

Geoprocessing Optimization in Grids

by

Shuo Liu

B.S., Nanjing University, 1995

M.S., Nanjing University, 1998

Submitted to the Graduate Faculty of
the School of Information Sciences in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

University of Pittsburgh

2005

UNIVERSITY OF PITTSBURGH
SCHOOL OF INFORMATION SCIENCES

This dissertation was presented

by

Shuo Liu

It was defended on

July 29, 2005

and approved by

Dr. Michael Lewis, Associate Professor, Information Science & Telecommunications

Dr. M. Talat Odman, Principal Research Engineer, Civil and Environmental Engineering,
Georgia Institute of Technology

Dr. Ralph Z. Roskies, Professor, Physics & Astronomy/Co-Scientific Director, Pittsburgh Super
Computing Center

Dr. Vladimir I. Zadorozhny, Assistant Professor, Information Science & Telecommunications

Dr. Hassan Karimi, Associate Professor, Information Science & Telecommunications
Dissertation Director

Copyright by Shuo Liu
2005

Geoprocessing Optimization in Grids

Shuo Liu, PhD

University of Pittsburgh, 2005

Geoprocessing is commonly used in solving problems across disciplines which feature geospatial data and/or phenomena. Geoprocessing requires specialized algorithms and more recently, due to large volumes of geospatial databases and complex geoprocessing operations, it has become data- and/or compute-intensive. The conventional approach, which is predominately based on centralized computing solutions, is unable to handle geoprocessing efficiently. To that end, there is a need for developing distributed geoprocessing solutions by taking advantage of existing and emerging advanced techniques and high-performance computing and communications resources. As an emerging new computing paradigm, grid computing offers a novel approach for integrating distributed computing resources and supporting collaboration across networks, making it suitable for geoprocessing. Although there have been research efforts applying grid computing in the geospatial domain, there is currently a void in the literature for a general geoprocessing optimization.

In this research, a new optimization technique for geoprocessing in grid systems, Geoprocessing Optimization in Grids (GOG), is designed and developed. The objective of GOG is to reduce overall response time with a reasonable cost. To meet this objective, GOG contains a set of algorithms, including a resource selection algorithm and a parallelism processing algorithm, to speed up query execution. GOG is validated by comparing its optimization time and estimated costs of generated execution plans with two existing optimization techniques. A proof of concept based on an application in air quality control is developed to demonstrate the advantages of GOG.

TABLE OF CONTENTS

PREFACE.....	xi
1. Introduction.....	1
1.1. A New Computing Paradigm for Geoprocessing	1
1.2. Distributed Computing for Geoprocessing	4
1.3. Grid Computing for Geoprocessing.....	7
1.4. Research Objectives and Contributions	8
1.5. Organization of the Dissertation	9
2. Background and Related Work.....	11
2.1. Geoprocessing and GIS.....	11
2.1.1. Geospatial Data.....	11
2.1.2. Geoprocessing.....	14
2.1.3. GIS Software and Spatial DBMS	18
GIS Software.....	18
Spatial DBMS	20
2.2. Grid Computing	23
2.2.1. Core Architecture and Middleware.....	24
Core Architecture.....	24
Middleware	26
2.2.2. Web Services and Service-Oriented Architecture	30
Web Services	30
Service-Oriented Architecture and Technology Stack	31
2.2.3. Open Grid Service Architecture.....	33
Grid Services.....	33
Open Grid Service Architecture Platform.....	36
2.3. Query Optimization in Distributed Databases	37
2.3.1. Dynamic Programming.....	38
2.3.2. Randomized Algorithms	41
2.3.3. Dynamic Data Allocation	42
Dynamic Replication	42
Cache Investment.....	44
2.3.4. Economic Model.....	46
2.4. Optimization Techniques in Grids	48
2.4.1. AppLeS and Condor	48
2.4.2. Geospatial Applications in Grids	53
3. Geoprocessing Optimization in Grids.....	55
3.1. Challenges.....	55
3.2. Assumptions in GOG.....	56
3.3. Optimization Strategy	60
Resource Selection.....	60

Parallelism.....	61
3.4. Architecture of Geoprocessing Optimization in Grids	64
3.5. Auxiliary Services.....	65
Environment Information Service (EIS).....	66
Database Information Service (DIS).....	66
Transmission Prediction Service (TPS).....	66
Geoprocessing Category Service (GCS).....	68
3.6. Resource Selection.....	68
3.7. Parallelism Processing	75
Intermediate Results.....	76
Detect Generic Parallelism	77
Detect Domain-Specific Parallelism.....	80
4. Experiment on Geoprocessing Optimization in Grids	82
4.1. Stages in Query Processing.....	82
4.2. Comparing Optimization Techniques	83
Complete Iteration Optimizer	83
Randomized Optimizer	85
4.3. Experiment Hypotheses	85
4.4. Experiment Design.....	87
4.4.1. Simulated Grid Environment	87
4.4.2. Query Testing.....	89
4.4.3. Experiment Scenario	90
4.5. Experiment Results and Analysis	91
4.5.1. QOT Analysis	91
4.5.2. QET Analysis.....	92
4.5.3. QPT Analysis	93
4.5.4. Analysis on Factors in the Ranking Function.....	96
4.6. Summary	98
5. Proof of Concept.....	99
5.1. Grid Testbed.....	99
5.2. Design of GCS	102
5.3. Design of the Proof of Concept	106
5.3.1. Air Emission Database.....	107
5.3.2. Query Testing.....	109
5.4. Experiment Results and Analysis	111
5.4.1. Q_WD Analysis	111
5.4.2. Q_CNT Analysis.....	113
5.4.3. Performance Comparison between WD and CNT.....	115
5.5. Summary	116
6. Conclusions and Future Research.....	117
6.1. Summary of the Research	117
6.2. Conclusions.....	118
6.3. Future Research	118
APPENDIX A.....	120
Experiment Results of Simulated Grid Environment	120
APPENDIX B	130

Query Optimization and Execution Times of Q_WD.....	130
Query Optimization and Execution Times of Q_CNT	130
APPENDIX C	131
Linear Regressions on WD Running Times in hosts GIS22 and GIS23	131
Linear Regressions on CNT Running Times in hosts GIS22 and GIS23	133
APPENDIX D.....	135
Entity-Relationship Diagram of Air Emission Database	135
APPENDIX E	137
Sample Execution Plans for Q_WD	137
BIBLIOGRAPHY.....	138

LIST OF TABLES

Table 2-1. Geoprocessing techniques and applications	14
Table 2-2. An SQL statement to create a trajectory table in Oracle Spatial	21
Table 2-3. A sample query in Oracle Spatial	22
Table 2-4. An SQL statement to create a trajectory table in Oracle Spatial	22
Table 2-5. A sample query in Oracle Spatial	23
Table 2-6. OGSA grid service interfaces (Foster et al. 2002)	35
Table 2-7. Sample ClassAds (Raman et al. 1998)	52
Table 4-1. Simulated databases and relations	87
Table 4-2. Simulated database replicas	88
Table 4-3. Simulated host configuration	88
Table 4-4. QPT gains in GOG	94
Table 5-1. Testbed configuration	100
Table 5-2. Summary of major relations in the air emission database	107

LIST OF FIGURES

Figure 1-1. Development gap between optical fiber, computer chips, and data storage (adopted from Stix 2001).....	2
Figure 1-2. Architecture of RPC (adopted from Birrell and Nelson 1984)	4
Figure 2-1. A view of vector data (Lo and Yeung 2002).....	12
Figure 2-2. A view of surface data represented in raster form (Lo and Yeung 2002).....	13
Figure 2-3. A TIN built on a DEM (Kreveld 1997).....	13
Figure 2-4. Reclassification and overlay in raster data (adopted from Lo and Yeung 2002).....	15
Figure 2-5. Point, line, and polygon buffers (adopted from Lo and Yeung 2002)	16
Figure 2-6. A shortest path between two locations in Pittsburgh, PA	17
Figure 2-7. Architecture of ArcInfo (Rigaux et al. 2002).....	19
Figure 2-8. Layered grid architecture (Foster et al. 2001).....	24
Figure 2-9. Major components of GRAM (Czajkowski et al. 1998).....	29
Figure 2-10. Overview of MDS architecture in Globus (Czajkowski et al. 2001)	30
Figure 2-11. Service-oriented architecture for Web services (Graham et al. 2002)	32
Figure 2-12. Technology stack diagram for Web services (Booth et al. 2003)	32
Figure 2-13. OGSA platform components and related profiles (Foster et al. 2003)	36
Figure 2-14. Dynamic programming algorithm (Kossmann and Stocker 2000)	39
Figure 2-15. A sample optimizing process of IDP (Kossmann and Stocker 2000).....	40
Figure 2-16. Moves defined for a search space of a bushy processing tree (Steinbrunn et al. 1997)	42
Figure 2-17. ADR diagram, adopted from Wolfson et al. (1997).....	44
Figure 2-18. Architecture of Mariposa (Stonebraker et al. 1996).....	48
Figure 2-19. Organization of an AppLeS agent (Berman and Wolski 1997)	50
Figure 2-20. Condor kernel (Thain et al. 2003).....	51
Figure 3-1. Orders of relations in AQT	59
Figure 3-2. The proposed strategy to optimize computation in grids.....	60
Figure 3-3. Classification of parallelism in grids.....	62
Figure 3-4. Sample parallelism in grids.....	63
Figure 3-5. Architecture of GOG.....	65
Figure 3-6. Workflow in the resource selection module.....	74
Figure 3-7. Workflow in the parallelism processing module	75
Figure 3-8. Structure of execution plan optimized by GOG.....	78
Figure 3-9. Algorithm to detect generic parallelism.....	79
Figure 3-10. Algorithm to detect domain-specific parallelism.....	81
Figure 4-1. Stages in query processing.....	82
Figure 4-2. Queries represented as AQT	90
Figure 4-3. Average QOT of GOG, CIO, and RO.....	91
Figure 4-4. Average QET of GOG, CIO, and RO	92

Figure 4-5. Average QPT of CIO, GOG, and RO	94
Figure 4-6. Average QPT of GOG and three variants	96
Figure 5-1. OGSA-DAI configuration in the testbed.....	101
Figure 5-2. Design of GCS	104
Figure 5-3. Pollution sources and wind trajectory buffers.....	108
Figure 5-4. Query Q_WD	110
Figure 5-5. Query Q_CNT	110
Figure 5-6. Query optimization times of Q_WD	112
Figure 5-7. Query execution times of Q_WD.....	112
Figure 5-8. QPT of Q_WD	113
Figure 5-9. Query optimization times of Q_CNT.....	114
Figure 5-10. Query execution times of Q_CNT	114
Figure 5-11. QPT of Q_CNT	115
Figure 5-12. Performances of WD and CNT operations	116

PREFACE

First of all, I would like to express my great appreciation to Dr. Hassan Karimi for his continuous direction and encouragement. I also wish to thank the members of my dissertation committee, Drs. Michael Lewis, M. Talat Odman, Ralph Z. Roskies, and Vladimir I. Zadorozhny, who have shared their research resources and insightful comments.

