleotide sequences contributed by research teams to each of the three databases are synchronized on a daily basis at each the collaborating organizations. The time frame of the data synchronization has decreased from weeks to a day over the period of 18 years.

The database consists of a collection of records of nucleotides. Each record includes nucleotide sequence and the information of submitters, references, source organisms, and the biological nature such as gene function and other property of the sequence, etc. The present database size for storing these records is in order of GB/TBytes. In the subsequent first post-genome decade, 270 billion base sequence pairs have been added to the existing collection of finished sequences. It has resulted in doubling the size of the database approximately every 18 months [44].
This type of emerging applications requires handling of large volumes of data and support for R/W of data in order to maintain data consistency. Next section deals with existing data solutions in DG-based DCI.

2.6. Existing data solutions in DG-based DCI

This section explores different data solutions that can be used for handing large volumes of data in emerging applications. As the commodity hardware underlying DG-based DCI gets cheaper with increased capacity of hard drives, the storage of Desktop Computer is underutilized. There are some solutions such as Freeloader [33], Farsite [34] [35], Stdchk [36], Blobseer [37], Attic [16], etc which utilises the unused space of Desktop Computers.

Freeloader [33] is an open-source, lightweight, highly decentralized storage cache system built on scavenged disk spaces. It employs an asymmetric striping technique to take advantage of the local space and I/O bandwidth at workstations that processes data. It can store scientific datasets that are much larger than the disk space of a desktop computer.

Farsite [34] [35] is a storage service that runs on the desktop computers of a large organization and provides the semantics of a central NTFS file server. It runs entirely on client machines. It achieves data availability and reliability through replication. To improve global data availability, it continuously monitors machine availability and relocates files accordingly to equalize availability across all files in the system.

Stdchk [36] is a solution that offers low-cost checkpointing storage for Desktop Grid. It applies write intensive I/O approach. It gives applications the access to the scavenged storage through a traditional file system interface. It provides fault tolerance for long-running high-throughput applications running on desktop grids.

BlobSeer [37] is generic data management system designed to support high-throughput data-intensive applications over a wide-area-network. It is based on versioning technique. It is used for massive data processing in applications like online transaction records, astronomy, supernova detection, etc.

Above mentioned data solutions [33] [34] [35] [36] [37] do not define the generic architecture for data management in DG environment and their performance is not good in handling large volumes of data.
Peer-to-Peer [57] systems have been used in the past for dealing with large volumes of data efficiently in a distributed environment. Attic [16] is an existing DG solution based on P2P for handling large volumes of data. Hence, Attic is considered as a preferred solution and hence its working is explained in detailed.

Attic [16] is the implementation of the Peer-to-Peer Architecture for Data-Intensive Cycle Sharing (ADICS) [39]. The data is provided by data provider to one of the Data Centre (DC) by using Data Lookup Service. Data is further replicated to other Data Centres by P2P technique. The worker fetches the desired data for the work unit from Data Centres.
Attic architecture consists of data providers, data lookup service, data centres, data seed, scheduler, and worker nodes.

1. Data Provider (DP) or Clients are centralized entities that are able to authorize to the Data Lookup Service and publish data that will be later transferred to the Data Centres.
2. Data Lookup Service (DLS) is the centralized entity that allows other network participants to publish new information, locate data on the network, and create or modify metadata.

3. Data Centre (DC) provides the distributed data management services that serves workers requests for data downloads. Data Centres independently contact the Data Lookup Service to receive pointers to download locations.
4. Data Seed node is a specialized instance of Data Centre that allows for propagation of data from third party entities (Data Provider/Client) to the Data Centre overlay.
5. Scheduler (Network Manager) is another centralized entity that responds to replication requests and ensures that data are being propagated to the Data Centre overlay.
6. Worker Nodes download the data from Data Centres after getting the location of data from DLS.

The working of Attic is as follows:

1. Data is published to a Data Lookup Service by Data Provider.
2. Data Centre query DLS for downloading data.
3. DLS provides Data Centre the data pointer containing endpoints associated with a metadata description.
4. Data Centres starts to download data from the endpoints specified in the pointer.
5. After downloading the data, Data Centre notifies the DLS that is has the data.
6. DLS updates its pointer with Data Centre's endpoint by adding it to the known list of replicas.
7. A worker node then invokes a request to DLS for data.
8. Worker proceeds to download data from the endpoints specified in the pointer from Data Centre(s).

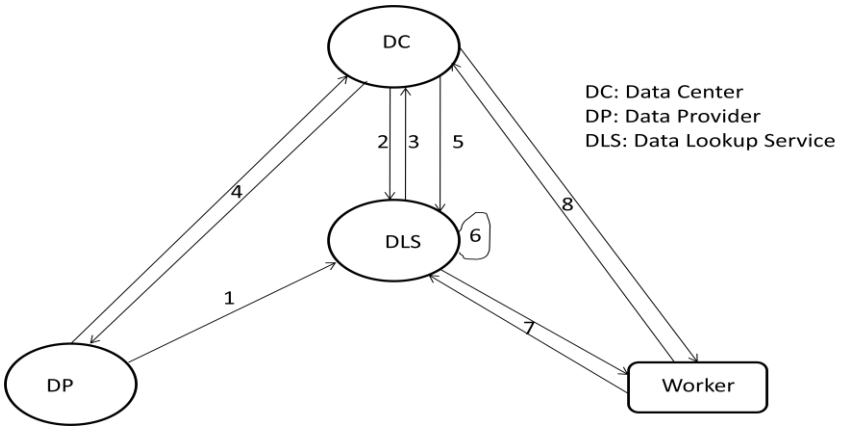

Figure 2.2: Attic Workflow

The performance of existing data solutions dealing with generic architecture for data management is not efficient in DG-based DCI due to issues like architecture for querying

data, data access and integration, support of R/W data, etc. Due to increasing demand of large volumes of data in scientific experiments, new types of applications are emerging. These applications require shared data storage, updating of some specific data from massive dataset, etc. Next section deals with the applicability of emerging applications scenario in DG environment.

2.7. Applicability of Read/Write of data in Desktop Grid

The data from the data centres in DG environment has to be replicated to other sites so that each site has the same set of data. As the data at one site is updated by some emerging application, the same data should be synchronised to other sites to maintain consistency of R/W of data.

Consider that Read/Write of data is applicable in DG-based DCI for the case scenario as described in section 2.5.

Minimum 3 sites are considered for the case scenario. The number of sites can be increased to incorporate more sites. Assume that following 3 sites are acting as Data Centres:

- DNA Data Bank of Japan (DDBJ) is represented as DC1
- European Nucleotide Archive (ENA) is represented as DC2
- GenBank is represented as DC3

1. Initially, assume that all the 3 sites (DC1, DC2, and DC3) have the same data. Researchers at each site use the data available for analysis.
2. After some period of time, research team at DDBJ site contributed new and updated data of nucleotide sequences. Then, this data has to be updated and replicated to all the other sites.
3. The process at step number 2 will be repeated for the new/updated data available at other sites i.e. ENA, GenBank.

DDBJ, ENA, and Genbank may be considered as different Data Centre sites in DG-based DCI. Initially, the data may be available at one or more sites. Then, this data has to be replicated to other sites so that each site has the same set of data. As the research team at one site updates the data, the same data should be synchronised to other sites to maintain data consistency.

Next section deals with suitability of Attic an existing DG-based data solution for emerging application requirements in order to handle large volumes of data and to maintain Read/Write of data.

2.7.1. Exploring Attic's suitability for R/W of data in Desktop Grid

Attic is an existing data solution based on P2P which is well established [57] for handling large volumes of data and so its suitability is being explored for the emerging applications. Accordingly to Attic model terminology, the following scenario is used to illustrate the operation of Read/Write of data in DG-based DCI.

Three institutes X, Y, and Z are acting as Data Centres DC1, DC2, and DC3 respectively for storing nucleotide sequences data:

1. X contains 100 worker nodes,
2. Y contains 100 worker nodes,
3. Z contains 100 worker nodes.

R/W of data can be generalised in the following 4 case scenarios:

1. Case 1 (Read only data available at a Single Provider): In this case only read only data is available at only one location.
2. Case 2 (Read only data available at multiple Providers): In this case only read only data is available at multiple locations. The data at multiple locations is also different.
3. Case 3 (Updated data available at Data Centre(s)): The data is updated at one location is to replicated to all other locations.
4. Case 4 (Worker download data from other worker nodes): The worker nodes also download the data from the other worker node in order to maintain data consistency.

Attic working in order to handle above mentioned cases are as follows:

2.7.1.1. Case 1 (Read only data available at a Single Provider)

1. Data is published to Data Lookup Service by Data Provider. Data Centre DC1 downloads the data after querying the data location from DLS.
2. Other Data Centres DC2, DC3 query DLS and download the data by P2P replication from DC1.
3. After some period of time, all the Data Centres will have the same data when all the data is downloaded by DC2, and DC3.
4. Worker nodes download the data for their work unit from Data Centres after querying the location from DLS.

Attic does not use dynamic strategies for data replication. It provides a fixed number of data replicas for replication in the network. Dynamic replication strategies [13] [14] [45] determine the replication at run time depending up on the parameter values. It improves the replication performance by reducing the latency, bandwidth, replication operations by

selecting optimal number of replicas, based on information available only at run time. The performance of data replication in Attic is not efficient as it uses fixed number of replicas in DG environment which degrades the performance.

2.7.1.2. Case 2 (Read only data available at multiple Providers)

1. Different set of data is published to Data Lookup Service by different Data Providers. Data Centre DC1 downloads the data from first location after querying DLS.
2. Later, Data Centre DC2 downloads the data from second location after querying DLS.
3. Data Centres DC1, DC2, and DC3 query DLS and download the data not available at their site from other Data Centres by P2P replication.
4. After some period of time, all the DCs will have same data.

In [39] the simulation and production environment of Attic, only one Data Centre at University of Cardiff is considered. Therefore, it not known whether this case had ever been implemented by Attic. So, performance of data replication in Attic solution is not known for this case scenario.

2.7.1.3. Case 3 (Updated data available at Data Centre(s))

1. Initially, assume that all the 3 Data Centres (DC1, DC2, and DC3) have the same set of data. Researchers use data available at each Data Centre for the analysis.
2. After some period of time, researchers at one Data Centre DC1 contribute new and updated data of nucleotide sequences.
3. This new and updated data will be communicated to DLS so that it can be downloaded by other Data Centres DC2 and DC3.

   Attic will treat updated data as new data and then it will replicate this updated data to other Data Centres. This will result in maintaining multiple versions of the same data at all the Data Centres. The cost of data storage and replicating updated data will increase significantly. So, performance of handling updated data is costly and inefficient in Attic for this case scenario.

2.7.1.4. Case 4 (Worker download data from other worker nodes)

In Attic, the workers are acting as data consumers. Worker can only download data from Data Centres. It may lead to performance degradation at Data Centres due to bandwidth

bottleneck. The case scenario when worker can download data from other workers is not addressed by Attic.

Existing Attic solution is not suitable for the emerging application requirements in DG environment due to the following reasons:

1. The performance of replication of Attic is not efficient since it does not uses dynamic replication strategies in handling large volume of data.
2. The performance of Read/Write of data in Attic is not efficient since it does not have concurrency mechanism to handle conflicting R/W data operations. It has to be modified to support concurrency mechanism. Existing Attic solution does not support of adding new functionality in form of API's.

Existing Attic solution is not very good at handling the emerging application requirements of R/W of data due to significant increase in cost of data storage and replicating updated data. There is a need for new architecture for the present Desktop Grid environment to support R/W of data due to emerging applications. Next section deals with the suitability of general data solutions used for handling large volumes of data in DG.

2.8. General data solutions used for handling large volumes of data

Some of the existing general data solutions used for handling large volumes of data are HDFS, Bitdew, etc. Hadoop Distributed File System (HDFS) [38] is used for large-scale data processing. It is used for executing MapReduce jobs. Its architecture is based on master-slave. Initially, the input data is split into smaller chunks, and a set of Map tasks is launched to process these small chunks in parallel. Then, the intermediate output generated is partitioned and transferred to corresponding Reduce tasks, where the reduce function is executed to produce final output.

In Hadoop, data is replicated three times (by default) to achieve data reliability and availability. HDFS adopts a relaxed consistency model where reordering of read and write operations are allowed as compared to traditional Distributed File System.

There has been attempt to integrate BOINC with MapReduce. Costa et al. [111] presented a system BOINC-MR to run MapReduce applications on top of BOINC. The system achieved performance increase of 64% in application turnaround time and reduction of 50% bandwidth as compared to BOINC system. The system deals with read-only data and uses only one master node to distribute and replicate data. A master node becomes a bottleneck as it might degrade the performance when large volumes of data are distributed.

The intermediate data generated by a Map task in Hadoop is stored on the local disk of a compute node without replication. DG's nodes are volatile, so this intermediate data may become unavailable, which in turn hinders the completion of reduce tasks. So, the corresponding Map task will be again executed on different node to generate the intermediate data. This leads to waste of lot of system resources.

1. Hadoop replicates the input as well as the output files on stable nodes. This type of replication is not desirable in DG due to the volatility of nodes.
2. Hadoop deals with read-only data.

So, Hadoop architecture is not appropriate for emerging applications in handling large volumes of data and Read/Write of data in DG-based DCI.

The BitDew [11] framework provides a programmable environment for data management and distribution services in Desktop Grids by using multi-protocols file transfers. The architecture is dependent upon a specific set of metadata for the data management operations. A single centralized server has been used for all data services. The performance will degrade due to bottleneck in server. The remote data storage has been provided between DG and Amazon S3 storage services [12]. The user has to tag each data as per the format provided by BitDew. This solution is not appropriate for handling large volumes of data in DG.

It is not appropriate to use the existing general data solutions used for handling large volumes of data in DG-based DCI. So, there is a need for new architecture for the present Desktop Grid for handling large volumes of data and to support R/W of data due to emerging applications.

2.9. Conclusion

Due to increasing demand of large volumes of data in scientific experiments, new types of applications are emerging. The performance of existing data solutions in SG, DG and CC dealing with large volumes of data is not efficient in DG-based DCI. It is also not appropriate to use the existing general data solutions like Hadoop, Bitdew for handling large volumes of data in DG-based DCI. So, there is a need of optimal architecture for handling large volumes of data in DG-based DCI. Emerging applications deals with Read/Write of data. So, there is a need to develop a data solution that supports R/W of data in emerging applications in DG-based DCI.

Chapter 3

Analysis and Design of Data access in DG-based DCI

3.1. Requirements for new data management solution

The performance of existing data solutions for handling large volumes of data is not efficient in DG-based DCI as described in chapter 2. New types of applications are emerging which require handling of large volumes and R/W of data. The modified data is generated by updating some of the values of existing data fields. This updated data has to be synchronised in all DG nodes for maintaining data consistency. Therefore, broad requirements for new data management solution in DG-based DCI are as follows:

1. To identify optimal architecture for handling large volumes of data in DG-based DCI.
2. To identify mechanism to support R/W of data in emerging applications.

The next section deals with the analysis of Peer-to-Peer based data solution for the fulfilment of the above requirements for the DG-based architecture.

3.2. Analysis of data management in Peer-to-Peer-based solution for Desktop Grid Computing

Peer-to-Peer (P2P) systems have been used in the past for dealing with large volumes of data efficiently in a large geographically distributed environment. So, P2P-based architecture is a promising solution for handling large volumes of data in DG-based DCI. Peer to Peer systems [56] are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority.

Some the characteristics of P2P as compared to traditional distributed computing are mentioned below:

- Symmetric role: In P2P, nodes can act as a client as well as a server.
- Scalability: P2P can scale up to thousands of nodes as compared to the traditional systems.

- Heterogeneity: In P2P, nodes can be slow or fast in terms of hardware capacity.
- Distributed control: There is no centralized structure in P2P.
- Dynamism: A peer can join or leave the network at any time.

3.2.1. P2P Architecture

P2P architecture can be broadly classified as Centralised or Decentralised. The architecture [57] of P2P systems in general is drawn below:

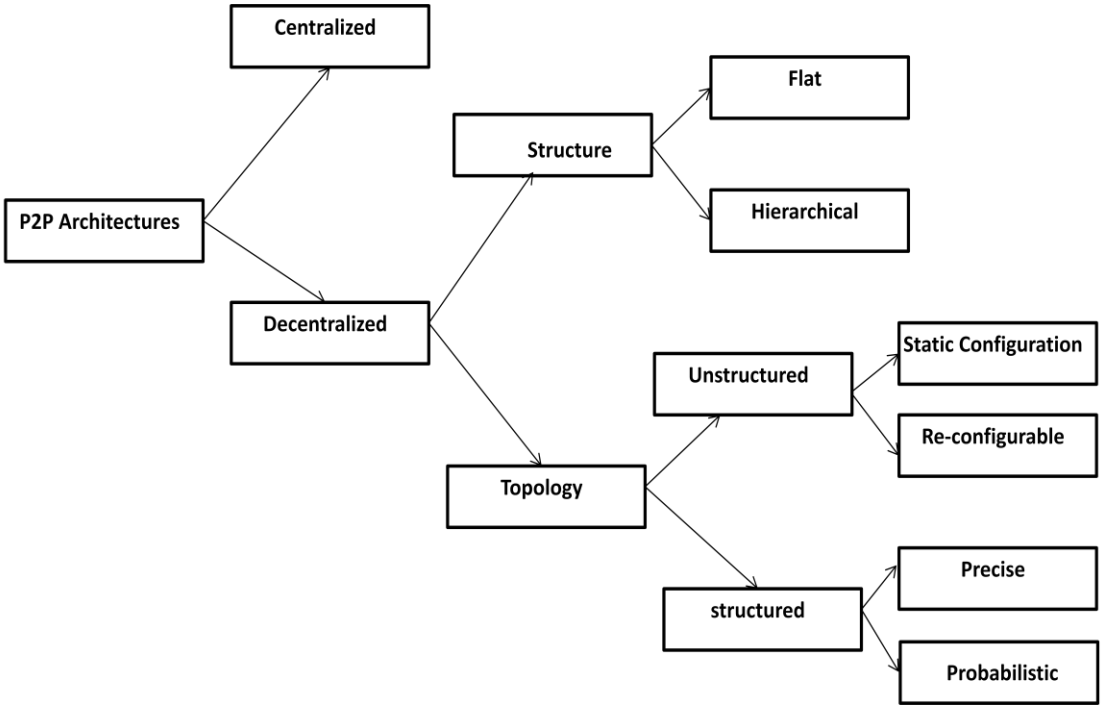

Figure 3.1: Topology of the P2P systems

In centralised P2P systems [57], there is one or more central server(s), which help peers to locate their desired resources or act as task scheduler to coordinate actions among them. To locate resources, a peer sends a message to the central server to determine the addresses of peers that contain the desired resources. Once the peer has the information/data, it can communicate directly with other peers. Thus every peer has to go to the centralised server to get the information about the data. So, centralized server becomes a bottleneck which degrades the performance of the system, e.g. Napster [58], etc. In decentralized P2P systems, each peer has only a partial view of the P2P network and offers data/services that may be relevant to only some queries/peers. The main challenge for this system is to locate the peer(s) offering service/data quickly. It is not only immune to a single point of failure but also has high performance, scalability, robustness compared to the centralised P2P systems. Most of the decentralised systems are non-hierarchical in nature.

Present P2P decentralized architecture [57] can be as classified according to the structure or the topology of the network. It can be either flat (single tier) or Hierarchical (multi-tier). Topology can be classified as structured or unstructured depending upon the network.
In Structured P2P systems, there is a mapping between data and peers. Only the metadata is inserted into the P2P network, data is private to the peer. It is prior known that how the queries will be forwarded to the other nodes in the system, e.g. Chord [59], Pastry [60]. The cost of maintaining the structured topology is very high.
In an unstructured P2P system, each peer is responsible for its own data, and keeps track of a set of neighbours for forwarding the query. There is no mapping between identifiers of data objects and those of peers, e.g. Freenet [61], etc. The main challenge in unstructured system is for locating data, completeness of results, non-deterministic response time, determination of the neighbour, etc.

The nodes in DG-based DCI are scalable, dynamic and volatile in nature. In P2P architecture [27], peers are also scalable, dynamic and volatile, and have distributed control properties for data handling. So there is a need of new architecture for the research in DG-based DCI in which nodes will have functionality of handling large volumes of data as P2P. The modified P2P-based architecture will be one of the promising solutions for handling large volumes of data in DG-based DCI for the research.

Due to the heterogeneous, dynamic and volatile environment of DG-based DCI, data in data intensive applications has to be replicated in multiple nodes for improving data availability and reducing response time. The performance and data availability in distributed systems is dependent on how the data is replicated between nodes. Thus, there is a need for addressing data replication for improving the performance of data intensive applications in this environment.

3.2.2. Related Work in Data Replication and Consistency

The performance of data-intensive applications in DG-based DCI can be improved by addressing the efficient data replication. The data replication in a system depends on factors like reduced response time, network bandwidth, number of file replicas, number of replicating operations, improved availability, memory optimisation, etc. Data consistency is considered for R/W data for data update. Updated data is synchronised between different nodes in order to maintain data consistency in DG-based DCI.

Depending upon the above mentioned factors of data replication and consistency, further comparative analysis of the relevant research has been done below for handling large volumes of data:

#	Author(s)	Architecture	Reduced response Time (Yes/No)	Network Bandwidth consumption Consideration (Yes/No)	Number of replica Consideration (Yes/No)	Reduced access cost Consideration (Yes/No)	Improved Availability Consideration (Yes/No)	Storage utilization	Read/ Write operations	Consistency Considered (Yes/No)
1	K. Ranganathan [20]	P2P	Y	N	Y	N	Y	Increased	R	N
2	Abdullah [21]	P2P (Structured)	Y	Y	N	N	Y	Increased	R	N
3	V. Ramasubramanian [22]	P2P (Structured)		N		Y	Y		R/W	N
4	Y. Chen [24]	P2P	Y	Y	Y	Y			R/W	N
5	H. Lamehamedi [26]	Multi-Tier	N	N	Y	Y	Y	Improved	R/W	
6	R.M. Rahman [32]	Multi-Tier	Y	N	Y	N	Y	Optimal	R/W	Y
7	MemaRoussopoulos [23]	P2P							R/W	
8	Anwitaman Datta [25]	P2P (Unstructured)	Y	Y	Y	Y			R/W	
9	Yi Hu [27]	P2P (Structured)		N	N		Y	Not Considered	R/W	Y
10	Xin Chen [28]	P2P (Structured)					Y	Not Considered	R/W	Y
11	Haiying Shen [29]	P2P (Structured)						Not Considered	R/W	Y
12	Liangzhong Yin [30]	P2P (Structured)	Y						R/W	Y
13	Haiying Shen[31]	P2P (Structured)			Y	Y			R/W	Y

Table 3.1: Comparative analysis of the data replication research

Where,

Y indicates Yes,

N indicates No,

R indicates Read operation,

W indicates Write operation.

Analysis from table 3.1 is described as follows:

1 Most of the research studies [20] [21] [22] [23] [24] [25] [27] [28] [29] [30] [31] involves P2P architecture for dealing with large volumes of data.
2 Research in some studies [27] [28] [29] [30] [31] [32] have focused both on R/W replication and consistency. The architecture of study [32] is Multi-Tier while others have structured P2P architecture. Research study [31] is focusing on solving data replication and consistency simultaneously
3 The focus of research studies [27] [28] [32] is mainly on improving the availability of data while [30] [32] have focused on reducing the response time.

Outcomes from the above analysis are as follows:

1 Most of the architecture dealing with large volumes of data uses P2P architecture. Thus, P2P architecture is promising architecture for dealing large volumes of data in DG-based DCI.
2 More efficient algorithms are needed for handling data replication in P2P system.
3 Efficient concurrency control mechanisms are required for maintaining Read/Write data consistency and replication simultaneously in P2P system.

Thus, performance of DG-based DCI can be improved by addressing the efficient data replication strategies and consistency simultaneously. There is a need for a solution which handles data replication and consistency simultaneously in this environment. Therefore, there is a need for a data concurrency mechanism for maintaining the data consistency in the proposed research. Next section deals with the concurrency control techniques that are used for maintaining consistency.

3.2.3. Concurrency Control mechanisms

Distributed computing environment consists of large number of nodes which may modify the same data simultaneously. Data concurrency is defined as accessing the same data by many nodes at the same time. The purpose of the data concurrency control [82] [112] [113] [114] is:

- To enforce isolation among conflicting transactions.

- To preserve database consistency through consistency preserving execution of transactions.
- To resolve read-write and write-write conflicts.

Therefore, there is a need for a data concurrency mechanism for maintaining the data consistency in the proposed research. Some of the concurrency control mechanisms which are used for maintaining data consistency are mentioned below:

a) Two Phase Commit Protocol (2PC) [112]: This algorithm is used for coordinating all the processes that participate in distributed systems to commit or abort the transaction. It consists of two phases: Request phase and Commit phase. Depending upon the outcome of all processes, the coordinator takes the decision to commit or abort the changes for a particular transaction.

b) Locking [112]: It involves the issuing and releasing of the locks on the desired data required by the process. Locking is an operation which secures permission to read/write a data item for a transaction. Locks can be shared (read data) or exclusive only (used for read/write of the data). This is also known as Two Phase Locking protocol.

c) Timestamp Ordering [82]: In this method, a time stamp is assigned to a transaction, and depending upon the value of the time stamp a conflict between transactions can be removed / avoided in reading or writing of data.

d) Multi-version concurrency control [82] [112]: This approach maintains a number of versions of a data item and allocates the right version to a read operation of a transaction. In this method, a new version is created for a database object when the write operation is done. This method avoids locking the data object and depending upon the operations performed, it provides a consistent view of the object by using the right version. In this mechanism, a read operation is never rejected.

e) Optimistic Concurrency Control [112]: In this technique only at the time of commit, serializability is checked and transactions are aborted in case of non-serializable schedules. A schedule is serializable if its outcome is equivalent to the outcome of all its transactions executed serially. Serializability can also be checked by created a precedence graph. Precedence graph deals with finding a cycle in the graph. If there is cycle in graph, then the transaction is aborted else the operation of the transactions can proceed for read/write operation.

f) Deferred update [111] [112]: In this method, the write operation on a data object is not updated to a persistent storage until the transaction holding that data is committed.

Concurrent control mechanisms are used for handles concurrent writes by clients through common consensus. For resolving the concurrent writes, a consensus is required between different peers in P2P systems. Consensus is the process of agreeing on one result among a group of participants when the peers are volatile. The Paxos algorithm [104] is used for resolving consensus in a network of volatile nodes in distributed systems.

P2P-based architecture will be used for the research in which peers nodes are heterogeneous, scalable, dynamic and volatile, and have distributed control properties for data handling. So there is a need for control mechanism which handles concurrent writes by peers through common consensus. 2PC protocol is one of the promising solutions for resolving consensus. Since 2PC assumes stable medium at each node, so time stamp ordering properties will be used for resolving the consensus in volatile and dynamic environment. Therefore, modified Two Phase Commit Protocol along with timestamp ordering properties will be used in the research to address concurrency mechanisms to maintain data consistency. The next section deals with the finite state diagram for the proposed DG-based architecture for data management.

3.3. Finite State Model for new data management solution

The requirements for the new data solution on the basis of analysis outcomes from section 3.2.2. are as follows:

1. The new architecture [20] [21] [22] [23] [24] [25] [27] [28] [29] [30] [31] for dealing with large volumes of data in DG-based DCI should be P2P-based.
2. Efficient concurrency techniques are needed for maintaining data consistency in DG-based DCI due to R/W data in emerging applications [40].
3. Efficient replication algorithms [27] [28] [29] [30] [31] [32] are needed for handling large volumes of data in DG-based DCI.

To address the above requirements following should be designed:

1. Optimal P2P-based DG architecture.
2. New algorithms for maintaining consistency of data due to data modifications.
3. New data replication algorithm.

The existing Desktop Grid architecture components as discussed in section 2.4 should be modified to incorporate the P2P techniques for handling large volumes of data. These P2P entities will be in addition to existing BOINC entities in the proposed DG-based DCI architecture. The proposed architecture for data management in DG-based DCI will

consist of P2P coordinator and P2P clients. P2P coordinator and P2P client interact with each other in the proposed architecture whenever data is modified in the DG environment. The proposed architecture is shown in figure 3.2.

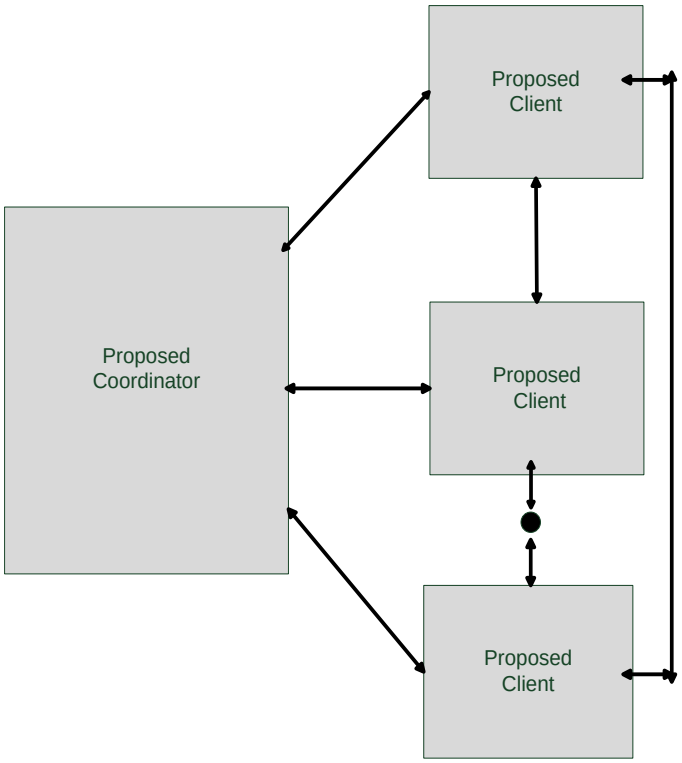

Figure 3.2: Proposed DG Architecture

The proposed architecture consists of the finite state models (FSM) for coordinator and client in DG-based DCI. In FSM, each state is represented by a circle and a transition by an arrow. A transition is labelled with an input that causes a transition.

When the coordinator receives the update request from a client, it adds it in a queue. Then it checks whether data received exists in the data log. If it exists, then it checks whether the update data request should be accepted or not. If the timestamp of the received data is less than or equal the timestamp in data log then it is accepted. The existing timestamp is incremented and then updated data along with new timestamp is sent to all the clients having same data for task execution. The finite state model for the coordinator is mentioned in the figure 3.3.

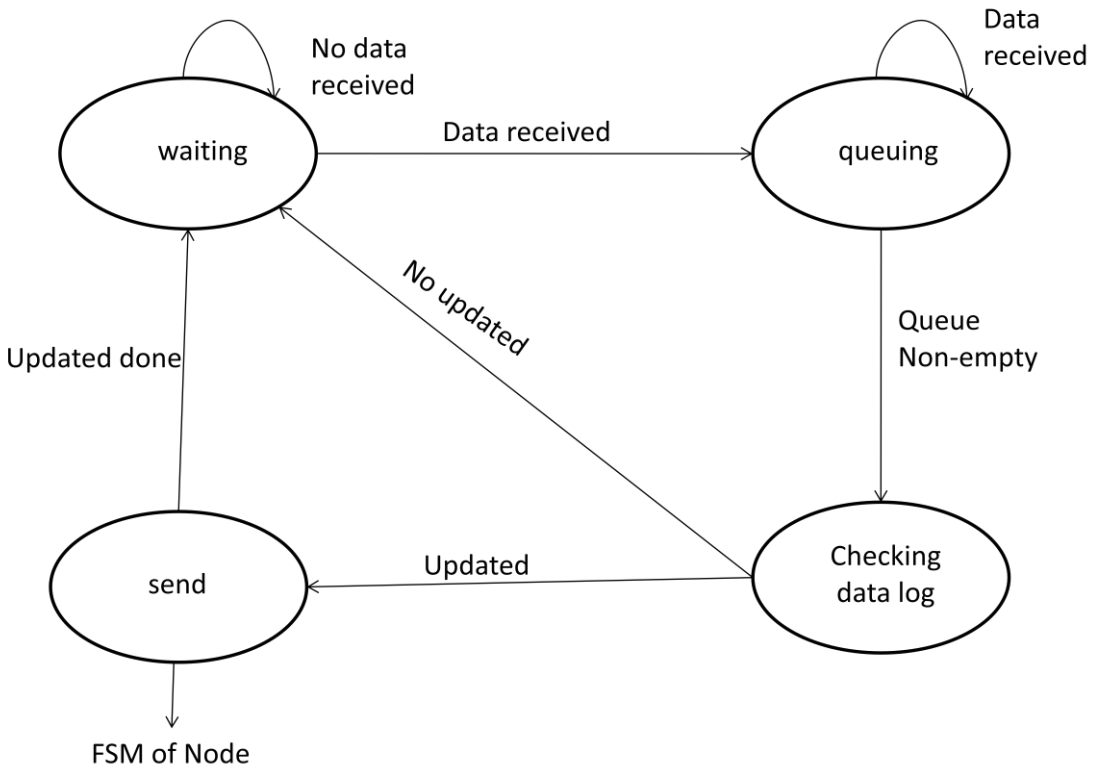

Figure 3.3: Finite state model for coordinator

The FSM of the coordinator has 4 states: waiting, queuing, send, and checking data log.

- Waiting: In this state, coordinator waits for request from client.
- Queuing: In this state, coordinator keeps the received data from the client(s) in a queue.
- Checking data log: It this state, coordinator checks for data updating request from a client depending upon the time stamping of data received.
- Send: In this state, coordinator sends the updated data and timestamp to the client.

The state transition table for the FSM of coordinator is mentioned in table 3.2.

Current State	**Input**	**Next State**	**Status**
Waiting	No data received	Waiting	None
	Data Received	queuing	Data is kept in queue
Queuing	Data received	Queuing	The data received is kept in queue if queue is empty.

	Queue non empty	Checking data log	The data received is checked for the validation condition in the data log if queue is not empty.
Checking data log	Not updated	Waiting	The data received is checked for update condition in the data log. When no update condition is satisfied then it goes to the waiting state.
	Updated	Send	The data received is checked for update condition in the data log. When update condition is satisfied then it goes to the send state.
Send	Updated (Updation finish)	Send (FSM of client)	Updated data is send to the client for updation.

Table 3.2: State transition table for FSM of coordinator

When the client is up for the first time, it sends a joining request to the coordinator. Once the joining request is accepted then it joins the network. Then, it starts accepting the data from the coordinator. It executed the task received from the coordinator. When an updated data request is received, it checks for the timestamp of data. If received timestamp is more than the existing timestamp then updated data is accepted. When the updated data is generated by the client itself, then this data along with existing timestamp is sent to the coordinator. A client goes down when it is disconnected from the network. The finite state model for the client is mentioned in the figure 3.4.

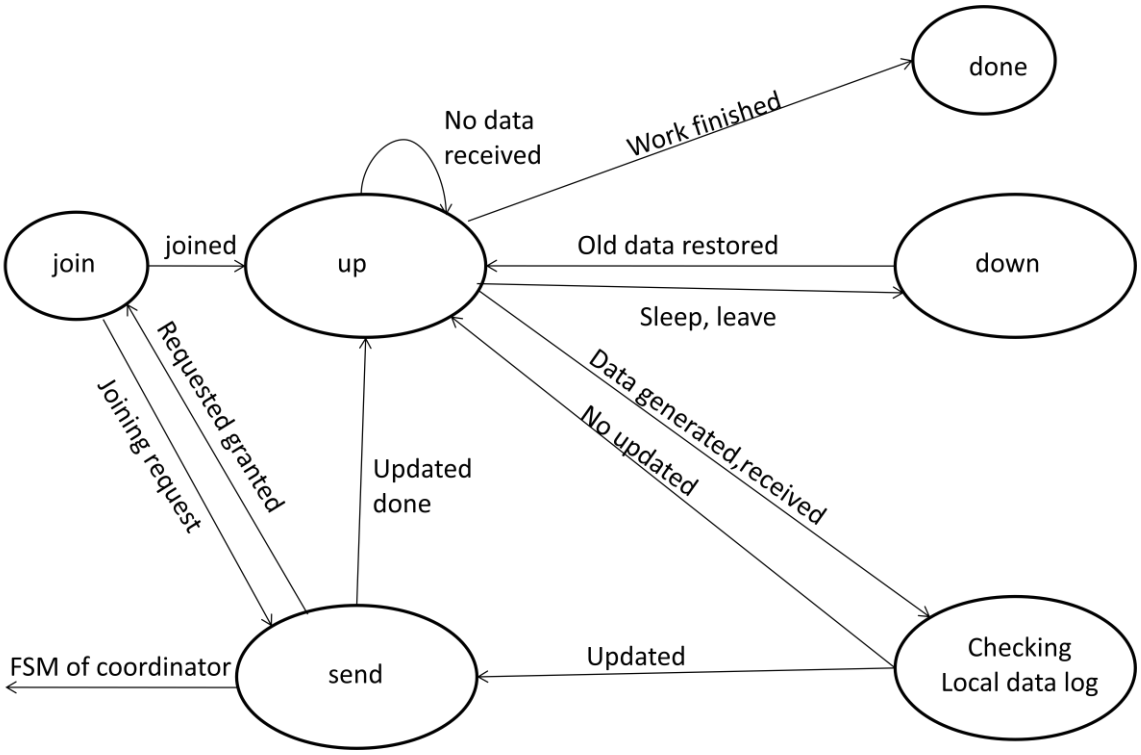

Figure 3.4: Finite state model for client

The FSM of the client has 6 states: join, up, down, done, checking local data log, and send.