I want to express my gratitude to the Department of Information Science and Telecommunications for the opportunity and financial support.

I thank the following colleagues and friends for their help and invaluable discussions: Chi Wu, Mark Holliman, Qiang Ye, and Christopher Jon Jursa.

This thesis would not have been possible without the support and understanding of my family. I thank my parents, Gaokui Liu and Xianyu Pan, for their love and the inspiration to pursue a Ph.D. I am very grateful to my uncle, Dr. Chunshi Chang, for his encouragement and invaluable knowledge. Last, but not the least, I would like to thank my wife, Xuan Yang, for the support and love that made this work possible.

1. Introduction

Geospatial Information System (GIS) technology has played an important role in solving problems across disciplines, from environmental science to civil engineering to wireless communications. As GIS technology becomes a key tool in information integration and decision making, the need for efficient geoprocessing¹ increases. Today's geoprocessing, due to large volumes of data (terabytes) and complex operations (e.g., simulations), requires high-computing resources (CPU, storage) for efficient implementations. Currently most geoprocessing operations are performed on centralized computing platforms. However, the emergence of new distributed computing solutions is expected to change this trend.

In the sections below, an overview of computing paradigms used in geoprocessing and motivations for taking a grid computing approach to geoprocessing is presented. The research objectives and contributions are discussed next, followed by the outline of the dissertation.

1.1. A New Computing Paradigm for Geoprocessing

Nowadays geoprocessing is usually conducted in a centralized fashion. First all required data are collected into a central storage. Next, a series of operations are launched to process the data and obtain the final result. This approach was sound and preferred when network connections were expensive and slow and data were often gathered in one place. However, recently the situation, as the result of unprecedented evolution in computing technology, has changed. One of the

¹ In this dissertation, geoprocessing is defined as any computation on geospatial data.

noticeable achievements is high-speed networks. On average, the network speed is doubled almost every 9 months (Foster 2002). In 1985, the NSFnet backbone among the National Science Foundation (NSF) supercomputing centers operated at 56 Kb/s. Today, the TeraGrid network (TeraGrid 2004) operates at 40 Gb/s in connecting its affiliating nodes and it is predicted (McRobbie et al. 2001) that the Global Network Access Points (GNAP) in the Global Terabit Research Network (GTRN) will be linked via terabit connection by 2006. Meanwhile, microprocessor performance is doubled roughly every 18 months (Foster 2002), leading to an increasing gap (Figure 1-1) between network capacity and microprocessor speed. The gap makes computing over networks more attractive than computing on a single site. It is being realized that high-computing powers can be obtained by integrating machines via networks, as demonstrated in SETI@Home (SETI@Home 2004) and other distributed computing projects.

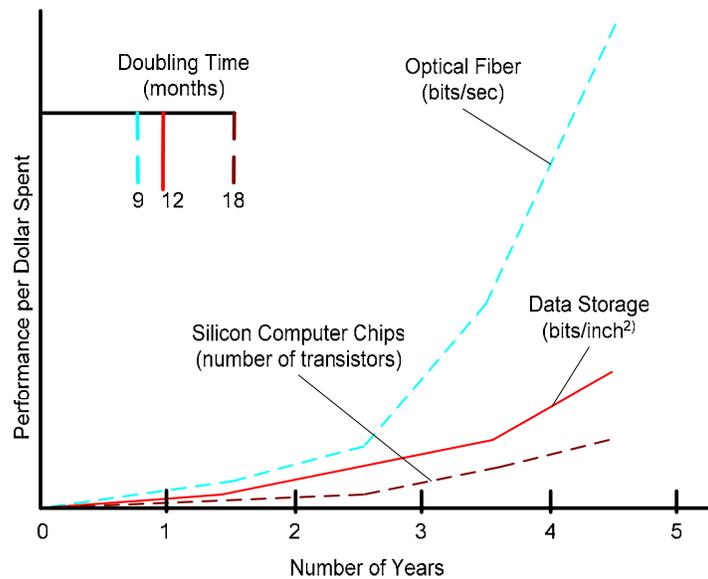


Figure 1-1. Development gap between optical fiber, computer chips, and data storage (adopted from Stix 2001)

Not only advances in computing technology are paving the way for development of new approaches to geoprocessing, challenging requirements from contemporary applications also demand such approaches. First, it is observed that geospatial data volumes have soared during the last decade. For instance, Terra, a spacecraft of the National Aeronautics and Space Administration's (NASA's) Earth Observing System, generates 194 GB data per day and Landsat 7, a satellite launched and operated by NASA, generates 150 GB data per day (Muntz et al. 2003). Crockett (1998) estimates that a single 1-meter resolution satellite image of land area on the Earth with only RGB channels will contain in the excess of 1 peta bytes (10^{15}) data. With the expected daily revisits of satellites in the near future (EROS 2003), the size of this data set will soon be doubled. Besides, there are other devices that also generate large volumes of geospatial data, such as sensor networks and embedded devices. These data sources create a huge amount of information whose analysis is beyond the capability of centralized computing environments. Second, in centralized computing all required data have to be available in a central node before starting a computing job. With a large volume of data being distributed in different sites, however, the cost of transferring a whole data set can be expensive, not to mention that some institutions only allow local access to their data. Lastly, distributed computing is more flexible to accommodate dynamic requests for geoprocessing. To an organization that needs to carry earth quake simulation once per year, it may not be worth to maintain a large computing facility just for a short time usage. Centralized systems offer little flexibility to accommodate this new challenge while distributed computing is potential for building computing-on-demand systems that can allocate computing resources dynamically based on requests.

1.2. Distributed Computing for Geoprocessing

Various distributed systems are built to integrate computing resources using different techniques. Most distributed computing techniques are based on Remote Procedure Call (RPC), a mechanism proposed by Birrell and Nelson (1984) that enables local processes to call remote procedures and hides communications from calling processes (Figure 1-2).

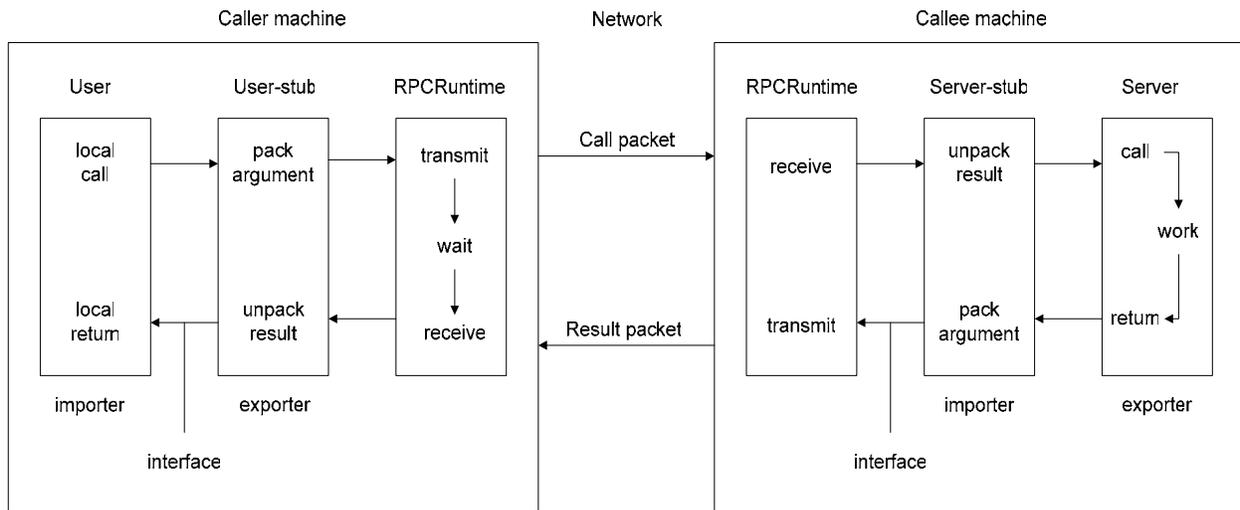


Figure 1-2. Architecture of RPC (adopted from Birrell and Nelson 1984)

Built upon RPC, distributed file systems, such as Network File System (NFS) (Callaghan 2000; Pawlowski et al. 1994; Sandberg et al. 1985) and Andrew File System (AFS) (Kistler 1996; Kistler and Satyanarayanan 1992; Satyanarayanan 1990), and distributed object-based systems, such as CORBA (Common Object Request Broker Architecture), from OMG (Object Management Group), and DCOM (Distributed Component Object Model), from Microsoft, provide users with high-level abstractions and tools for application development. For instance, NFS builds an interface called Virtual File System (VFS) to provide transparent file access in a

system. In CORBA, proxies and skeletons are created at client and server sides, respectively, to support operations on remote objects.

Despite the benefits current distributed computing techniques offer, they have weaknesses and limitations. For instance, object-based distributed systems offer a convenient approach to model business logic and hide implementation details but lack the support for massive data processing (Haynos 2004). In CORBA, object references are location dependent and become invalid as objects move; this is suitable for local-area distributed systems but not for large-scale systems (Tanenbaum and Steen 2002). There are also interoperability issues by different CORBA vendors (Haynos 2004). In general, CORBA is best suited for highly reliable, pre-compiled, tightly-coupled systems rather than dynamic, Internet-based systems (Haynos 2004). DCOM's availability only to Windows limits its application in heterogeneous environments. On the other hand, distributed file systems provide transparent access to data dispersed over multiple sites, making them more favorable in projects where data integration is the primary concern. The major limitation of distributed file systems is their primitive way to build applications: developers often have to work with low-level RPC and deal with intricate implementation issues and heterogeneous resources, which often distracts them from business logic and delays project development.

Grid computing has emerged as a new approach to overcome the limitations of existing techniques. As stated by Foster et al. (2001), grid computing is designed to build computing infrastructures that support controlled resource sharing and collaboration across multiple organizations. Many of its major advantages come from its adaptation of service-oriented design.

In a service-oriented architecture, entities are represented in the form of services that can be identified and invoked through message exchange. Messages define service interfaces and hide implementation issues from service requestors. By adopting service-oriented design, grid computing achieves transparency in integrating heterogeneous resources and avoids making implementation-related assumptions such as object orientation. In addition, support of late binding and service virtualization in service-oriented design gives rise to more flexibility in grid computing as well. Unlike distributed systems that utilize static binding, service requests can be bound to service instances at run time so that changes in service instances/implementations are transparent to service requestors. Late binding also makes self-assembling and self-healing possible. Service virtualization means that services can be composed as integrations of other services which in turn could also be virtualized. Combining service virtualization together with late binding, application developers can dynamically assemble or dismantle services according to changes in user requirements, which makes computing-on-demand possible.

More specifically, with inherited strengths from precedent distributed computing techniques, grid computing offers desired features for contemporary geoprocessing. It builds resource-sharing systems using services as basic operating units that provide interoperability and transparency among various geospatial operations. For GIS applications with dynamic demands, late-binding and service virtualization help grid computing shift loads to sites that can handle them more efficiently. Furthermore, built-in support for large data processing in grids makes it more preferable for data-intensive geoprocessing to other distributed computing techniques. In summary, with all these attractive characteristics, grid computing is becoming a promising approach to support geoprocessing in distributed, dynamic, and heterogeneous environments.

1.3. Grid Computing for Geoprocessing

Grid computing has attracted the attention and momentum from academia and industry after it emerged in the end of last century. By the dawn of the new century grid computing had experienced three major stages in its evolution (Roure et al. 2003). The first stage is usually known as “metacomputing”, a term invented by Catlett and Smarr (1992). Most metacomputing research conducted prior to the mid-1990s was focused on integrating computing resources for a set of compute- /data-intensive tasks. The second stage, late 1990s, saw a rapid development in middleware, which is deemed as a key to constructing a distributed, large-scale, computational infrastructure to support various compute- and data-intensive applications over heterogeneous networks (Foster and Kesselman 1998b). Example middlewares included Globus Toolkit (Foster and Kesselman 1998a; Foster et al. 2002) and Legion (Chapin et al. 1999; Grimshaw et al. 2002; Grimshaw and Wulf 1997). The third stage started from the new millennium when the research focus in grid computing was shifted to global collaboration via a service-oriented view. This trend can be seen in the work of Global Grid Forum (GGF 2004), an open source organization devoted to grid computing.

Despite advances in grid technology, much research effort currently is concentrated on building a working system with little attention to such issues as performance. As in many other computing areas, having functional grids is only the first step to creating global, grid-based information infrastructures. Given the current development of grid computing, there are some issues that should be investigated when applying grid technology to geoprocessing.

- Application performance. One of users' major concerns is how well their applications perform. As mentioned earlier, a promising potential of grids is performance gain by employing superior resources and parallel execution. There is, however, a cost that comes with this gain, i.e., overheads of conducting distributed computation. How to improve performance while reducing overheads is an open question.
- Optimization complexity. The difficulty of developing optimization techniques in grids is two folds. First, computing resources in grids not only are heterogeneous, but may also be replicated at different locations. Second, a grid system may consist of a large number of nodes across wide geographic areas. These features make grids distinct from conventional distributed systems and pose challenges to development of optimization techniques.
- Special techniques for geoprocessing. Considering large volumes of data and data- and compute-intensive operations involved in today's geoprocessing operations, grid computing is seen as a suitable computing environment for geoprocessing. However, most existing optimization techniques built for generic operations (e.g., relational operations) are unsuitable for geoprocessing since they do not recognize spatial operations. Thus there is a need for optimization techniques that meet the requirements of geoprocessing operations.

1.4. Research Objectives and Contributions

A novel technique, Geoprocessing Optimization in Grid (GOG), to optimize grid-based geoprocessing is designed and developed in this research. The objective of GOG is to reduce overall response time for geospatial applications with a reasonable cost. GOG takes into account

performance factors of grids for optimal execution plans. It is based on a modular, two-phase optimization strategy: in the first phase search space is limited by selecting candidate resources according to a ranking function and in the second phase parallel executions are detected and processed.

GOG features a parallelism processing module to handle spatial operations. This module analyzes requested geoprocessing operations and for each identified operation it will check for possible parallel executions and will suggest an optimal one. One advantage of having a dedicated module for parallel geoprocessing is that it could be modified for problems in other domains (e.g., computational biology) without affecting other modules. This strategy would make GOG applicable to other application domains.

This research yields the following contributions:

- A query optimization methodology in grid systems.
- A performance-based ranking function in grids.
- A new index to evaluate transmission capacity of a host for a given query.
- A module to detect and execute parallelism for geoprocessing.

1.5. Organization of the Dissertation

The remainder of the dissertation is organized as follows. Chapter 2 provides background information on grid computing and query optimization techniques as well as related work in geospatial domain. The proposed methodology is presented in Chapter 3, where the architecture of GOG and its components are discussed. In Chapter 4 the results of performance comparison

between GOG and other optimization techniques are reported and analyzed. Chapter 5 describes proof of concept where GOG is applied to an air quality control application. Chapter 6 concludes the dissertation and presents future research directions.

2. Background and Related Work

Optimizing geoprocessing in grid systems involves geoprocessing operations, grid computing approaches and techniques and optimization techniques. In this chapter a brief description of geoprocessing and GIS is provided as a background, followed by an overview of grid computing, including latest developments and grid services as the result of merging grid computing and Web services. Query optimization techniques for distributed databases are discussed next. In the end of this chapter, optimization techniques in grid systems, both generic and specific for geospatial applications, are presented.

2.1. Geoprocessing and GIS

A key component in geospatial applications is geoprocessing, i.e., a computation on geospatial data. GIS software packages support a number of geoprocessing operations. Geoprocessing can be either basic operations (e.g., buffering) or complex operations composed of several basic operations (e.g., floodplain modeling). For a better understanding of geoprocessing, different types of geospatial data and geoprocessing techniques are presented in this section. GIS software packages and spatial DBMS are also discussed.

2.1.1. Geospatial Data

Geospatial data can be categorized into two basic types: vector and raster. Vector data (Figure 2-1) present geospatial features as discrete points, lines, and polygons. Geographical objects that are discrete and with identifiable geospatial extent, such as buildings, highways and rivers, usually take this form. Raster data (Figure 2-2) depict features in a grid of cells with attribute or spectral values. Raster data are not suitable for representing individually identifiable objects but

often used in geospatial analysis tools. A common type of raster data is satellite image. In contemporary geospatial databases, vector data are mostly stored as lists of coordinates while raster data take the form of a grid of square or rectangular cells with some attribute values.

Some geographical objects or phenomena can be represented in either raster or vector form. For instance, terrain elevations can be presented in vector form as Triangulated Irregular Network (TIN) or in raster form as Digital Elevation Models (DEM) (Figure 2-3). In a DEM, an area is divided into a grid of rectangles of the same size and an elevation is sampled at the center of each rectangle. On the other hand, elevations in a TIN are sampled using irregular triangles throughout a given space to form an approximation of the sampled surface.

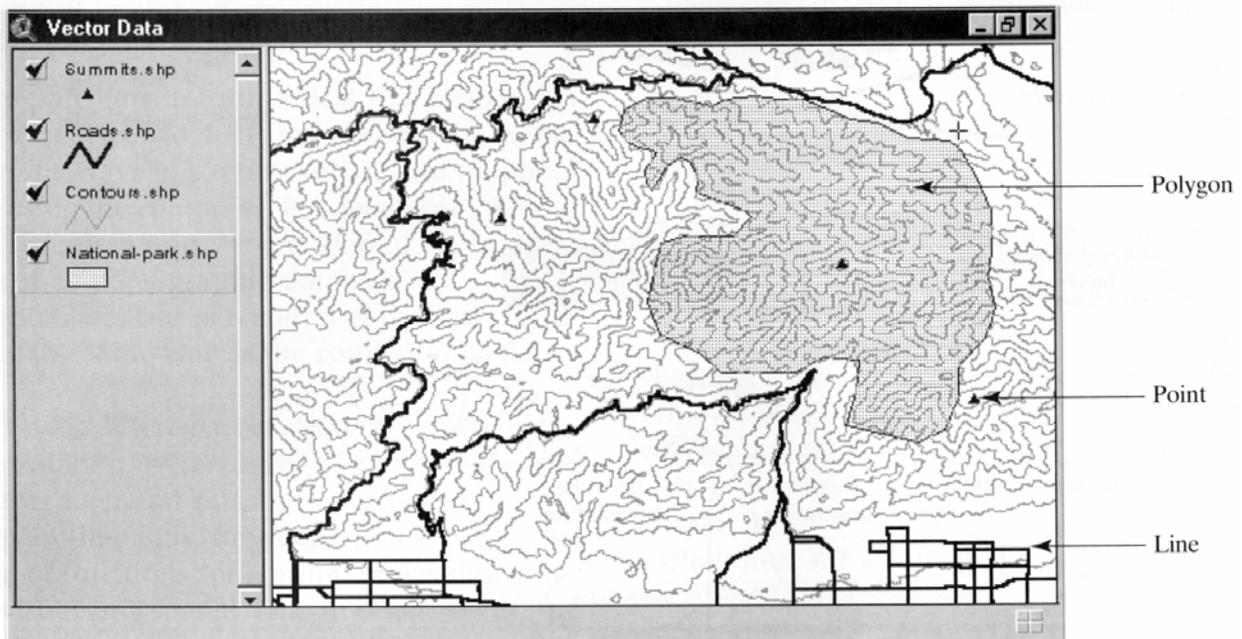


Figure 2-1. A view of vector data (Lo and Yeung 2002)