- Join: In this state, a client wants to join the network. It sends a join request to the coordinator.
- Up: In this state, a client has already joined the network and it is ready for accepting the data and task from coordinator.
- Down: In this state, a client is disconnected from the network.
- Done: In this state, the task given to the client is completed by the client.
- Checking local data log: In this state, a client checks for the timestamp of the data received timestamp. If received timestamp is more than the existing timestamp then updated data is accepted. If data is generated by the client itself, then this data along with existing timestamp is sent to the coordinator.
- Send: In this state, the client sends the update data, joining request to the coordinator. It also accepts the updating data request from the coordinator send by another client.

The state transition table for the FSM of client is mentioned in table 3.3.

Current State	**Input**	**Next State**	**Status**
Join	Joined	Up	The client request for joining the Desktop Grid is accepted by the coordinator.
	Joining Request	Send	Request is send to the coordinator.
Up	No data received	Up	None
	Receiving generated data	Checking local data log	When data update is received, it is checked in the local data log for the updation.
	Work finished	Done	The task is finished with the data.
	Old data resorted	Down	The client is disconnected when it leaves the network due to user interruption or when client goes for sleep.
Down	Old data restored	Up	Old data is restored in the client.
Checking local data log	No updation	Up	The data received is checked in the local data log. When condition is not satisfied then it goes to the up state.
	Updation	Send	The data received is checked for update condition in the local data log. When update condition is satisfied then it goes to the send state.
Send	Updated (Updation finish)	Up	Updated data is send to the coordinator for updation.
	Requested granted	Send (FSM of coordinator)	The request for joining the network is sent to the coordinator.

Table 3.3: State transition table for FSM of a client

New algorithms will be needed to maintain the consistency and replication of Read/write of data as per the FSM of coordinator and client.

3.3.1. Proposed Algorithms based on FSM of coordinator and client

As per the FSM of coordinator and client, the coordinator is used for resolving the conflict due to R/W data operations. Conflicts occur when two or more modified data are sent by the clients to the coordinator concurrently. This results in inconsistent data in the system. This problem can be resolved by having a timestamp along with the modified data item. The coordinator creates a time stamp for each of the modified data item when received in the proposed algorithm. The coordinator then resolves the data conflict by using conflict resolving algorithm. For maintaining data consistency another algorithm is required. This describes the basic need of two consistency algorithms in proposed architecture. The detailed working of the consistency algorithms is mentioned in the section 4.2. of next chapter.

Once the consistency of data is maintained by the coordinator, then the modified data has to be replicated to all other clients having same set of data. The algorithm is needed which measures the data replication performance between different clients for different paths. Once the frequent paths having efficient replication are found, data replication is applied along these paths. This describes the need of data replication algorithm. The detailed working of replication algorithms is mentioned in the section 4.3. of next chapter.

The next section deals with the sequence diagrams for new data management solution for maintain data consistency for the DG-based architecture.

3.4. Sequence diagrams for new data management solution

The coordinator will accept the input from the client(s) for resolving conflict for R/W of data. Coordinator creates new timestamp for each new/updated value of data item received. Clients send modified value of data item and previous received timestamp to coordinator. A set of case scenarios have been identified from the cases mentioned in chapter 2 for the emerging applications to handle large volumes and R/W of data in DG-based DCI. These case scenarios are based on the FSM of coordinator and client as mentioned in section 3.3. In order to prove the working of new data solution, a set of scenarios is created with 1 coordinator and maximum 3 clients in order to simplify. This is a sample demonstration of activities that would be required in a case scenario. The number of clients may increase or decrease, but the interaction between them would follow similar pattern. Hence, it can be considered as an exhaustive list meeting the needs of a generalised situation.

Number of case scenarios for justifying the proposed solution can be limited as per the following list:

1. One client modifies data.
2. Two clients modify data having same value of timestamp and data.
3. Two clients modify data having same data value but having different timestamps.
4. Two clients modify data having different data value and timestamps.
5. Two clients modify data and then one client goes down whose data update request is first accepted.
6. Two clients modify data item and 3rd one is busy in computing.
7. Two clients modify more than 2 or more data item.

The assumptions considered for the data management solution are as follows:

1. Initially one coordinator is present in DG system.
2. Coordinator will only accept request for reading/updating data of data item(s) from clients.
3. For simplicity timestamping is started at 00 and values of data item consists of one or two values.
4. Coordinator maintains list of clients containing data item. "Send to all" message for a particular data item is only for those clients which are present in the list of coordinator.

Notations used for the case scenarios are as follows:

Send <x=3, 00>➔C represents, send x=3 to Coordinator with previous value of timestamp 00.

Where,

C: Coordinator

x: data item

x=3: data item x value is 3

00: previous timestamp of x received from coordinator C

x': new local updated value of x

y': new local updated value of y

3.4.1. Case Scenario 1: Only one node (n1) modifies data item x

Consider the following initial values in DG environment,

1. Data item x=1.
2. Number of clients =2

3. Clients: n1(x=1), n2(x=1); timestamp: 00

The sequence diagram for this case scenario is mentioned in the figure 3.5.

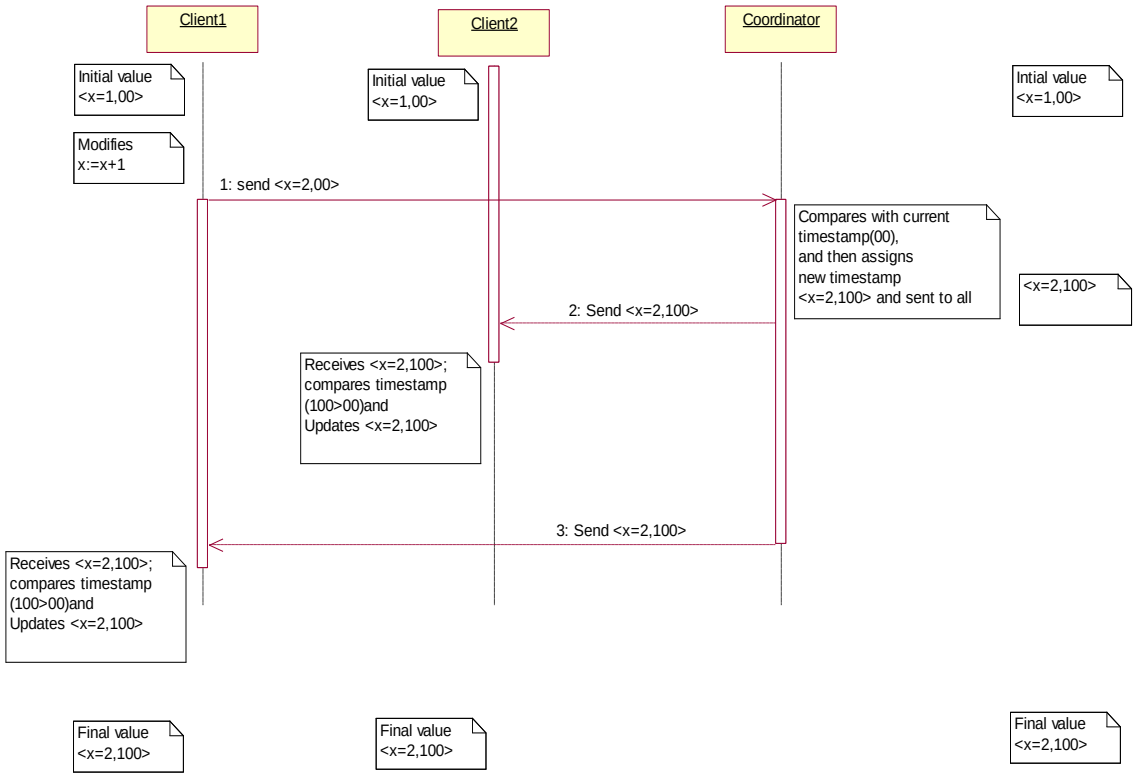

Figure 3.5: Sequence diagram for case scenario 1

The execution order detail of figure 3.5 is mentioned in table 3.4.

Execution order	**Client n1**	**Client n2**	**Processing done by Coordinator C (assignment of <value, timestamp>)**	**Global Value of <x, timestamp> at coordinator**
1	x=1	x=1		<x=1,00>
2	x:=x+1 {x=1; x'=2}	x=1		<x=1,00>
3	SEND <x=2,00>➔C (where 00 represents last timestamp received from C)			

4	{x=1; x'=2}	x=1	Receives < x=2,00> from n1; Compares with present timestamp (00) and then assigns new timestamp <x=2,100>	<x=1,00>
5			Send to all clients <x=2,100>	<x=2,100>
6	Receives <x=2,100>; compares timestamp(100>00) and Updates <x=2,100>	Receives <x=2,100>; compares timestamp(100>00) and Updates <x=2,100>		<x=2,100>
7	<x=2,100>	<x=2,100>		<x=2,100>

Table 3.4: Execution order of case scenario 1

3.4.2. Case Scenario 2: Clients n1 and n2 modifies data having same value of timestamp and data.

Consider the following initial values in DG environment:

1. Data item x=1.
2. Number of clients =2
3. Clients: n1(x=1), n2(x=1); timestamp: 00

The sequence diagram for this case scenario is mentioned in the figure 3.6.

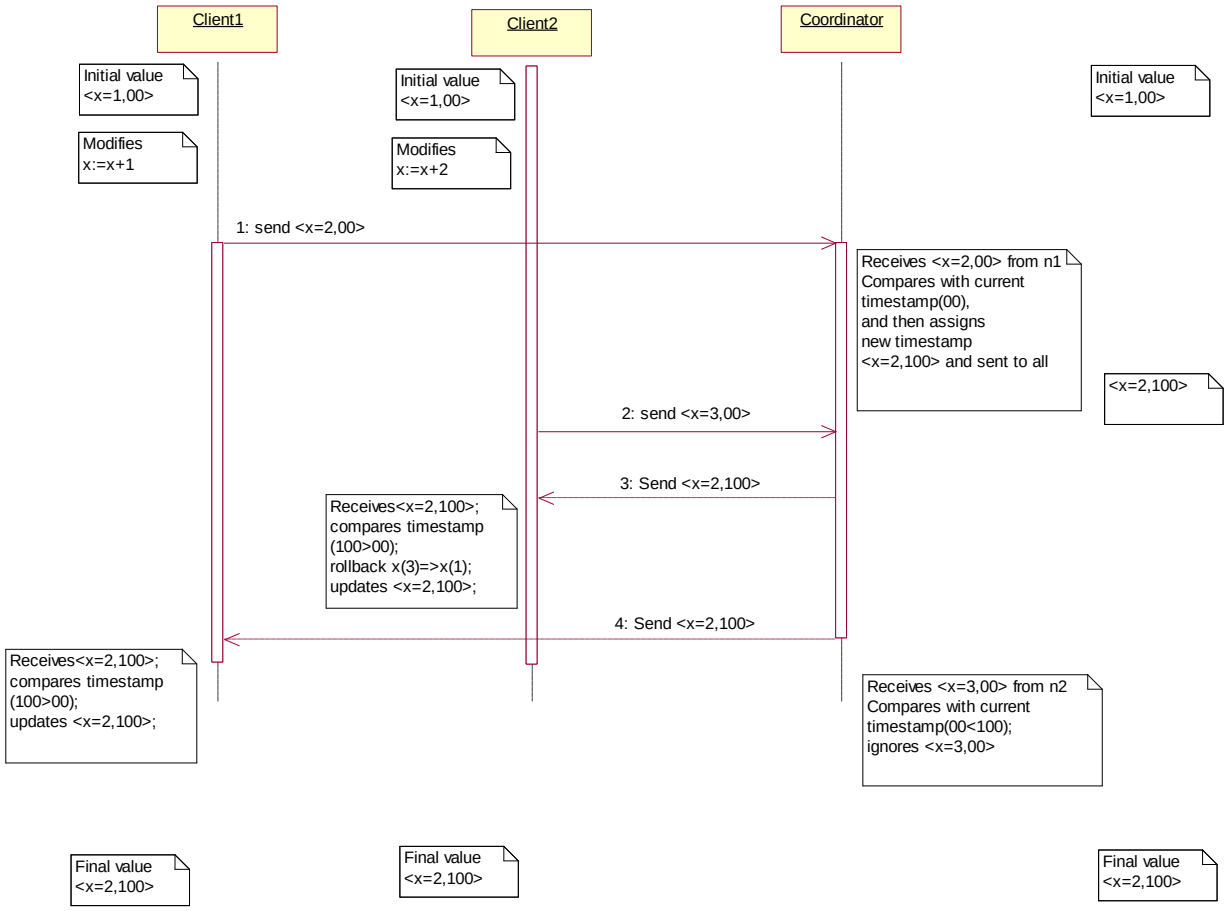

Figure 3.6: Sequence diagram for case scenario 2

The execution order detail of figure 3.6 is mentioned in table 3.5.

Execution order	***Client n1***	***Client n2***	***Processing done by Coordinator C (assignment of <value, timestamp>)***	***Global Value of <x, timestamp> at coordinator***
1	*x=1*	*x=1*		*<x=1,00>*
2	*x:=x+1 (x'=2)*	*x:=x+2 (x'=3)*		*<x=1,00>*
3	*SEND <x=2,00> ➔C (where 00 represents last timestamp received from C)*	*SEND <x=3,00> ➔C (where 00 represents last timestamp received from C)*		*<x=1,00>*
4	*{x=1; x'=2}*	*{x=1;x'=3}*	*Receives < x=2,00> from n1 first; Compares with present timestamp*	*<x=1,00>*

			(00) and then *assigns new timestamp* *<x=2,100>*	
5			*Send to all clients* *<x=2,100>*	*<x=2,100>*
6			*Receives <x=3,00> from n2;* *Compares with present timestamp (100) and ignores <x=3,00>*	*<x=2,100>*
7	*Receives <x=2,100>; compares new timestamp>old (100>00)* *and* *updates x=2* *{<x=2,100>}*	*Receives<x=2,100>; compares new timestamp>old (100>00)* *and* *rollback x'(3)➔ x(1),* *updates x=2* *{<x=2,100>}*		*<x=2,100>*
8	*<x=2,100>*	*<x=2,100>*		*<x=2,100>*

Table 3.5: Execution order of case scenario 2

3.4.3. Case Scenario 3: Clients n1, n2 modifies data item x having same data value but having different timestamps.

Consider the following initial values in DG environment:

1. Data item x.
2. Number of clients =2
3. Clients: n1(x=2, 200), n2(x=2, 100)
4. Timestamps (200, 100)

The sequence diagram for this case scenario is mentioned in the figure 3.7.

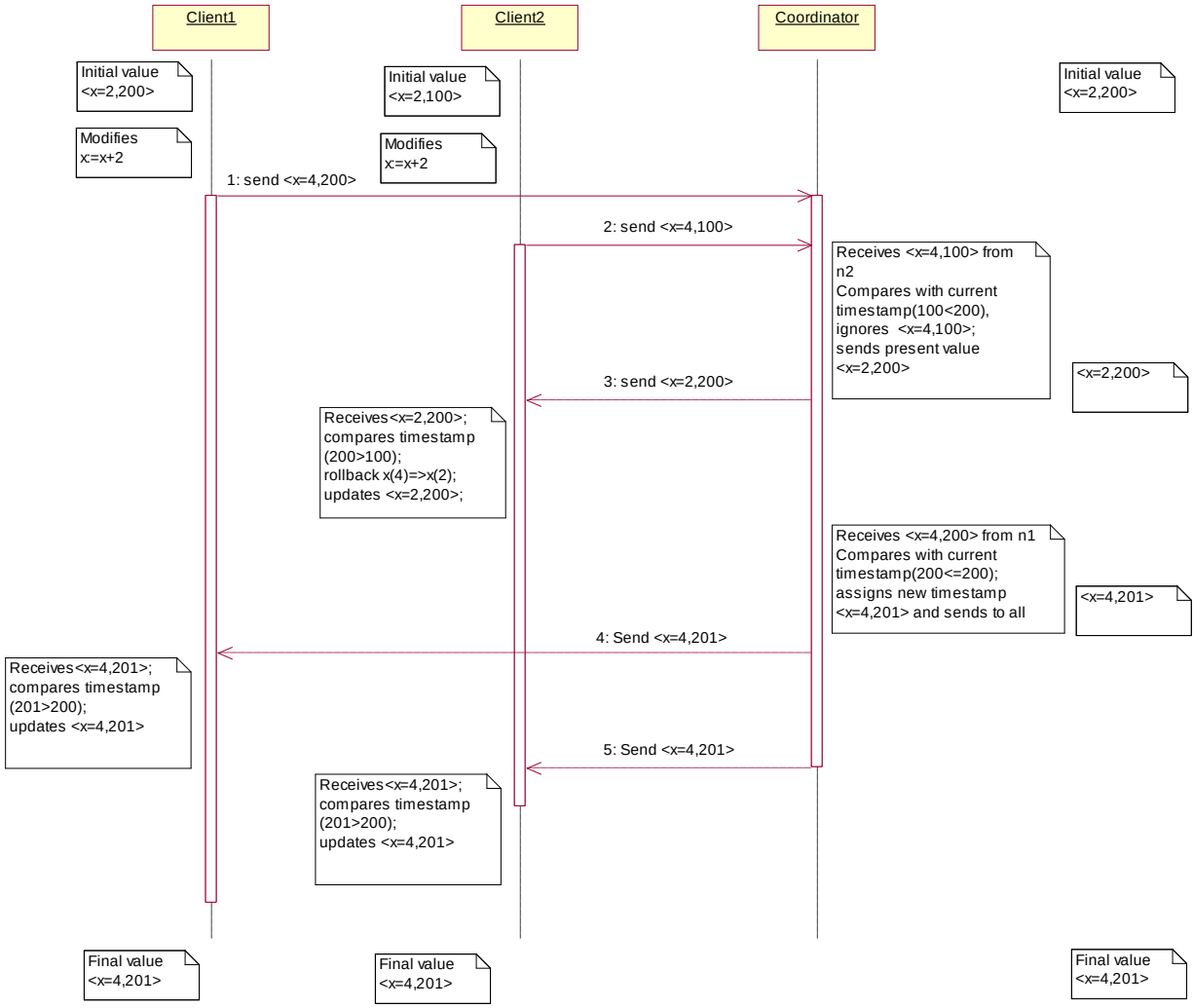

Figure 3.7: Sequence diagram for case scenario 3

The execution order detail of figure 3.7 is mentioned in table 3.6.

Execution order	***Client n1***	***Client n2***	***Processing done by Coordinator C (assignment of <value, timestamp>)***	***Global Value of <x, timestamp> at coordinator***
1	*<x=2,200>* *x:=x+2 (x'=4)*	*<x=2,100>* *x:=x+2 (x'=4)*		*<x=2,200>*
2	*SEND<x=4,200>➔C* *(where 200 represents last timestamp received*	*SEND <x=4,100>➔C* *(where 100 represents last timestamp received*		*<x=2,200>*

	from C)	*from C)*		
3			*Receives < x=4,100> from n2 first;* *Compares with present timestamp (200),* *ignores < x=4,100>* *And send present value of x <x=2,200>*	*<x=2,200>*
4			*Receives < x=4,200> from n1;* *Compares with present timestamp (200),* *assigns new timestamp* *<x=4,201>*	*<x=2,200>*
5			*Sends <x=4,201> to all*	*<x=4,201>*
6		*Receives <x=2,200>;* *Compares new>old timestamp (100>200), if no* *Rollback x'=4 ➔x=2,* *Updates x <x=2,200>*		*<x=4,201>*
7	*Receives <x=4,201>* *Compares new>old timestamp(201>200, if yes* *Updates x:2 ➔4*	*<x=2,200>*		*<x=4,201>*
8	*<x=4,201>*	*Receives <x=4,201>* *Compares new>old timestamp(201>101, if yes* *Updates x:2 ➔4*		*<x=4,201>*
9	*<x=4,201>*	*<x=4,201>*		*<x=4,201>*

Table 3.6: Execution order of case scenario 3

3.4.4. Case Scenario 4: Clients n1 and n2 modifies data item x having different data value and timestamp values.

Consider the following initial values in DG environment:

1. Data item x=1.
2. Number of clients =2
3. Clients: n1(x=2,200), n2(x=1,100)

The sequence diagram for this case scenario is mentioned in the figure 3.8.

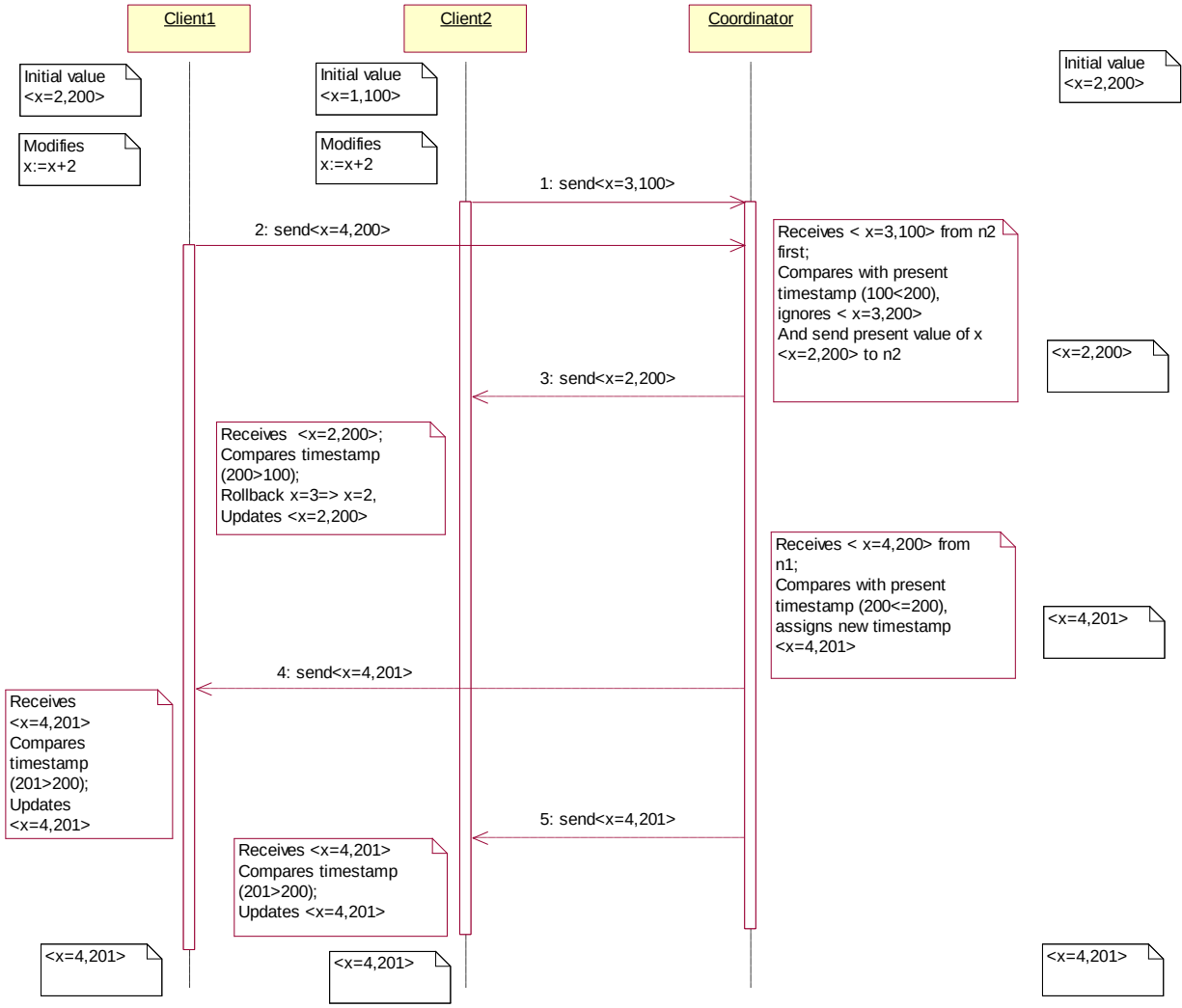

Figure 3.8: Sequence diagram for case scenario 4

The execution order detail of figure 3.8 is mentioned in table 3.7.

Execution order	***Client n1***	***Client n2***	***Processing done by Coordinator C (assignment of***	***Global Value of <x, timestamp> at***

			<value, timestamp>)	coordinator
1	<x=2,200> x:=x+2 (x'=4)	<x=1,100> x:=x+2 (x'=3)		<x=2,200>
2	SEND <x=4,200> ➔C (where 200 represents last timestamp received from C)	SEND <x=3,100> ➔C (where 100 represents last timestamp received from C)		<x=2,200>
3			Receives < x=3,100> from n2 first; Compares with present timestamp (200), ignores < x=3,200> And send present value of x <x=2,200> to n2	<x=2,200>
4			Receives < x=4,200> from n1; Compares with present timestamp (200), assigns new timestamp <x=4,201>	<x=2,200>
5			Sends <x=4,101> to all	<x=4,201>
6		Receives <x=2,200>; Compares new>old timestamp (200>100), if yes Rollback x'=3 ➔x=1, Updates x:1 ➔2 <x=2,200>		<x=4,201>
7	Receives <x=4,201> Compares new>old timestamp(201>200, if yes Updates x:2 ➔4	<x=2,200>		<x=4,201>
8	<x=4,201>	Receives <x=4,201>		<x=4,201>

		Compares new>old timestamp(201>200, if yes *Updates x:2 ➔4*		
9	*<x=4,201>*	*<x=4,201>*		*<x=4,201>*

Table 3.7: Execution order of case scenario 4

3.4.5. Case Scenario 5: Two clients modify data and then one client goes down whose data update request is first accepted.

We assume that network is fault tolerant. Consider the following initial values in DG environment:

1. Data item x=1.
2. Number of clients =2
3. clients: n1(x=2,200), n2(x=1,100)
4. Client n2 goes down

The sequence diagram for this case scenario is mentioned in the figure 3.9.

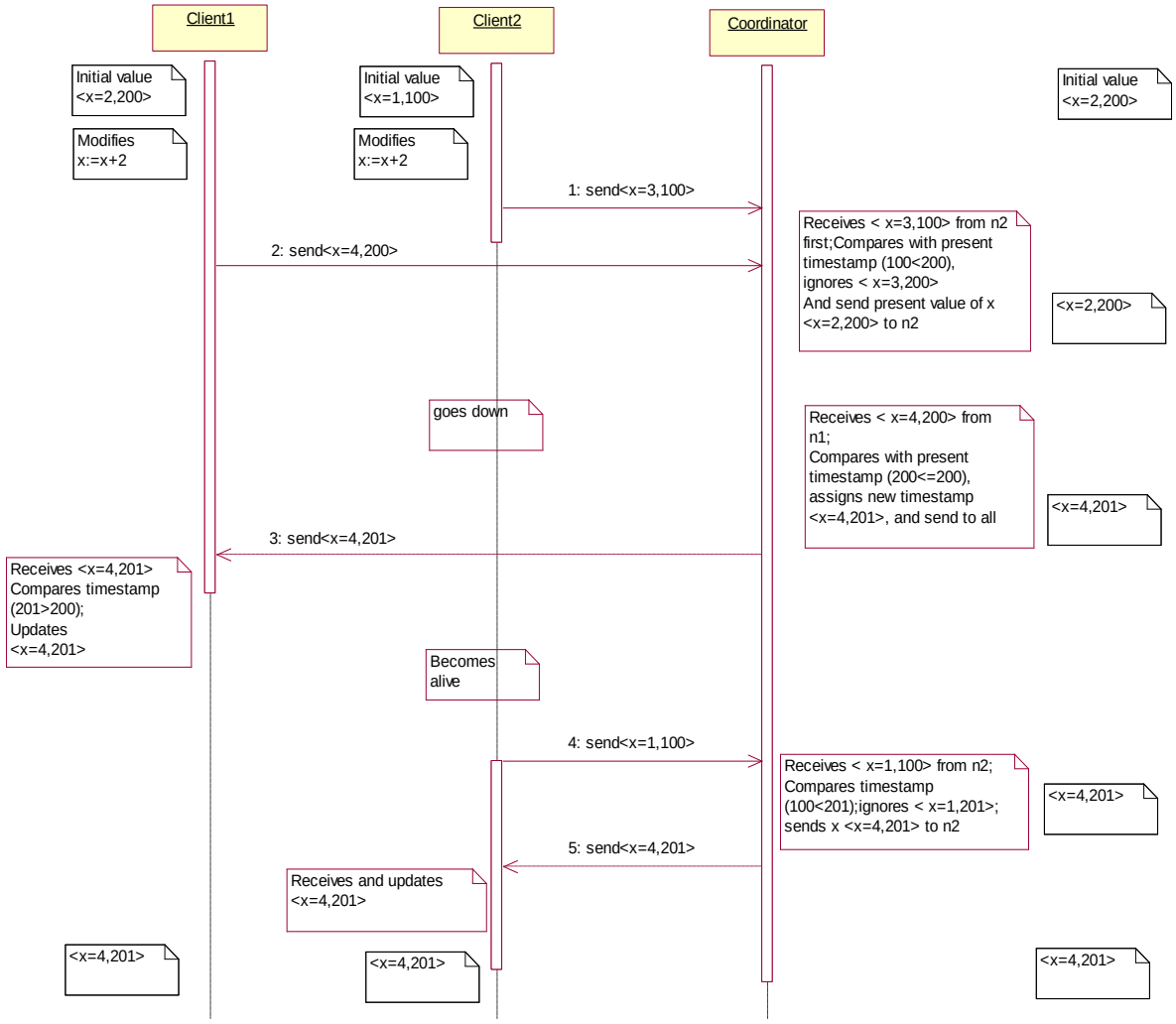

Figure 3.9: Sequence diagram for case scenario 5

The execution order detail of figure 3.9 is mentioned in table 3.8.

Execution order	***Client n1***	***Client n2***	***Processing done by Coordinator C (assignment of <value, timestamp>)***	***Global Value of <x, timestamp> at coordinator***
1	*<x=2,200> x:=x+2 (x'=4)*	*<x=1,100> x:=x+2 (x'=3)*		*<x=2,200>*
2	*SEND <x=4,200> ➔C (where 200 represents last*	*SEND <x=3,100> ➔C (where 100*		*<x=2,200>*

	timestamp received from C)	*represents last timestamp received from C)*		
3			*Receives < x=3,100> from n2 first;* *Compares with present timestamp (200),* *ignores < x=3,200>* *And send present value of x <x=2,200> to n2*	*<x=2,200>*
4			*Receives < x=4,200> from n1;* *Compares with present timestamp (200),* *assigns new timestamp* *<x=4,201>*	*<x=2,200>*
5			*Sends <x=4,201> to all*	*<x=4,201>*
6		*Goes down*		*<x=4,201>*
7	*Receives <x=4,201>* *Compares new>old timestamp(201>200, if yes* *Updates x:2 ➔4*	*Goes down*		*<x=4,201>*
8	*<x=4,201>*	*Goes down*		*<x=4,201>*
9	*<x=4,201>*	*Goes down*		*<x=4,201>*
10		*Becomes alive* *SEND <x=1,100> ➔C* *(where 100 represents last timestamp received from C)*		*<x=4,201>*
11			*Receives < x=1,100> from n2;* *Compares with present timestamp (201),* *ignores < x=1,201>;* *And* *send present value of x <x=4,201>*	*<x=4,201>*

			to n2	
12	*<x=4,201>*	*Receives and updates it* *<x=4,201>*		*<x=4,201>*
13	*<x=4,201>*	*<x=4,201>*		*<x=4,201>*

Table 3.8: Execution order of case scenario 5

3.4.6. Case Scenario 6: Two clients modify data item and 3rd one is busy in heavy computing.

The client is busy in heavy computing due to the applications e.g. parameter sweep applications. Consider the following initial values in DG environment:

1. Data item x=1.
2. Number of clients =3
3. Clients: n1(x=2,200), n2(x=1,100) , n3(x=2,200)

The sequence diagram for this case scenario is mentioned in the figure 3.10.

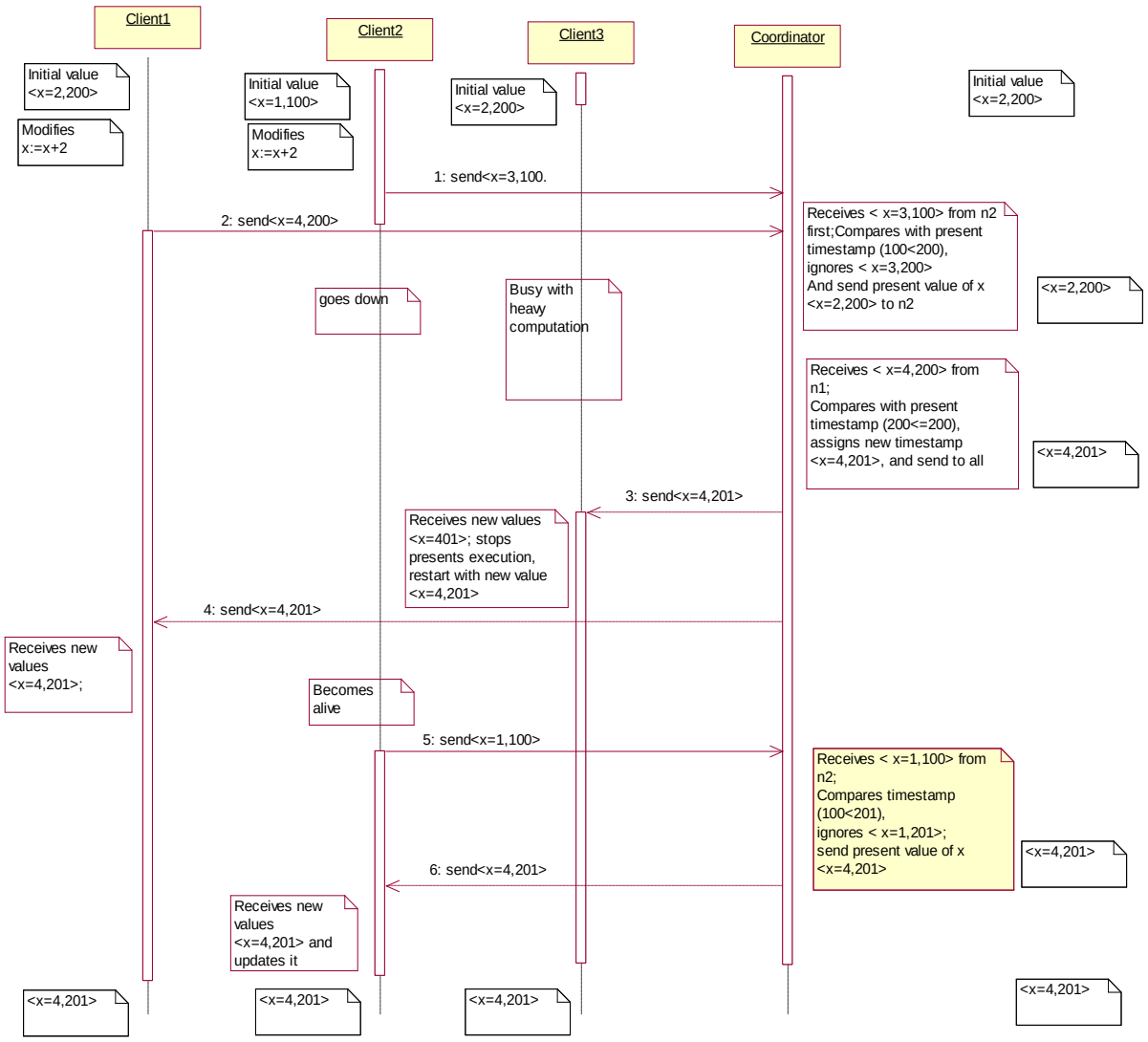

Figure 3.10: Sequence diagram for case scenario 6

The execution order detail of figure 3.10 is mentioned in table 3.9.