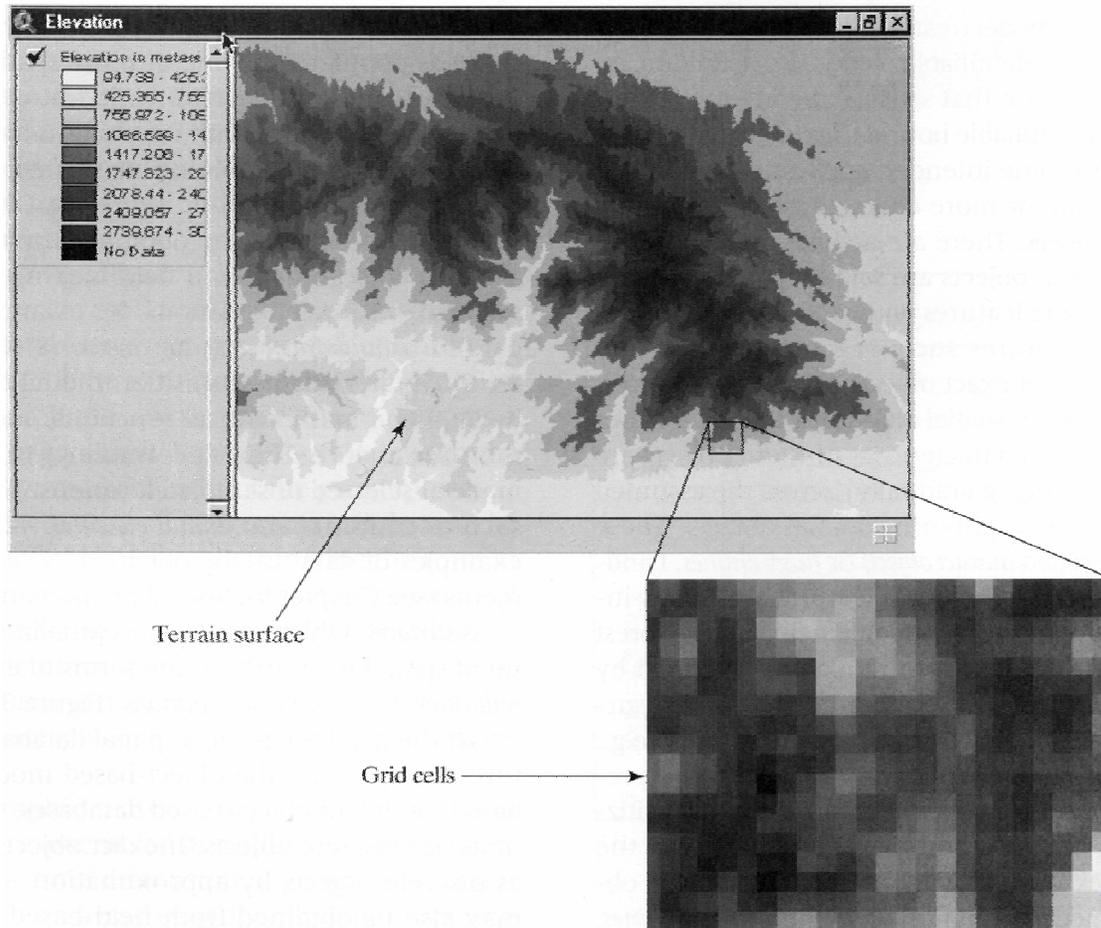


Figure 2-2. A view of surface data represented in raster form (Lo and Yeung 2002)

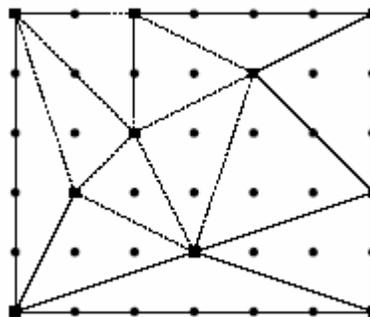


Figure 2-3. A TIN built on a DEM (Kreveld 1997)

Vector and raster data are suitable for different applications. For instance, due to its explicit representation of topology, vector form is preferred for network analysis (Burrough and McDonnell 1998). On the other hand, raster data are simpler for certain computations since all entities have the same regular shape. But the large storage requirement of raster data and low spatial resolution in large grid cells are disadvantages in certain applications.

2.1.2. Geoprocessing

There are two types of geoprocessing, raster-based and vector-based, that are applied to raster and vector data, respectively. Some common geoprocessing techniques and applications are listed in Table 2-1.

Table 2-1. Geoprocessing techniques and applications

Geoprocessing Technique		Sample Application
Raster-based	Filtering	Image quality enhancement
	Reclassification	Terrain analysis
	Overlay	Change detection
	Aggregation	Environmental modeling at regional or global levels
Vector-based	Buffering	Natural resource management
	Geocoding	Address lookup
	Attribute database query	Urban management
	Network analysis	Transportation

Many raster-based geoprocessing techniques are for digital image processing. Filtering, for instance, is a technique for image enhancement (Jensen 1996). In filtering, an $m*n$ filtering window (kernel) is formed and passes all cells in an image. The brightness of each cell is multiplied by the values in the filtering window and the average of multiplications is set as the

brightness of the cell in the center of the filtering window. Filtering can be used in analysis of linear features such as roads (Karimi and Liu 2004). Other raster-based geoprocessing includes, but not limited to, reclassification and overlay. Reclassification creates a new layer by applying either logical or arithmetic operators to the value in each cell of an input layer. Overlay also uses logical and arithmetic operators in integrating two or more layers. Figure 2-4 shows an example of reclassification and overlay in two layers. Another common raster geoprocessing is aggregation. In aggregation, raster data collected from small regions are down-sampled to form a large layer with fewer cells.

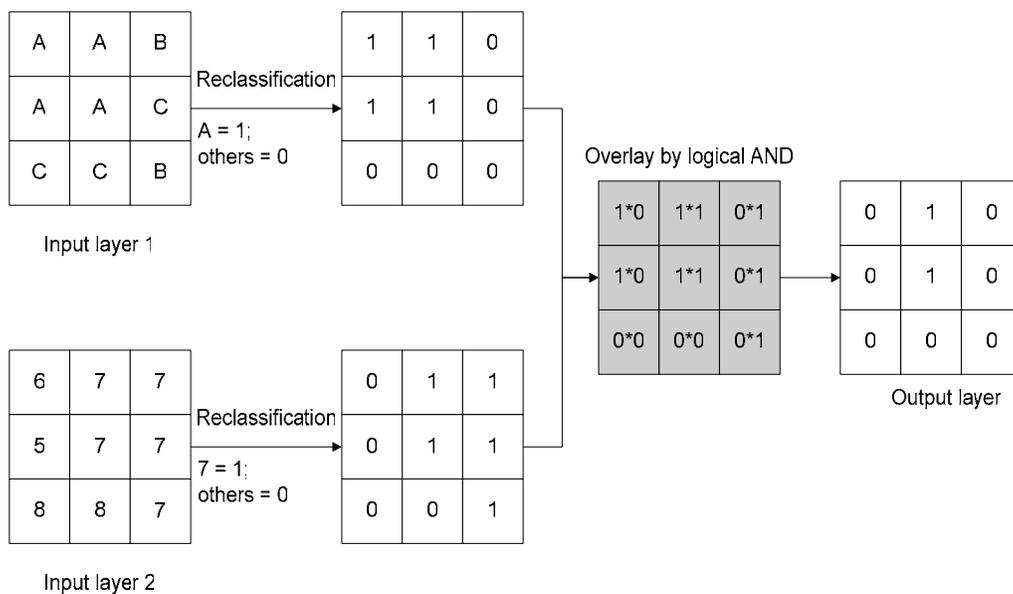


Figure 2-4. Reclassification and overlay in raster data (adopted from Lo and Yeung 2002)

Vector-based geoprocessing also plays an important role in geospatial applications. Due to the efficiency in storage of vector data and accuracy in computation, vector-based geoprocessing has been widely applied in many GIS applications. One frequently used vector-based geoprocessing is buffering. A buffer is an area surrounding a geospatial object with a specific distance. Samples

buffers around points, lines and polygons are shown in Figure 2-5. Buffering is often conducted to identify the area of interest, followed by other analytical tools. Geocoding is another useful geoprocessing (Karimi et al. 2004) that computes coordinates for a given address such as “135 N. Bellefield Ave., Pittsburgh, PA 15213”. Attribute queries in the form of Structured Query Language (SQL) have become an important geoprocessing as more and more attribute data are stored in either internal tables (e.g., INFO tables in ArcInfo) or external DBMS. SQL provides flexibility to applications to query and update geospatial data. For example, a query to find land parcels that are larger than 100 meter² in Pittsburgh can be expressed as:

```
SELECT parcel_id FROM land_parcel WHERE area>100 AND city="Pittsburgh"
```

Several DBMS provide SQL with spatial extension so that some geoprocessing operations can be embedded in SQL statements. Such support for geoprocessing is discussed in next section.

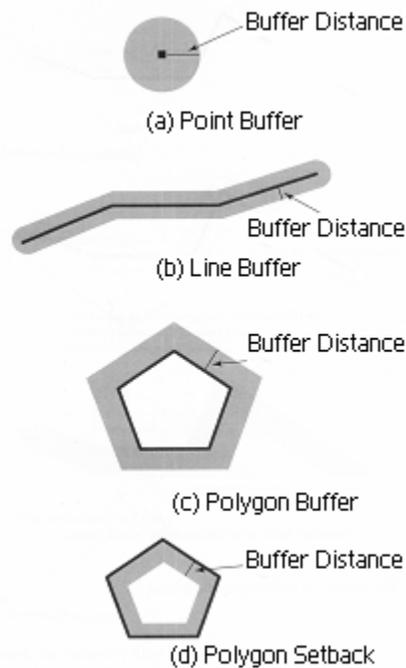


Figure 2-5. Point, line, and polygon buffers (adopted from Lo and Yeung 2002)

One vector-based geoprocessing that is widely used in transportation engineering and urban planning is network analysis. Network analysis is carried out in network layers where geospatial features are topologically organized as segments and junctions (Lo and Yeung 2002). Segments are linear features with start and end points (such as a street segment or a part of a telephone line) that intersect with each other at junctions. A segment can be associated with optional attributes, e.g., length or cost. There are a number of network analysis applications, such as pathfinding, tracing, and allocation. A shortest path between two locations, for example, can be found by applying Dijkstra's algorithm (1959) to a street network (Figure 2-6).

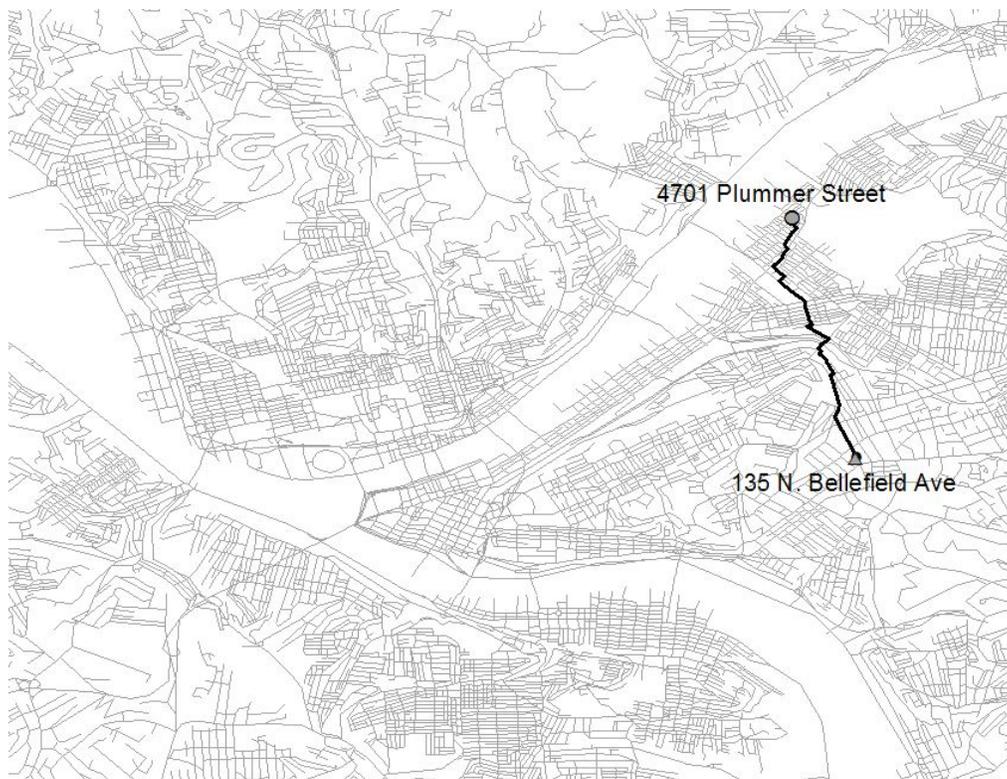


Figure 2-6. A shortest path between two locations in Pittsburgh, PA

It is worth of knowing that despite the differences between raster- and vector-based geoprocessing, most current GIS software is able to handle both types of geoprocessing, as discussed in the next section.

2.1.3. GIS Software and Spatial DBMS

GIS Software

The history of GIS can be traced back to 1970s when first GIS, Canada GIS (Tomlinson 1998), and Geographical Information Retrieval and Analysis System (Mitchell et al. 1977) were built. Due to the limit in computing capacity and data availability, early GIS were “dedicated systems” (Rigaux et al. 2002) in that they processed application-specific data with proprietary structures in mainframes.

Entering 1980s, as advances in hardware and software technology led to lower computing cost, minicomputers became a suitable platform for GIS software. A representative GIS package in this period is ArcInfo by Environmental Systems Research Institute (ESRI). As one of the first vector-based GIS (Lo and Yeung 2002), ArcInfo provides a number of tools for classification, thematic analysis, network analysis, terrain modeling, and statistical analysis. Spatial data in ArcInfo are stored in three modes: vector, raster or grid, and TIN. The three types of spatial data are managed by a module called *Arc* and linked to non-spatial data via internal object identifiers (ID) that are under control of *Info* module. *Arc* and *Info* modules are loosely coupled in such a way that either of them can be replaced by other modules that provide similar functionalities. For instance, non-spatial data can be managed in a DBMS such as Oracle instead of the *Info* module. The architecture of ArcInfo is depicted in Figure 2-7.

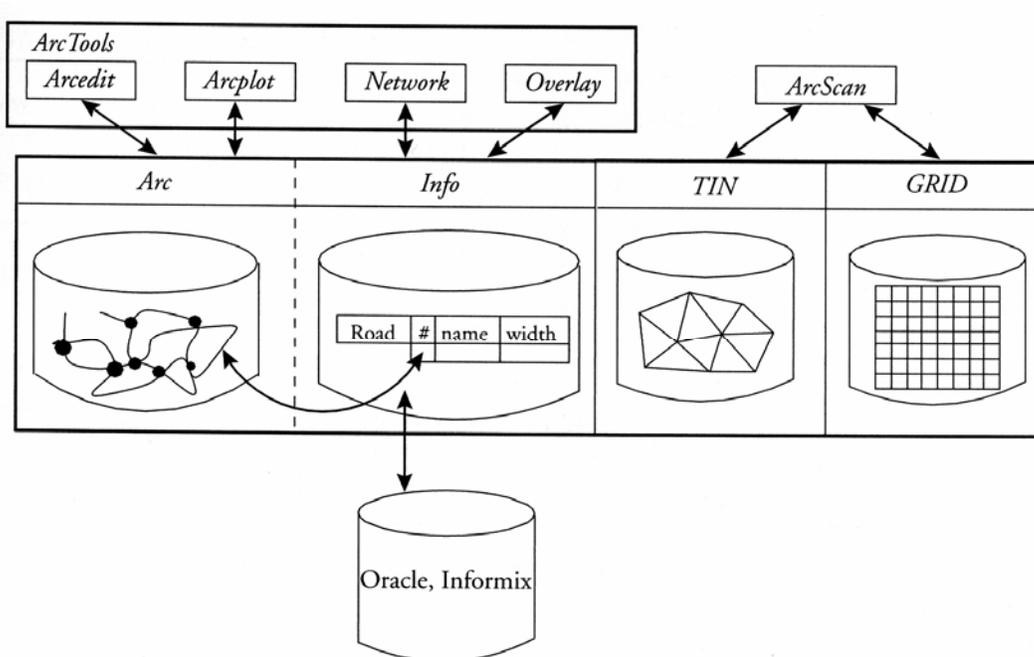


Figure 2-7. Architecture of ArcInfo (Rigaux et al. 2002)

In addition to mainframes and minicomputers, GIS software has found its way to personal computers (PC) due to the popularity and growing power of PC in last two decades. Such GIS are called “desktop GIS”. Although most desktop GIS support a number of geoprocessing operations, they are more focused on data integration, visualization and easy usage. One widely used desktop GIS is ArcView by ESRI. ArcView provides a set of tools for spatial query and analysis that are similar to those in ArcInfo but with some limitations. A key feature of ArcView is the easiness to integrate with applications, text editors, and DBMS (Rigaux et al. 2002). Many formats of raster data (e.g., TIFF, ERDAS) can be accessed directly from ArcView. Additional analytical functionalities can be added as extensions to ArcView (such as Network Analyst and 3D Analyst). Other desktop GIS include MapInfo and Intergraph Geomedia that offer similar geospatial functionalities in PC.

A latest development in GIS is Web-based GIS that provide access to geospatial data and geospatial analysis tools over the Web. Currently most Web-based GIS are primarily focused on online map publishing and simple geoprocessing (e.g., geocoding and attribute queries). There are some commercial products available for Web-based GIS, such as ArcIMS by ESRI, MapXtreme by MapInfo, and WebMap by Geomedia. It is expected that as the technology in wireless and mobile computing becomes more mature, the demand for Web-based GIS is expected will increase.

Spatial DBMS

Due to growing requests for geoprocessing and natural connection between geoprocessing and databases, more and more DBMS are equipped with extensions or modules to support geoprocessing and management of geospatial data. Such systems are often referred to as spatial DBMS. Rigaux et al. (2002) suggest five requirements that a spatial DBMS should fulfill in addition to relational-/object-oriented functionalities:

- Extend logical data representation to geospatial data.
- Integrate geospatial functions into query language.
- Have an efficient physical representation of geospatial data.
- Provide efficient data access to geospatial data, such as R-tree indexing (Guttman 1984).
- Implement new algorithms of important relational query processing, such as join, for geospatial data.