Execution Order	***Client n1***	***Client n2***	***Client n3***	***Processing done by Coordinator C (assignment of <value, timestamp>)***	***Global Value of <x, timestamp> at coordinator***
1	*<x=2,200> x:=x+2 (x'=4)*	*<x=1,100> x:=x+2 (x'=3)*	*<x=2,200>*		*<x=2,200>*
2	*SEND*	*SEND*	*Busy in heavy*		*<x=2,200>*

	<x=4,200>➔C *(where 200 represents last timestamp received from C)*	*<x=3,100>➔C* *(where 100 represents last timestamp received from C)*	*computation*		
3			*Busy in heavy computation*	*Receives < x=3,100> from n2 first;* *Compares with present timestamp (200),* *ignores < x=3,200>* *And send present value of x <x=2,200> to n2*	*<x=2,200>*
4				*Receives < x=4,200> from n1;* *Compares with present timestamp (200),* *assigns new timestamp* *<x=4,201>*	*<x=2,200>*
5				*Sends <x=4,201> to all*	*<x=4,201>*
6	*Receives new values <x=401>;*		*Receives new values <x=401>; stops presents execution, restart with new value*		*<x=4,201>*
7	*<x=4,201>*	*Goes down*	*<x=4,201>*		*<x=4,201>*
8	*<x=4,201>*	*Goes down*			*<x=4,201>*
9		*Becomes alive* *SEND <x=1,100>➔C* *(where 100*	*Starts doing heavy computation with new value of x*		*<x=4,201>*

		represents last timestamp received from C)			
11			*busy*	*Receives < x=1,100> from n2;* *Compares with present timestamp (201),* *ignores < x=1,201>;* *And* *send present value of x <x=4,201>*	*<x=4,201>*
12	*<x=4,201>*	*Receives and updates x* *<x=4,201>*	*<x=4,201>*		*<x=4,201>*
13	*<x=4,201>*	*<x=4,201>*	*<x=4,201>*		*<x=4,201>*

Table 3.9: Execution order of case scenario 6

3.4.7. Case Scenario 7: Two clients modifies more than 2 or more data item

3.4.7.1. Case Scenario 7.1: Two clients modifies more than 2 or more data item and data item y has dependency on x (y = y + x)

Consider the following initial values in DG environment:

1. Data item x=1, y=1
2. Number of clients =2
3. Clients: n1(x=1, 00), (y=1, 00) ; n2(x=1, 00), (y=1, 00)

The sequence diagram for this case scenario is mentioned in the figure 3.11.

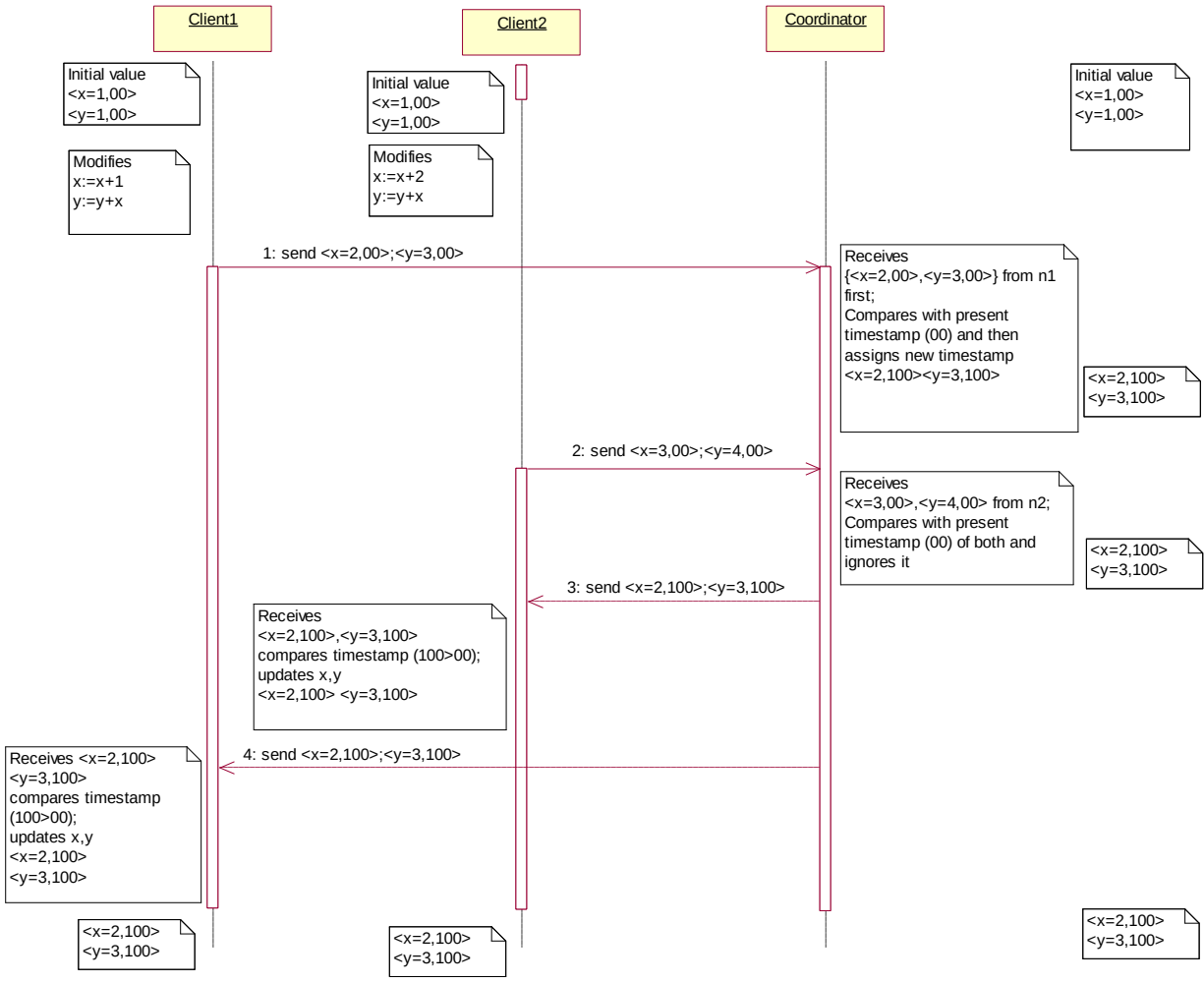

Figure 3.11: Sequence diagram for case scenario 7.1

The execution order detail of figure 3.11 is mentioned in table 3.10.

Execution order	***Client n1***	***Client n2***	***Processing done by Coordinator C (assignment of <value, timestamp>)***	***Global Value of <x, timestamp> at coordinator***
1	*x=1,y=1*	*x=1,y=1*		*<x=1,00> <y=1,00>*
2	*x:=x+1; y:=y+x;* *(x'=2, y'=3)*	*x:=x+2 y:=y+x;* *(x'=3, y'=4)*		*<x=1,00> <y=1,00>*
3	*SEND {<x=2,00>,<y=3,00>}➔C* *(where 00 represents last timestamp received from C)*	*SEND {<x=3,00>,<y=4,00>}➔C* *(where 00 represents last timestamp received from C)*		*<x=1,00> <y=1,00>*

4	*{x=1, x'=2;y=1,y'=3}*	*{x=1,x'=3;y=1,y'=4}*	*Receives {<x=2,00>,<y=3,00>} from n1 first;* *Compares with present timestamp (00) and then* *assigns new timestamp* *<x=2,100>* *<y=3,100>*	*<x=1,00>* *<y=1,00>*
5			*Send to all nodes* *<x=2,100>* *<y=3,100>*	*<x=2,100>* *<y=3,100>*
6			*Receives* *{<x=3,00>,<y=4,00>} from n2;* *Compares with present timestamp (00) of both and ignores it*	*<x=2,100>* *<y=3,100>*
7	*Receives <x=2,100> <y=3,100>* *compares new timestamp>old (100>00)* *and* *updates x,y* *<x=2,100>* *<y=3,100>*	*Receives <x=2,100> <y=3,100>* *compares new timestamp>old (100>00)* *and* *rollback x',y',* *updates x,y* *<x=2,100>* *<y=3,100>*		*<x=2,100>* *<y=3,100>*
8	*<x=2,100>* *<y=3,100>*	*<x=2,100>* *<y=3,100>*		*<x=2,100>* *<y=3,100>*

Table 3.10: Execution order of case scenario 7.1

3.4.7.2. Case Scenario 7.2: Two clients modifies more than 2 or more data item and data time y has dependency on x (y= y + x) in client n1

Consider the following initial values in DG environment:

1. Client n1(x=1,00), (y=1,00);
2. Client n2(x=1, 00).

The sequence diagram for this case scenario is mentioned in the figure 3.12.

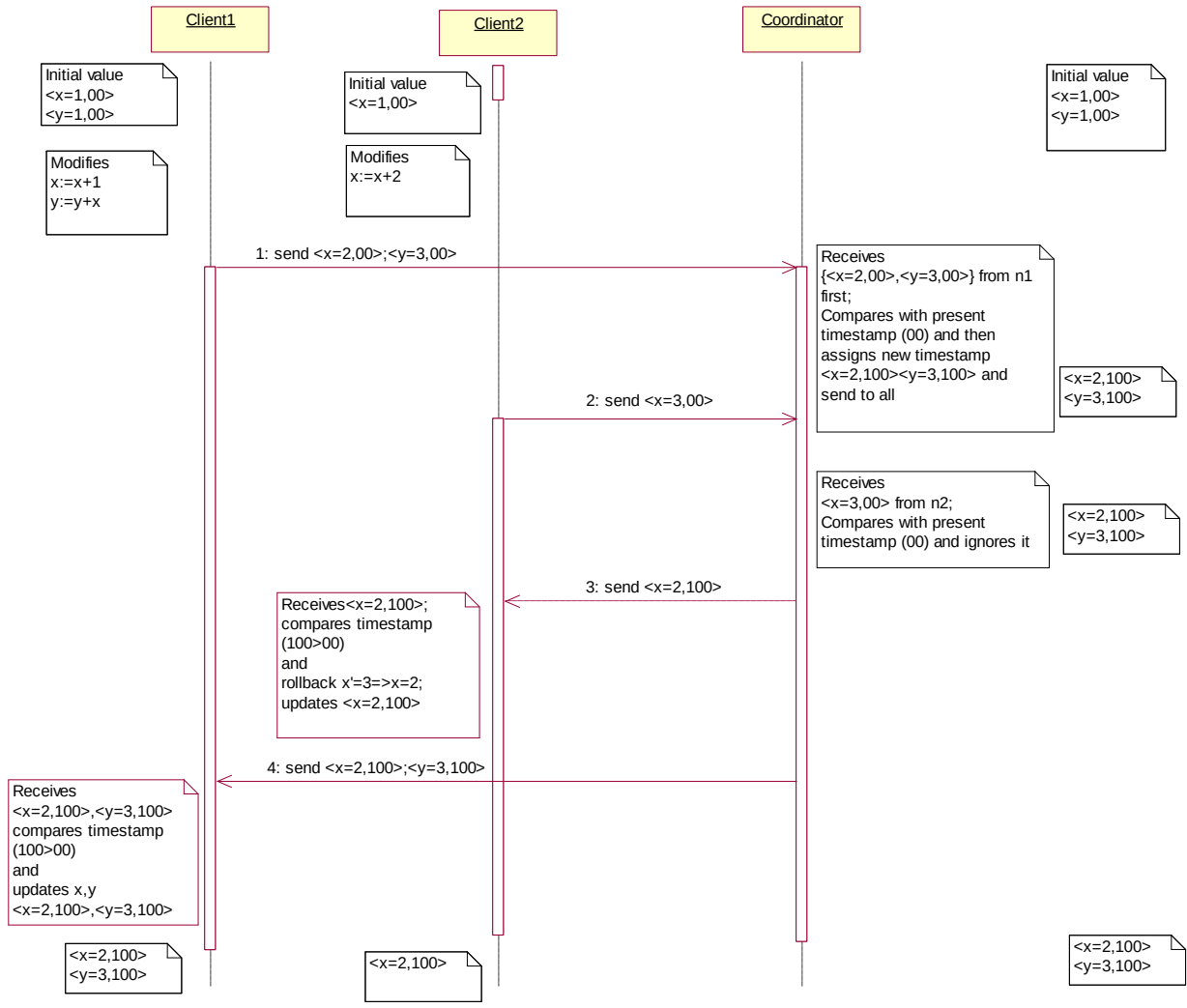

Figure 3.12: Sequence diagram for case scenario 7.2

The execution order details of figure 3.12 is mentioned in table 3.11

Execution order	***Client n1***	***Client n2***	***Processing done by Coordinator C (assignment of <value, timestamp>)***	***Global Value of <x, timestamp> at coordinator***
1	*x=1,y=1*	*x=1*		*<x=1,00> <y=1,00>*
2	*x:=x+1;* ***y=y+x;*** *(x'=2, y'=3)*	*x:=x+2*		*<x=1,00> <y=1,00>*
3	SEND	SEND		<x=1,00>

	{<x=2,00>,<y=3,00>}➔C (where 00 represents last timestamp received from C)	{<x=3,00>}➔C (where 00 represents last timestamp received from C)		<y=1,00>
4	*{x=1, x'=2;y=1,y'=3}*	*{x=1,x'=3}*	*Receives {<x=2,00>,<y=3,00>} from n1 first;* *Compares with present timestamp (00) and then* *assigns new timestamp* *<x=2,100>* *<y=3,100>*	*<x=1,00>* *<y=1,00>*
5			*Send to all nodes having x/y : {<x=2,100>|<y=3,100>*	*<x=2,100>* *<y=3,100>*
6			*Receives <x=3, 00> from n2;* *Compares with present timestamp (100) and then* *Ignores* *<x=3,00>*	*<x=2,100>* *<y=3,100>*
7	*Receives <x=2,100> <y=3,100>* *compares new timestamp>old (100>00)* *and* *updates x,y* *<x=2,100>* *<y=3,100>*	*Receives<x=2,100>; compares new timestamp>old (100>00)* *and* *rollback x' ➔x* *updates x* *<x=2,100>*		
	<x=2,100> *<y=3,100>*	*<x=2,100>*		*<x=2,100>* *<y=3,100>*

Table 3.11: Execution order of case scenario 7.2

3.4.7.3.Case Scenario 7.3: Two clients modifies more than 2 or more data item and data item y has dependency on x (y = y +x) in client n1

Consider the following initial values in DG environment:

1. Client n1(x=1,00), (y=1,00);

2. Client n2(x=1, 00).

The sequence diagram for this case scenario is mentioned in the figure 3.13.

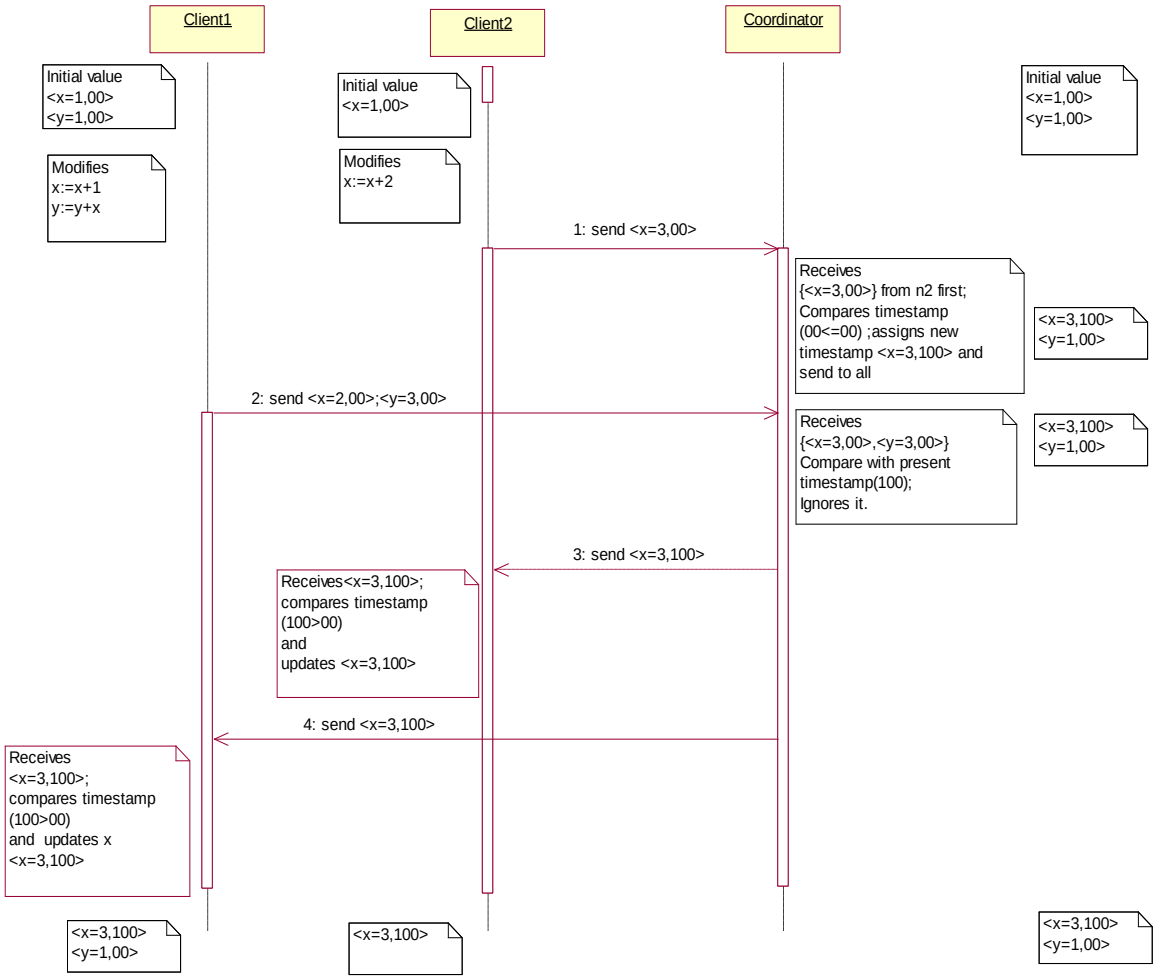

Figure 3.13: Sequence diagram for case scenario 7.3

The execution order details of figure 3.13 is mentioned in table 3.12

Execution order	***Client n1***	***Client n2***	***Processing done by Coordinator C (assignment of <value, timestamp>)***	***Global Value of <x, timestamp> at coordinator***
1	*x=1,y=1*	*x=1*		*<x=1,00> <y=1,00>*
2	*x:=x+1; y=y+x; (x'=2, y'=3)*	*x:=x+2 x'=3*		*<x=1,00> <y=1,00>*
3	*SEND*	*SEND*		*<x=1,00>*

	{<x=2,00>,<y=3,00>}➔C *(where 00 represents last timestamp received from C)*	*{<x=3,00>}➔C* *(where 00 represents last timestamp received from C)*		*<y=1,00>*
4	{x=1, x'=2;y=1,y'=3}	{x=1,x'=3}	Receives {<x=3,00>} from n2 first; Compares with present timestamp (00) and then assigns new timestamp <x=3,100>	<x=1,00> <y=1,00>
5			Send to all nodes having x : <x=3,100>	<x=3,100> <y=1,00>
6	Receives <x=3,100> compares new timestamp>old (100>00); Rollback: x'➔x; y'➔y; updates x,y <x=3,100> <y=1,00>	Receives<x=3,100>; compares new timestamp>old (100>00) and updates x <x=3,100>	Receives {<x=3,00>,<y=3,00>} Compare with present timestamp(100); Ignores it.	<x=2,100> <y=1,00>
7	<x=3,100> <y=1,00>	<x=3,100>		<x=3,100> <y=1,00>

Table 3.12: Execution order of case scenario 7.3

3.5. Conclusion

P2P-based architecture will be used for the research for handling large volumes of data in DG-based DCI. It will have peers nodes which are heterogeneous, scalable, dynamic and volatile, and have distributed control properties for data handling. Consequently, for maintaining replication and consistency simultaneously in proposed architecture, modified Two Phase Protocol along with time stamp properties will be used for handling data concurrency. New algorithms will be required to improve the overall performance in the proposed architecture.

Chapter 4

A Peer-to-Peer Architecture for handling large volumes of data in DG-based DCI

A novel P2P-based architecture is proposed for handling large volumes of data in DG-based DCI in this chapter from the analysis outcomes of data management in P2P-based solutions for desktop grid in chapter 3. The architecture's detailed working and algorithms related to the data handling of large volumes are discussed in this chapter. The first three sections discuss the three main contributions of P2P-based architecture, data consistency, and data replication algorithms. The fourth section discusses the relationship between the contributed algorithms. The fifth section concludes with all the contributions made in this research.

4.1. Proposed Desktop Grid Architecture

This section introduces a novel P2P-based architecture for handling larger volumes of data and maintaining consistency of R/W of data in DG-based DCI. The existing Desktop Grid architecture components as discussed in section 2.4 are modified to incorporate the proposed P2P techniques for handling large volumes of data. These P2P entities are in addition to existing BOINC entities in the proposed DG-based DCI architecture. The BOINC components are modified as follows to accommodate proposed architecture.

- BOINC Client: Proposed architecture consists of existing BOINC component and P2P component. P2P component is used for dealing data downloading and uploading related to the task assigned to BOINC component. P2P component communicates with coordinator for data inputs while BOINC component communicates to the coordinator for the task assignment.
- Queue Manager: It is modified for storing the information related to a query for a specific data by different clients.
- DC-API Plugin: This component is replaced by P2P coordinator and P2P client components for data distribution. The client enquires about the data location from the coordinator and then downloads the data from the data seed or other nodes.

There is no change in the BOINC scheduler and Job database components.

In the proposed architecture, the coordinator and clients will consist of P2P components as well as existing components of BOINC client [6]. P2P components are used for handling

large volumes of data and maintaining data consistency due to R/W of data. The proposed architecture is novel in the following aspects:

- It will support large volumes of data by using the P2P techniques.
- It will support R/W of data by using proposed data consistency techniques
- Replicator component will address the dynamic data planning replication strategies.

The proposed DG architecture is drawn in figure 4.1

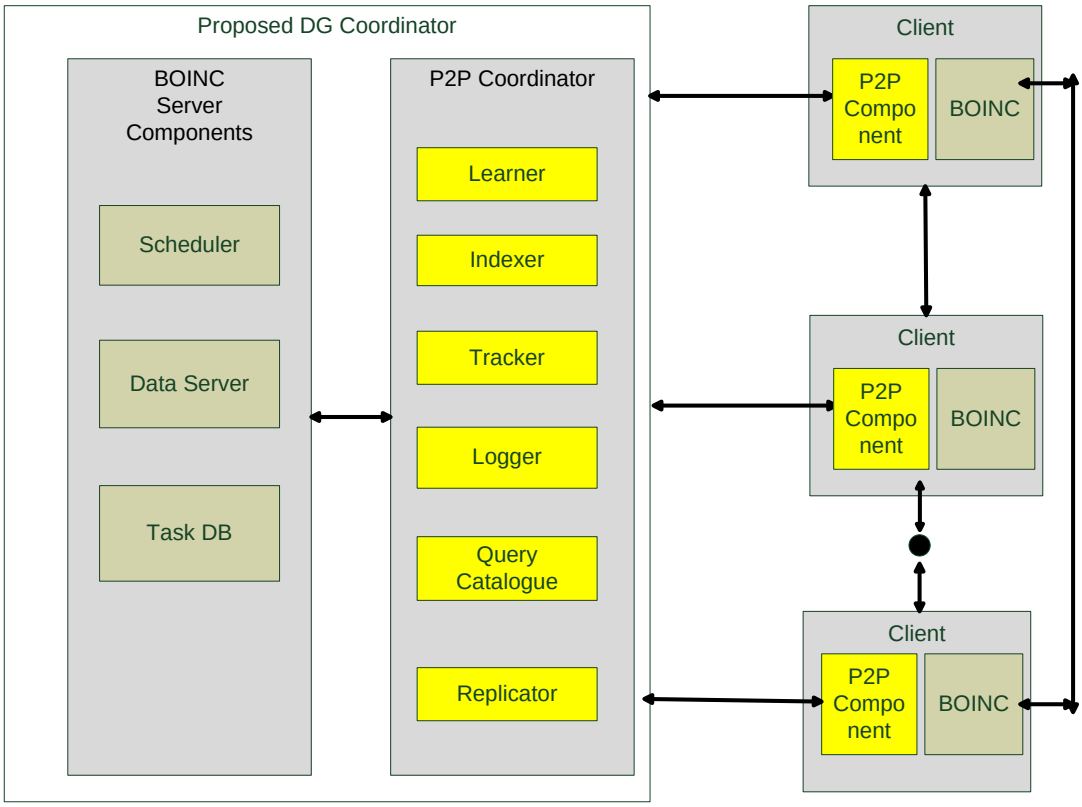

Figure 4.1: Proposed detailed DG architecture

The working of the proposed coordinator architecture is explained in the next section.

4.1.1. Coordinator Architecture

A P2P Coordinator accepts the query from the clients for Read/Write data operation in the DG-based DCI. In case of any conflict arising due to concurrent writes, it resolves it by using algorithms as mentioned in section 4.2. Proposed P2P Coordinator architecture for the R/W data replication will have following components apart from the existing architecture:

1. Logger: This component is used for storing all R/W operations for a specific data.
2. Tracker: This component is used for maintaining information about the location of clients.
3. Replicator: This component is used for determining the frequent access data paths for the clients where data replication should be applied. It finds the frequent access patterns of data from the data obtained from the logger and query catalogue.

4. Learner: This component uses the conflict resolving algorithm for any conflicts that arises during the concurrent write operations by the clients.
5. Indexer: This component stores the data related to the other clients who have access to the modified data.
6. Query catalogue: This module is used for storing the information related to the query for a specific data by different clients.
7. Data Storage element: It is used for storing the data in the forms of files.
8. P2P Mediator: It establishes and maintains P2P communication between nodes.

The proposed architecture for the P2P coordinator with new components is shown in figure 4.2:

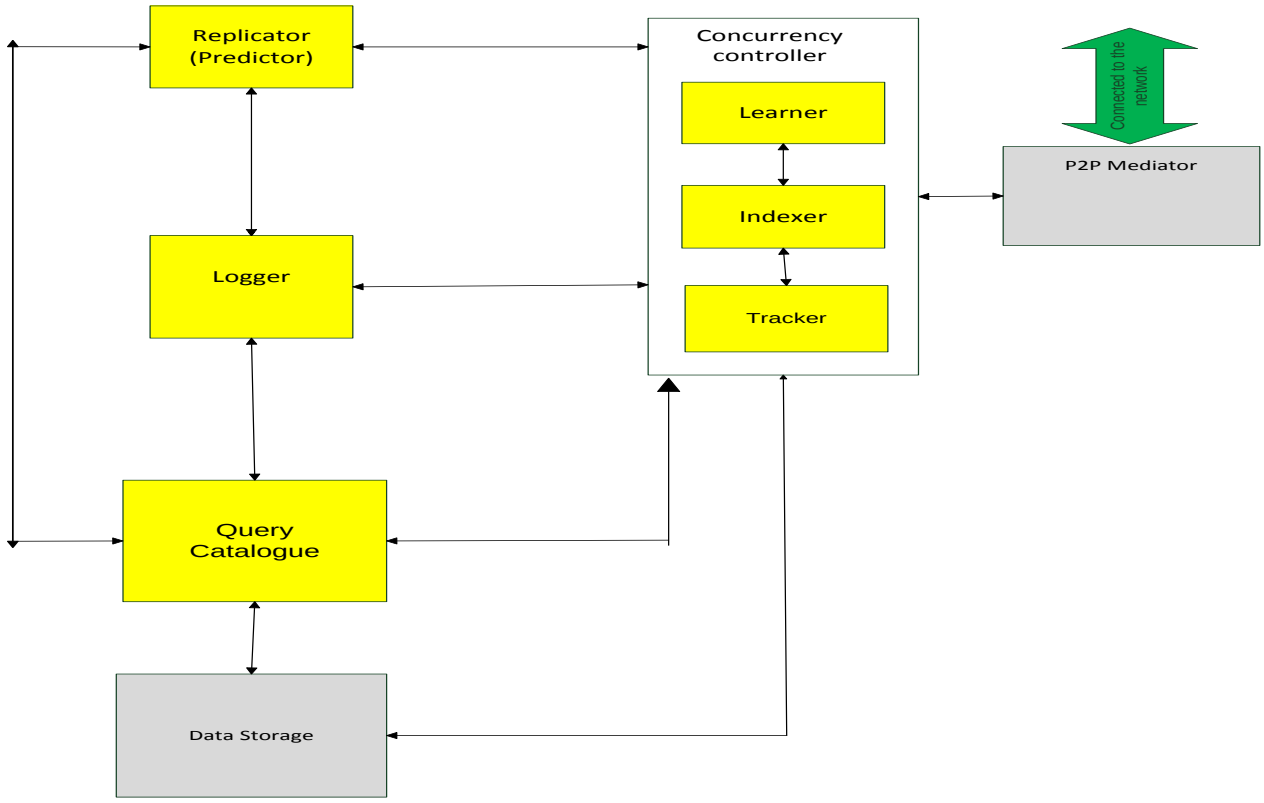

Figure 4.2: Proposed P2P Coordinator architecture

The components with yellow background are new and have been added to the existing architecture.

When a request from P2P client comes to the coordinator, first it goes to the concurrency controller which consists of leaner, indexer, and tracker. Learner calls the conflict resolving algorithms to resolve the concurrent write operations by using the logger and

indexer. R/W operation details along with the nodes and data accessed is recorded in the logger. Query catalogue records the details of any query done by the clients for a particular data. Depending upon the data in logger and query catalogue, replicator will call the data replication algorithms. Replication algorithms will first finds the frequent access paths and then apply it to improve the replication performance. After regular intervals, the data statistics from replicator component of P2P clients is collected for the analysis. The working of the proposed client architecture is explained in the next section.

4.1.2. Client Architecture

DG-based DCI consists of many P2P clients in the proposed architecture. The client sends request to the coordinator for data in the DG-based DCI. Depending upon the data, the client receives the list of the clients having the desired data from the coordinator. All the components are local to a client are used for improving its replication performance. A request is send to the P2P coordinator whenever data is updated by the client. Proposed P2P client architecture for the R/W data replication will have following components apart from the existing architecture components:

1. Logger: This component is used for storing all the R/W operations related to a specific data in P2P client. This component is local to the client.
2. Replicator: This component determines which data has to be replicated in the network depending upon the analysis of logger. It acts as a distributed component which sends this information to coordinator.
3. Indexer: It is also used for storing the information/data related to the query for a specific data by different clients/coordinator. This component is local to the client.
4. Data Storage element: It is used for storing the data in the forms of files.
5. P2P Mediator: It establishes and maintains P2P communication between client and coordinator.

The proposed architecture for the P2P client with new components is shown in figure 4.3:

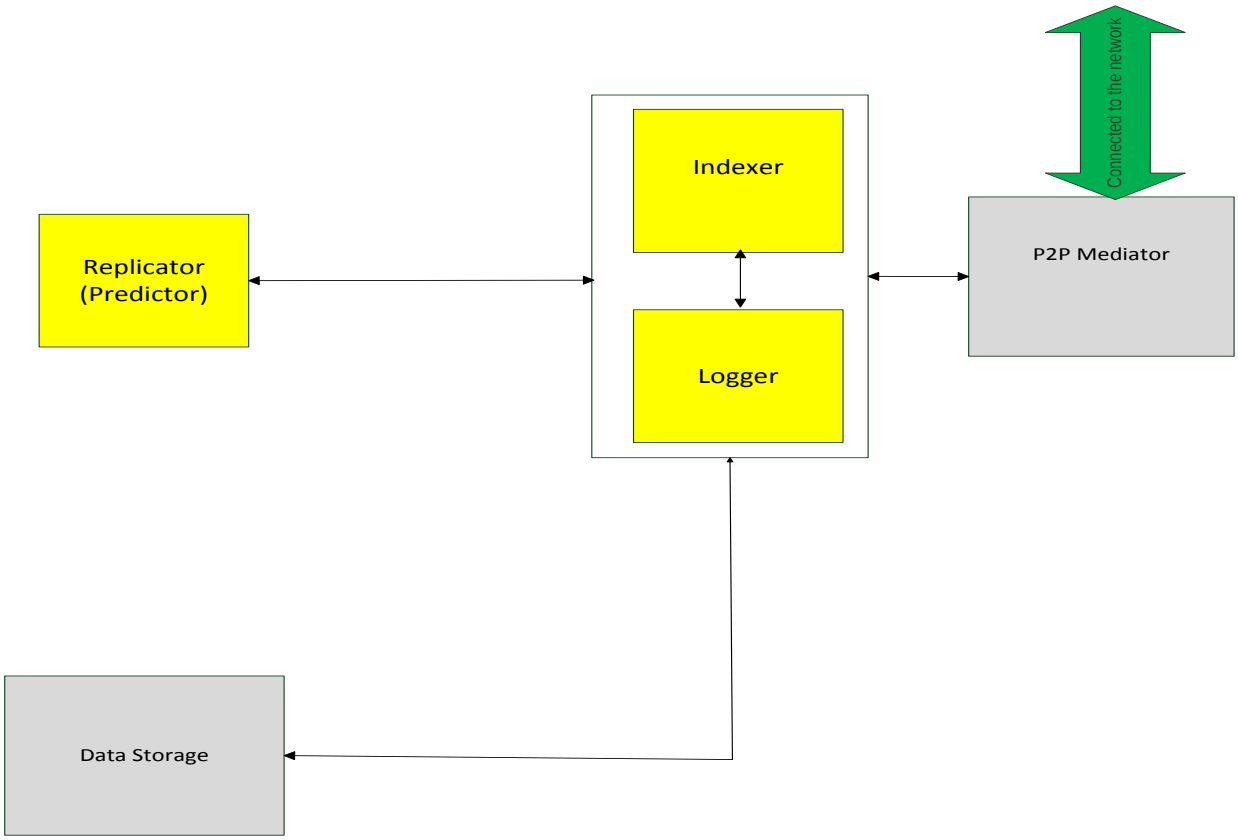

Figure 4.3: Proposed P2P Client Architecture

The next section compares the above mentioned architecture with the existing architecture.

4.1.3. Comparison of Existing and Proposed architecture

The comparison of existing DG architecture at University of Westminster [17] and proposed architecture is mentioned in table 4.1.

	Present architecture of DG		Proposed P2P client		Proposed P2P coordinator	
	Yes/**N**o (Y/N)	**C**entralised/ **D**istributed (C/D)	**Y**es/**N**o (Y/N)	**L**ocal/ **D**istributed (L/D)	**Y**es/**N**o (Y/N)	**C**entralised/ **D**istributed (C/D)
Replicator	N	-	Y	D	Y	D
Logger	N	-	Y	L	Y	D
Tracker	N		N	-	Y	D
Query Catalogue/Manager	Y	-	N	-	Y	C
Learner	N	-	N	-	Y	C
Indexer	N	-	Y	L	Y	D

Table 4.1: Comparison of the components present in the proposed architecture

Where,

Y means that the particular component is present in the architecture.

N means that the particular component is not present in the architecture.

C means centralised.

L means local.

D means distributed.

Attic [16] is an existing P2P-based solution for handling large volumes of data in desktop grid. So, the comparative analysis of Attic with the proposed architecture for handling large volumes of data is mentioned in table 4.2.

Sl. No.	Feature	Attic	Proposed Architecture
1	P2P Data transfer between Data Centre and worker	Yes	Yes
2	P2P Data transfer between worker and worker	No	Yes
3	R/W Data Conflict resolving	No	Yes
4	R/W Data Consistency	No	Yes
5	Dynamic Replication	No	Yes
6	Distributed Tracker	No	Yes
7	Client Replicator	No	Yes
8	Query Catalogue	No	Yes

Table 4.2: Comparison of Attic and proposed architecture

The next section deals with the suitability of the proposed DG architecture for R/W of data requirements in emerging applications.

4.1.4. Suitability of the proposed architecture for R/W of data

The suitability of proposed architecture for R/W of data is considered for the emerging case scenario as described in section 2.5 of chapter 2. The figure 4.4 is used for the explanation of suitability of proposed architecture for described below 4 cases.