Many existing commercial/open source DBMS fall under the category of spatial DBMS, such as Oracle, PostgreSQL, and IBM DB2. Oracle Spatial, for instance, is a module that includes a set of operators and subprograms to enable storage, access, and analysis of geospatial data in an Oracle database (Murray et al. 2003). Geometric data are encapsulated in a special data type called *SDO_GEOMETRY*. An instance of *SDO_GEOMETRY* can be either an atomic element (e.g., point, line string, or polygon) or an ordered list of elements. The later one can be used to model complex geometries, e.g., an island inside a lake. Thus a relation with a column of *SDO_GEOMETRY* is a theme where geospatial objects are represented as rows. A sample statement in the form of SQL to create a wind trajectory table in Oracle Spatial is shown in Table 2-2. Field “*shape*” is of *SDO_GEOMETRY* type and stores geometric information of each trajectory. Oracle Spatial provides a number of geospatial operators and functions that can be mixed with relational operators in SQL. Table 2-3 demonstrates a query to find how many trajectories in year 2003 intersect with each other. *SDO_ANYINTERACT* is a spatial subprogram that returns true if two input geometries have non-disjoint spatial relationship.

Table 2-2. An SQL statement to create a trajectory table in Oracle Spatial

```
CREATE TABLE trajectory (  
    id number(6) primary key not null,  
    start_year NUMBER(4),  
    start_month NUMBER(2),  
    start_day NUMBER(2),  
    start_hour NUMBER(2),  
    start_lat NUMBER(8,3),  
    start_lon NUMBER(8,3),  
    start_level NUMBER(8,3),  
    shape SDO_GEOMETRY);
```

Table 2-3. A sample query in Oracle Spatial

```
SELECT COUNT(t1.*) FROM trajectory t1, trajectory t2
WHERE SDO.ANYINTERACT(t1.shape, t2.shape) = 'TRUE'
AND t1.start_year = 2003
AND t2.start_year = 2003
AND t1.id != t2.id;
```

Besides commercial products, support for geoprocessing can be found in open source DBMS such as PostgreSQL as well. Several geometric data types are available in PostgreSQL, e.g., point, line segment, path and polygon. Geometric data of objects in a theme are organized as a column together with other attribute data in a relation for that theme. Similar to Oracle Spatial, PostgreSQL can also handle queries that have both relational and geospatial operations. For instance, the above two Oracle Spatial queries can be expressed in PostgreSQL as follows. Again, field “*shape*” is the column for geometric data. “*path*” is a geometric data type of a list of connected points. Geometric operator “*?#*” tests if two geometries intersect. In addition, an extension called PostGIS by Refractions Research attaches to PostgreSQL some geospatial functionalities (e.g., support to geographic/projection coordinate systems).

Table 2-4. An SQL statement to create a trajectory table in Oracle Spatial

```
CREATE TABLE trajectory (
  id integer,
  start_year integer,
  start_month integer,
  start_day integer,
  start_hour integer,
  start_lat double precision,
  start_lon double precision,
  start_level double precision,
  shape path,
  PRIMARY KEY (id));
```

Table 2-5. A sample query in Oracle Spatial

```
SELECT COUNT(t1.*) FROM trajectory t1, trajectory t2
WHERE t1.shape ?# t2.shape
      AND t1.start_year = 2003
      AND t2.start_year = 2003
      AND t1.id != t2.id;
```

Despite the differences in representation of geospatial data and implementation of geoprocessing between spatial DBMS, they enjoy one advantage over conventional GIS, i.e., using a high-level query language (extended SQL) capable of expressing both relational and geospatial criteria, as seen in queries listed above. On the other hand, functionalities of geoprocessing are limited in spatial DBMS due to requirements of logical and physical data structures in DBMS. For instance, currently there is little support for raster-based geoprocessing in spatial DBMS since it is difficult to fit raster data into relational DBMS.

2.2. Grid Computing

A definition of grid computing is given by Foster et al. (2001): “coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations (VO)”. Several issues are highlighted in this definition:

- “*Resource*” has a broad meaning which includes data, CPU cycles, and storage.
- *Resource sharing* is controlled. Upon receiving requests from users, resource providers can decide who are allowed to use their resources and how they are shared.
- *Collaboration* across multiple institutions/individuals is enabled in the form of VOs, which are formed by various participants willing to share resources in completing some work (Casanova 2002). Relationships among VO members can dynamically change over time; not only the members, but also the access they have to others’ resources.

Due to its wide reference and acceptance, the above definition is adopted in this dissertation.

2.2.1. Core Architecture and Middleware

Core Architecture

A layered architecture proposed by Foster et al. (2001) is shown in Figure 2-8. This architecture has been adopted by many research projects and organizations, e.g., e-Science Project (e-Science 2004). This core architecture is high level and quite independent of specific implementations. As shown together with the corresponding Internet protocols (Figure 2-8), this architecture is composed of five stacked layers: fabric, connectivity, resource, collective, and application.

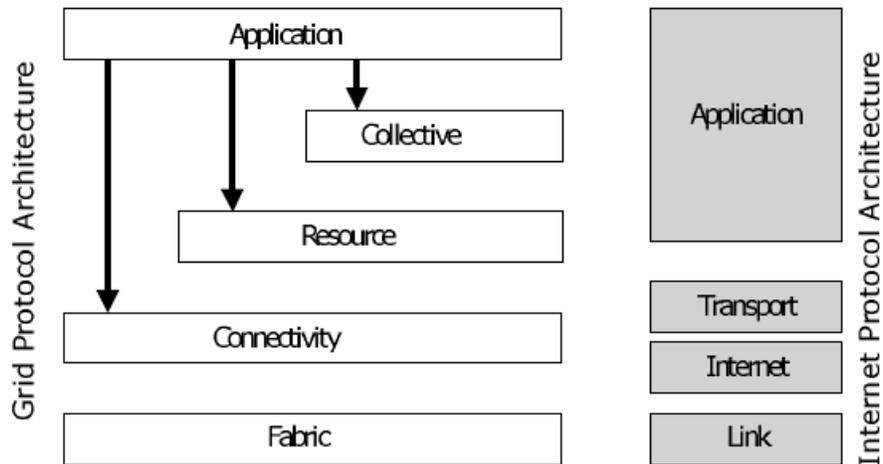


Figure 2-8. Layered grid architecture (Foster et al. 2001)

At the lowest level, the fabric layer provides access to local resources by implementing resource-specific operations for various native systems. Two basic mechanisms are offered by the fabric layer:

- enquiry for structure, status and capabilities of resources; and
- resource management to support control of delivered quality of service.

The connectivity layer specifies a set of protocols for communication and authentication. The TCP/IP protocol, or other network protocols, can be adopted to exchange information among fabric layers. Built upon communication protocols, the authentication protocols offer secure mechanisms to verify users and resources. Authentications considered as important for grid computing are:

- Single sign-on. After logging in to a VO the first time, users must be able to access various resources without being checked for identity again.
- Delegation. Users should be able to assign a program to access authorized resources on their behalf. This program in turn can launch other programs with restricted rights.
- Integration with local security solutions. Grid security should interoperate with local solutions instead of replacing them.
- User-based trust relationships. When users are authorized for multiple resources, they should be able to use them together without interactions with security administrators.

Protocols implemented in the resource layer enable sharing of individual resources. Corresponding to the two mechanisms in the fabric layer, the two classes of protocols needed in the resource layer are:

- information protocols: retrieve information about the structure and status of a resource; and
- management protocols: negotiate access to a resource and specify requirements and operations.

Complementing the resource layer, the collective layer focuses on global status and interactions among collections of resources. Some of functionalities that can be implemented in this layer are:

- directory services for resource discovery;
- co-allocation, scheduling and monitoring services for efficient and effective task execution; and
- collaboratory services to support information exchange among large number of users.

At the highest level of the architecture, the application layer comprises VO applications built upon the lower layers.

Middleware

Currently there are three middleware packages available for building grid systems: Globus Toolkit (Foster and Kesselman 1998a; Foster et al. 2002), Legion (Chapin et al. 1999; Grimshaw et al. 2002; Grimshaw and Wulf 1997), and UNICORE (Erwin et al. 2002; Erwin and Snelling 2001; Streit et al. 2005). Globus Toolkit provides a set of open-source libraries for building grid services and applications. Components in Globus are Grid Resource Allocation and Management (GRAM) protocol, Meta Directory Service (MDS), and Grid Security Infrastructure (GSI).

Legion takes an object-oriented approach to building a single, coherent virtual machine for grid computing. In a Legion-enabled grid system, various computing resources (e.g., data sources and applications) are represented as Legion objects. Legion objects are created and managed by corresponding classes or metaclasses. An object can respond to other objects in a system through its methods. Legion defines Application Programming Interfaces (APIs) for object interaction but does not require specific implementation languages or network protocols. The UNICORE middleware is built upon a layered architecture that includes user, server, and target system tiers. The user tier provides a graphical interface for users to exploit services available in grids. Users' computing tasks are sent to the server tier as Abstract Job Objects (AJO). The server tier consists of two components, a Gateway and a Network Job Supervisor (NJS). The Gateway performs authentication and authorization on incoming requests. The NJS is responsible for mapping abstract resource descriptions specified in an AJO to resources in target systems. The mapped AJO is then passed to the Target System Interface (TSI) in the target system tier. The TSI interacts with underlying local resource management systems, such as Computing Center Software (Hovestadt et al. 2003), to execute user-specified tasks and retrieve results back to the client tier.

Globus Toolkit has gained much attention from the academic community and has been applied to several back-bone projects, e.g., TeraGrid (TeraGrid 2004). Due to its conformity to the core grid architecture and its acceptance by the academic community, it is worthwhile to describe the structure and components of this middleware.

At the fabric layer, Globus primarily uses existing protocols and interfaces to interoperate with local platforms. It also supports functions to probe resource status in case such functions are absent in local sites.

In the authentication layer, the public-key based GSI protocols (Butler et al. 2000; Foster et al. 1998) are employed. Extending the Transport Layer Security (TLS) protocols (Dierks and Allen 1999), GSI supports the security characteristics stated earlier, e.g., single sign-on and delegation.