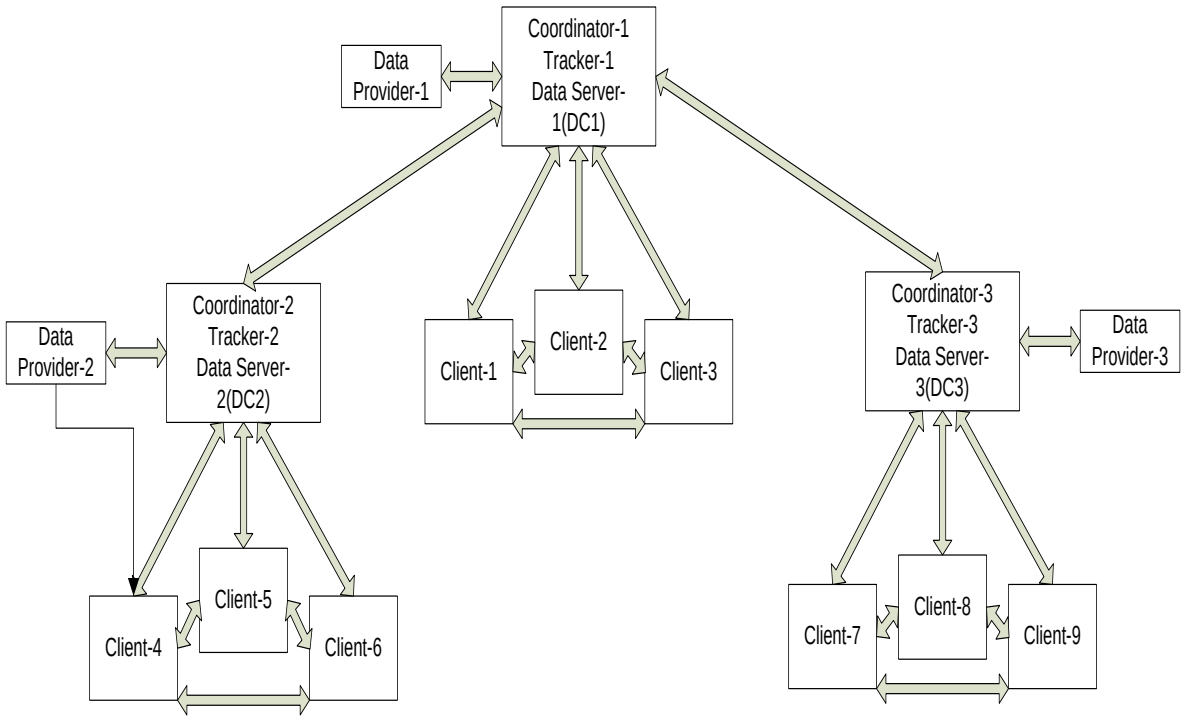

Figure 4.4: Suitability of proposed architecture in emerging applications

The above figure consists of 3 different sites. Each site has one coordinator and 3 clients. Coordinator is also acting as a tracker and data server/data centre. A site can provide the initial data through data provider. Then data is replicated to other sites through coordinator and clients. In case of data update, the update has to be synchronised to all the sites for maintaining data consistency.

The four case scenarios as described in section 2.7 for emerging applications are checked for the suitability of proposed DG architecture. The data server of proposed architecture is assumed to the data centre in the cases described below:

4.1.4.1. Case 1 (Read only data available at a single Provider)

1. Data is published to a tracker of coordinator by Data Provider. Data Centre (DC) DC1 downloads the data after querying the data location from tracker.
2. Other Data Centres DC2, DC3 query the tracker and download the data by P2P replication from DC1.
3. After some period of time all the Data Centres will have the same data when all the data is downloaded by DC2 and DC3.
4. Clients download the data for their work unit from Data Centres after querying the location from DLS.

Efficient replication strategies improve the performance of replication by reducing latency, bandwidth, number of replicas, and replicating operations. In proposed architecture, data replication between different Data Centres will apply efficient replication strategies using proposed algorithms. This will improve the overall performance of data replication in the proposed architecture.

4.1.4.2. Case 2 (Read only data available at multiple Providers)

1. Different set of data is published to tracker by different Data Providers. Data Centre DC1 downloads the data from first location after querying tracker.
2. Later Data Centre DC2 downloads the same data from second location after querying tracker.
3. Data Centres DC1, DC2, and DC3 query tracker and download the data not available at their site from other Data Centres by P2P replication.
4. After some time of time, all the DC's will have same data.

This case scenario is handled in the proposed architecture by using the distributed replication component present in coordinator and client. This will provide data replication support to multiple providers in the proposed architecture.

4.1.4.3. Case 3 (Updated data available at Data Centre(s))

1. Initially, assume that all the 3 Data Centres (DC1, DC2, and DC3) have the same set of data. Researchers use data available at each Data Centre for their analysis.
2. After some period of time, researchers at one Data Centre DC1 contribute new and updated data.
3. This new and updated data will be communicated to Coordinator. Coordinator will resolve the R/W data conflicts at one site and then will communicate the change to other coordinators.

In the proposed architecture, this case scenario is handled by resolving the Read/Write data conflicts by the coordinators in Data Centres. Data consistency is maintained at all the 3 Data Centres by applying proposed algorithms. This will improve the performance of data consistency in the proposed architecture.

4.1.4.4. Case 4 (Clients download the data from other clients)

In proposed architecture, the clients will also act as data source to other clients. This will improve the overall data replication performance in the proposed architecture.

Proposed architecture is more suitable for the emerging application requirements of data Read/Write in DG environment due to the following reasons:

1. The improvement in performance is achieved by applying dynamic replication strategies.
2. The performance of data consistency is improved by using concurrency mechanism for handling R/W of data.
3. The replication performance is further improved by supporting the P2P data transfer between the client nodes.

Thus, proposed architecture is more suitable for the emerging application requirements by maintaining replication and consistency simultaneously. The working of new components present in the proposed architecture of P2P coordinator and P2P clients are mentioned in the next section.

4.1.5. Algorithms used in proposed architecture

As per the FSM of coordinator and client in section 3.3., coordinator resolves data conflict and maintains data consistency between different clients. Two algorithms are needed for addressing data consistency issues. The requirements for data consistency algorithms are as follows:

- Clients will send modified data to Coordinator.
- Coordinator should resolve data conflicts by using time stamping [82].
- Coordinator should maintain data consistency between clients.

The proposed architecture of P2P coordinator and P2P clients interact with each other for R/W operations and to improve the overall performance of the system. The proposed architecture use two algorithms for maintain data consistencies are as follows:

1. Conflict Resolving Algorithm (Algorithm-1)**:** This algorithm is used for resolving conflicts when the concurrent Read/Write data operations(s) are submitted to the coordinator. The detailed working of algorithm is mentioned in section 4.2.1.
2. Consistency Algorithm (Algorithm-2)**:** This algorithm is used by P2P Coordinator for maintaining data consistency in the DG-based DCI environment due to Read/Write data operations. The detailed working of algorithm is mentioned in section 4.2.2.

The modified data is later replicated to all other clients. Algorithms are needed for addressing data replication. The requirements for data consistency algorithms are as follows:

- It should measure the replication performance along different paths in the DG.
- It should then find frequent paths for replication in the DG to improve performance.

The proposed architecture uses two algorithms for efficient data replication in order to improve the performance. The algorithms are as follows:

3. Replication performance measurement Algorithm (Algorithm-3): This algorithm measures the performance of data replication of Algorithm-1 and Algorithm-2 in the DG-based DCI environment. The detailed working of algorithm is mentioned in section 4.3.1.
4. Replication performance improvement Algorithm (Algorithm-4): This algorithm is used for planning data distribution strategies. It firsts finds the frequent access paths of data replication from the data statistics of Algorithm-3. Then, this algorithm applies data replication in these frequent access paths to improve the replication performance. The detailed working of algorithm is mentioned in section 4.3.2.

4.2. Data Consistency Algorithms

Concurrency control techniques [46] are widely used in a distributed system to resolve Read/Write and Write/Write conflicts that arises due to concurrent R/W operations. In our proposed architecture, we have used Algorithm-1, and Algorithm-2 to address this issue in DG-based DCI. The following assumptions are considered for the proposed algorithms:

1. P2P Coordinator is up all the time.
2. The P2P client's may join/leave network at any time.
3. Initially, client 1 sends the request for the modification of the data to the coordinator and then the subsequent process starts.

4.2.1. Conflict Resolving Algorithm (Algorithm-1)

This algorithm is used in P2P coordinator for resolving the conflict due to R/W data operations between different P2P clients. Conflicts occur when two or more transactions acquire/commit the modified data item concurrently. Due to this, different nodes in DG-based DCI will have inconsistent data. This problem can be resolved by having a timestamp along with the modified data item. In our algorithm, the coordinator creates a time stamp for each of the modified data time received. Each data item is associated with a timestamp at the coordinator. The assumed value of timestamp is initialised from zero for all data items in order to minimize the

complexity that might rise by using the real clock value. Furthermore to add the simplicity, this timestamp value will increase by one. The participating members in this algorithm are as follows:

1. P2P Clients (C_1, . , . , C_n)
2. Coordinator (L)

Algorithm-1 consists of two parts:

1. Algorithm-1_Client (Client side): In this part, Client C_i sends new modified value(s) to Coordinator L along with current timestamp.
2. Algorithm-1_Coor (Coordinator side): In this part, Coordinator resolves conflicts of modified data item by creating new timestamp depending upon some criterions.

The working of algorithm is as follows:

1. *P2P Client(s) (C_1, C_2, etc) sends a request for data change along with its current timestamp to the Coordinator L.*
2. *Coordinator receives the change request from different nodes. It puts into queue according to the received order.*
3. *Coordinator selects first request from the queue.*
4. *It checks received time stamping value of data item from a node.*
    1. *If timestamp is equal to the global value, it assigns a new time stamping t (greater than the last stored global value for this data item) to the data value received.*
        1. *Then it checks, all the nodes (C_1,.,., C_n) having the same data in its list.*
        2. *Then it sends the new value of data item, along with new timestamp to all the nodes.*
        3. *Then it checks for next request from queue.*
    2. *Else it ignores the modified data item, and checks for next request.*

The following notations are used in Algorithm-1:

1. x: Data item x,
2. v': Value of new data item x,
3. t: Current timestamp of data item x whose initial value is considered to be 00,
4. T: Global timestamp of data item x in coordinator,
5. Δt: Change in timestamp whose value is assumed to be 1 for simplicity,
6. t': New timestamp of data item x,
7. C_i: Client i,

8. L: Coordinator.
9. ➔: indicates is changed to.

Algorithm-1

Algorithm-1_Client

Input: [x (v'), t] (Client C_i sends modified value of data item x along with current timestamp)

Output: Modified value is accepted/ rejected by coordinator

1. *Client modifies the value of data item x: x(v)➔x(v')*
2. *Send [x(v'),t] ➔ coordinator (L)*
3. *Wait for response from L*
    1. *If Response=Yes,*
        1. *Modified value accepted*
    2. *Else*
        1. *Modified value rejected*

Algorithm-1_Coor

Input: [x (v'), t] (Modified value of data item x along with timestamp t)

T (Current timestamp of data item x in coordinator)

Output: Timestamp generated or not for modified data item x(v')

1. *Receive [x(v', t)]*
2. *Checks for global timestamp for data item x*
    a. *If T=t then*
        i. *Generate a new timestamp t'=t + Δt for data item x,*
        ii. *T=t',*
        iii. *Sends [x (v', t')] to all the clients having data item x.*
    b. *Else*
        i. *Ignore the request*

4.2.2. Consistency Algorithm (Algorithm-2)

This algorithm is used for maintaining data consistency of R/W operations between different P2P clients. The consistency management is handled by the P2P coordinator. It uses the modified version of the Two Phase Protocol (2PC) [46] to resolve the conflict. 2PC is used for coordinating all the processes that participate in distributed systems to commit or abort the transaction. It consists of two phases: Request, and Commit Phase.

Depending upon the outcome of all the processes, the P2P coordinator takes the decision to commit or abort the changes for a particular transaction. The participating members in this Algorithm are as follows:

1. P2P Clients (C_1, . , . , C_n)
2. Coordinator (L)

Major steps involved in this algorithm are as follows:

1. P2P client will sent a request for data change to the Coordinator L.
2. Coordinator L will provide the necessary time stamping for the modified data value sent by the P2P client.
3. In DG-based DCI for maintaining data consistency for the modified data with other P2P clients involves messages sending by the coordinator to other P2P clients. Message sending involves 3 steps for maintaining data consistency:
    1. Request: Modified data send by the P2P client to the coordinator.
    2. Prepare: Coordinator send the modified data along with new time stamp to the P2P clients having the same data.
    3. Commit: P2P clients receive the data from the coordinator, commit the change(s) in data and then send the acknowledgement to the P2P coordinator. Coordinator then sends the acknowledgement to the P2P client who has initiated the request for the modified data.

The working of algorithm is as follows:

1. *P2P Client(s) (C_1, C_2, etc) sends a request for data change along with its current timestamp to the Coordinator L.*
2. *Coordinator receives the change request from different nodes. It puts into queue according to the received order.*
3. *Coordinator selects first request from the queue.*
4. *It checks received time stamping value of data item from a node.*
    1. *If none of the clients have the same data, L sends an acknowledgement (ack) to the C_1 to commit the changes.*
    2. *If timestamp is equal to the global value, it assigns a new time stamping t (greater than the last stored global value for this data item) to the data value received.*
        1. *Then it checks all the nodes (C_1,.,., C_n) having the same data in the list.*

2. *Then it sends the new value of data item, along with new timestamp to all the nodes.*
3. *Then it checks for next request from queue.*

3. *Else it ignores the modified data item, and checks for next request.*

5. *P2P Client(s) checks the value of the timestamp received from the coordinator,*
    1. *If the value of timestamp t (is first time received), client accepts it and send an acknowledgement to L.*
    2. *If the value of timestamp t is greater than the previous values it promises to ignore all the previous values, accepts it, and send an acknowledgement to the L.*
    3. *If the value of timestamp t is less than the previous values, it ignores it.*
6. *Coordinator wait's for the acknowledgement for the new timestamp t from the list of the clients as mentioned in 4.2.1. for a time period.*
    1. *If no ack is received from the clients, then go to step no 4.2.2.*
    2. *Else commits the data and informs P2P client who has initiated the request for modified data.*
7. *The value v and the time stamping details are stored in the logger, indexer, and learner in P2P coordinator and P2P clients.*

The following notations used in Algorithm-2:

1. x: Data item x,
2. v': Value of new data item x,
3. t: Current timestamp of data item x,
4. T: Global timestamp of data item x in coordinator,
5. Δt: Change in timestamp whose value is assumed to be 1 for simplicity,
6. t': New timestamp of data item x,
7. C_i: Client i,
8. L: Coordinator.
9. ➔: indicates is changed to.

Algorithm-2 consists of two parts:

1. Algorithm-2_Client (Client side): In this part, Client C_i sends new modified value to Coordinator L along with current timestamp.
2. Algorithm-2_Coor (Coordinator side): In this part, Coordinator maintains the consistency of modified value of data item in the network.

Algorithm-2

Algorithm-2_Client

Input: Modified data along with current timestamp [x(v'), t]

Output: Modified value x(v') is committed/rolled back

1. *Client modifies the value of data item x(v)➔x(v')*
2. *Sends [x(v'),t] to the coordinator L*
3. *Wait for response from L*
    1. *If Response=Yes,*
        1. *Modified value accepted [x(v'), t'].*
        2. *Commit the modified values.*
        3. *Sends the acknowledgement to coordinator*
    2. *Else*
        1. *Modified value rejected*
        2. *Rollback to previous values [x(v),t]*

Algorithm-2_Coor

Input:

Modified value [x (v'), t]

Current timestamp t,

List of clients (C_2, ., ., C_n) having value x(v)

Output: Modified value accepted(Y)/Rejected (N)

1. *Receive [x(v', t)]*
2. *Checks for global timestamp for data item x*
    a. *If T=t then*
        i. *Generate a new timestamp t'=t + Δt for data item x,*
        ii. *T=t',*
        iii. *Check all the clients in list having same data item x*
        iv. *If List is null*
            1. *Then sends [ack,[x(v'),t']] to Client C_i*
        v. *else*
            1. *Then send [x(v'),t'] to all clients in List*
            2. *Waits from acknowledgement from clients*
            3. *Send [x(v'),t'] to client C_i*
    b. *Else*
        i. *Ignore the request*

Each client in List L receives [x(v'), t']

Do

1. *For value x(v'),*
2. *If t< t', // for all previous value of times stamp or for first time stamp*
    i. *Ignores all previous value of x(v)*
    ii. *Commits new data time and timestamp value [x(v'),t']*
    iii. *Send [ack, x(v'),t'] to Coordinator*
3. *Else if t> t'*
    i. *Ignores x(v'),*
    ii. *Sends previous value and timestamp [x(v),t] to Coordinator*

Done

4.3. Data Replication Algorithms

Data replication algorithms are used for measuring and improving the performance of data replication in DG-based DCI. In proposed architecture, Algorithm-3 and Algorithm-4 are used to address data replication. Algorithm-3 deals with measurement of performance of data replication in DG while Algorithm-4 deals with planning data distribution strategies in order to improve data replication performance.

4.3.1. Replication performance measurement Algorithm (Algorithm-3)

This algorithm is used for measuring the performance of replication in the DG-based DCI. The performance of replication is depending upon the factors like number of clients, dataset size, number of pieces in dataset, number of pieces modified, number of tasks, size of task, routing of the network, bandwidth of link, latency of link, etc.

Algorithm-3

1. *Measure the following values from the data collected in Algorithm-1and Algorithm-2:*
    a. *Measurement of* total tasks execution time and number of messages passed *for* varying size of data.
    b. *Measurement of* total tasks execution time and number of messages passed *for* varying number of modifications of data.
    c. *Measurement of* total tasks execution time and number of messages passed *for* varying number of clients.

d. *Measurement of* total tasks execution time and number of messages passed *for* varying number of size of task.
e. *Measurement of* total tasks execution time *for* varying number of task set.
f. *Measurement of* total tasks execution time *for* varying bandwidth
g. *Measurement of* total tasks execution time *for varying* latency of link.

2. *Determination of most significant parameters from the above measurements.*
3. *Compare the replication performance on these significant parameters for the proposed and existing architecture in Desktop Grid in DCI.*

4.3.2. Replication performance improvement Algorithm (Algorithm-4)

Algorithm-4 is used for planning the data distribution strategies and improving data replication performance in DG-based DCI environment. Algorithm-4 first find the frequent data access path patterns for data replication from the data replication statistics collected by Algorithm-3 in P2P coordinator and P2P clients. It then applies the data replication on the frequent access paths to improve the performance. This algorithm works in both P2P coordinator as well as in P2P clients.

Two algorithms have been selected for generating the frequent access paths for the data replication:

1. Frequent Pattern (FP) growth algorithm [19]: This algorithm is selected for generating the frequent paths of replication. This algorithm has better performance **[19]** as compared to other existing data mining algorithms in finding frequent paths of replication.
2. Prim's Minimum Spanning Tree algorithm [47]: This algorithm is widely used for finding the shortest path in a network. This algorithm is selected in order to replicate data with minimum cost in the spanning tree of the network. Presently, this algorithm assumes stable nodes in a network. This algorithm is modified to handle network with volatile nodes similar to DG environment.

4.3.2.1. Frequent Pattern (FP) growth algorithm

FP growth algorithm [19] is used to generate the rules for the different data access patterns for the R/W data. This algorithm is used for generating the frequent access patterns from the data stored in P2P clients and P2P coordinator. This algorithm compresses the dataset representing frequent items into a frequent pattern tree, which retains the item set association information. It then divides the compressed dataset into a set of conditional

dataset; each associated with one frequent item, and mines each dataset separately. For each pattern fragment only its associated data sets need to be examined.

The following notations are used in FPgrowth algorithm:

1. D: A dataset of items,
2. F: A set of frequent items,
3. L: List,
4. p: First element in list,
5. P: Remaining items in list,
6. T: Tree,
7. Support count: It is the number of transactions in which a data item value appears in whole set of transactions.

This algorithm consists of two major steps:

1. Construction of Frequent Pattern tree (FP_Tree),
2. Mining FP_Tree for finding frequent item sets.

FP_growth Algorithm

Input:

- A dataset D consists of data item accesses along with the P2P client.
- Minimum support threshold value: Initially, minimum support threshold value is varied to generate a set of rules. Then, a stable value is selected depending upon the number of rules generated.

Output: The complete set of frequent patterns.

1. Construction of Frequent Pattern tree

1. *Scan the dataset D. Collect the set of frequent items (F) and their support counts. Sort F in descending order of their support count.*
2. *Create the root of a FP-tree, and label it as "null." For each row in D do the following:*
    a. *Select and sort the frequent items in D according to the order of L.*
    b. *Let the sorted frequent item list be [p|P], where p is the first element and P is the remaining list.*
    c. *Call insert_tree([p|P], T), which is performed as follows:*

i. *If T has a child N such that N.item-name = p.item-name, then increment N's count by 1;*

ii. *Else create a new node N, and let its count be 1, its parent link be linked to T, and its node-link to the nodes with the same item-name via the node-link structure.*

iii. *If P is nonempty, call insert_tree (P, N) recursively.*

The above constructed FP_Tree is mined by calling FP_growth (FP_Tree, null) as mentioned below:

2. Mining FP_Tree

FP_growth (Tree, α)

1. *If Tree contains a single path P then*
    a. *For each combination (denoted as β) of the nodes in the path P*
    b. *Generate pattern β U α with support_count = minimum support count of nodes in β;*
2. *Else for each a_i in the header of Tree*
    a. *Generate pattern β =a_i U α with support_count=a_i.support count;*
    b. *Construct β's conditional pattern base and then β's conditional FP_treeTree$_\beta$;*
    c. *If Tree$_\beta \neq \emptyset$ then*
        i. *Call FP_growth (Tree$_\beta$, β);*

Once the frequent access path patterns are discovered by the FP growth, then the Algorithm-4 replicates the data in the frequent access path patterns in order to improve replication performance.

4.3.2.2. Prim's Minimum Spanning Tree algorithm

A minimum spanning tree (MST) defines the subset of edges that connects all the clients in a network in a cost effective (minimum) manner without forming a cycle. The client that provides the initial data acts as the starting node for this algorithm.

The following notations are used in Prim's algorithm [47]:

1. G: A set of edges in graph,
2. V: A set of vertexes,
3. R: Root node,

4. Q: Min-priority queue,
5. v.key: Minimum weight of any edge connecting v to a vertex in the tree,
6. v.π: Names the parent of vin the tree,
7. w: Weight of the edge.

The working of the Prim's algorithm is as follows:

1. Start from the first node.
2. Find the edge that has minimum weight from all known nodes.
3. Stop when the tree covers all the nodes.

Prim's algorithm (G, w, v) [47]

1. for each u Є G.V
    a. u.key = ∞
    b. u. π =NIL
2. r.key =0
3. Q = G.V
4. while Q≠Ø
    a. u =EXTRACT-MIN(Q)
    b. for each v Є G.Adj[u]
    c. if v Є Q and w(u,v) <v.key
        i. v. π =u
        ii. v.key=w(u,v)

This algorithm generates the spanning tree which connects all the clients in where the replication cost is minimum i.e. more replication will be done in minimum cost.

In DG-based DCI, the clients can leave or join the network at instance of time. Since FP_growth and existing spanning tree algorithms consider stable clients. So, these algorithms have to be modified in order to adapt the change in behaviour of clients in DG-based DCI. A new algorithm is proposed in the next section to the dynamic and volatile environment of DG-based DCI.

4.3.2.3. Adaptive multipath Spanning Tree algorithm (Algorithm-4)

A new adaptive multipath spanning tree for replication has been proposed in DG environment. This algorithm uses the concept of minimum spanning tree. Initially, the k-

path spanning tree algorithm is used for finding the optimal path for replication in DG environment. It uses the concept of finding the best paths in the DG-based DCI. A best path is defined as the cost of edge of each client (minimum cost to go from source client to destination client neighbour). A P2P client stored the edge cost of its neighbours, so this cost is updated if some neighbours leave/join the network. k signifies the number of best paths stored in a client in the network. Initially, the value of k is assumed to be equal to 2. It forms the set of MST by using the existing prim's algorithm.

The following notations are used in this algorithm:

1. G: A set of edges between clients,
2. G_0: A set of edges in initial graph,
3. V:A set of vertexes represented by clients,
4. S^k: A set of k-Minimum Spanning Tree.

The algorithm uses the following steps for building the adaptive multipath spanning tree:

1. Each node stores k-best paths for the spanning tree.
2. Once a client leaves the network in S_k, the cost of edge (between sources to destination) increases, and then the spanning tree is updated by using the next best path in S_{k+1}
3. Once a new client joins the network in S_k, if the cost of the edge (between sources to destination) decreases, and then the new path is added to the spanning tree.

Adaptive multipath Spanning Tree (Algorithm 4)

Input: k value

Output: Set of multi-path MST: $S^K = S_1 \cup S_2 \ldots US_k$

1. *Go=G*
2. *S_0=null*
3. *for i=(1..k)*
4. *Do*
5. *S_i=MST (Gi-1)*
6. *$S^i=S^{i-1} \cup S_i$*
7. *$G_i=G_{i-1} - S_0$*
8. *Done*

This algorithm outputs a sub graph which consists of k-spanning tree. The replication is applied on the minimum spanning tree path generated by the above algorithm. The

replication is measured again in order to find the improvement in the replication performance.

4.4. Relationship between Algorithms

The four algorithms are executed in P2P clients and P2P Coordinator in DG-based DCI environment. All the four algorithms are inter-dependent on each other and are executed in a distributed environment. The relative order of execution of algorithms is as follows:

1. First Algorithm-1 will be executed to resolve any conflict due to data writes in the P2P coordinator.
2. Then Algorithm-2 will address data consistency.
3. Algorithm-3 will gather the statistics from the outcomes of Algorithm-1, Algorithm-2.
4. Then Algorithm-4 will be applied to improve the replication performance.

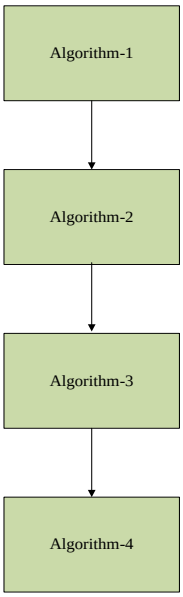

Figure 4.5: Relative order of execution of algorithms

The dependencies of four algorithms are mentioned below:

1. Algorithm-2 maintains data consistency for the R/W data which arises when the conflict writes requests are resolved by calling Algorithm-1 in coordinator.
2. Algorithm-3 collects the data statistics from Algorithm-1, Algorithm-2
3. Algorithm-4 is dependent on Algorithm-3 to get all data statistics for implementing the improvement in data replication performance.

So, dependencies between algorithms are represented as follows:

1. Algorithm-2➔ Algorithm-1
2. Algorithm-3➔ Algorithm-1, Algorithm-2
3. Algorithm-4➔ Algorithm-3

Where, ➔ indicates is dependent on

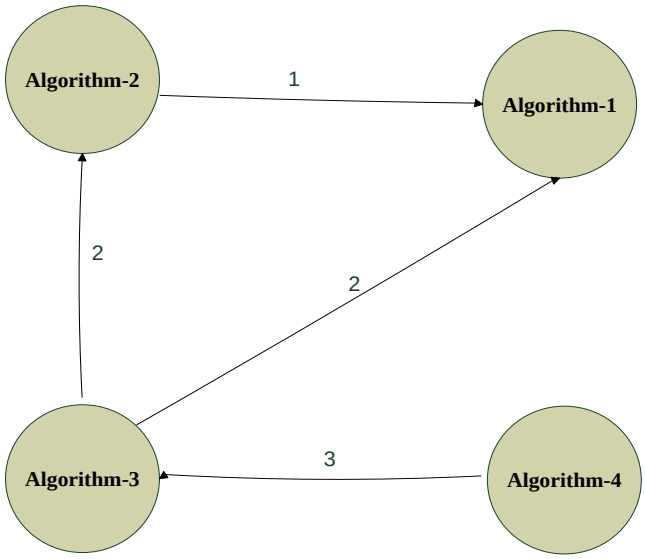

Figure 4.6: Dependency diagram between four algorithms

4.5. Proposed Research Contributions

The proposed research contributions for handling large volumes in order to maintain data replication and consistency in DG-based DCI are as follows:

Data Consistency Algorithms: This proposed research contribution deals with development of new concurrency control techniques for the new P2P-based architecture in DG-based DCI. It uses modified 2PC protocol for improving the performance of data replication and consistency simultaneously.

RC1$_1$: This contribution deals with a proposed algorithm that is used for resolving conflicts due to concurrent Read/Write operations. This algorithm is used in P2P coordinator.

RC1$_2$: This contribution deals with a proposed algorithm that is used for maintaining data consistency for conflict Read/Write data operations. This algorithm is used in both P2P coordinator and P2P clients. It uses the modified version of Two Phase Commit Protocol.

Data Replication Algorithms: This proposed research contribution deals with development of efficient algorithms for handling data replication for the new P2P-based architecture in DG-based DCI.

RC2$_1$: This algorithm is used for measuring the performance of replication in DG-based DCI. This algorithm also maintains data replication statistics of different clients.

$RC2_2$: This contribution deals with a proposed algorithm that is used for data planning and distribution strategies. It firsts finds the optimal access paths from the data collected from the replication. This algorithm then applies data replication to improve the performance of data replication. This algorithm is used in P2P coordinator as well as P2P clients.

P2P-based Architecture: This proposed research contribution deals with a novel P2P-based architecture for handling large volume of data in DG-based DCI.

$RC3_1$: This contribution deals with new P2P Coordinator architecture. It accepts and resolves any write operations for solving the data consistency due to concurrent writes by multiple clients. The P2P Coordinator is also acting as a tracker in the P2P-based architecture. A tracker maintains information about the location of clients in a network. It coordinates the transfer of files among P2P clients.

$RC3_2$: This contribution deals with new P2P client architecture. The client issues a request of Read/Write data to the client/coordinator respectively in the DG-based DCI.

The above proposed contributions are validated by demonstration of working examples and extensive experimental analysis in the chapter 6.

4.6. Conclusion

The goal of the research is to identify a novel architecture and new algorithms in DG-based DCI which are optimal for handling large volumes of data. The three research contributions related to P2P architecture, Data Consistency and Data replication have been identified and designed. New architecture has been identified and proposed in DG-based DCI to improve the performance of large volumes of data handling. New architecture has been reconstructed by the applying the good and proven aspects of P2P architecture for handling large volumes of data. The detailed workings of proposed P2P coordinator and P2P client's architectures have been mentioned in this chapter. Efficient algorithms have been proposed for handling the Read/Write data consistency and replication. The performance of the proposed architecture will be proved by experimental analysis of the proposed algorithms for data consistency and replication.

Chapter 5

Experimental Testbed and Simulation Design

5.1. Experimental Testbed

This section introduces the experimental testbed, infrastructure and its system constraints used for performing experiments.

5.1.1. Overview

An adequate testbed plays an importance role in the validation of all the proposed contributions in the research. The main requirement of the testbed is to allow the deployment and testing of the present and proposed architecture. Initially, the comparative analysis of the existing simulators/solutions has been done depending upon the requirements of proposed contributions.

Simulation is a good technique used for predicting the behaviour of new system in real world by creating a model for it. Some of the benefits of the selecting simulation for the research are as follows:

1. It allows repeatable experiments to be conducted.
2. The result remains limited to the testbed in real world implementations.
3. It allows more experimental scenarios to be performed.
4. It is possible for others to reproduce the results of the simulation.

The criteria for selection of tools are dependent upon the implementation and good support of the proposed architecture. Some of the criteria of selecting the tools are as follows:

1. Architecture support: Support for existing DG/Cloud based-architecture.
2. Data replication support to perform experiments.
3. Good support for adding new features to the existing architecture.

The comparative analysis of the existing simulators is mentioned in the table 5.1.

Sl. No.	OverSim [1]	PeerSim P2P [2]	SimGrid [3]	OptorSim[4]
Features	It is an open source overlay and peer-to-peer network simulation framework.	It is a simulation environment for P2P protocols.	It is used as a Grid, P2P, and Cloud simulator.	It is used for testing dynamic replication strategies for optimising the data location within a Grid.
Architecture	P2P	P2P	P2P, Grid,	Data Grid

support			Cloud	
Support for adding new functionality for the research	Normal	Normal	Good	Poor
Relevance towards data replication	Less	Normal	Normal	More (Uses P2P internally in replica optimiser)
Documentation	Good	Normal	Good	Poor
Language used for development	C++	Java	C/Java	Java

Table 5.1: Comparison of existing simulators

SimGrid is more suitable from the above comparative simulator's analysis for the research experiments due to the following reasons:

1. Some experiments results from simulator have been verified with actual results obtained in realistic environment. The mathematical analysis has been done for the verification of the results obtained through simulator.
2. It has good programming environment support for Grid, P2P, Cloud systems.
3. It has layered architecture. New component/Code can be easily added to the existing simulator architecture. The code may be added to SimGrid component interface without actually modifying the existing source code.
4. Good support and documentation.
5. The support of multiple languages (C/Java) for experimentation.

5.1.2. Infrastructure

The testbed has been developed to test and validate the proposed contributions as mentioned in the section 4.4. SimGrid [3] is a toolkit that provides core functionalities for the simulation of distributed applications in heterogeneous distributed environments. It provides the support for Grids, P2P, cloud, and HPC environment. The different components of SimGrid toolkit are as follow:

1. MSG: It is simple programming environment which describes how to setup and control your simulation.
2. SMPI: It is a programming environment for the simulation of MPI applications.

3. SimDag: It is programming environment for simulating parallel task scheduling with DAG(Direct Acyclic Graphs) models.
4. SURF: It is the internal kernel of the SimGrid simulator. It provides the functionalities to simulate a virtual platform. It contains the platform models.
5. XBT: It consists of data structures, portability support, grounding features (logging, exception support, unit testing, etc), and other features.
6. TRACE: It is used for tracing mechanism to know how much power is used for each host and how much bandwidth is used for each link of the platform.

The SimGrid architecture is mentioned in the figure 5.1.

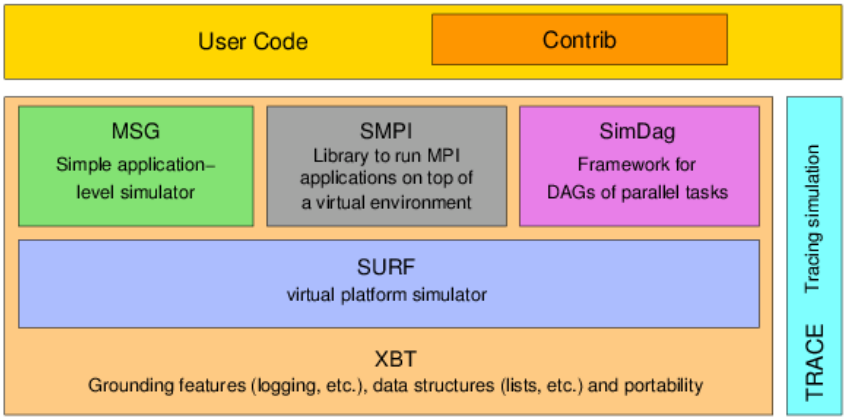

Figure 5.1: SimGrid [3] architecture

The algorithm code has to be written by the developer in the MSG environment. This framework uses java programming for preparing each scenario of the simulation for the proposed contributions. The next section deals with the system constraints for the simulation of the experiments.

5.1.3. System Constraints

The proposed architecture is subjected to several design and implementation constraints. The testbed constraints which are considered while performing the experimentation are as follows:

1. Clients may be up/down at any instance of the time.
2. New clients can join the network at any instance of the time.
3. Clients can send the messages to other nodes when they are up.
4. Messages sent between the clients are not corrupted/ lost in the network.
5. Messages sent to the other clients may take time a deterministic time to deliver.

5.2. Simulation Design

In order to validate the proposed research contributions as mentioned in chapter-5, testbed will be evaluated by initially programming of different components of proposed architecture on SimGrid toolkit.

Simulation design consists of following steps:

1. Initially, an algorithm is selected for the simulation.
2. The input resources and parameters required for an algorithm are selected.
3. The structures of the input files consisting of these parameters are defined.
4. The construction of experiment is done for the algorithm. The detailed code of the algorithm is mentioned in the developer section of the SimGrid toolkit.
5. The experiment is conducted on the simulator for developed algorithm and input parameters are defined.
6. The results of the experiment are collected for the analysis purpose.
7. The performance in terms of total execution time, number of messages exchanged or number of Read/Write operations are compared for proposed algorithm with respect to existing algorithm.

The steps comprising of simulation process are mentioned in the figure 5.2

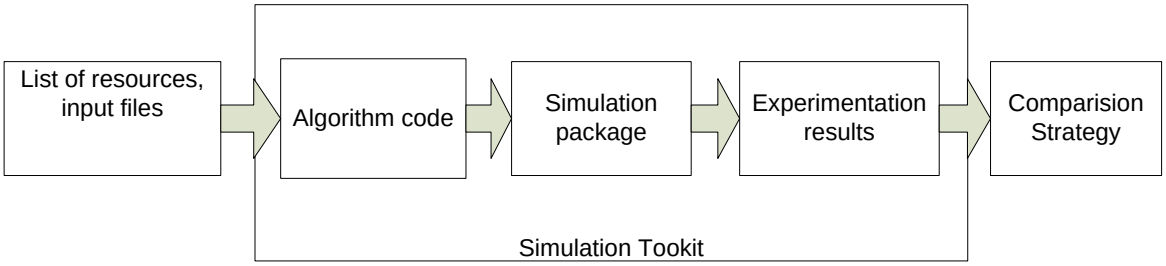

Figure 5.2: Simulation process

The details of each step are mentioned in next sections.

5.2.1. Overview

Initially, a set of experiments will implement the basic components of exiting DG architecture on SimGrid Toolkit. Then, existing P2P coordinator/clients are implemented. Then, the proposed architecture consisting of P2P coordinator/clients with existing BOINC components are implemented. Then a set of experiments will be conducted for the proposed algorithms as mentioned in section 6.2. The architecture of the figure 5.3 is considered for the performing set of experiments.

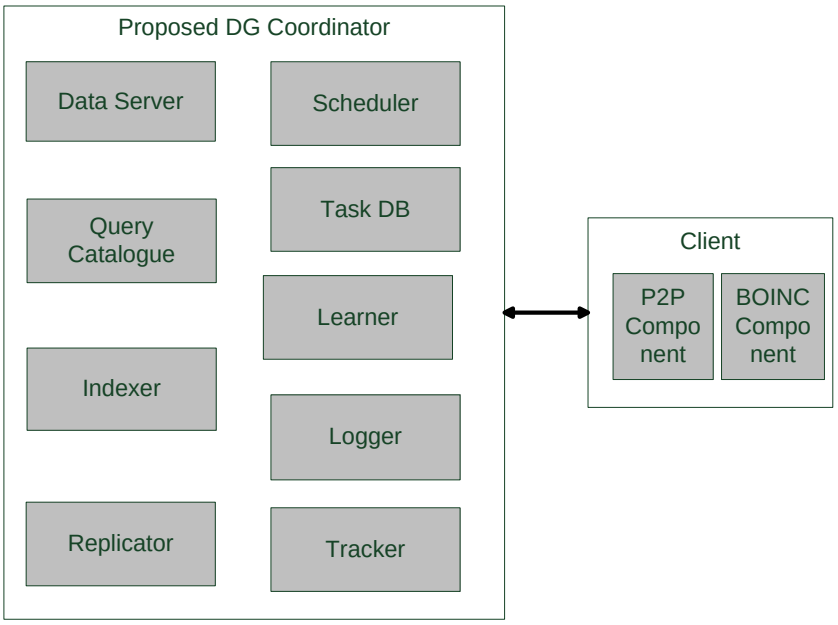

Figure 5.3: Proposed DG architecture

The simulation process consists of the following three things:

1. Developer's code: The developers write the code of the simulated algorithm for the proposed architecture.
2. Simulation input: The input for the simulation is the finite set of resources/parameters required for conducting the experiments. Depending upon the experiments the parameters required for the experimentation is changed in the input files.
3. Simulation output: The results of executing an experiment are written to the output file.

   The simulation architecture is mentioned in the figure 5.4.

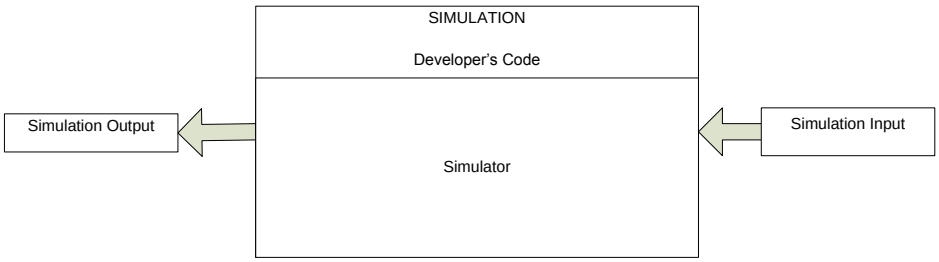

Figure 5.4: Simulation Architecture

5.2.2. Construction of Experiments

A set of experiments is first identified for each of the proposed contributions as per section 4.5. It consists of the following set of experiments:

1. Data Consistency Experiments
2. Data Replication Experiments

3. P2P-based architecture Experiments
4. Comparative performance analysis of the P2P-based architecture with the existing DG-based architecture.

Then the simulation architecture of each experiment is identified. The input parameters are identified for each experiment. Then the corresponding code for each experiment is written in the simulation environment.

The next section deals with the parameters considered for the experiments in the proposed architecture.

5.2.3. Parameters considered for the Test Data

The proposed DG architecture consists of coordinator and clients. They are connected via a network. The coordinator and clients are connected via links in the network. The link has bandwidth and latency associated with it. Network has a routing topology for sending the messages, data, task, etc between coordinator and client. The coordinator sends the task information i.e. task size, number of tasks to clients. Coordinator and clients machine's processing power is used for processing the task, message, etc in the network. In the proposed architecture, a client acts as a data seed initially. The data from data seed is distributed to other clients by using P2P techniques. Client communicates to the coordinator when a piece of data is modified by a client in the emerging applications. Coordinator communicates the changes to other clients by using algorithms to maintain data consistency. The information about the environment parameters i.e. coordinator, number of clients, dataset size, number of pieces in dataset, number of pieces modified, data seed, task set, size of task set, communication size of task, processing power of coordinator, processing power of clients, routing of the network, bandwidth of each link, latency of each link are mentioned in the input files of the simulator. The input parameters for the simulation are mentioned in two input files:

1. Platform file: It consists of configuration details parameters to be used for the simulation. It gives the description of the platform on which application has to be executed.
2. Deployment file: It consists of the deployments parameters used for the simulation. It gives the information what has to be deployed in which location.

The parameters considered for the test data for the experiments are as follows:

1. No. of coordinator: It represents the number of coordinator present in the system.
2. No. of clients: It represents the number of clients to which the data and tasks has to be distributed.
3. Dataset size: It represents the set of data expressed in bytes.

4. No. of pieces in dataset: It represents the number of pieces in a dataset. Each dataset comprises of number of file pieces. Each piece has 2/5 number of blocks.
5. No. of pieces modified: It represents the number of pieces of dataset modified from the entire dataset.
6. Data seed: Data seed is the client which has dataset initially. This client acts as a data seed. It is 1 when it is acting as data seed else it is 0.
7. Task set: It represent the total number of tasks in a set which has to be distributed to the clients.
8. Size of task set: Task size is value of the processing amount (in flop) needed to process the task. A task may be defined by a computing amount, a message size and some private data.
9. Communication size of task: It the size of the application code data (in bytes) to be transferred along with the task from coordinator to client.
10. Processing power of Coordinator: It the processing power of the coordinator in Hertz.
11. Processing power of clients: It the processing power of the client in Hertz.
12. Routing of the network: It represents the type of routing used for the simulation.
13. Bandwidth of each link: It represents the bandwidth in bytes between two links.
14. Latency of each link: It represents the latency of each link in seconds.

Next section discusses the structure of the test data for the experiments.

5.2.4. Structure of Test Data

The input test data for the experiments are provided in two input xml files: Platform file and Deployment file.

The structure of the input platform file for the test data is as follows:

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="3">
<AS id="AS0" routing="Full">
<cluster id="my_cluster_1" prefix="n-" suffix=""
radical="0-5"   power="10000000" bw="12500000" lat="0.000514433"/>
</AS>
</platform>
```

Where,

AS: It represents Autonomous System which are networks also known as LAN. AS0 represents the id of AS in above example.

routing: It represents type of routing used in the network. Full routing means that all the clients are connected with each other.

radical: It is the number of total clients in the network.

power: It is the peak number FLOPS the CPU can manage. It is expressed in flop/s.

bw: It represents the bandwidth of the network.

lat: It represents the latency of each link

The structure of the deployment file for the input test data is as follows:

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="3">
<process host="n-0" function="np2p.Coordinator"><argument value="5"/><argument
value="500000000"/><argument value="10"/><argument value="5"/><argument
value="1"/><argument value="10"/><argument value="2"/></process>
<process host="n-1" function="np2p.Client"><argument value="0"/><argument
value="0" /></process>
<process host="n-2" function="np2p.Client"><argument value="1"/><argument
value="0" /></process>
<process host="n-3" function="np2p.Client"><argument value="2"/><argument
value="0" /></process>
<process host="n-4" function="np2p.Client"><argument value="3"/><argument
value="0" /></process>
<process host="n-5" function="np2p.Client"><argument value="4"/><argument
value="1" /></process>
</platform>
```

Where,

Host: It represents a computer in where code can be executed and information can be sent or received.

Np2p. Coordinator function of the code has following arguments:

- Amount of tasks to dispatch=5,
- Computation size of each task=500000000,
- Communication size of each one=10,
- Number of np2p.Clients waiting for orders=5,
- Data item present if 1 else 0,
- No. of data items=10,
- Data item modified=2

Np2p.Client of the code has following arguments:

- 1st argument shows the number of client,
- 2nd argument shows that if client will modify(1) the data items

Client n-5 is acting as a data seed in the above deployment file.

5.2.5. Experiment Simulation

An experiment simulation consists of the following steps:

1. Finding the architecture for the experiment: It deals with determining the architecture for the experiment to be conducted. A example of an architecture for the experiment which measures the performance of algorithm-1 and algorithm-2 in terms of total execution time, number of messages exchanged for varying data size for a number of clients is shown below:

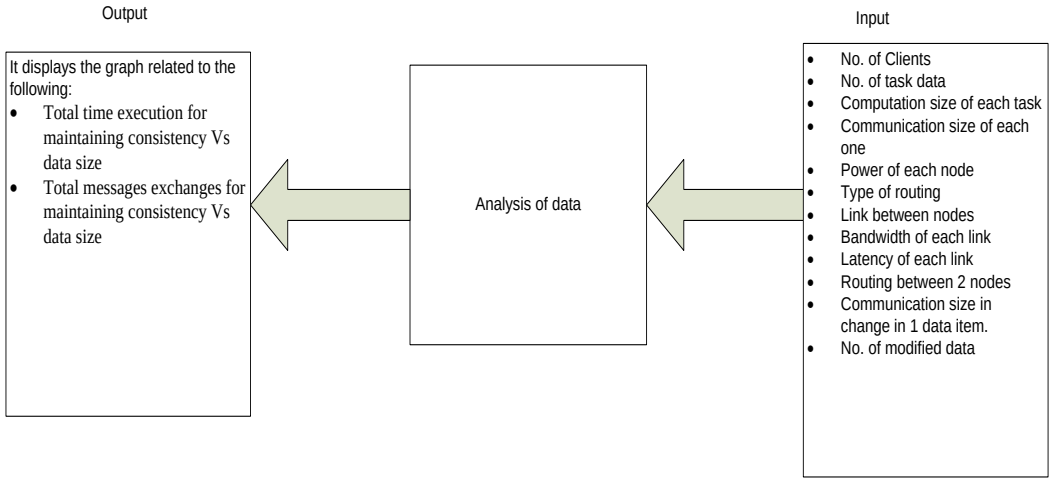

Figure 5.5: Generic architecture for an experiment

2. Data generation for platform file: It deals with the generation of the platform file for the experiment. Its structure is similar to the platform file as mentioned in the section 5.2.5.
3. Data generation for deployment file:It deals with the generation of the deployment file for the experiment. Its structure is similar to the deployment file as mentioned in the section 5.2.5.
4. Implementation of the user code in the MSG component of the SimGrid toolkit: It deals with the implementation of the code in java programming in the SimGrid toolkit.
5. Task Execution in the simulated environment: It deals with the execution of the experiment for a set of input files. Source code files are complied first, and then experiment is executed as follows:

   java algo/AlgoTestalgo\platform.xml algo\deploy.xml >out_exp.txt 2>&1

Where,

AlgoTest: Complied source file is called from algo package.

platform.xml: The configuration of the network is mentioned in this file.

Deploy.xml: The configuration details of the each node in the network are mentioned in this file.

out_exp.txt: The file name where output is redirected.

6. Collecting the results: It involves collecting the results of performance parameters from the output file.

Depending upon the network configuration, number of clients, number of tasks, and other parameters, the details in platform.xml and deploy.xml files are changed. The complied source program may remain same for the experiment.

5.2.6. Representation of Results

Several experiments were conducted to verify that the simulation model is valid for the working of proposed architecture for the experimentation purpose.

Initially basic architecture of existing DG was implemented in order to validate the working of DG. The distribution and processing of tasks in basic DG architecture are assigned to the coordinator.

Input: It consists of input from files platform.xml and deploy.xml. Files contains information about the number of coordinator (1) , number of tasks (2), number of clients (2), computation size of tasks, communication size of tasks, computational power of coordinator and clients, number of links, links details in terms of bandwidth, latency, etc.

The output of the experiment was analysed by displaying the messages for the distribution, processing of tasks between coordinator and clients. It also measured the total task execution time. The time taken for each task execution was same for both clients since both clients have same configuration. This experiments shows that the simulation model is valid.

Another experiment for the existing DG was also conducted by increasing the number of tasks for multiples clients. Two clients were assigned the tasks by the coordinator. The task execution time calculated remains same for both clients when client's configuration is maintained same. This also validated that the simulation model is valid.

Another experiment was conducted by increasing the total number of tasks and number of clients. The working of DG was analysed for distribution of tasks and messages displayed

during processing. The output displayed was similar to the working of existing DG. This also validates that the simulation model used is valid.

One experiment dealing with distribution and data processing between 1 P2P Coordinator and 2 P2P clients was conducted. The input to this experiment is described as follows:
Input: It consists of input from two files: platform.xml, deployment.xml. It consists of information related to total number of nodes, number of tracker (1), number of peers (2), seed node, cluster information, bandwidth and latency between the links of nodes, etc.
The Output of the experiment shows that the working of P2P by displaying the messages for the data distribution, processing, between coordinator/tracker and clients. It also displays the total time of execution in data distribution for all nodes. The coordinator is also acting as Tracker in this experiment. Initially, all the 2 P2P clients send join request to the tracker. Client-1 is acting as a seed in the network. Once all the peers have complete data, the status of all clients becomes 1111111111. In the end, it displayed the total execution time of tasks. The working of the P2P system was further verified by analysing the output by increasing the number of clients in the system.

Other experiments were also conducted for the validation of simulation model. For example, the figure 5.6 illustrates an example of representation of execution time calculated for varying size of dataset for 50 tasks and 50 clients. This graph is prepared from the data obtained from the output file generated after executing the experiment as mentioned in the section 5.2.6. The intention of this type of graph is to analyse the performance in terms of execution time for varying dataset size. The execution time does not increase significantly when the dataset size is increased 10 times. Similarly, the performance of existing and proposed architecture is analysed in form of comparison graphs by varying different parameters.

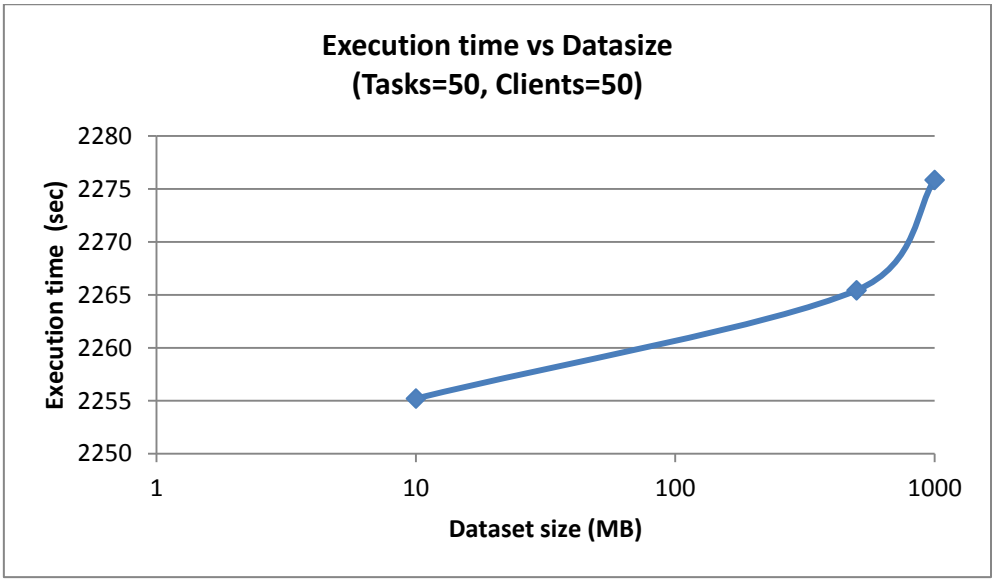

Figure 5.6: Graph of execution time for varying data size

5.2.7. Comparison Strategy

The performance of the proposed architecture is compared with the existing architecture for the experiments. The performance during the experiments is measured in terms of the following factors:

1. Execution time: It is the total execution time measured for performing an experiment.
2. Number of messages passed: It is total number of message passed between coordinator and clients for performing an experiment.
3. Number of Read/Write operations: It is total number of Read/Write operation performed for maintaining the consistency in the simulation architecture.

The outputs from an experiment are collected for both proposed and existing architecture. Then the performance of both architectures are compared from the output in terms of total execution time, number of messages passed, number of Read/Write operations, etc.

5.3. Conclusions

This chapter encompassed two objectives. The first objective was to introduce the experimental testbed architecture. The requirements of the simulation environment along with its constraints were discussed. The second objective was to present the simulation design for the conduction of experiments. The structure of data used, experiment simulation, and result representation were also discussed in this chapter.

Chapter 6

Experimental Results

6.1. Introduction

The purpose of this chapter is to perform experiments to validate the contributions identified in chapter 5. The aim of this chapter is to find out the scenarios and conditions in which proposed architecture outperforms present architecture as well as when it does not. This chapter is divided in two sections – experimental results and its post-mortem analysis. The first section describes the methodology for performing experiments and summarized results corresponding to it. The second section deals with the analysis of the experiments result outcomes and provides the ways to improve the performance in the proposed architecture.

6.2. Experiments Results

6.2.1. Overview

The SimGrid testbed is designed with the purpose of adding new functionality to the existing architecture model. The new features in the proposed architecture are Read/Write of data and handling large volumes of data. It then compares the performance of proposed architecture model with the existing architecture model. The performance of each architecture model is evaluated on the total execution time and total number of messages exchanged. The total execution time is the time to distribute tasks and data between coordinator and clients and time taken to execute the tasks on the clients. The total number of messages exchanged is the number of messages exchanged in distribution of large volumes of data between clients and coordinator. A set of experiments is identified by adding new features to the existing architecture in order to improve the performance for handling large volumes of data. The experiments performed are classified into three broad categories as mentioned in table 6.1.

#	Experiment	Experiment Description
1	Data Consistency	These experiments are classified into two sub categories: Conflict resolving and Data consistency maintenance.

2	Data Replication	These experiments are classified into two sub categories: Replication performance measurement and Replication planning strategies.
3	Comparative Performance Analysis of proposed and existing architecture	These experiments deal with the performance comparison of proposed architecture with the Attic solution used in existing DG architecture.

Table 6.1: Classification of experiments performed

Data consistency experiments deals with the data conflict resolution and maintenance of data consistency for R/W of data in DG-based DCI. Data replication experiments deals with replication performance measurement and replication planning strategies in order to improve the performance. Comparative performance experiments deals with the performance analysis for handling large volumes of data when the proposed architecture outperforms existing architecture and when it does not.

Each experiment consists of one or more scenarios. Each scenario defines different value of the parameters considered and it consists of one or more test cases. Each test case consists of one or more executions, where each execution has different value of the parameters considered.

The simulation of the experiments is based on institutional DG-based DCI. The minimum and maximum parameters values are used as per the actual information provided by university staff.

- Routing of the network,
- Bandwidth of link (Gb),
- Latency of link (Microsec),
- The Processing power of Coordinator (GHz),
- Datasize (in MB's),
- Size of tasks.
- Number of tasks,
- No. of coordinator

Some of the parameters value like number of pieces in dataset, number of pieces modified are considered due to the emerging application requirements. In the experiments, the number of clients is considered as 50 in order to simplify analysis. The rationale of considering these parameters values is that it should be comparable to the existing DG environment.

So, different parameters values are considered for the experimentation purpose as mentioned table 6.2.

		Min value	Max value	Mean	Considered value
1	No. of coordinator	1	1	1	1
2	No. of clients (homogenous campus network)	1	1800	900	50
3	Dataset size (MB)	10	1000	505	500
4	No. of pieces in dataset	1	10	5.5	10
5	No. of pieces modified	0	10	5	10
6	Data seed	1	1	1	1
7	Task set	1	100	50	50
8	Size of task set (in flops)	5E+12	5E+12	5E+12	5E+12
9	Size of task (Bytes)	20	9216	4618	4618
10	Processing power of Coordinator (GHz)	3	7	5	5
11	Processing power of client (GHz)	1.5	3.5	2.5	2.5
12	Routing of the network	Tree topology (Clique)	Tree topology (Clique)	Tree topology (Clique)	Tree topology (Clique)
13	Bandwidth of link (Gb)	1	10	5.5	5.5
14	Latency of link (Microsec)	0.37	6.7	3.535	3.5

Table 6.2: Parameters value considered for the experimentation

Where,

Min value represents s the minimum value of a parameter in institutional DG-based DCI.

Max value represents s the maximum value of a parameter in institutional DG-based DCI.

Considered mean value is the value of a parameter selected for performing experiments.

Performance analysis of experiments are performed for min, max and mean values of one parameter while considering the mean value of all other parameters. This process is repeated for all parameters in order to find the significant parameters which have more variation in the value of the performance. Then ranking of these parameters is done in order of significance. Then extensive experimentation is done by varying the values of these significant parameters.

6.2.2. Data Consistency Experiments

6.2.2.1. Overview

The purpose of this type of experiments is to maintain the data consistency in DG-based DCI. These experiments are further classified in two: Data conflict resolving, and Data consistency maintenance experiments.

The algorithms dealing with data consistency are Algorithm-1 and Algorithm-2. The designs of these algorithms are described in chapter 4. The Algorithm-1 resolves the data conflicts while Algorithm-2 maintains the data consistency due to the R/W operations in DG-based DCI. The scenarios for these experiments are mentioned in the next section.

6.2.2.2. Scenario and Test Cases

The scenarios for data consistency experiments are classified in two categories as mentioned in the table 6.3.

Scenario Number	Experiment	Experiment Description
DC1	Conflict Resolving	This scenarios deal with the data conflict resolving by using Algorithm-1.
DC2	Consistency Maintenance	This scenarios deal with the data consistency maintenance by using Algorithm-2.

Table 6.3: Data consistency experiments scenarios

The test cases and the execution analysis for DC1 and DC2 scenarios are mentioned in the next section.

6.2.2.2.1. Scenario DC1: Conflict Resolving

In scenario DC1, the numbers of data modifications are increased gradually in a dataset. A dataset consists of many data pieces. The modification is applied on a piece of data. The size of one piece of data is assumed to be 10MB and all pieces are of equal size. The modification is done on different data pieces. The purpose of this scenario is to find out the performance in terms of total execution time for resolving data conflicts when multiple clients send data concurrently.

Test Cases: DC1

In scenario DC1, numbers of modifications of data pieces are varied from 2 to 50 in a dataset. The value of other parameters is kept constant. The data considered for this scenario DC1 is as follows:

- Processing power of coordinator = 5.0 GHz,
- Processing power of client = 2.5 GHz,
- Computation size of each task = 5.00e+12 Flops,
- Communication size of each task =4618 Bytes,
- Bandwidth of link= 5.5Gb,
- Latency of link =3.53 microsecond,
- Dataset size =100 GB,
- Number of tasks = 2,
- Number of clients = 2.

This scenario consists of four test cases: DC1TC1, DC1TC2, DC1TC3, and DC1TC4 as listed in table 6.4.

Test Case Number	Parameter Considered (number of modifications)
DC1TC1	2
DC1TC2	5
DC1TC3	10
DC1TC4	50

Table 6.4: Scenario DC1 test cases

DC1 Test Case Architecture

The architecture diagram for scenario DC1 test cases is mentioned in the Figure 6.1. The architecture accepts input parameters value like number of tasks, computation size of task,

communication size of each task, coordinator, number of clients, type of routing, bandwidth of links, latency of links, processing power of coordinator, processing power of client, etc for the Algorithm-1 in the simulator. Algorithm-1 as described in section 4.2.1 then resolves the conflicts due to the modification of data by clients. It displays the total execution time for resolving the data conflict.

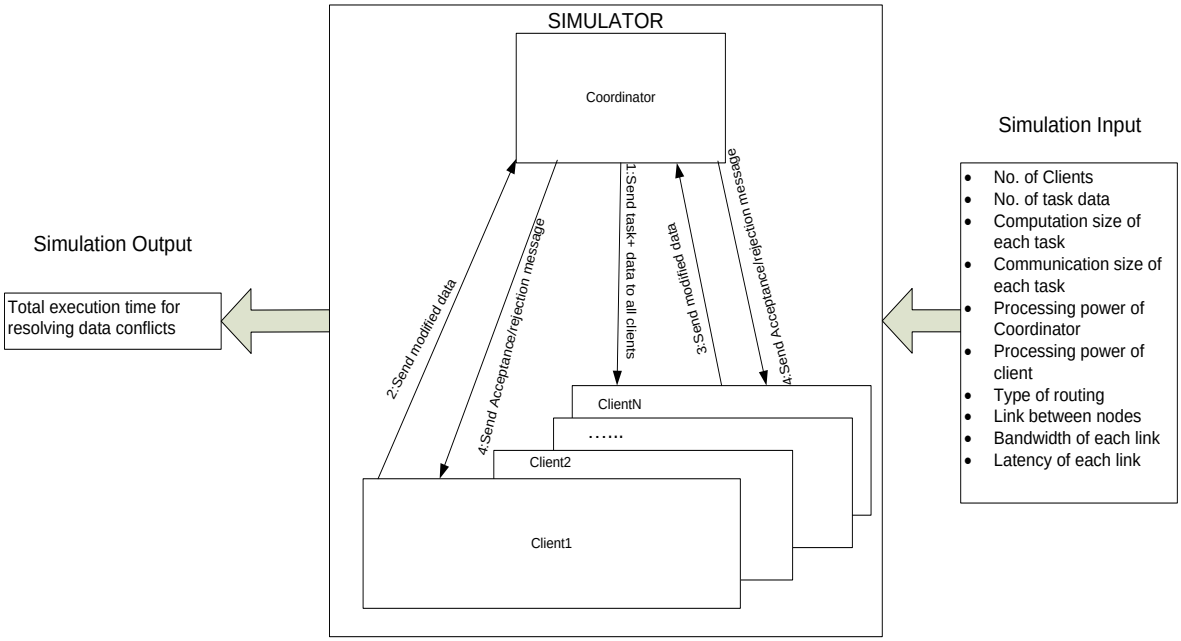

Figure 6.1: Architecture diagram for scenario DC1

In DC1 scenario, each test case is associated with a single execution which displays the total execution time for resolving data conflicts. The DC1 test case executions are listed in the table 6.5.

Test Case No.	Test Case Execution No.	Number of Modifications	Total Execution Time (sec)
DC1TC1	DC1TC1E1	2	609.18
DC1TC2	DC1TC2E1	5	618.18
DC1TC3	DC1TC3E1	10	633.18
DC1TC4	DC1TC4E1	50	753.18

Table 6.5: DC1 test case executions

Scenario Execution Analysis: DC1

The execution trend for DC1 test cases is mentioned in figure 6.2. The x-axis represents number of data modifications and y-axis represents the total execution time taken in seconds for resolving data conflicts.

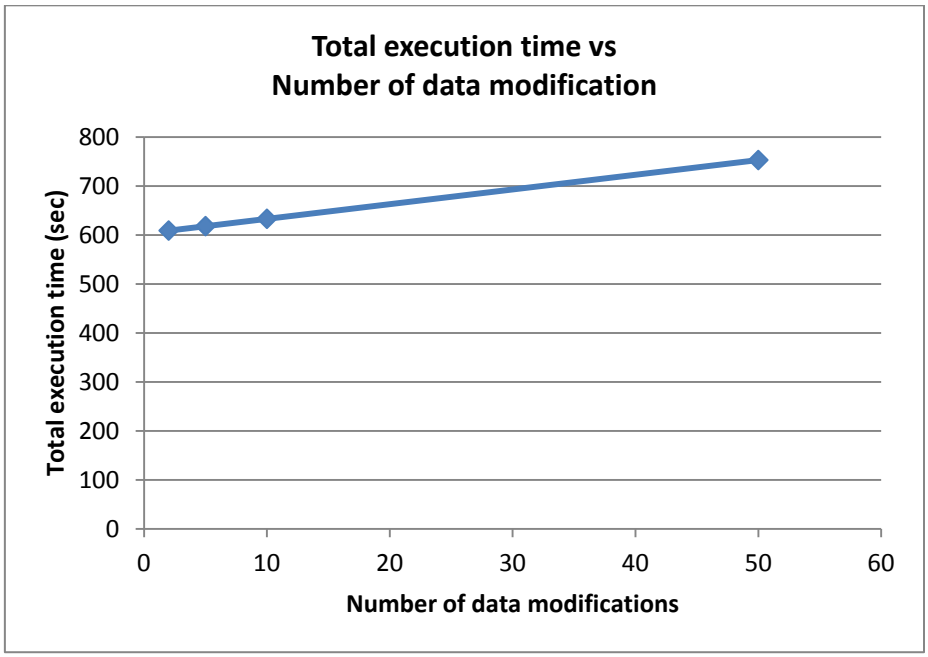

Figure 6.2: Execution trend for scenario DC1

The above figure shows the total execution time taken to resolve data conflicts for number of modifications of data by Algorithm-1. It shows that the total execution time increases gradually when number of data modifications increases many folds for resolving data conflicts.

6.2.2.2.2. Scenario DC2: Consistency Maintenance

In scenario DC2, the numbers of modifications are increased gradually in a dataset for many clients. The purpose of this scenario is to measure performance for maintaining data consistency for multiple clients. The performance is measured in terms of total execution time in DG-based DCI.

Test Cases: DC2

In scenario DC2, numbers of modifications of data in a dataset are varied from 10 to 1000. The value of other parameters is kept constant. The data considered for this DC2 scenario is as follows:

- Processing power of coordinator = 5.0 GHz,
- Processing power of client = 2.5 GHz,

- Computation size of each task = 5.00e+12 Flops,
- Communication size of each task = 4618 Bytes,
- Bandwidth = 5.5 Gb,
- Latency =3.53 microsecond,
- Dataset size =100 GB,
- Number of clients = 50,
- Number of tasks = 50.

The DC2's scenario consists of 8 test cases: DC2TC1, DC2TC2, DC2TC3, DC2TC4, DC2TC5, DC2TC6, DC2TC7, and DC2TC8 as listed in table 6.6.

Test Case Number	Parameter Considered (number of modifications)
DC2TC1	10
DC2TC2	20
DC2TC3	50
DC2TC4	100
DC2TC5	200
DC2TC6	500
DC2TC7	700
DC2TC8	1000

Table 6.6: Scenario DC2 test cases

DC2 Test Case Architecture

The architecture diagram for data consistency scenario DC2 test cases is mentioned in the figure 6.3. Algorithm-2 in the architecture accepts input parameters value like number of tasks, computation size of task, communication size of each task, coordinator, number of clients, type of routing, bandwidth of links, latency of links, processing power of coordinator, processing power of client, etc. Algorithm-2 maintains data consistency after data conflicts are resolved by the Algorithm-1. It displays the total execution time taken for maintaining the data consistency in all the clients in DG-based DCI.

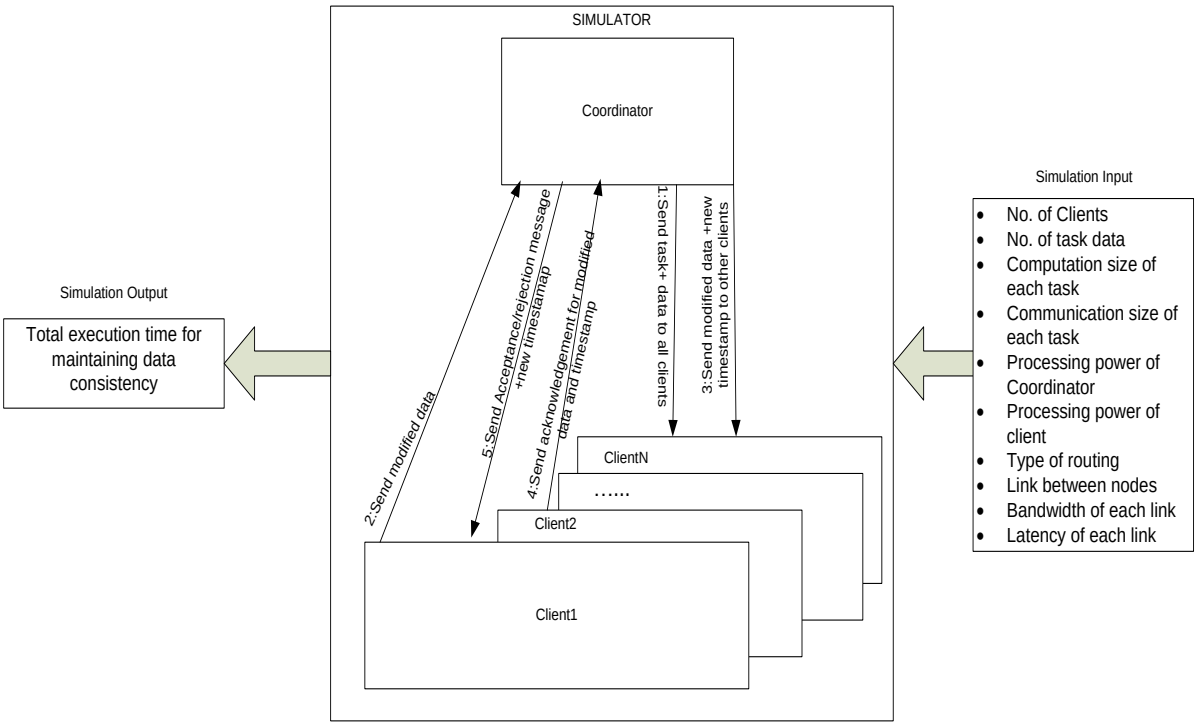

Figure 6.3: Architecture diagram for scenario DC2

In DC2 scenario, each test case is associated with a single execution. The DC2 test cases executions are listed in the table 6.7.

Test Case No.	Test Case Execution No.	Number of Modifications	Total Execution Time (sec)
DC2TC1	DC2TC1E1	10	23853
DC2TC2	DC2TC2E1	20	24363
DC2TC3	DC2TC3E1	50	25893
DC2TC4	DC2TC4E1	100	28443
DC2TC5	DC2TC5E1	200	33543
DC2TC6	DC2TC6E1	500	48843
DC2TC7	DC2TC7E1	700	59043
DC2TC8	DC2TC8E1	1000	74343

Table 6.7: Test case DC2 executions

Scenario Execution Analysis: DC2

The execution trend for DC2 test cases is mentioned in the figure 6.4. The x-axis represents number of data modifications and y-axis represents the total execution time

taken in seconds for maintaining data consistency. The log to the base 10 is used for representing the number of modifications in a dataset in x-axis.

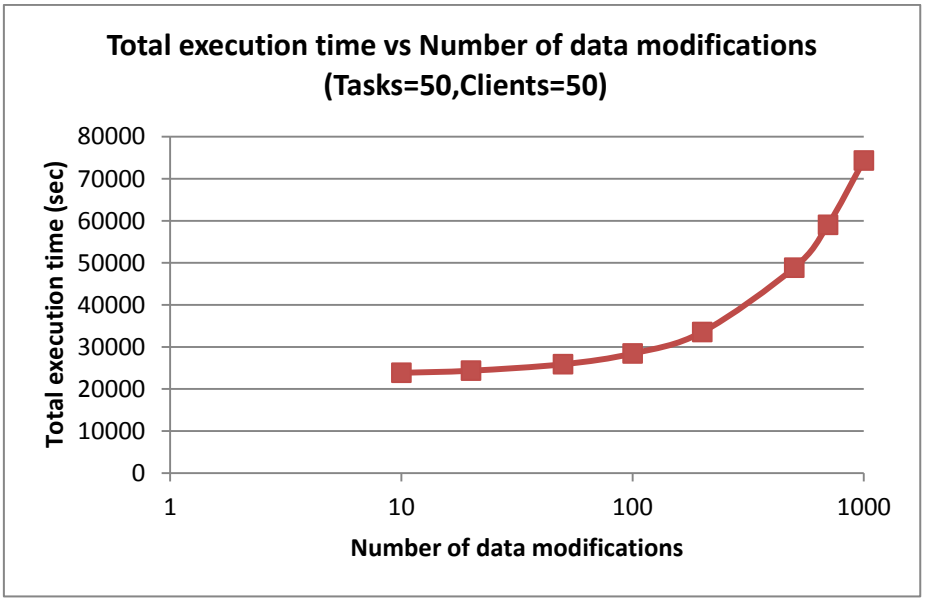

Figure 6.4: Execution trend for scenario DC2

The above figure shows the total execution time taken by Algorithm-2 to maintain data consistency for multiple clients. The above graph depicts that the total execution time is linearly dependent upon number of data modifications.

6.2.2.3. Data Consistency Scenario Analysis

The Algorithm-1 and Algorithm-2 resolves data conflicts and maintains data consistency for the modifications of data in DG-based DCI. The total execution time taken by Algorithm-1 and Algorithm-2 is linearly dependent on the number of data modifications.

Thus proposed algorithms validates the research contribution towards knowledge in development of new concurrency control techniques for handling R/W of data in proposed P2P-based architecture for DG-based DCI.

6.2.3. Data Replication Experiments

6.2.3.1. Overview

The purpose of replication experiments is to first measure the performance and then plan replication strategies to improve performance in DG-based DCI. The performance is measured in terms of total execution time. These experiments are classified in two:

replication measurement and replication planning strategies experiments. The algorithms dealing with data replication are Algorithm-3 and Algorithm-4. The designs of these algorithms are described in chapter 4.

The Algorithm-3 is used for the measurement of the replication performance. The performance of replication is depending upon the parameters like number of clients, dataset size, number of pieces in dataset, number of pieces modified, number of tasks, size of task, routing of the network, bandwidth of link, latency of link, etc. The performance is measured by varying different parameters.