Within the resource layer, the HTTP-based GRAM protocol is to allocate resources and monitor job execution (Czajkowski et al. 1998). Major components of GRAM are depicted in Figure 2-9. GRAM provides a client library for applications to send requests to the gatekeeper, which is a simple component running at a remote site. Upon receiving a request, the gatekeeper conducts mutual authentication of clients and resources and generates a job manager to execute tasks in local hosts. The GRAM reporter is used to update the resource information in MDS.

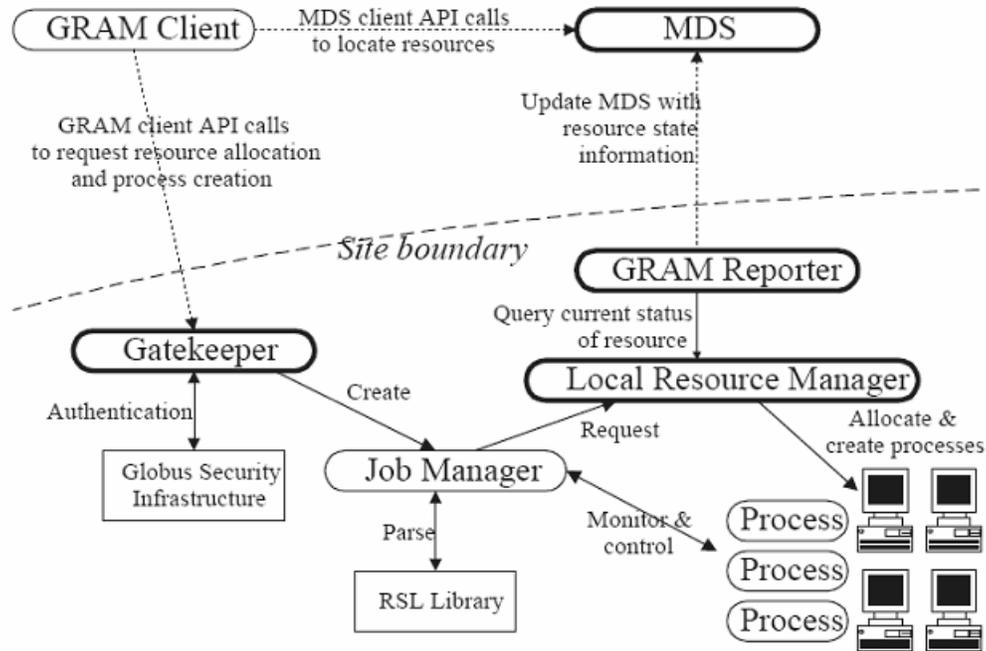


Figure 2-9. Major components of GRAM (Czajkowski et al. 1998)

MDS is one of the functionalities provided in Globus at the collective layer for resource registration and discovery. It is comprised of two types of members: service providers and aggregate directory services (Czajkowski et al. 2001), as shown in Figure 2-10. Service providers publish their services in one or several aggregate directory services for some VO via GRid Registration Protocol (GRRP) while applications or aggregate directory services use GRid Information Protocol (GRIP) to query information about resources. GRRP is a soft-state protocol in that information about a resource may be discarded if there are no subsequent notifications from the service provider for some time. Using GRRP, several directory services can also participate in a VO to form the VO's aggregate directory service. GRIP uses Lightweight Directory Access Protocol (LDAP) as data model, query language and protocol. In addition to providing static information like LDAP, GRIP can generate dynamic information upon a request as well.

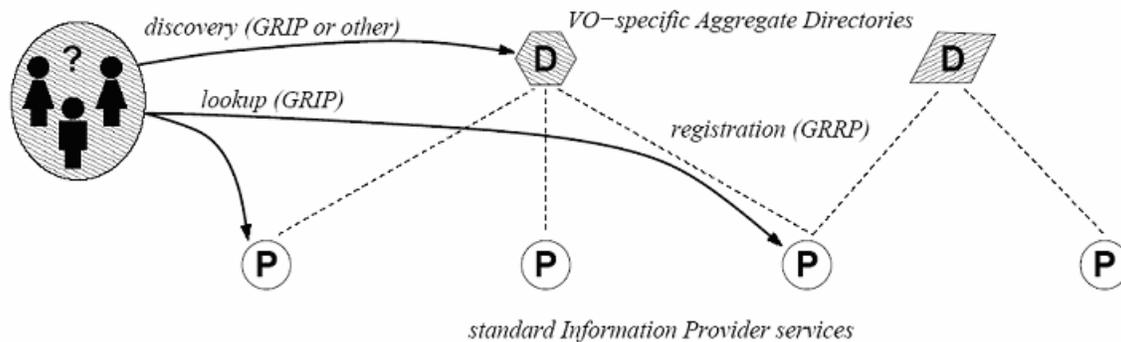


Figure 2-10. Overview of MDS architecture in Globus (Czajkowski et al. 2001)

2.2.2. Web Services and Service-Oriented Architecture

Web Services

Web services, an emerging distributed computing technique, have recently caught the attention of researchers from both academia and industry. Web services are being widely adopted due to several distinctive properties such as separation of service description from service implementation and platform neutrality. It is perceived by the grid community that Web services would leverage grid computing through providing complementing services (e.g., dynamic discovery and composition of services) and numerous tools (Foster et al. 2002). The integration of grid and Web services has been investigated in the proposed Open Grid Services Architecture (OGSA). A brief description of Web services is given below as a reference before introducing OGSA.

A definition of Web services given by Booth et al. (2003) states that Web services are software systems that support interoperability at machine-to-machine level over networks. Web services advertise themselves using a machine-processable interface, such as Web Service Description

Language (WSDL), through which other systems can interact with them. Communication between a Web service and its client is typically carried in the form of Simple Object Access Protocol (SOAP) messages. Considering that this definition is adopted by the World Wide Web Consortium (W3C) as the draft for Web services standard, the concept of Web services in the remainder of this document is aligned with this definition.

Service-Oriented Architecture and Technology Stack

Graham et al. (2002) suggest a service-oriented architecture for Web services (Figure 2-11) that includes three roles: service requestor, service registry and service provider. A service provider is responsible for generating a service description, publishing that description to one or more service registries, and responding invocation messages from service requestors. A service requestor is a customer of a Web service that can either be a human being or a program/agent. The customer checks for a service description in some service registry and then binds to the Web service. The responsibility of a service registry is receiving service descriptions from service providers and matching users' queries with them.

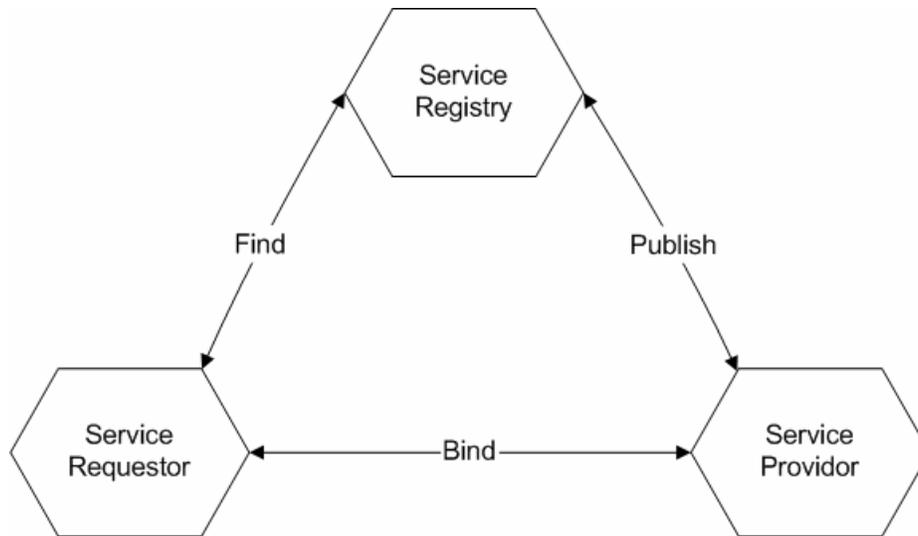


Figure 2-11. Service-oriented architecture for Web services (Graham et al. 2002)

Various technologies, including SOAP, XML, and WSDL, are involved in enabling interactions within this service-oriented architecture. A stack diagram of these component technologies is illustrated in Figure 2-12.

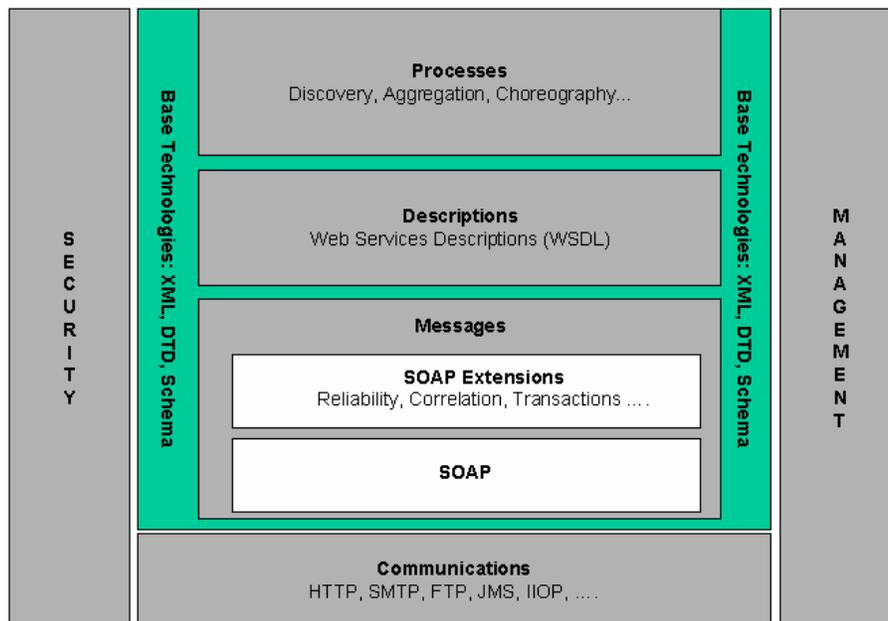


Figure 2-12. Technology stack diagram for Web services (Booth et al. 2003)

At the lowest level in the stack diagram are the communication mechanisms that can be based on a variety of protocols (HTTP, SMTP and others). The level above the communications level is the messages level, representing how a message is exchanged between providers and requestors. SOAP is considered to be a robust and powerful framework for functionalities in this layer. The descriptions level is for creating a common understanding of message structure and data types for both service providers and users. A proposed standard (WSDL) is currently used to describe the invocation syntax of Web services. Languages for handling semantic contents, such as Resource Description Framework (RDF) and Web Ontology Language (OWL), can fit in this level in future. The processes level contains high-level tools that require process descriptions, such as process aggregation, and service discovery according to specific criteria. Universal Description Discovery and Integration (UDDI) is one of the techniques used in service registration and discovery. In addition, a couple of mechanisms for service aggregation/orchestration, such as Web Services Flow Languages (WSFL) (Leymann 2001) and Xlang (Thatte 2001), have been proposed and can be placed at this level as well. As the base technology for Web services, eXtended Markup Language (XML) is placed in the vertical column passing through the upper three levels on the diagram. The two columns on both sides of the diagram show the security and management technologies that impact each level of the diagram.