Algorithm-4 is used for planning data distribution strategies and to improve data replication performance in DG-based DCI. Algorithm-4 first finds the frequent data access paths for data replication from data collected by Algorithm-3.Two algorithms FP tree algorithm and Prim algorithm are used for finding frequent access paths for data replication in DG-based DCI. An adaptive multipath spanning tree designed in chapter 4 is used for planning data replication strategies in DG-based DCI.

6.2.3.2. Scenario and Test Cases

The scenarios for these experiments are classified in two broad categories as mentioned in the table 6.8.

Scenario Number	Experiment	Experiment Description
DR1	Replication Performance Measurement	This scenarios deal with the measurement of replication performance by using Algorithm-3.
DR2	Replication Planning Strategies	This scenarios deal with generating frequent access paths and planning replication strategies.

Table 6.8: Data replication experiments scenarios

The test cases and execution analysis for DR1 and DR2 scenarios are mentioned in the next section.

6.2.3.2.1. Scenario DR1: Replication Performance Measurement

The purpose of DR1 scenario is to measure the replication performance in terms of total execution time. The replication performance in DG-based DCI is dependent upon parameters like dataset size, number of data modifications, number of clients, communication size, number of tasks, processing power of coordinator, processing power

of client, bandwidth of link, latency of link, etc. The performance is first measured for varying these parameters and then finding the significant parameters which affect the replication performance.

Test case architecture: DR1

The architecture diagram for DR1scenario for data replication performance measurement is mentioned in the Figure 6.5. The architecture accepts input parameters value like dataset size, number of data modifications, number of clients, number of tasks, processing power of client, bandwidth of link, latency of link, communication size of task, processing power of coordinator, etc as the input for the simulator. The total execution time is measured by varying one parameter value while keeping the other parameter value constant.

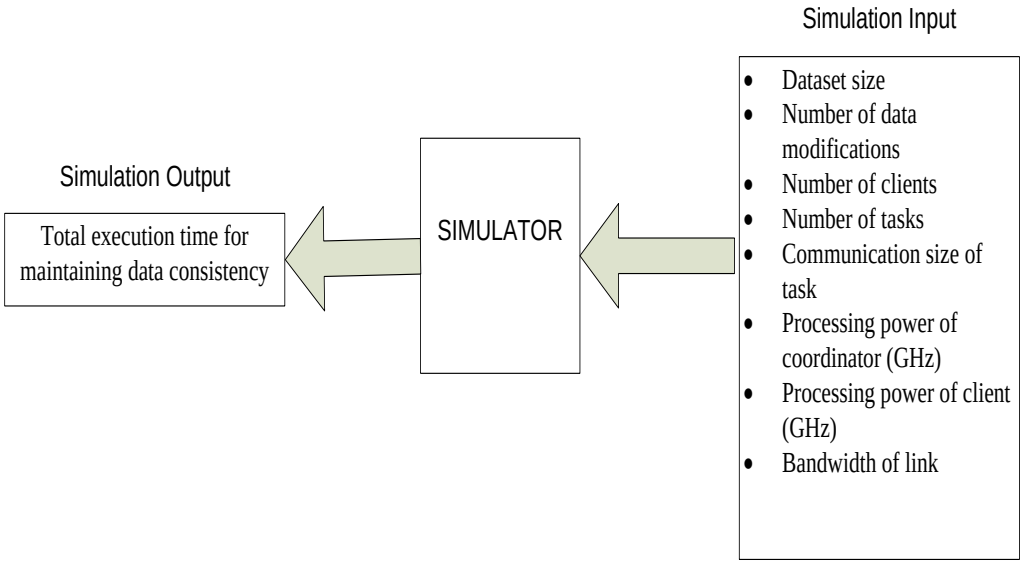

Figure 6.5: Architecture diagram for scenario DR1

Test cases: DR1

In scenario DR1, test cases deals with parameters which influence the replication performance in DG-based DCI. The total execution time is measured by varying a parameter value and keeping the other parameter's mean value as constant. Initially each test case is run on 3 values: the minimum, maximum and its mean value representing the realist values from DG environment. Multiple experiments were executed on 3 values in order to drive the most influential parameters, after which those parameters will be further explored in depth by taking multiple values. The DR1 scenario consists of test cases as listed in table 6.9.

#	Test Case Number	Parameter Considered
1	DR1TC1	Dataset size (MB)
2	DR1TC2	Number of data modifications
3	DR1TC3	Number of clients
4	DR1TC4	Number of tasks
5	DR1TC5	Processing power of client (GHz)
6	DR1TC6	Bandwidth of link
7	DR1TC7	Communication size of task
8	DR1TC8	Processing power of coordinator (GHz)
9	DR1TC9	Dataset size (GB)

Table 6.9: DR1 Scenario test cases

DR1TC1

In DR1TC1 test case, total execution time is measured by varying dataset size in MB and by keeping other parameters value at their mean in realistic DG-based DCI environment.

Test case executions: DR1TC1

The dataset size is varied from 10 to 1000 MB. The data considered for this test case is as follows:

- Processing power of coordinator = 5.0 GHz,
- Processing power of client = 2.5 GHz,
- Computation size of each task = 5.00e+12 Flops,
- Communication size of each task = 4618 Bytes,
- Bandwidth = 5.5 Gb,
- Latency =3.53 microsecond,
- Number of tasks = 50,
- Number of clients = 50,
- Number of modifications =10.

Each test case is associated with 3 executions. Table 6.10 lists the executions corresponding to test case DR1TC1.

Test Case No.	Test Case Execution No.	Dataset size (MB)	Total Execution time (sec)
DR1TC1	DR1TC1E1	10	2255.21

	DR1TC1E2	500	2265.43
	DR1TC1E3	1000	2275.85

Table 6.10: Test case DR1TC1 executions

Test case execution analysis: DR1TC1

The execution trend for test case DR1TC1 is mentioned in the figure 6.6. The x-axis represents dataset size in MB and y-axis represents the total execution time taken in seconds for replication.

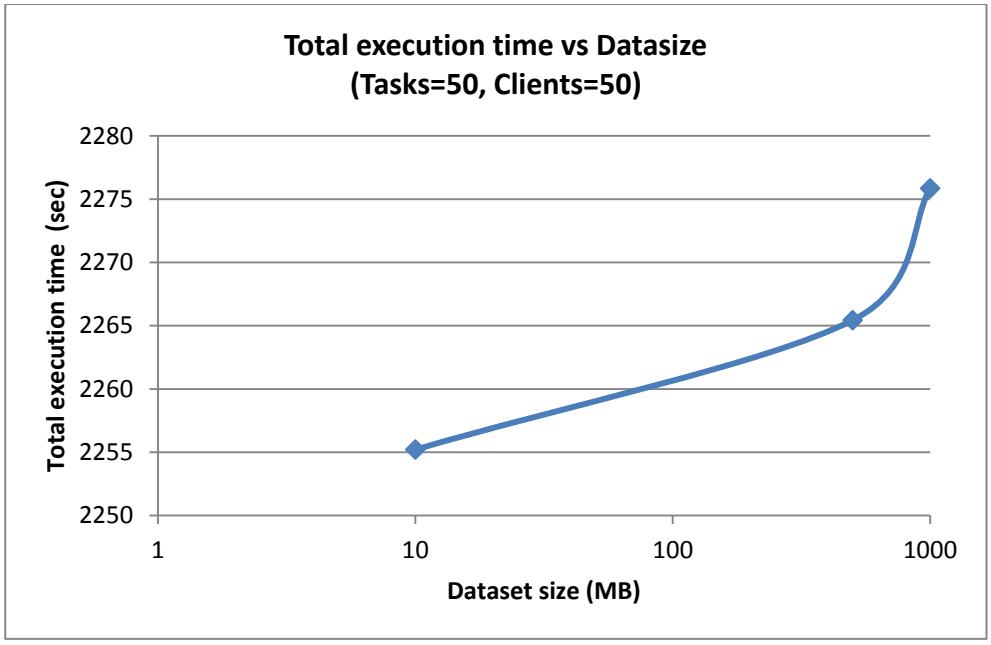

Figure 6.6: Execution trend for test case DR1TC1

The above figure shows the total execution time taken by varying dataset size. It shows that the total execution time increases gradually when dataset size is increased significantly.

DR1TC2

In DR1TC2 test case, total execution time is measured by varying number of modifications and by keeping other parameters value at their mean in realistic DG-based DCI environment.

Test case executions: DR1TC2

The number of modifications is varied from 1 to 10. The data considered for this test case is as follows:

- Processing power of coordinator = 5.0 GHz,
- Processing power of client = 2.5 GHz,
- Computation size of each task = 5.00e+12 Flops,
- Communication size of each task = 4618 Bytes,
- Bandwidth = 5.5 Gb,
- Latency =3.53 microsecond,
- Number of tasks = 50,
- Number of clients = 50,
- Dataset size = 500 MB.

Each test case is associated with 3 executions. Table 6.11 lists the executions corresponding to test case DR1TC2.

Test Case No.	Test Case Execution No.	No. of modifications	Total execution time (sec)
DR1TC2	DR1TC2E1	1	2051.91
	DR1TC2E2	5	2255.91
	DR1TC2E3	10	2510.91

Table 6.11: Test case DR1TC2 executions

Test case execution analysis: DR1TC2

The execution trend for test case DR1TC2 is mentioned in the figure 6.7. The x-axis represents number of data modifications and y-axis represents the total execution time taken in seconds for replication.

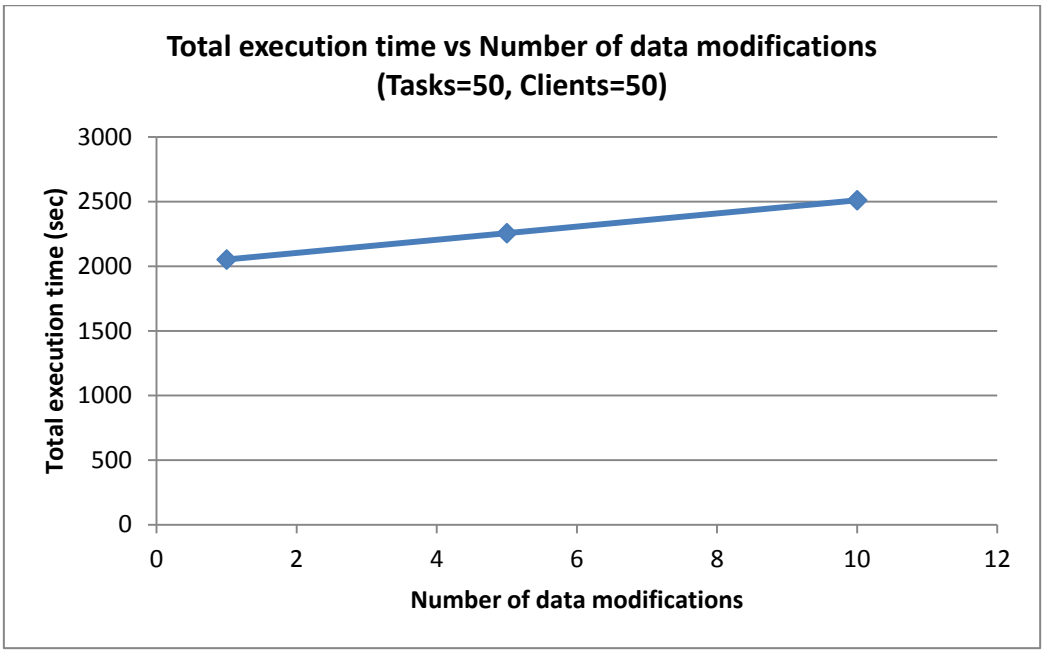

Figure 6.7: Execution trend for test case DR1TC2

The above figure shows the total execution time taken by varying number of modifications. It shows that the total execution time increases linearly when number of modifications significantly.

DR1TC3

In DR1TC3 test case, total execution time is measured by varying number of clients and by keeping other parameters value at their mean in realistic DG-based DCI environment.

Test case executions: DR1TC3

The number of clients is varied from 10 to 100. The data considered for this test case is as follows:

- Processing power of coordinator = 5.0 GHz,
- Processing power of client = 2.5 GHz,
- Computation size of each task = 5.00e+12 Flops,
- Communication size of each task = 4618 Bytes,
- Bandwidth = 5.5 Gb,
- Latency =3.53 microsecond,
- Dataset size = 500 MB,
- Number of modifications =10.

Each test case is associated with 3 executions. Table 6.12 lists the executions corresponding to test case DR1TC3.

Test Case No.	Test Case Execution No.	No. of clients	Total execution time (sec)
DR1TC3	DR1TC3E1	10	2055.19
DR1TC3	DR1TC3E2	50	2255.91
DR1TC3	DR1TC3E3	100	2506.83

Table 6.12: Test case DR1TC3 executions

Test case execution analysis: DR1TC3

The execution trend for test case DR1TC3 is mentioned in the figure 6.8. The x-axis represents number of clients and y-axis represents the total execution time taken in seconds for replication.

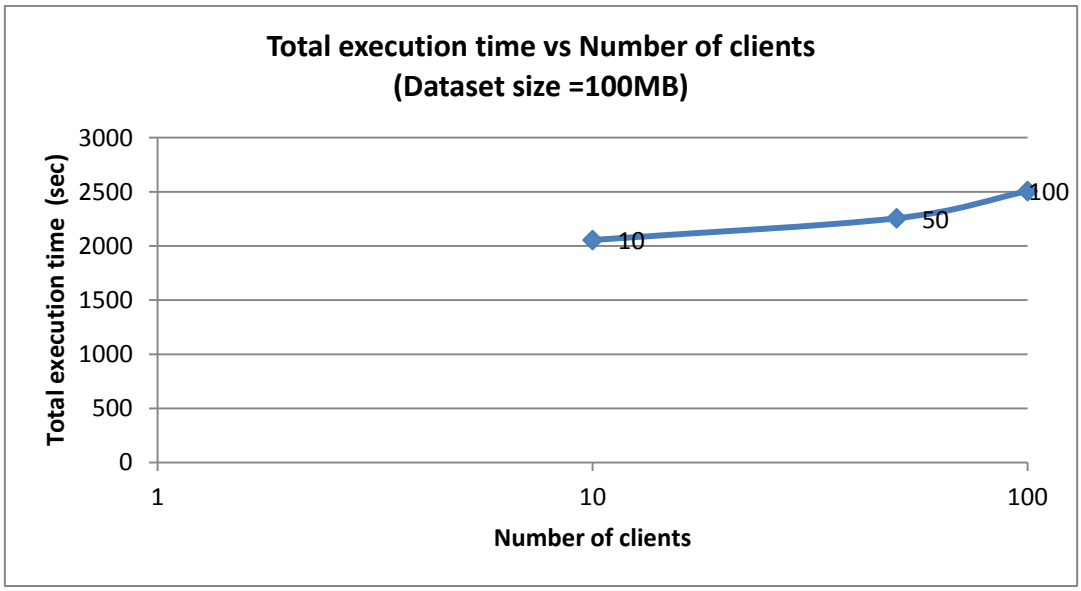

Figure 6.8: Execution trend for test case DR1TC3

The above figure shows the total execution time taken by varying number of clients. It shows that the total execution time increases gradually when numbers of clients are increased.

DR1TC4

In test case DR1TC4, total execution time for replication is measured by varying number of tasks and by keeping mean value of other parameters constant.

Test case executions: DR1TC4

The number of tasks is varied from 10 to 100. The data considered for this test case is as follows:

- Processing power of coordinator = 5.0 GHz,
- Processing power of client = 2.5 GHz,
- Computation size of each task = 5.00e+12 Flops,
- Communication size of each task = 4618 Bytes,
- Bandwidth = 5.5 Gb,
- Latency =3.53 microsecond,
- Dataset size = 500 MB,
- Number of modifications =10.

Each test case is associated with 3 executions. Table 6.13 lists the executions corresponding to test case DR1TC4.

Test Case No.	Test Case Execution No.	Number of Tasks	Total execution time (sec)
DR1TC4	DR1TC4E1	10	2055.19
DR1TC4	DR1TC4E2	50	2255.91
DR1TC4	DR1TC4E3	100	2506.83

Table 6.13: Test case DR1TC4 executions

Test case execution analysis: DR1TC4

The execution trend for test case DR1TC4 is mentioned in the figure 6.9. The x-axis represents number of tasks and y-axis represents the total execution time taken in seconds for replication.

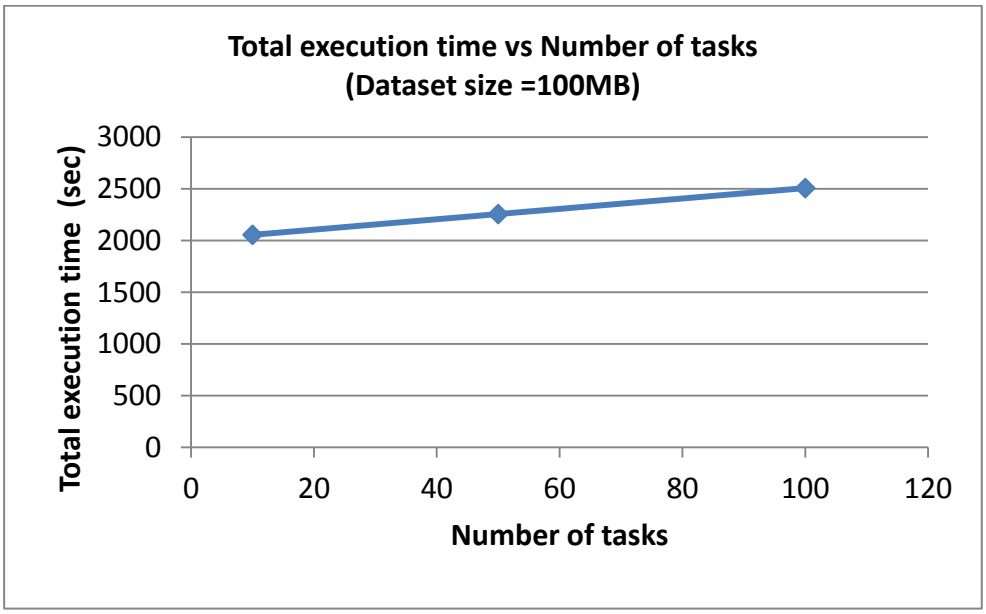

Figure 6.9: Execution trend for test case DR1TC4

The above figure shows the total execution time taken by varying number of tasks. It shows that the total execution time increases gradually when numbers of tasks and clients are increased.

DR1TC5

In DR1TC5 test case, total execution time is measured by varying by keeping other parameters value at their mean in realistic DG-based DCI environment.

Test case executions: DR1TC5

The processing power of client is varied from 1.5 to 3.5 GHz. The data considered for this test case is as follows:

- Processing power of coordinator = 5.0 GHz,
- Computation size of each task = 5.00e+12 Flops,
- Communication size of each task = 4618 Bytes,
- Bandwidth = 5.5 Gb,
- Latency =3.53 microsecond,
- Dataset size = 500 MB,
- Number of modifications =10,
- Number of clients =50,
- Number of tasks =50.

Each test case is associated with 3 executions. Table 6.14 lists the executions corresponding to test case DR1TC5.

Test Case No.	Test Case Execution No.	Processing power of client (GHz)	Total execution time (sec)
DR1TC5	DR1TC5E1	1.5	3589.25
DR1TC5	DR1TC5E2	2.5	2255.91
DR1TC5	DR1TC5E3	3.5	1684.49

Table 6.14: Test case DR1TC5 executions

Test case execution analysis: DR1TC5

The execution trend for test case DR1TC5 is mentioned in the figure 6.10. The x-axis represents processing power of client in GHz and y-axis represents the total execution time taken in seconds for replication.

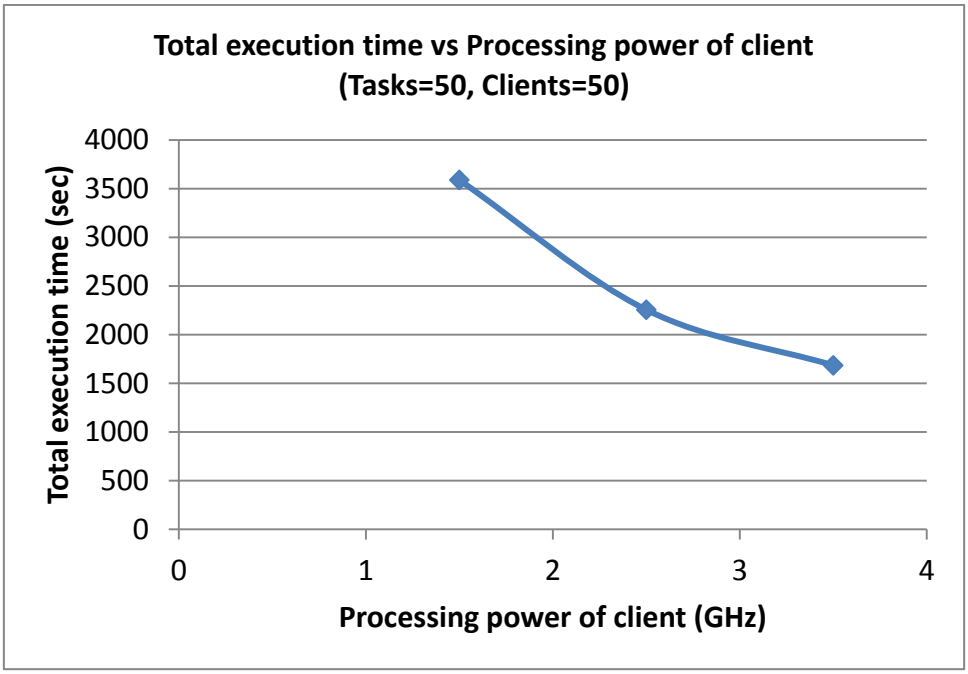

Figure 6.10: Execution trend for test case DR1TC5

The above figure shows the total execution time taken by varying processing power of client. It shows as processing power of client is increased, the execution time also decreases.

DR1TC6

In DR1TC6 test case, total execution time is measured by varying bandwidth and latency of link and by keeping other parameters value at their mean in realistic DG-based DCI environment.

Test case executions: DR1TC6

The bandwidth of link is varied from 1 to 10 Gb. The data considered for this test case is as follows:

- Processing power of coordinator = 5.0 GHz,
- Processing power of client = 2.5 GHz,
- Computation size of each task = 5.00e+12 Flops,
- Communication size of each task = 4618 Bytes,

- Dataset size = 500 MB,
- Number of modifications =10,
- Number of clients =50,
- Number of tasks =50.

Each test case is associated with 3 executions. Table 6.15 lists the executions corresponding to test case DR1TC6.

Test Case No.	Test Case Execution No.	Bandwidth of link (Gb)	Latency of link (Microsec)	Total execution time (sec)
DR1TC6	DR1TC6E1	1	0.37	2255.09
	DR1TC6E2	5.5	3.535	2255.009
	DR1TC6E3	10	6.7	2255.01

Table 6.15: Test case DR1TC6 executions

Test case execution analysis: DR1TC6

The execution trend for test case DR1TC6 is mentioned in the figure 6.11. The x-axis represents bandwidth in Gb and y-axis represents the total execution time taken in seconds for replication.

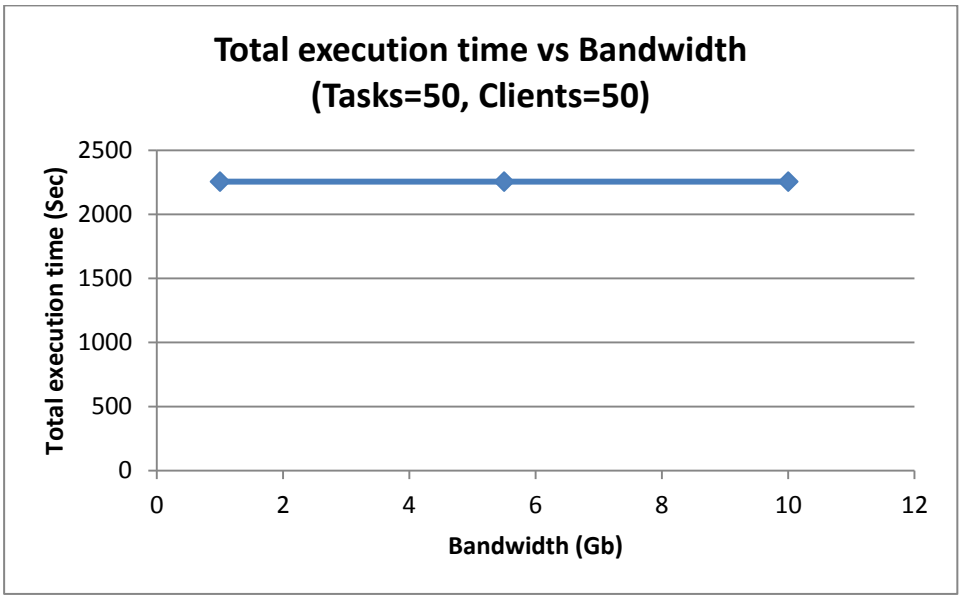

Figure 6.11: Execution trend for test case DR1TC6

The above figure shows the total execution time taken by varying bandwidth and latency of the link. There is very small variation in execution time when bandwidth and latency of link are varied.

DR1TC7

In DR1TC7 test case, total execution time is measured by varying task communication size and by keeping other parameters value at their mean in realistic DG-based DCI environment.

Test case executions: DR1TC7

The task communication size is varied from 20 to 9216 Bytes. The data considered for this test case is as follows:

- Processing power of coordinator = 5.0 GHz,
- Processing power of client = 2.5 GHz,
- Computation size of each task = 5.00e+12 Flops,
- Dataset size = 500 MB,
- Number of modifications =10,
- Number of clients =50,
- Number of tasks =50,
- Bandwidth = 5.5 Gb,
- Latency =3.53 microsecond.

In the current scenario, each test case is associated with 3 executions. Table 6.16 lists the executions corresponding to test cases DR1TC7.

Test Case No.	Test Case Execution No.	Communication size (Bytes)	Total execution time (sec)
DR1TC7	DR1TC7E1	20	2255.91
	DR1TC7E2	4618	2255.91
	DR1TC7E3	9216	2255.92

Table 6.16: Test case DR1TC7 executions

Test case execution analysis: DR1TC7

The execution trend for test case DR1TC7 is mentioned in the figure 6.12. The x-axis represents task communication size in Bytes and y-axis represents the total execution time taken in seconds for replication. The log to the base 10 is used for representing the value of communication size in x-axis.

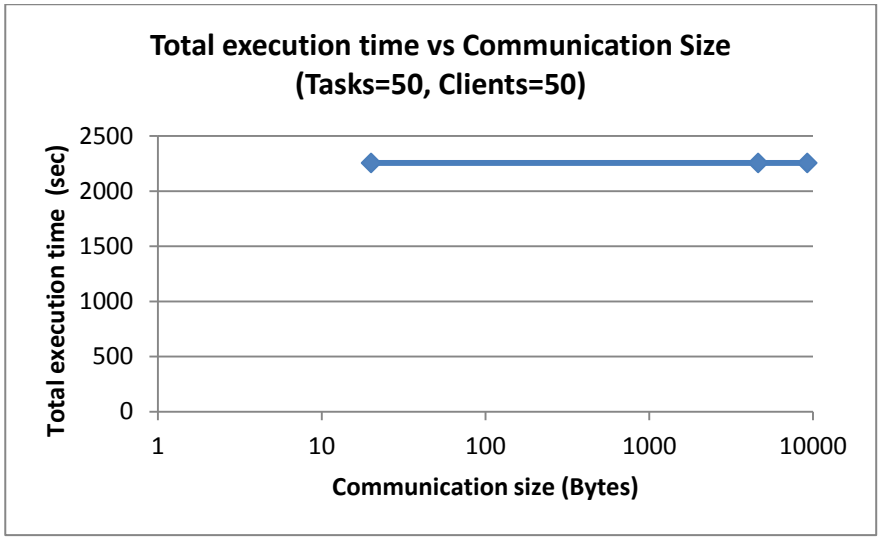

Figure 6.12: Execution trend for test case DR1TC7

The figure 6.12 shows the total execution time taken by varying task communication size. There is small variation in total execution time when communication size of task is varied.

DR1TC8

In DR1TC8 test case, total execution time is measured by varying processing power of coordinator and by keeping other parameters value at their mean in realistic DG-based DCI environment.

Test case executions: DR1TC8

The processing power of coordinator is varied from 3 to 5 GHz. The data considered for this test case is as follows:

- Processing power of client = 2.5 GHz,
- Computation size of each task = 5.00e+12 Flops,
- Communication size of each task = 4618 Bytes,
- Dataset size = 500 MB,
- Number of modifications =10,
- Number of clients =50,
- Number of tasks =50,
- Bandwidth = 5.5 Gb,
- Latency =3.53 microsecond.

In the current scenario, each test case is associated with 3 executions. Table 6.17 lists the executions corresponding to test cases DR1TC8.

Test Case No.	Test Case Execution No.	Processing power of Coordinator (GHz)	Total execution time (sec)
DR1TC8	DR1TC8E1	3	2255.91
	DR1TC8E2	5	2255.91
	DR1TC8E3	7	2255.91

Table 6.17: Test case DR1TC8 executions

Test case execution analysis: DR1TC8

The execution trend for test case DR1TC8 is mentioned in the figure 6.13. The x-axis represents processing power of coordinator in GHz and y-axis represents the total execution time taken in seconds for replication.

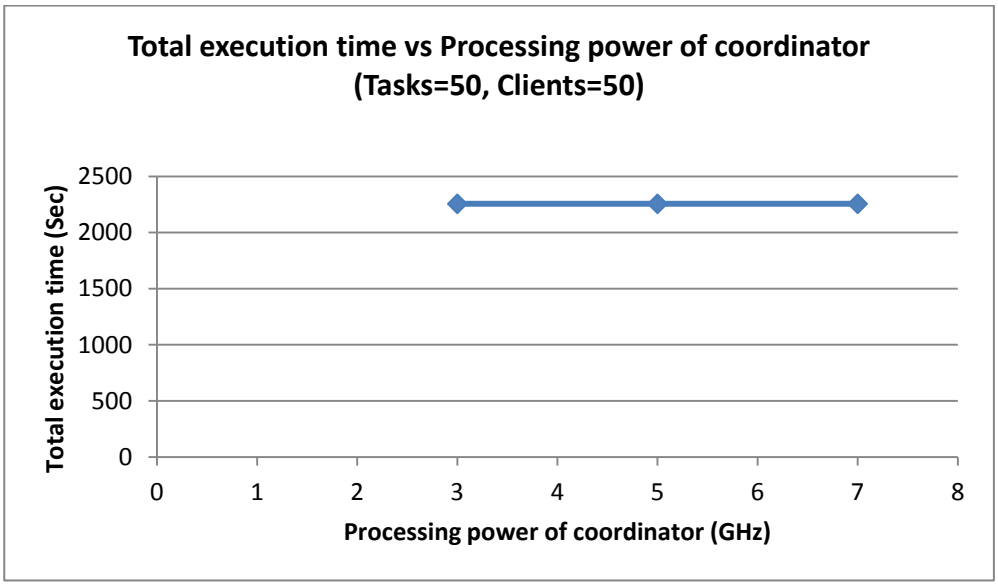

Figure 6.13: Execution trend for test case DR1TC8

The above figure shows the total execution time taken by varying processing power of coordinator. There is no variation in execution time when processing power of coordinator is varied.

DR1TC9

In DR1TC9 test case, total execution time is measured by varying dataset size in GB and by keeping other parameters value at their mean in realistic DG-based DCI environment.

Test case executions: DR1TC9

The dataset size is varied from 10 to 1000 GB. The data considered for this test case is as follows:

- Processing power of coordinator = 5.0 GHz,
- Processing power of client = 2.5 GHz,
- Computation size of each task = 5.00e+12 Flops,
- Communication size of each task = 4618 Bytes,
- Number of modifications =10,
- Number of clients =50,
- Number of tasks =50,
- Bandwidth = 5.5 Gb,
- Latency =3.53 microsecond.

In the current scenario, each test case is associated with 3 executions. Table 6.18 lists the executions corresponding to test cases DR1TC1.

Test Case No.	Test Case Execution No.	Dataset size (GB)	Execution time (sec)
DR1TC9	DR1TC9E1	10	2468.44
	DR1TC9E2	500	12927.01
	DR1TC9E3	1000	23598.99

Table 6.18: Test case DR1TC9 executions

Test case execution analysis: DR1TC9

The execution trend for test case DR1TC9 is mentioned in the figure 6.14. The x-axis represents dataset size in GB and y-axis represents the total execution time taken in seconds for replication.

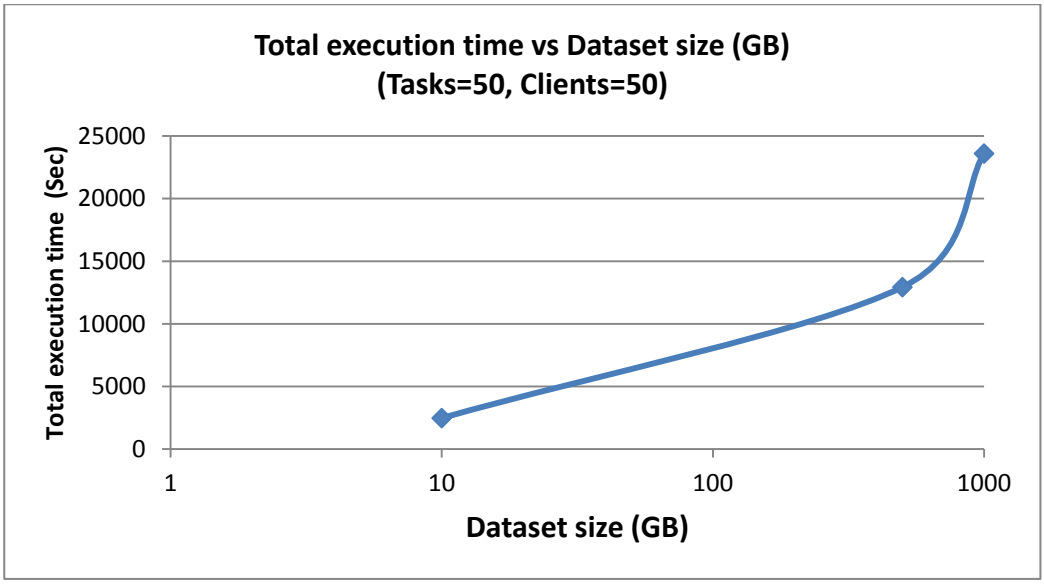

Figure 6.14: Execution trend for test case DR1TC9

The above figure shows the total execution time taken by varying dataset size in GB. The execution time increases gradually when data size is increased significantly.

6.2.3.2.1.1. DR1 Scenario Analysis

From the analysis of the above DR1 test cases, the ranking of parameters which influence the performance of replication is mentioned in table 6.18. The ranking of the parameters is done on standard deviation for the total execution time. The standard deviation is a measure of how widely values are dispersed from the mean value of a parameter. Standard deviation is a well established method to derive the dispersion between the values, so in this research it is used to find the most significant parameters.

The standard deviation is calculated by using the following formula:

Standard deviation (Parameter) = $\sqrt{\sum(x - x')^2/(n-1)}$

Where, x indicates the parameter value

x' indicates the average parameter value

n indicates the sample size whose value is 3.

The higher the standard deviation, higher parameter value will influence the performance and higher will be the ranking of parameter. The total execution time as per the parameter's significance are mentioned in table 6.19.

		Execution time (sec) for parameter			
Rank	Parameter Type	Min value	Max Value	Mean Value considered	Standard deviation
1	Dataset size (GB)	2468.44	23598.99	12927.01	10565.4500
2	Processing power of client	3589.25	1684.49	2255.91	977.4500
3	No. of modifications	2051.91	2510.91	2255.91	229.9700
4	No. of clients	2055.19	2506.83	2255.91	226.2800
5	Task set	2055.19	2506.83	2255.91	226.2800
6	Data size (MB)	2255.21	2275.85	2265.43	10.320
7	Bandwidth of link (Gb)	2255.09	2255.01	2255.01	0.0465
8	Task Communication size	2255.91	2255.92	2255.91	0.0058
9	Processing power of Coordinator	2255.91	2255.91	2255.91	0.0000

Table 6.19: Ranking of experimentation input parameters

As per table 6.19, the significant 3 parameters influencing performance are dataset size, processing power of clients, and number of modifications. In institutional DG-based DCI, the clients have same processing power, so the most significant parameters considered for influencing the performance are dataset size and number of modifications. Hence this result helps us identify less influencing parameters effecting performance, so that major attention can be paid to the difference making parameters. These two parameters are considered for further experiments in the comparative analysis of the proposed and existing architecture.

6.2.3.2.2. Scenario DR2: Replication Planning Strategies

The purpose of DR2 scenario is to first find the frequent access paths for replication and then plan for replication strategies to improve performance in DG-based DCI. DR2 scenario test cases are designed to find the replication path length which maximizes the total replication in DG-based DCI. The DR2's scenario consists of test cases as listed in table 6.20.

#	Test Case Number	Algorithm used
1	DR2TC1	Frequent pattern tree algorithm
2	DR2TC2	Prim algorithm
3	DR2TC3	Adaptive spanning tree algorithm

Table 6.20: Scenario DR2 test cases

Test Case: DR2TC1

The purpose of DR2TC1 test case is to determine the frequent access paths to improve the replication performance in DG-based DCI. It generates frequent access paths for replication by using FP-tree algorithm from the data collected from the Algorithm-3. The FP-tree algorithm and its working are described in the chapter 4.The input data to this algorithm is in the form of adjacency matrix. The data in the adjacency matrix shows the number of pieces of data transferred from one client to another by using P2P approach. This algorithm then generates the frequent access paths used for data transfer in the proposed architecture. The performance improves when replication is applied on the frequent access paths generated.

DR2TC1 Test case architecture

The architecture accepts input data in the form of adjacency matrix by FP-tree algorithm to generate tree with frequent access paths for data replication. The architecture diagram for the DR2TC1 test case is mentioned in the figure 6.15.

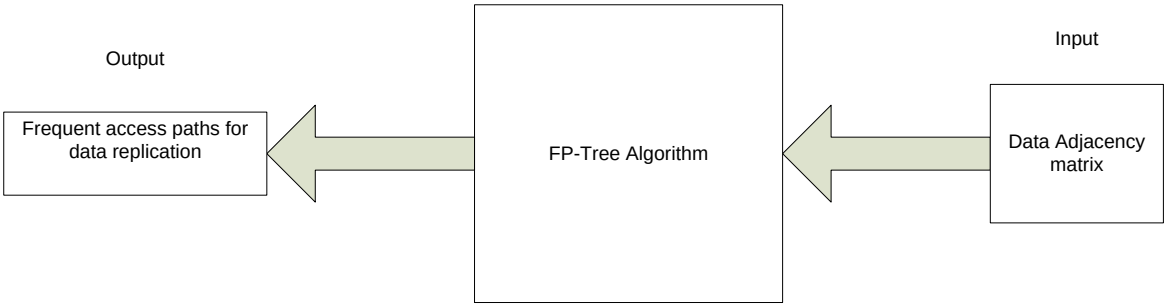

Figure 6.15: Architecture diagram for DR2TC1 test case

Test case executions: DR2TC1

Each test case is associated with 4 executions. Table 6.21 lists the executions corresponding to test case DR2TC1.

Test Case No.	Test Case Execution No.	Input Data	No. of Nodes (clients and coordinator)
DR2TC1	DR2TC1E1	Adjacency matrix for data replication	5
	DR2TC1E2	Adjacency matrix for data replication	11
	DR2TC1E3	Adjacency matrix for data replication	21
	DR2TC1E4	Adjacency matrix for data replication	51

Table 6.21: DR2TC1 Test case executions

The adjacency data matrix format used in Algorithm-4 for 4 clients and 1 coordinator is mentioned in table 6.22.

	Client-1	Client-2	Client-3	Client-4
Client-1	0	4	4	0
Client-2	0	0	0	0
Client-3	0	0	0	4
Client-4	0	0	0	0

Table 6.22: Adjacency data matrix format of 5 nodes for Algorithm-4

The value in a cell of adjacency cell represents the number of pieces transferred from one client to another client by using P2P techniques in the proposed architecture. Coordinator is acting as a tracker, so there is no data transfer between clients and coordinator. The location of the data is stored is informed to the coordinator by the clients. The data in the adjacency matrix is changed depending upon the number of clients and coordinator in the proposed architecture.

Test case execution analysis: DR2TC1

The frequent access paths generated by applying FP tree algorithm on DR2TC1test case executions are listed in table 6.23.

Test Case Execution No.	No. of Nodes (clients and	Frequent access paths for replication

	coordinator)	
DR2TC1E1	5	1➔2, 1➔3, 3➔4.
DR2TC1E2	11	1➔2, 1➔3, 4➔3, 6 ➔2, 7➔2, 9➔3, 8➔9, 10➔2.
DR2TC1E3	21	1➔2,1➔3,4➔3,5➔3,6➔2,9➔3, 10➔2, 11➔2,14➔3,15➔3,20➔2,18➔19.
DR2TC1E4	51	1➔2, 1➔3, 36➔2, 33➔3, 37➔2, 39➔2, 43➔3, 48➔2.

Table 6.23: Frequent access paths generated for DR2TC1 test case executions

Where,

➔Indicates the data access path for replication,

Number indicates the client number.

The frequent paths generated for test case DR2TC1E1 execution by applying FP tree are from client1 to client 2, client 1 to client 3, and client 3 to 4. The performance of the replication in DG-based DCI improves when the data replication is applied on these frequent access paths. Similarly, data replication is applied on the frequent access paths generated for other test cases.

Test case: DR2TC2

The purpose of DR2TC2 test case is to determine the minimum spanning tree for providing maximum replication. It generates the minimum spanning tree by applying Prim's algorithm to improve replication performance from the data collected by algorithm-3in DG-based DCI. The working of prim algorithm is described in the chapter 4. The input to this algorithm is replication data in the form of adjacency matrix. This algorithm then generates the minimum spanning tree where replication should be applied to improve the replication.

DR2TC2 Test case architecture

The architecture accepts input data in the form of adjacency matrix which is then processed by prim algorithm to generate a spanning tree having maximum replication. The architecture diagram for the test case DR2TC2 is mentioned in the figure 6.16.

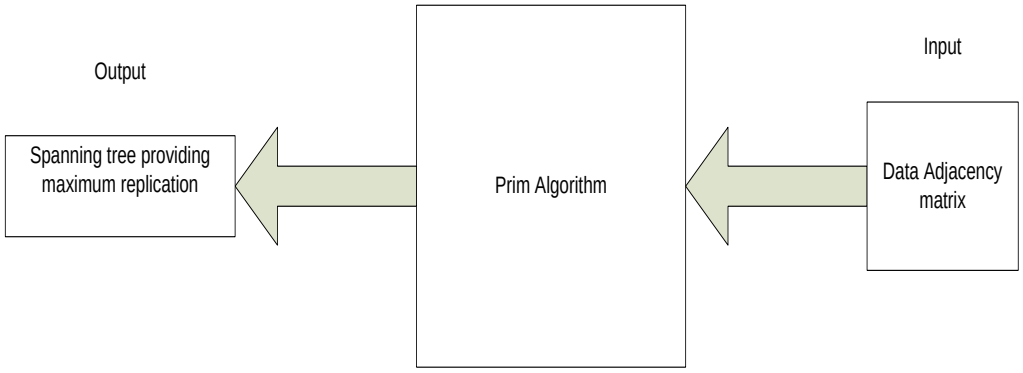

Figure 6.16: Architecture diagram for DR2TC2 test case

Test case executions: DR2TC2

Each test case is associated with 2 executions. The data format of adjacency matrix used is input to Prim's algorithm is same as mentioned in table 6.21.Table 6.24 lists the executions corresponding to test case DR2TC2.

Test Case No.	Test Case Execution No.	Input Data	No. of Nodes (clients and coordinator)
DR2TC2	DR2TC2E1	Adjacency matrix for data replication	5
	DR2TC2E2	Adjacency matrix for data replication	11

Table 6.24: DR2TC2 Test case executions

Test case execution analysis: DR2TC2

The spanning trees generated by applying Prim's algorithm on DR2TC2 test case executions are listed in table 6.25.

Test Case Execution No.	No. of Clients and Coordinator	Spanning tree for replication	Spanning tree path length for replication
DR2TC2E1	5	1➔ 2,1➔3,3➔4	12
DR2TC2E2	11	1➔2,1➔3,2➔5,2➔6, 2➔7,2➔10, 3➔4,3➔9, 4➔8,	26

Table 6.25: DR2TC2 Test case executions outcomes

Where,

➔Indicates the data access path for replication,

Number indicates the client number.

The minimum spanning tree generated for test case DR2TC2E1 execution by applying Prim algorithm from client1 to client 2, client 1 to client 3, and client 3 to 4. The spanning tree path length for replication represents the total number of data transferred between nodes for replication in the spanning tree.

The spanning tree for replication for test case execution DR2TC2E1 is mentioned in the figure 6.17. The vertex represents the client number and weight between two vertexes represents the number of data pieces in a dataset transferred by using P2P techniques in DG-based DCI. The total spanning tree length for maximum replication for this test case is 12.

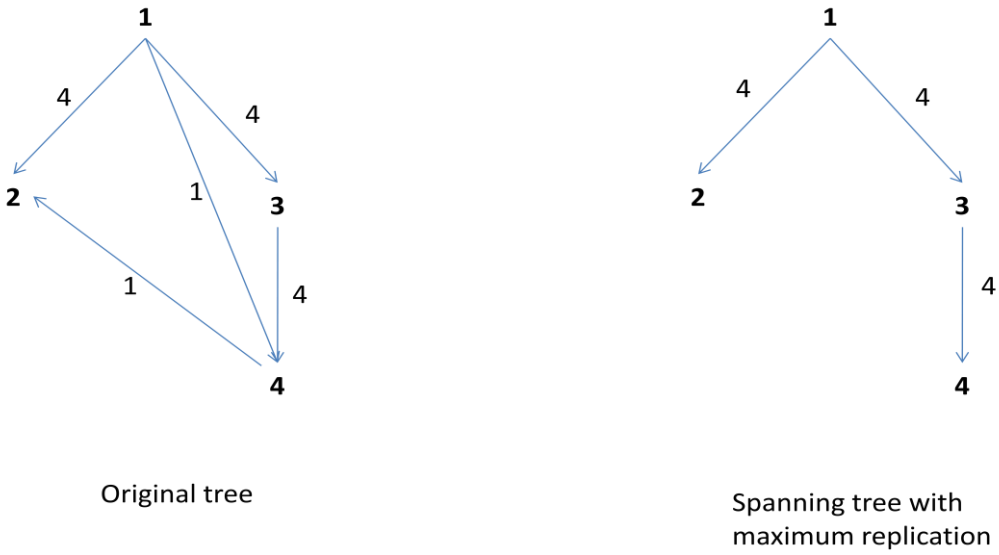

Figure 6.17: Spanning tree for replication for DR2TC2E1 execution

The spanning tree for replication for test case execution DR2TC2E2 is mentioned in the figure 6.18.The total spanning tree length for maximum replication for this test case is 12.

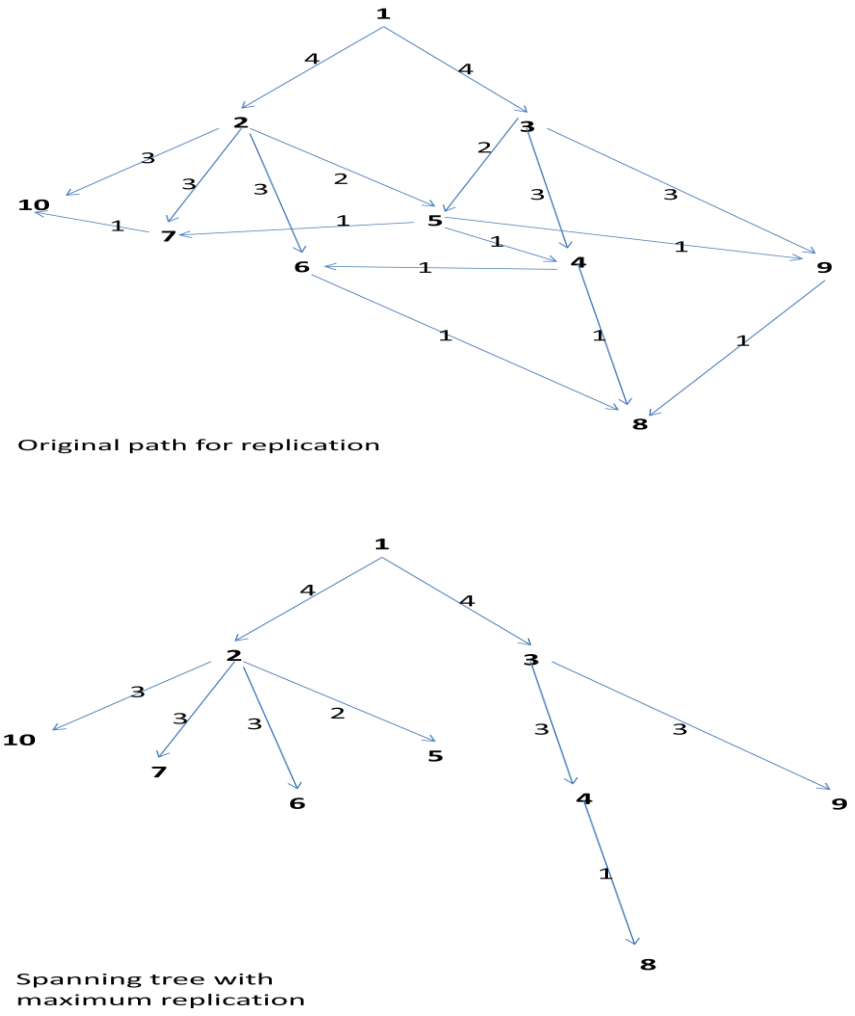

Figure 6.18: Spanning tree for replication for DR2TC2E2 execution

The performance of the replication in DG-based DCI improves when the data replication is applied on the minimum generated spanning tree paths. Similarly, data replication is applied on the frequent access paths generated for other test cases.

Test Case: DR2TC3

In DG environment, the clients can leave/join the network at any time. So an adaptive algorithm is required for DG-based DCI. The purpose of DR2TC3 test case is to determine the best access paths in DG-based DCI to improve the replication performance. A best access path is defined as the minimum cost of edge of each client having maximum data transfer from source client to destination client neighbour. It generates best access paths for replication by using adaptive multipath spanning tree from the replication data collected from the Algorithm-3. The design and working of adaptive multipath spanning

tree is described in the chapter 4. The input to this algorithm is replication data in the form of adjacency matrix. If some client goes down, then this algorithm generates the new best access paths for replication.

DR2TC3 Test case architecture

The architecture accepts input data in the form of adjacency matrix which is then processed by adaptive multipath spanning tree algorithm to generate best paths for replication. The architecture diagram for the test case DR2TC3 is mentioned in the Figure 6.19.

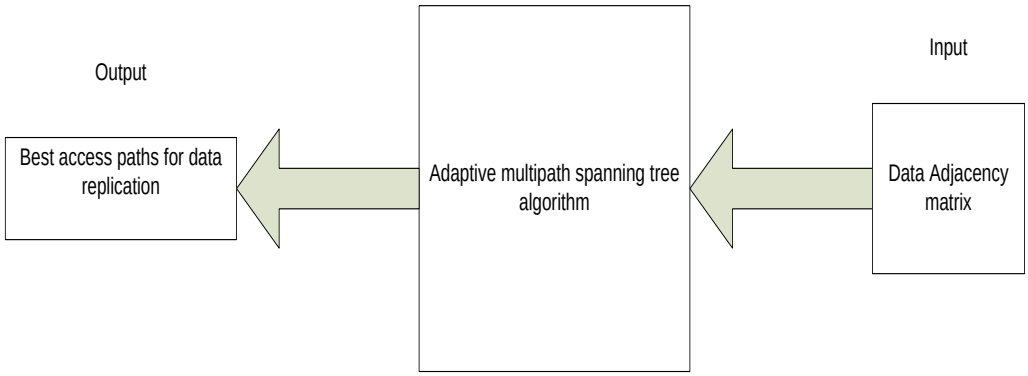

Figure 6.19: Architecture diagram for DR2TC3 test case

Test case executions: DR2TC3

Each test case is associated with two executions. The data format of adjacency matrix used is input to Prim's algorithm is same as mentioned in table 6.21.Table 6.26 lists the execution corresponding to test case DR2TC3.

Test Case No.	Test Case Execution No.	Input Data	No. of Nodes (Clients and Coordinator)
DR2TC3	DR2TC3E1	Adjacency matrix for data replication	7
	DR2TC3E2	Adjacency matrix for data replication	7

Table 6.26: DR2TC3 Test case executions

Test case execution analysis: DR2TC3

The spanning tree generated by applying adaptive spanning tree algorithm on DR2TC3 executions is listed in table 6.27.

Test Case Execution No.	No. of nodes (Clients and Coordinator)	Client number down (not working)	Best access paths for replication	Spanning tree path length for replication
DR2TC3E1	7	-	1➔2, 1➔6, 2➔3, 2➔5, 3➔4.	16
DR2TC3E2	7	3	1➔2, 1➔6, 5➔4, 2➔5.	11

Table 6.27: Test case DR2TC3 executions outcomes

Where,

➔Indicates the data access path for replication,

Number indicates the client number.

The spanning tree generated for test case DR2TC3E1 execution by applying adaptive multipath spanning tree is mentioned in the figure 6.20.

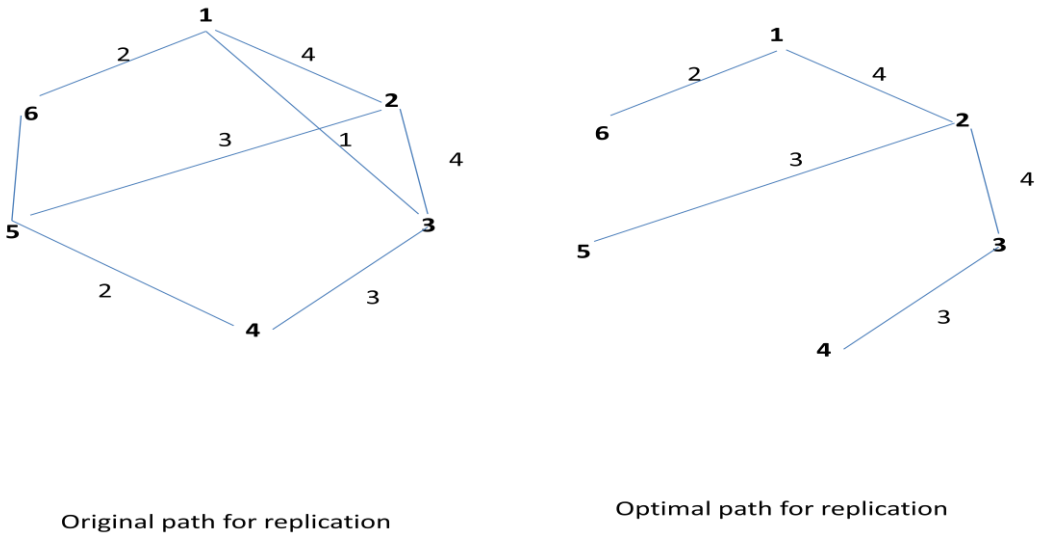

Figure 6.20: Spanning tree path generated by DR2TC3E1 execution

The vertex represents the client number and weight between two vertexes represents the number of data pieces in a dataset transferred by using P2P techniques in DG-based DCI. The spanning tree path length for replication represents the total number of data transferred between nodes for replication in the spanning tree.

The spanning tree for replication for test case execution DR2TC3E2 is mentioned in the figure 6.21.

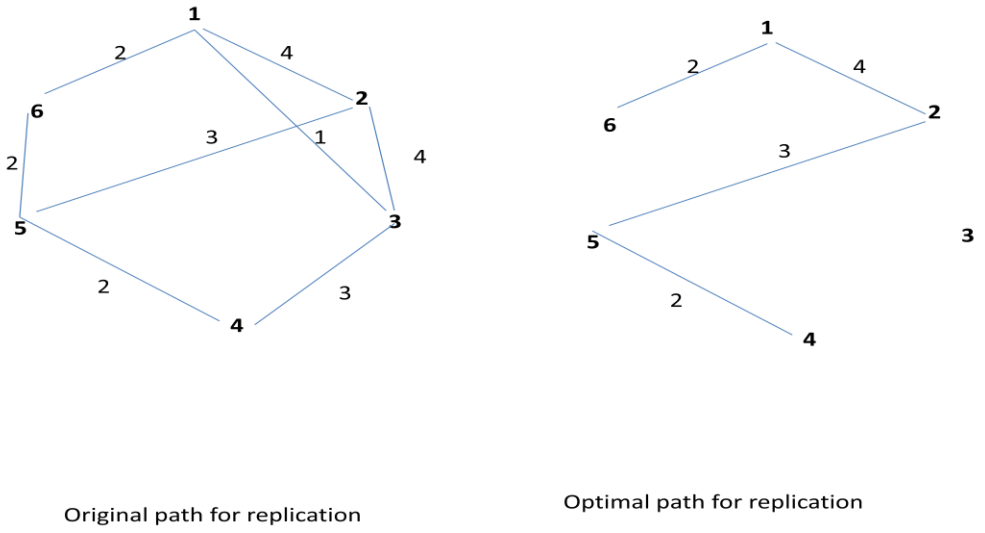

Figure 6.21: Spanning tree path generated by DR2TC3E2 execution

Adaptive multipath spanning tree algorithm generates the new spanning tree for the replication when some of the nodes are down in DG-based DCI. The performance of the replication improves when the data replication is applied on the above new generated spanning tree paths.

6.2.3.2.2.1. Data replication Scenario DR2 analysis

The Algorithm-4 generates the frequent access paths for maximum replication in DG-based DCI. The replication cost is reduced when replication is applied on the frequent path generated. Thus proposed algorithms validate the research contribution towards knowledge in development of novel algorithm for handling large volumes of data in proposed P2P-based architecture for DG-based DCI.

6.2.4. P2P-based Architecture Experiments

6.2.4.1. Overview

The purpose of this type of experiments is to compare the relative performance of proposed architecture with the existing architecture in DG-based DCI. As per the analysis outcomes from the DR1 scenario analysis, the significant parameters identified for the comparative analysis are data size and number of modifications. So, the performance comparison of experiments is done on these parameters for the proposed and existing architecture. The performance is measured in terms of total time execution and number of messages passed for the comparative analysis of architectures.

6.2.4.2. Scenario

The scenarios for the comparative analysis of the experiments are classified in four scenarios as mentioned in the table 6.28.

Scenario No.	Description
PDG1	This scenarios deal with the varying data size when number of modifications is kept constant in proposed architecture.
EDG1	This scenarios deal with the varying data size when number of modifications is kept constant in existing architecture.
PDG2	This scenarios deal with the varying number of modifications when data size is kept constant in proposed architecture.
EDG2	This scenarios deal with the varying number of modifications when data size is kept constant in existing architecture.

Table 6.28: Comparison of proposed and existing architecture experiments Scenarios

Where,

PDG indicates Proposed Desktop Grid,

EDG indicates Existing Desktop Grid.

6.2.4.3. Test Cases

6.2.4.3.1. PDG1 and EDG1 Test Cases

The data considered for PDG1 and EDG1 test cases is as follows:

- Processing power of coordinator = 5.0 GHz,
- Processing power of client = 2.5 GHz,
- Computation size of each task = 5.00e+12 Flops,
- Communication size of each task = 4618 Bytes,
- Bandwidth = 5.5 Gb,
- Latency =3.53 microsecond,
- Number of tasks = 50,
- Number of clients = 50,
- Number of modifications= 200.

In test case PDG1, total execution time for replication is measured by varying dataset size. The dataset size is varied from 10 to 100000 MB. The PDG1 scenario consists of test case as listed in table 6.29.

Scenario No.	Test Case Number	Parameter considered
PDG1	PDG1TC1	Dataset size (MB)

Table 6.29: PDG1 scenario test case

In the scenario PDG1, test case PDG1TC1 is associated with 13 executions. Table 6.30 lists the executions corresponding to test cases PDG1TC1.

Test Case No.	Test Case Execution No.	Dataset size in MB	Total Execution time for proposed architecture
PDG1TC1	PDG1TC1E1	10	12200.21
	PDG1TC1E2	20	12200.42
	PDG1TC1E3	50	12201.05
	PDG1TC1E4	100	12202.09
	PDG1TC1E5	200	12204.17
	PDG1TC1E6	500	12210.43
	PDG1TC1E7	1000	12220.85
	PDG1TC1E8	2000	12241.69
	PDG1TC1E9	5000	12304.22
	PDG1TC1E10	10000	12408.44
	PDG1TC1E11	20000	12616.88
	PDG1TC1E12	50000	13242.19
	PDG1TC1E13	100000	14284.39

Table 6.30 Test case PDG1TC1executions

In test case EDG1, total execution time for replication is measured by varying dataset size. The dataset size is varied from 10 to 100000 MB. The PDG1 scenario consists of test case as listed in table 6.31. We have taken the assumption that when a modification is done in existing architecture then entire dataset has to be replicated in all the nodes in DG-based DCI.

Scenario no.	Test Case Number	Parameter considered
EDG1	EDG1TC1	Dataset size (MB)

Table 6.31: EDG1 scenario test cases

In the scenario EDG1, test case EDG1TC1 is associated with 13 executions. Table 6.32 lists the executions corresponding to test cases EDG1TC1.

Test Case No.	Test Case Execution No.	Dataset size in MB	Total Execution time for existing architecture
EDG1TC1	EDG1TC1E1	10	53000.21
	EDG1TC1E2	20	53000.42
	EDG1TC1E3	50	53001.05
	EDG1TC1E4	100	53002.09
	EDG1TC1E5	200	53004.18
	EDG1TC1E6	500	53010.43
	EDG1TC1E7	1000	53020.85
	EDG1TC1E8	2000	53041.69
	EDG1TC1E9	5000	53104.23
	EDG1TC1E10	10000	53208.45
	EDG1TC1E11	20000	53416.89
	EDG1TC1E12	50000	54042.19
	EDG1TC1E13	100000	55084.39

Table 6.32: Test case EDG1TC1 executions

6.2.4.3.1.1. PDG1 and EDG1 Scenario Execution Analysis

The comparative analysis of the execution trends for the test cases PDG1TC1 and EDG1TC1 for the same set of input data is illustrated in figure 6.22. The x-axis represents dataset size in MB and y-axis represents the total execution time taken in seconds for replication. The log to the base 10 is used for representing the value of dataset size in x-axis.

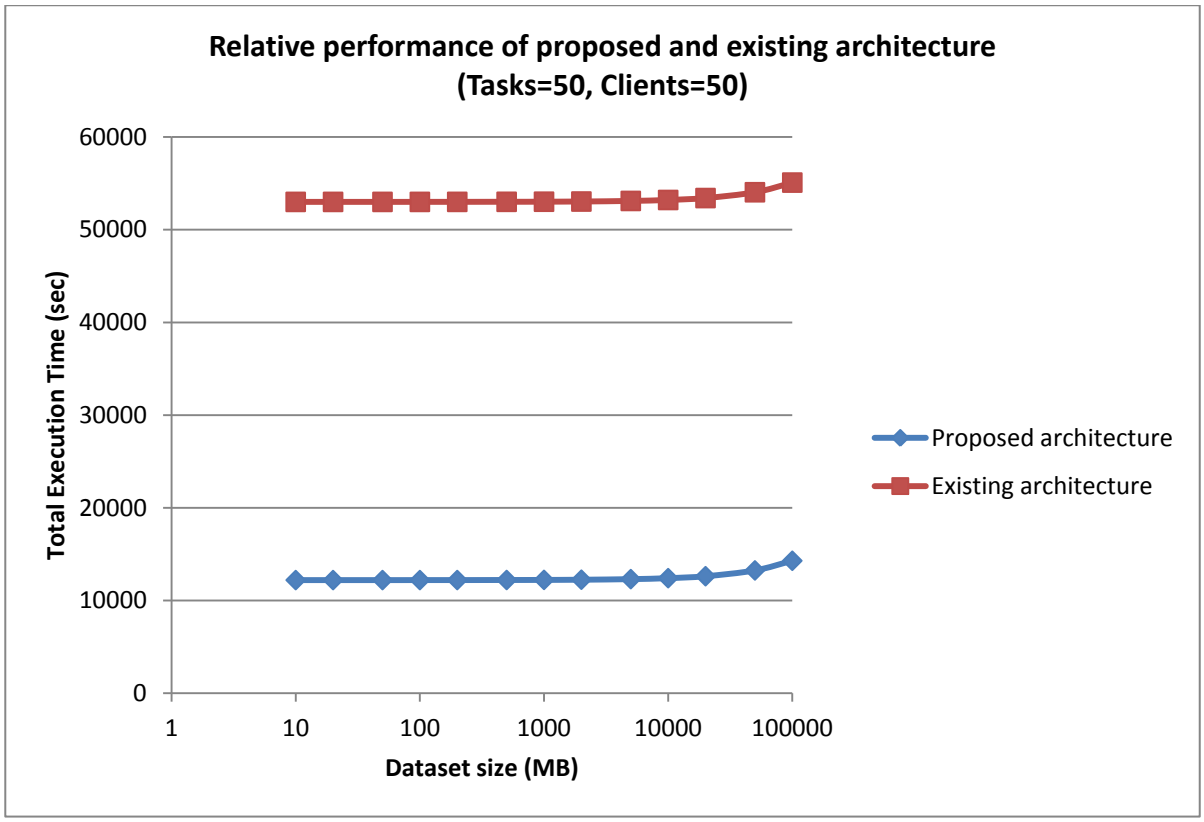

Figure 6.22: Comparative analysis of test cases PDG1TC1 and EDG1TC1

The performance of proposed architecture is better than the existing architecture for same number of modifications. The total execution time is increased by 0.17% when the dataset size is increased by 100 times from 10 to 1000 MB. The total execution time is increased by 1.71% when the dataset size is increased by 1000 times from 10 to 10000 MB. The total execution time is increased by 17.08% when the dataset size is increased by 10000 times from 10 to 100000 MB. The execution time increases very less when dataset size is increased significantly.

6.2.4.3.2. PDG2 and EDG2 Test Cases

The data considered for PDG2 and EDG2 test cases is as follows:

- Processing power of coordinator = 5.0 GHz,
- Processing power of client = 2.5 GHz,
- Computation size of each task = 5.00e+12 Flops,
- Communication size of each task = 4618 Bytes,
- Bandwidth = 5.5 Gb,
- Latency =3.53 microsecond,
- Number of tasks = 50,
- Number of clients = 50,

- Dataset size= 100 GB.

In test case PDG2, total execution time and messages exchanged for replication is measured by varying number of data modifications. The data modifications are varied from 10 to 1000. The PDG2 scenario consists of test case as listed in table 6.33.

Scenario No.	Test Case Number	Parameters considered
PDG2	PDG2TC1	Number of modifications

Table 6.33: PDG2 scenario test case

In scenario PDG2, test case PDG2TC1 is associated with eight executions. Table 6.34 lists the executions corresponding to test cases PDG2TC1.

Test Case No.	Test Case Execution No.	Number of modifications	Total Execution time for proposed architecture	Number of messages exchanged for maintaining consistency in proposed approach
PDG2TC1	PDG2TC1E1	10	23853	560
	PDG2TC1E2	20	24363	1070
	PDG2TC1E3	50	25893	2600
	PDG2TC1E4	100	28443	5150
	PDG2TC1E5	200	33543	10250
	PDG2TC1E6	500	48843	25550
	PDG2TC1E7	700	59043	35750
	PDG2TC1E8	1000	74343	51050

Table 6.34: Test case PDG2TC1 executions

In test case EDG2, total execution time and messages exchanged for replication is measured by varying number of data modifications. The data modifications are varied from 10 to 1000. The EDG2 scenario consists of test case as listed in table 6.35.

Scenario No.	Test Case Number	Parameters considered
EDG2	EDG2TC1	Number of modifications

Table 6.35: EDG2 scenario test case

In scenario EDG2, each test case EDG2TC1 is associated with 8 executions. Table 6.36 lists the executions corresponding to test cases EDG2TC1.

Test Case No.	Test Case Execution No.	Number of modifications	Total Execution time for existing architecture*	Number of messages exchanged for maintaining consistency in existing approach*
EDG2TC1	EDG2TC1E1	10	74343	51050
	EDG2TC1E2	20	74343	51050
	EDG2TC1E3	50	74343	51050
	EDG2TC1E4	100	74343	51050
	EDG2TC1E5	200	74343	51050
	EDG2TC1E6	500	74343	51050
	EDG2TC1E7	700	74343	51050
	EDG2TC1E8	1000	74343	51050

Table 6.36: Test case EDG2TC1 executions

* Since existing DG-based architecture does not support modifications of data.

We have taken the assumption that when a modification is done in existing architecture then entire dataset has to be replicated in all the nodes in DG-based DCI. So, the value of total execution time and number of messages exchanged in existing approach will be same as for 10 data modifications.

6.2.4.3.2.1. PDG1 and EDG1 Scenario Execution Analysis

Figure 6.23 illustrates the comparative analysis of performance in terms of execution time for the test cases PDG2TC1 and EDG2TC1 for the same set of input data. The x-axis represents number of modifications in a dataset and y-axis represents the total execution time taken in seconds for replication. The log to the base 10 is used for representing the value of modifications in x-axis.

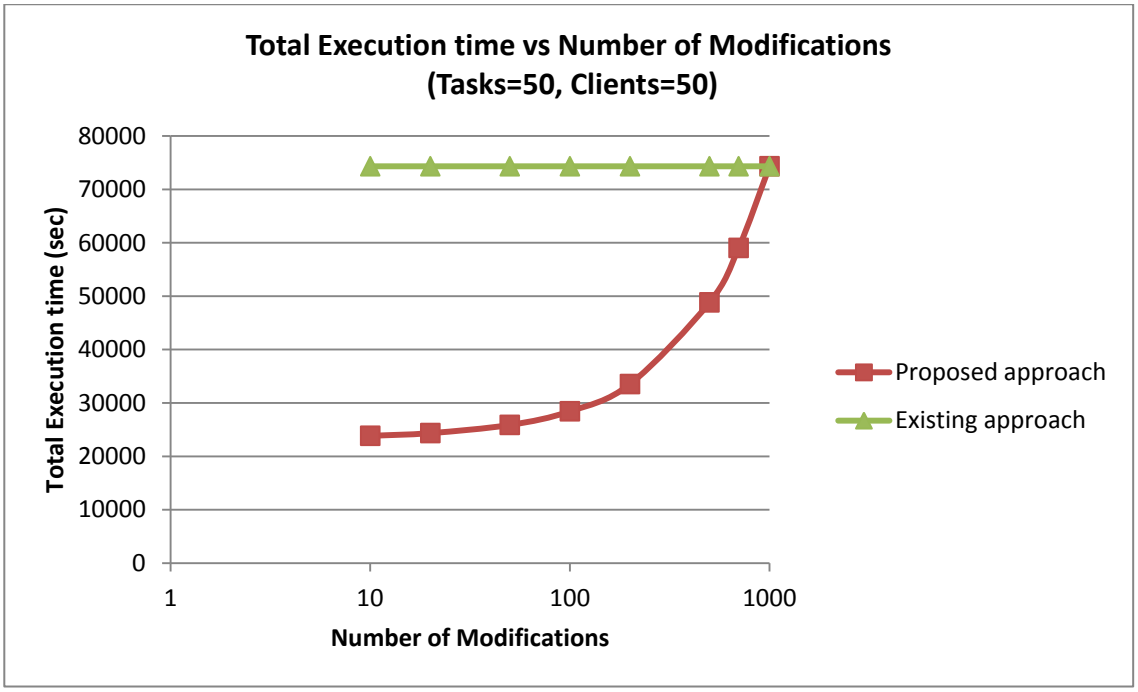

Figure 6.23: Comparative analysis of execution time for test cases PDG2TC1 and EDG2TC1

The above figure shows that the total execution time in proposed approach is less when numbers of modifications are less than the total number of pieces in a dataset. When number of modifications are equal or more than the total number of pieces in a dataset then the existing architecture performs well. The number of modifications will be less than the total number of pieces in a dataset in DG-based DCI. So, the proposed architecture performs better in terms of total execution time than the existing architecture for these test cases.

Figure 6.24 illustrates the comparative analysis of performance in terms of message exchanges for the test cases PDG2TC1 and EDG2TC1 for the same set of input data. The x-axis represents number of modifications in a dataset and y-axis represents the number of messages exchanged for replication. The log to the base 10 is used for representing the value of modifications in x-axis.

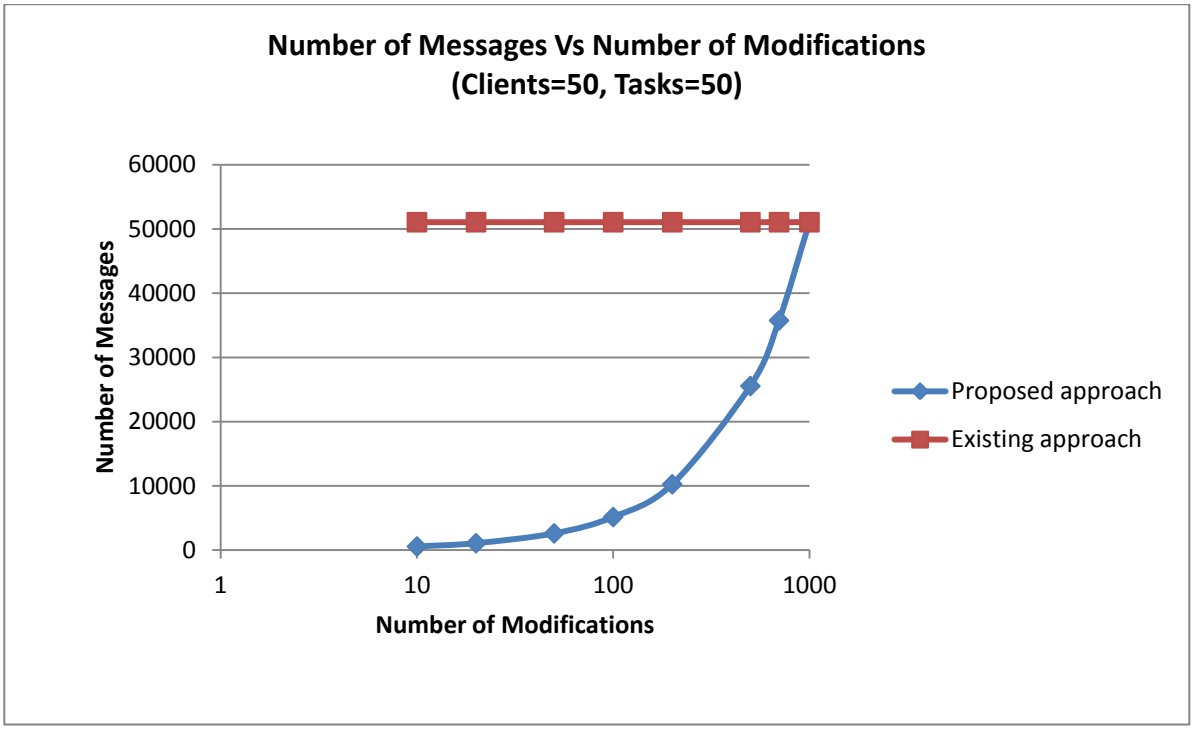

Figure 6.24: Comparative analysis of message exchanged for test cases PDG2TC1 and EDG2TC1

The above figure shows that the total number of messages exchanged in proposed approach is less when numbers of modifications are less than the total number of pieces in a dataset. When number of modifications are equal or more than the total number of pieces in a dataset then the existing architecture performs well. So, the proposed architecture performs better in terms of total number of messages exchanged than the existing architecture for these test cases.

6.2.4.4. Scenario Analysis

The proposed architecture performs better than the existing architecture as numbers of modifications in a dataset is less in DG-based DCI. Also, figure 6.23 and 6.24 shows that there is positive correlation between number of messages exchanged and execution time. The total execution time and number of messages exchanged increases as number of modifications increases in a dataset. Thus proposed architecture validates the research contribution towards knowledge in design and development of a novel architecture for handling large volumes of data in DG-based DCI.

6.3. Post-mortem Analysis

The proposed architecture performs better than the existing architecture when numbers of modifications in a dataset is less than the total dataset size in DG-based DCI. The new

features of Read/Write and handling large volumes of data are achieved in the proposed architecture. Concurrent data conflicts and maintaining data consistency due to modifications of data is achieved by using Algorithm-1, Algorithm-2, Adaptive multipath spanning tree algorithm in DG-based DCI. The experimental analysis validates the research contribution towards knowledge in design and development of a novel architecture for handling R/W and large volumes of data in DG-based DCI.

6.4. Conclusion

The objective of this chapter was to perform a comparative performance analysis between proposed and existing architecture models when new features of data R/W and handling large volumes of data are added to the existing architecture model in DG-based DCI. This was achieved by performing a set of experiments on the testbed. The experiments results served the purpose of finding out the scenarios and conditions in which proposed architecture outperforms present architecture as well as when it does not. In conclusion, the main outcomes of this chapter are the following:

- Set of experiments were organised in three categories: Data Consistency Experiments, Data Replication Experiments, and P2P-based Architecture Experiments where each contains multiple scenarios.
- Analysis of the result outcomes in different scenarios for the set of experiments.
- Justification of the proposed architecture for the scenario in which it outperforms existing architecture.

Chapter 7

Conclusions

7.1. Summary

The objective of this research is to introduce a novel architecture, developing algorithms for data conflict resolution, maintaining data consistency, data replication for handling large volumes of data and R/W of data in Desktop Grid-based DCI. This research has led to the creation of new P2P-based architecture that has added new functionality of handling R/W of data in the existing DG-based DCI. These features will be valuable to the emerging application in DG-based DCI. The problem statement includes a generalisation of the process of handling large volumes of data in DG-based DCI.

Chapter 1 introduces the research framework. The goal of the research is to identify architecture and algorithms in DG-based DCI which are efficient for handling large volumes of data. The descriptions of main research contributions to the research have been discussed in this chapter. A novel P2P-based architecture has been identified in DG-based DCI. New architecture has been reconstructed by the applying the good and proven aspects of P2P architecture for handling large volumes of data.

Chapter 2 provides a thorough background research and related work in DG-based DCI. The main aspects of this chapter were a discussion of related work and solution for handling large volumes of data in DG-based DCI. The outcome of this chapter was that the performance of existing data solutions dealing with generic architecture for data management is not efficient in DG-based DCI. Due to increasing demand of large volumes of data in scientific experiments, new types of applications are emerging. These applications require shared data storage, updating of some specific data from massive dataset, etc. The conclusion of this chapter was that the existing general data solutions are not appropriate for handling large volumes of data in DG-based DCI. There is a need for new architecture for the present Desktop Grid for handling large volumes of data and to support R/W of data due to emerging applications.

Chapter 3 deals with the suitability analysis of existing P2P-based techniques for handling large volumes of data in DG-based DCI. It also deals with the analysis of handling R/W of data in P2P-based architecture for the emerging applications in DG-based DCI. The outcomes of this chapter are that P2P-based architecture is suitable for the research for handling large volumes of data in DG-based DCI. The replication and consistency are handled in the proposed architecture by using modified Two Phase Protocol along with

time stamp properties for data concurrency. The need of new algorithms required to improve the overall performance in the proposed architecture has been addressed in this chapter.

Chapter 4 identifies a novel architecture and new algorithms in DG-based DCI for handling large volumes of data. New architecture has been identified and proposed in DG-based DCIto improve the performance of large volumes of data handling. New architecture has been reconstructed by the applying the good and proven aspects of P2P architecture for handling large volumes of data. The workings of proposed P2P coordinator and P2P client's architectures have been mentioned in this chapter.This chapter also deals with the design of efficient algorithms for handling the R/W of data consistency and replication.

Chapter 5 introduces the objectives related to experimental testbed and simulation design. The first objective was to introduce the experimental testbed architecture. The requirements of the simulation environment along with its constraints were discussed in it. The second objective was to present the simulation design for the conduction of experiments. This chapter also discusses the structure of data used, experiment simulation, and representation of results.

Chapter 6 describes the experiments results which were based on the implementation of novel architecture and algorithms for handling large volumes of data. This chapter performs a comparative performance analysis between proposed and existing architecture models when new features of data R/W and handling large volumes of data are added to the existing architecture model in DG-based DCI. It was achieved by performing a set of experiments on the testbed designed in chapter 5.The analysis of the result outcomes in different scenarios for the set of experiments are conducted in this chapter. The experiments results served the purpose of finding out the scenarios and conditions and its justification in which proposed architecture outperforms present architecture as well as when it does not.

7.2. Knowledge contributions

The contributions to knowledge in the research are classified in three broad categories as follows:

1. Data Consistency Algorithms: This research contribution deals with development of new concurrency control techniques for the novel P2P-based architecture in DG-based DCI. It uses modified 2PC protocol for improving the performance of data replication and consistency simultaneously. The contribution deals with a proposed algorithm that is used

for resolving conflicts due to concurrent Read/Write operations in P2P coordinator. The contribution also deals with an algorithm used for maintaining data consistency for conflict Read/Write data operations in P2P coordinator and P2P clients.

2. Data Replication Algorithm: This research contribution deals with development of new algorithms for handling data replication for the new P2P-based architecture in DG-based DCI. The first contributed algorithm is used for measuring the performance of replication in DG-based DCI. It also maintains data replication statistics of different clients. The second contributed algorithm is used for data planning and distribution strategies. It firsts finds the optimal access paths from the data collected from the replication and then applies data replication to improve the performance of data replication.

3. P2P-based Architecture: This research contribution deals with a novel P2P-based architecture for handling large volume of data in DG-based DCI. The first contribution deals with a new P2P-based coordinator architecture which accepts and resolves any write operations due to concurrent writes. The coordinator also acts as a tracker which maintains information about the location of clients in the P2P-based architecture. It coordinates the transfer of files among P2P clients. The second contribution deals with a new P2P client architecture. The P2P client issues a request of Read/Write of data to the client/coordinator in DG-based DCI.

As per the analysis described in section 6.2.4.4. of chapter 6, the proposed architecture performs better in comparison to the existing Attic DG-based architecture. In conclusion, this research has considerably attained its aim and objectives by demonstrating how the P2P-based architecture improves the performance of handling large volumes of data in DG-based DCI. This was demonstrated by performing the experiments on the designed testbed. The comparative analysis between the existing and proposed architecture demonstrates that the proposed P2P-based architecture is good for handling large volumes of data as well as for maintaining data consistency due to R/W of data in emerging applications. As a result, the novelty points for this research are new P2P-based architecture, new algorithms for dealing R/W of data in order to maintain data consistency, and efficient algorithms for data replication in DG-based DCI. The strengths of this research are a new architecture which is able to handle large volumes of data, maintain data consistency and replication, and improvement in performance in terms of execution time. The limitations of the proposed architecture are computation and message overhead.

It imposes an extra layer of complexity but the strengths of this research overweigh the mentioned limitations for handling large volumes of data.

7.3. Future work

The areas identified as future work for the research work are listed as follows:

1. Automatic Replication: At the current state, the replication strategy uses a multi path spanning tree algorithm for replication. Data mining techniques can be applied to improve the replication performance. This technique could be based on machine learning algorithms that recognise complex patterns and make smart decision based on these patterns to improve the performance.
2. Additional features in P2P-based architecture: Presently, the replication is based on historical data, the model can be expanded to support for real time data replication. It may further improve the replication performance.
3. Support for database metrics: The database metrics like total transactions performed per sec may be added to measure the measurement of concurrent writes in DG-based DCI.
4. Multiple coordinators: Present architecture consists of one P2P coordinator in DG-based DCI. This may be further expanded by having multiple coordinators in DG-based DCI.
5. Performance measurement: Presently, the performance is measured in terms of total execution time and number of messages exchanged. The performance metrics can be extended by adding other measures like memory utilisation in client and coordinator, average number of message processed to maintain data consistency by each client.

Appendix

The format of the sample deployment and platform file used for the experimentation purpose is mentioned below:

Deployment.xml

The sample file deployment file used for the experimentation for 50 clients is mentioned below:

```
<?xml version='1.0'?>

<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">

<platform version="3">

  <!-- Amount of tasks to dispatch, Computation size of each task, Communication size of each one, Amount of np2p.Clients waiting for orders, Data item present if 1 else 0,No. of data items, data pieces modified-->

  <process host="n-0" function="np2p.Coordinator"><argument value="50"/><argument value="5000000000000"/><argument value="10490378"/><argument value="50"/><argument value="1"/><argument value="1000"/><argument value="1000"/></process>

  <process host="n-1" function="np2p.Client"><argument value="0" /><argument value="0" /> </process> <!-- 1st argument shows the number of client, 2nd argument that client will modify the data items-->

  <process host="n-2" function="np2p.Client"><argument value="1" /><argument value="0" /></process>

  <process host="n-3" function="np2p.Client"><argument value="2" /><argument value="0" /></process>

  <process host="n-4" function="np2p.Client"><argument value="3" /><argument value="0" /></process>

  <process host="n-5" function="np2p.Client"><argument value="4" /><argument value="0" /></process>

  <process host="n-6" function="np2p.Client"><argument value="5" /><argument value="0" /></process>

  <process host="n-7" function="np2p.Client"><argument value="6" /><argument value="0" /></process>

  <process host="n-8" function="np2p.Client"><argument value="7" /><argument value="0" /></process>

  <process host="n-9" function="np2p.Client"><argument value="8" /><argument value="0" /></process>
```

<process host="n-10" function="np2p.Client"><argument value="9" /><argument value="0" /></process>

<process host="n-11" function="np2p.Client"><argument value="10" /><argument value="0" /></process>

<process host="n-12" function="np2p.Client"><argument value="11" /><argument value="0" /></process>

<process host="n-13" function="np2p.Client"><argument value="12" /><argument value="0" /></process>

<process host="n-14" function="np2p.Client"><argument value="13" /><argument value="0" /></process>

<process host="n-15" function="np2p.Client"><argument value="14" /><argument value="0" /></process>

<process host="n-16" function="np2p.Client"><argument value="15" /><argument value="0" /></process>

<process host="n-17" function="np2p.Client"><argument value="16" /><argument value="0" /></process>

<process host="n-18" function="np2p.Client"><argument value="17" /><argument value="0" /></process>

<process host="n-19" function="np2p.Client"><argument value="18" /><argument value="0" /></process>

<process host="n-20" function="np2p.Client"><argument value="19" /><argument value="0" /></process>

<process host="n-21" function="np2p.Client"><argument value="20" /><argument value="0" /></process>

<process host="n-22" function="np2p.Client"><argument value="21" /><argument value="0" /></process>

<process host="n-23" function="np2p.Client"><argument value="22" /><argument value="0" /></process>

<process host="n-24" function="np2p.Client"><argument value="23" /><argument value="0" /></process>

<process host="n-25" function="np2p.Client"><argument value="24" /><argument value="0" /></process>

<process host="n-26" function="np2p.Client"><argument value="25" /><argument value="0" /></process>

<process host="n-27" function="np2p.Client"><argument value="26" /><argument value="0" /></process>

<process host="n-28" function="np2p.Client"><argument value="27" /><argument value="0" /></process>

<process host="n-29" function="np2p.Client"><argument value="28" /><argument value="0" /></process>

<process host="n-30" function="np2p.Client"><argument value="29" /><argument value="0" /></process>

<process host="n-31" function="np2p.Client"><argument value="30" /><argument value="0" /></process>

<process host="n-32" function="np2p.Client"><argument value="31" /><argument value="0" /></process>

<process host="n-33" function="np2p.Client"><argument value="32" /><argument value="0" /></process>

<process host="n-34" function="np2p.Client"><argument value="33" /><argument value="0" /></process>

<process host="n-35" function="np2p.Client"><argument value="34" /><argument value="0" /></process>

<process host="n-36" function="np2p.Client"><argument value="35" /><argument value="0" /></process>

<process host="n-37" function="np2p.Client"><argument value="36" /><argument value="0" /></process>

<process host="n-38" function="np2p.Client"><argument value="37" /><argument value="0" /></process>

<process host="n-39" function="np2p.Client"><argument value="38" /><argument value="0" /></process>

<process host="n-40" function="np2p.Client"><argument value="39" /><argument value="0" /></process>

<process host="n-41" function="np2p.Client"><argument value="40" /><argument value="0" /></process>

<process host="n-42" function="np2p.Client"><argument value="41" /><argument value="0" /></process>

<process host="n-43" function="np2p.Client"><argument value="42" /><argument value="0" /></process>

<process host="n-44" function="np2p.Client"><argument value="43" /><argument value="0" /></process>

<process host="n-45" function="np2p.Client"><argument value="44" /><argument value="0" /></process>

<process host="n-46" function="np2p.Client"><argument value="45" /><argument value="0" /></process>

<process host="n-47" function="np2p.Client"><argument value="46" /><argument value="0" /></process>

<process host="n-48" function="np2p.Client"><argument value="47" /><argument value="0" /></process>

<process host="n-49" function="np2p.Client"><argument value="48" /><argument value="0" /></process>

<process host="n-50" function="np2p.Client"><argument value="49" /><argument value="1" /></process>

</platform>

Platform.xml

The sample file platform file used for the experimentation for 50 clients is mentioned below:

<?xml version='1.0'?>

<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">

<platform version="3">

<AS id="AS0" routing="Full">

<host id="n-0" power="5.0E9" />

<host id="n-1" power="2.5E9" />

<host id="n-2" power="2.5E9" />

<host id="n-3" power="2.5E9" />

<host id="n-4" power="2.5E9" />

<host id="n-5" power="2.5E9" />

<host id="n-6" power="2.5E9" />

<host id="n-7" power="2.5E9" />

<host id="n-8" power="2.5E9" />

<host id="n-9" power="2.5E9" />

<host id="n-10" power="2.5E9" />

<!— the details are continued till host number 50 -->

```
.....
  <host id="n-50" power="2.5E9" />

  <link id="2671" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2672" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2673" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2674" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2675" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2676" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2677" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2678" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2679" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2680" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2681" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2682" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2683" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2684" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2685" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2686" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2687" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2688" bandwidth="5.5E9" latency="3.535E-6" />
  <link id="2689" bandwidth="5.5E9" latency="3.535E-6" />
.....
<!— the details are continued for the rest of the link id -->
  .....
  <route src="n-0" dst="n-1" symmetrical="NO">
    <link_ctn id="2672"/>
    <link_ctn id="2722"/>
```

```
</route>

<route src="n-0" dst="n-2" symmetrical="NO">
  <link_ctn id="2673"/>
  <link_ctn id="2773"/>
</route>

<route src="n-0" dst="n-3" symmetrical="NO">
  <link_ctn id="2674"/>
  <link_ctn id="2824"/>
</route>

<route src="n-0" dst="n-4" symmetrical="NO">
  <link_ctn id="2675"/>
  <link_ctn id="2875"/>
</route>

<route src="n-0" dst="n-5" symmetrical="NO">
  <link_ctn id="2676"/>
  <link_ctn id="2926"/>
</route>

<route src="n-0" dst="n-6" symmetrical="NO">
  <link_ctn id="2677"/>
  <link_ctn id="2977"/>
</route>
<!— the details are continued for rest of the links -->
.....
```

<route src="n-0" dst="n-50" symmetrical="NO">

<link_ctn id="2721"/>

<link_ctn id="5221"/>

</route>

<route src="n-1" dst="n-0" symmetrical="NO">

<link_ctn id="2672"/>

<link_ctn id="2722"/>

</route>

.....

<!— the details are continued for rest of the links -->

.....

<route src="n-50" dst="n-45" symmetrical="NO">

<link_ctn id="5016"/>

<link_ctn id="5266"/>

</route>

<route src="n-50" dst="n-46" symmetrical="NO">

<link_ctn id="5067"/>

<link_ctn id="5267"/>

</route>

<route src="n-50" dst="n-47" symmetrical="NO">

<link_ctn id="5118"/>

<link_ctn id="5268"/>

</route>

<route src="n-50" dst="n-48" symmetrical="NO">

<link_ctn id="5169"/>

```
    <link_ctn id="5269"/>
  </route>

  <route src="n-50" dst="n-49" symmetrical="NO">
    <link_ctn id="5220"/>
    <link_ctn id="5270"/>
  </route>
 </AS>
</platform>
```

Bibliography

[1] http://www.oversim.org/wiki (accessed on June, 2012)

[2] http://peersim.sourceforge.net/ (accessed on June, 2012)

[3] http://simgrid.gforge.inria.fr/index.php (accessed on June, 2012)

[4] http://sourceforge.net/projects/optorsim/ (accessed on June, 2012)

[5] Mustafee N, Taylor S.J.E., "Speeding up simulation applications using WinGrid", Concurrency and Computation: Practice and Experience, Vol. 21 No. 11, pp. 1504-23 (Special Issue on Distributed Simulation, Virtual Environments and Real Time Applications), 2009

[6] Computing with BOINC, http://boinc.berkeley.edu/trac/wiki/ProjectMain (2007).

[7] Thain D., Livny M., Building reliable clients and services, In: Foster, I., Kesselman, C. (eds.) The Grid: Blueprint for a New Computing Infrastructure, pp. 285–318. Morgan Kaufman, San Francisco (2004).

[8] Cirne, W., Labs of the world Unite, Journal of Grid Computing 4(3), pp. 225–246 (2006).

[9] Fedak, G., et al., XtremWeb: a generic global computing platform, Proceedings of 1st IEEE International Symposium on Cluster Computing and the Grid CCGRID'2001, Special Session Global Computing on Personal Devices, pp. 582–587. IEEE, Piscataway (2001).

[10] DC-API Manual, http://www.desktopgrid.hu/ (2010).

[11] Gilles Fedak, Haiwu He, Franck Cappello, BitDew: A data management and distribution service with multi-protocol file transfer and metadata abstraction, Journal of Network and Computer Applications 32 (2009), pp. 961–975.

[12] Amazon Simple Storage Service: Getting Started Guide, API Version 2006-03-01, http://s3.amazonaws.com/awsdocs/S3/latest/s3-gsg.pdf (accessed on 2011).

[13] A survey of dynamic replication strategies for improving data availability in data grids, Future Generation Computer Systems, 28, 2012, pp. 337–349.

[14] Leyli Mohammad Khanli, AyazIsazadeh, Tahmuras N. Shishavan, PHFS: A dynamic replication method, to decrease access latency in the multi-tier data grid, Future Generation Computer Systems, 27, 2011, pp. 233–244.

[15] The University of Westminster Local Desktop Grid, http://wgrass.wmin.ac.uk/index.php/Desktop_Grid:Westminster_Local_DG (2010).

[16] Attics Overview, http://www.atticfs.org/ (accessed on 2011).

[17] Peter Kacsuk, Jozsef Kovacs, Zoltan Farkas, Attile Csaba Marosi, Gabor Gombas, Zoltan Balaton, SZTAKI Desktop Grid (SZDG): A flexible and scalable Desktop Grid System, J of Grid Computing , 2009, pp. 439-461

[18] A Roadmap for Desktop Grids for eScience, July 2012, http://desktopgridfederation.org/road-map (accesses on 28/9/2013)

[19] Jiawei Han, Micheline Kamber, Jian Pei, Data Mining Concepts and Techniques, Third Edition, Morgan Kaufmann Publishing, 2012.

[20] K. Ranganathan, A. Iamnitchi, I. Foster, Improving data availability through dynamic model-driven replication in large peer-to-peer communities, CCGrid, 2002, pp. 376.

[21] Abdullah, M. Othman, H. Ibrahim, M.N. Sulaiman, A.T. Othman, Decentralized replication strategies for P2P based scientific data grid, International Symposium on Information Technology, ITSim 2008, Vol. 3, pp. 1–8.

[22] V. Ramasubramanian, EG Sirer, Beehive: exploiting power law query distributions for O(1) lookup performance in peer to peer overlays, in Proceedings of the 1st USENIX symposium on Networked Systems Design and Implementation(NSDI), 2004, pp. 331-342.

[23] Mema Roussopoulos, Mary Baker, CUP: Controlled Update Propagation in Peer-to-Peer Networks, in proceedings of the USENIX annual Technical Conference, 2003.

[24] Y. Chen, R.H. Katz, J. Kubaiatowicz, Dynamic replica placement for scalable content delivery, in proceeding of the 1st international workshop on P2P systems(IPTPS), 2002, pp. 306-318.

[25] Anwitaman Datta, Manfred Hauswirth, Karl Aberer, Updates in Highly Unreliable, Replicated P2P systems, 23rd international conference on distributed computing systems, 2003.

[26] H. Lamehamedi, Z. Shentu, B. Szymanski, E. Deelman, Simulation of dynamic data replication strategies in data grids, 2003.

[27] Yi Hu, Min Feng, Laxmi N. Bhuyan, A Balanced Consistency Maintenance Protocol for Structured P2P Systems, Proceedings of the 29th conference on Information communications, INFOCOM'10, pp. 286-290.

[28] Xin Chen, Shansi Ren, Haining Wang, Xiaodong Zhang, SCOPE: Scalable Consistency Maintenance in Structured P2P Systems, IEEE INFOCOM 2005 (2005), pp. 1502-1513.

[29] Haiying Shen, CORP: A Cooperative File Replication Protocol for Structured P2P Networks, International Conference on High Performance Computing (HiPC), 2009.

[30] Liangzhong Yin and Guohong Cao, DUP: Dynamic-tree Based Update Propagation in Peer-to-Peer Networks, Proceedings of the 21st International Conference on Data Engineering, ICDE 2005.

[31] Haiying Helen Shen, IRM: Integrated File Replication and Consistency Maintenance in P2P Systems, IEEE Transactions on Parallel and Distributed Systems, Vol. 21 No. 1, January 2010.

[32] R.M. Rahman, K. Barker, R. Alhajj, Replica placement in data grid: Considering utility and risk, 2005.

[33] Sudharshan S., Vazhkudai, Xiaosong Ma, Vincent W. Freeh, Jonathan W. Strickland, Nandan Tammineedi, and Stephen L. Scott. FreeLoader: Scavenging Desktop Storage Resources for Bulk, Transient Data. In Proceedings of Supercomputing, 2005.

[34] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In Proceedings of the 5th Symposium on Operating Systems Design and Implementation, 2002.

[35] John R. Douceur, Roger P. Wattenhofer, Optimizing File Availability in a Secure Serverless Distributed File System. In Reliable Distributed Systems, 2001. Proceedings. 20th IEEE Symposium on, pages 4 –13, 2001.

[36] Samer Al-Kiswany, MateiRipeanu, Sudharshan S. Vazhkudai, and Abdullah Gharaibeh, stdchk: A Checkpoint Storage System for Desktop Grid Computing. Distributed Computing Systems, International Conference on, 0:613–624, 2008.

[37] B. Nicolae, G. Antoniu, and L. Bouge, Blobseer: Efficient data management for data-intensive applications distributed at large-scale, in Parallel Distributed Processing, Workshops and PhD Forum (IPDPSW), 2010 IEEE International Symposium on, April 2010, pp. 1 –4.

[38] HDFS Users Guide, http://hadoop.apache.org/docs/stable/hdfs_user_guide.html (accessed on 28/9/2013)

[39] Ian Robert Kelley, Data management in dynamic distributed computing environments, PhD Thesis, June 2012

[40] http://insdc.org (accessed on Sep, 2013)

[41] http://www.ebi.ac.uk/ena/about/about (accessed on Sep, 2013)

[42] http://www.ddbj.nig.ac.jp/intro-e.html (accessed on Sep, 2013)

[43] http://www.ncbi.nlm.nih.gov/genbank/ (accessed on Sep, 2013)

[44] http://www.nature.com/news/2009/091021/full/464670a.html (accessed on Sep, 2013)

[45] Atakan Dogan, A study on performance of dynamic file replication algorithms for real-time file access in Data Grids,Future Generation Computer Systems,25,2009, pp. 829-839

[46] A. Silberschatz, Henry F. Korth, S. Sudarshan, Database System Concepts, McGraw-Hill, Sixth Edition, 2010

[47] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Third Edition,2009

[48] Mustafee N, Taylor S.J.E., "Speeding up simulation applications using WinGrid", Concurrency and Computation: Practice and Experience, Vol. 21 No. 11, pp. 1504-23 (Special Issue on Distributed Simulation, Virtual Environments and Real Time Applications).

[49] Computing with BOINC, http://boinc.berkeley.edu/trac/wiki/ProjectMain (2007).

[50] Thain D., Livny M., Building reliable clients and services, In: Foster, I., Kesselman, C. (eds.) The Grid: Blueprint for a New Computing Infrastructure, pp. 285–318. Morgan Kaufman, San Francisco (2004).

[51] Cirne, W., Labs of the world Unite, Journal of Grid Computing 4(3), pp. 225–246 (2006).

[52] Fedak, G., et al., XtremWeb: a generic global computing platform, Proceedings of 1st IEEE International Symposium on Cluster Computing and the Grid CCGRID'2001, Special Session Global Computing on Personal Devices, pp. 582–587. IEEE, Piscataway (2001).

[53] DC-API Manual, http://www.desktopgrid.hu/ (2010).

[54] Peter Mell, Tim Grance, The NIST Definition of Cloud Computing, http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc (2009).

[55] Daniel J. Abadi, "Data Management in the Cloud: Limitations and Opportunities", IEEE Data Engineering Bulletin, Volume 32, Number 1, March 2009.

[56] S. Androutsellis Theotokis, D. Spinellis, A Survey of Content Distribution Technologies, ACM Computing Surveys, 36(4), December 2004.

[57] QuangHieu Vu, Mihai Lupu, Beng Chin Ooi, Peer-to-Peer Computing: Principles and Applications, Springer, 2010, pp. 12-24.

[58] Napster, http://www.napster.com (accessed in 2012)

[59] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 149-160.

[60] Antony Rowstron, Peter Druschel, Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems, Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001). Heidelberg, Germany, 2001.

[61] Langley, A., Freenet: Peer-to-Peer: Harnessing the power of disruptive technologies, O'Reilly & Associates Inc., 2001, pp. 123–132.

[62] Foster, I. and Kesselman, C., The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann, 1998, pp. 15-52.

[63] Fran Berman, Geoffrey Fox, Tony Hey, Grid Computing – Making the Global Infrastructure a Reality, John Wiley & Sons, 2003.

[64] OGSA-DAI 4.1 Documentation, http://www.ogsadai.org.uk/about/index.php (2011).

[65] GRelC DAS Documentation, http://grelc.unile.it/documentation.php (2008).

[66] Giovanni Aloisio, Massimo Cafaro, Sandro Fiore, Maria Mirto, The GRelC Library: A Basic Pillar in the Grid Relational Catalog Architecture, Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04), 2004.

[67] G. Aloisioa, M. Cafaroa, S. Fiorea and M. Mirtoa, The grid relational catalog project, Advances in Parallel Computing Volume 14, 2005, pp. 129-155.

[68] Sandro Fiore, Alessandro Negro, Salvatore Vadacca, Massimo Cafaro, Maria Mirto and Giovanni Aloisio, Advanced Grid Database Management with the GRelC Data Access Service, Parallel and Distributed Processing and Applications, Lecture Notes in Computer Science, 2007, Volume 4742/2007, pp. 683-694.

[69] Using the AMGA Metadata Catalog, http://egee-uig.web.cern.ch/egee-uig/production_pages/UsingAMGA.html (2008).

[70] G-DSE documents, http://www.grid.inaf.it/index.php?option=com_content&view=article&id=73&Itemid=91 (2007).

[71] Giuliano Taffoni, Edgardo Ambrosi, Claudio Vuerli, Fabio Pasian, The Grid Data Source Engine Batch Query System, Distributed and Parallel Systems 2008, pp. 189-194.

[72] Gilles Fedak, Haiwu He, Franck Cappello, BitDew: A data management and distribution service with multi-protocol file transfer and metadata abstraction, Journal of Network and Computer Applications 32 (2009), pp. 961–975.

[73] Amazon Simple Storage Service: Getting Started Guide, API Version 2006-03-01, http://s3.amazonaws.com/awsdocs/S3/latest/s3-gsg.pdf (2011).

[74] The Mobius Project at OSU BMI, http://projectmobius.osu.edu/ (2012).

[75] USER GUIDE FOR EDG SPITFIRE 2.1.8, http://edg-wp2.web.cern.ch/edg-wp2/spitfire/documentation.html (2003).

[76] OGF – Production Grid Infrastructure, Use Case Collection, Version 1.0, http://www.ogf.org/documents/GFD.180.pdf (2011) .

[77] Sim, Soshani, A Storage Resource Manager Interface Specification version 2.2, http://www.ogf.org/Public_Comment_Docs/Documents/2007-10/OGF-GSM-SRM-v2.2.pdf (2007).

[78] Rajasekar A., Storage resource broker—managing distributed data in a Grid, J. Computer Society, India, 33(4), 42–54 (2003).

[79] GT 5.0.2 User's Guide, http://www.globus.org/toolkit/docs/5.0/5.0.2/user/gtUserGuide.pdf (2010).

[80] Teragid about, https://www.teragrid.org/ (2011).

[81] NGS Overview, http://www.ngs.ac.uk/tools/overview (2011).

[82] Ramez Elmasri, Shamkant Navathe, Fundamentals of Database Systems, Pearson, 6th Edition, 2011, Chapter 21- 22.

[83] OGSA-DAI 4.0 Documentation, http://ogsa-dai.sourceforge.net/documentation/ogsadai4.0/ogsadai4.0-axis/DQPOverview.html (2010).

[84] Kukla Tamas, Integrating the OGSA-DAI to the P-GRADE portal, MSc thesis, Cranfield University, School of Engineering, 2007.

[85] Tamas Kiss, Tamas Kukla, Achieving Interoperation of Grid Data Resources via Workflow Level Integration, J Grid Computing (2009) 7:355–374.

[86] Kun Wang,Yuejian Xie, Sanli Li, Xiaoying Wang, Performance Analysis of the OGSA-DAI 3.0 software, ITNG '08 Proceedings of the Fifth International Conference on Information Technology, 2008.

[87] Helen X. Xiang, Experiences Running OGSA-DQP Queries against a Heterogeneous Distributed Scientific Database, 15th International Conference on Parallel and Distributed Systems, 8-11 Dec 2009.

[88] P-GRADE Portal Features and Releases, http://portal.p-grade.hu/ (2010).

[89] O.D. team. (2007, Aug.) Ogsa-daiwsi/wsrf 2.2 performance improvements. [Online]. Available: http://www.ogsadai.org.uk/documentation/scenarios/performance

[90] Etienne Urbah, Peter Kacsuk, Zoltan Farkas, Gilles Fedak, Gabor Kecskemeti, Oleg Lodygensky, Attila Marosi, Zoltan Balaton, Gabriel Caillat, Gabor Gombas, Adam Kornafeld, Jozsef Kovacs, Haiwu He, Robert Lovas, EDGeS: Bridging EGEE to BOINC and XtremWeb, Journal of Grid Computing (2009) , pp. 335–354.

[91] The University of Westminster Local Desktop Grid, http://wgrass.wmin.ac.uk/index.php/Desktop_Grid:Westminster_Local_DG (2010) .

[92] Gartner Says Cloud Computing is The Top Technology Trend in 2010, http://www.cloudave.com/1323/gartner-says-cloud-computing-is-the-top-technology-trend-in-2010 (2010).

[93] Cheat Sheet: G-Cloud, http://www.silicon.com/management/ceo-essentials/2010/02/10/cheat-sheet-g-cloud-39745449/ (2010).

[94] About Windows Azure, http://msdn.microsoft.com/en-us/library/dd179442.aspx (Nov, 2010).

[95] Google App Engine Documentation, http://code.google.com/appengine/downloads.html#Download_the_Google_App_Engine_Documentation (2011) .

[96] WebSphere CloudBurstAppliance , http://www-01.ibm.com/software/webservers/cloudburst/ (2010).

[97] Force.com Workbook, http://www.salesforce.com/us/developer/docs/workbook/workbook.pdf (2011).

[98] Amazon Elastic Compute Cloud: Getting Started Guide API Version 2011-02-28, http://aws.amazon.com/documentation/ec2/ (2011).

[99] GoGrid Getting Started Guide, http://wiki.gogrid.com/wiki/index.php/Getting_Started_Guide (2010).

[100] Manage and Secure Your IT Infrastructure with VMware vCloud, http://www.vmware.com/products/vcloud/overview.html (2011).

[101] Introducing Eucalyptus 2.0, http://open.eucalyptus.com/wiki/IntroducingEucalyptus_v2.0 (2011).

[102] About the OpenNebula.org Project, http://opennebula.org/about:about (2011).

[103] P. Kacsuk, A. Marosi, M. Kozlovszky, S. Ács and Z. Farkas, Parameter Sweep Job Submission to Clouds, Grids, Clouds and Virtualization Computer Communications and Networks, 2011, pp. 123-141.

[104] Leslie Lamport, Paxos made simple, 2001 http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf (accessed on 2012).

[105] Interoperation Between Desktop Grid and Cloud, Breaking down the interoperability between XtremWeb-HEP and CRANE, http://grid.hust.edu.cn/xhshi/projects/DG-Cloud.html (Accessed in 2012)

[106] Christopher J. Reynolds, Stephen Winter, Gabor Z. Terstyanszky, Tamas Kiss, Pamela Greenwell, Sandor Acs, Peter Kacsuk, Scientific Workflow Makespan

Reduction through Cloud Augmented Desktop Grids, IEEE Third International Conference on Cloud Computing Technology and Science, 2011

[107] http://edgi-project.eu/ (accessed on 2012)

[108] http://gow.epsrc.ac.uk/NGBOViewGrant.aspx?GrantRef=EP/I034254/1 (accessed on 2011)

[109] http:// boing.berkeley.edu/trac/wiki/DesktopGrid (accessed on 2014)

[110] Anderson, D.P., BOINC: A system for public-resource computing and storage, In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, GRID 2004, pp. 4-10.

[111] Raghu Ramakrishnan, Johannes Gehrke, Database Management Systems, Mcgraw-Hill, 3rd Edition, Addison-Wesley, 2006, Chapter 17.

[112] Henry F Korth, Abraham Silberschatz, S. Sudurshan, Database system concepts, 5th Edition, McGraw-Hill, 2006

[113] Thomas Connolly, Carolyn Begg, Database Systems-A Practical Approach to design, Implementation and Management, 3rd Edition, Addison-Wesley, 2002.

[114] Anderson, D.P.: BOINC: a system for public-resource computing and storage. In: Proceedings of the 5th IEEE/ACM International GRID Workshop. Pittsburgh, USA, 2004