2.2.3. Open Grid Service Architecture

Grid Services

A major objective of grid computing is to enable coordinated resource sharing in VOs. Foster et al. (2002) argue that virtualization is important to grid computing in that it allows consistent

access to heterogeneous resources, mapping between multiple logical resource instances, and composition of services to create complex services. They point out that a service-oriented approach simplifies virtualization since it encapsulates various implementations with a common interface. Thus Web services can complement grid computing in realizing resource sharing. The concept of grid services is then developed to combine Web services and grid.

A grid service is referred to as a Web service that supports a set of standard interfaces and conforms to specific conventions. The standard interfaces are listed in Table 2-6. Grid service instances are dynamically created by the *Factory* interface. After its creation, every instance is assigned a globally unique name, Grid Service Handle (GSH), to distinguish it from other instances. Since the instance- or protocol-specific information about the instance may vary over the lifetime of the instance, this information is encapsulated in another abstraction called Grid Service Reference (GSR). GSH and GSR can be related via the *HandleMap* interface. This interface is defined to return a valid GSR for a given GSH. A grid service can be explicitly terminated via the *Destroy* operation. Thus upon failure of an operation, a system can reclaim associated services and state. The *SetTerminationTime* operation is used for soft-state lifetime management of grid service instances. Soft-state protocols require a system to send a stream of subsequent “keepalive” messages to keep its state at a remote location, which makes systems resilient to failure of single message loss.

Grid services have to follow conventions that address naming and upgradeability. Since complex distributed computing environments like grids require service upgrade be carried independently (Foster et al. 2002), there should be a way to offer clients compatible services if the specific

version they look for is not available. Through upgradeability convention, clients are able to identify when a service changes and when it is backward-compatible.

Table 2-6. OGSA grid service interfaces (Foster et al. 2002)

PortType	Operation	Description
GridService	FindServiceData	Query a variety of information about the grid service instance, including basic introspection information (handle, reference, primary key, home handleMap: terms to be defined), richer per-interface information, and service-specific information (e.g., service instances known to a registry). Extensible support for various query languages.
	SetTerminationTime	Set (and get) termination time for grid service instance.
	Destroy	Terminate grid service instance.
NotificationSource	SubscribeTo-NotificationTopic	Subscribe to notifications of service-related events, based on message type and interest statement. Allows for delivery via third party messaging services.
NotificationSink	DeliverNotification	Carry out asynchronous delivery of notification messages.
Registry	RgisterService	Conduct soft-state registration of grid service handles.
	UnregisterService	Deregister a grid service handle.
Factory	CreateService	Create new grid service instance.
HandleMap	FindByHandle	Return Grid Service Reference currently associated with supplied Grid Service Handle.

Open Grid Service Architecture Platform

Open Grid Service Architecture (OGSA) is currently under development in the OGSA working group (OGSA-WG 2004) of the Global Grid Forum (GGF 2004). The purpose of OGSA is to standardize approaches to solving common problems in grid systems, such as communications among services, negotiation of authorization, service discovery, and management of a set of services (Foster et al. 2003). An architecture proposed by OGSA-WG is shown in Figure 2-13.

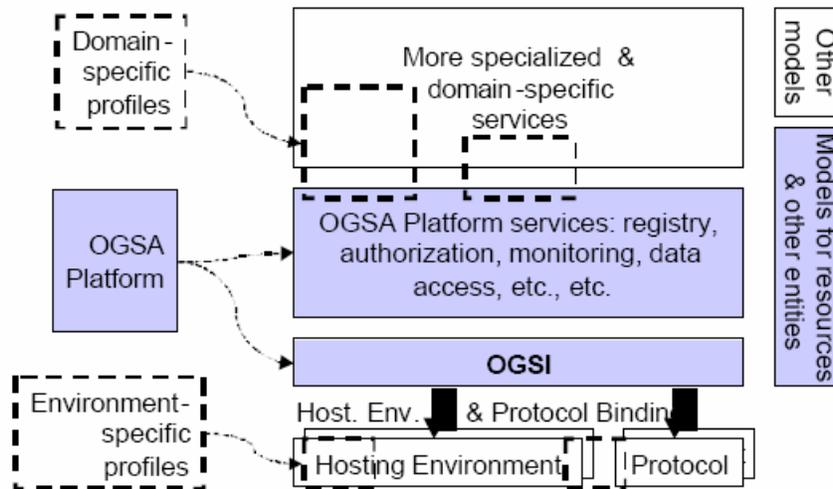


Figure 2-13. OGSA platform components and related profiles (Foster et al. 2003)

OGSA has three principal elements (shaded boxes in Figure 2-13): Open Grid Services Infrastructure (OGSI), OGSA services, and OGSA models. OGSI defines the building blocks for grid systems, i.e., mechanisms for creating, managing grid services and exchanging information among them. It also includes the conventions that regulate interaction between a client and a grid service. Built upon OGSI are OGSA services, which define interfaces and associated behaviors to large-scale systems but not included in OGSI. For instance, service discovery, data access, data integration, messaging and monitoring all fall in this layer. The “OGSA models” element

provides models for common resource and service types to support interface specifications listed above.

A set of environment-related profiles are suggested in OGSA to complement the principal elements (boxes with dashed lines in Figure 2-13). The profiles of hosting environment bindings are aimed to enable portability of grid service implementations. For example, grid services can be made portable among OGSI-enabled Java2 Enterprise Edition (J2EE) systems through an “OGSA J2EE Profile” with standardized Java APIs. The profiles of protocol bindings deal with interoperability among different grid services. Transport and authentication mechanisms are left undefined in OGSI and treated as binding properties in binding profiles. Thus services have the flexibility to choose transport and authentication implementations based on different needs. Profiles of domain-specific services address issues in designing interfaces and models for specific domains. For instance, common interfaces and models for distributed databases can be specified in the “OGSI Database Profile”.

2.3. Query Optimization in Distributed Databases

A large portion of data involved in geoprocessing is usually stored in relational tables. Thus distributed query optimization techniques for relational databases can be applied to grid-based geoprocessing. This section presents an overview of latest developments in query optimization from the query structure and data allocation perspectives and is concluded with the discussion of the economic model as a novel optimization approach.

2.3.1. Dynamic Programming

Selinger et al. (1979) take a dynamic programming approach in IBM's System R project and it has been adopted in most commercial database systems (Kossmann and Stocker 2000). This dynamic programming algorithm works in a bottom-up fashion. It first builds all possible access paths (the actual data structure and algorithm to be used to access the data) for n tables that are involved in a query. It then iterates two-way join plans based on the access paths. The three-way and n -way plans are generated in a similar manner. The n -way plans are complete plans for the query. The advantage of this dynamic programming algorithm is that in each iteration inferior plans are discarded by a pruning function as shown in lines 3 and 10 of Figure 2-14. Thus, the complexity in optimization is significantly reduced. In a distributed system, however, an operation *scan(A at Site 1)* should not be pruned right away even if it costs more than *scan(A at Site 2)*. This is because *scan(A at Site 1)* may result in a better plan afterwards, for instance, when the next operation requires that the result of *scan(A)* to be in Site 1. In such situations, *scan(A at Site 1)* can only be pruned if the cost of *scan(A at Site 2)* plus the cost of shipping the result from Site 2 to Site 1 is greater than the cost of *scan(A at Site 1)*.

Input: SPJ query q on relations R_1, \dots, R_n
Output: A query plan for q

```

1: for  $i = 1$  to  $n$  do {
2:   optPlan( $\{R_i\}$ ) = accessPlans( $R_i$ )
3:   prunePlans(optPlan( $\{R_i\}$ ))
4: }
5: for  $i = 2$  to  $n$  do {
6:   for all  $S \subset \{R_1, \dots, R_n\}$  such that  $|S| = i$  do {
7:     optPlan( $S$ ) =  $\emptyset$ 
8:     for all  $O \subset S$  do {
9:       optPlan( $S$ ) = optPlan( $S$ )  $\cup$  joinPlans(optPlan( $O$ ), optPlan( $S \setminus O$ ))
10:      prunePlans(optPlan( $S$ ))
11:     }
12:   }
13: }
14: finalizePlans(optPlan( $\{R_1, \dots, R_n\}$ ))
15: prunePlans(optPlan( $\{R_1, \dots, R_n\}$ ))
16: return optPlan( $\{R_1, \dots, R_n\}$ )

```

Figure 2-14. Dynamic programming algorithm (Kossmann and Stocker 2000)

Although dynamic programming is superior over the enumeration of the entire search space (i.e., searching all candidate execution plans), it can still be prohibitive as queries become complex (Steinbrunn et al. 1997). In processing complex queries that require more memory than a machine can provide, dynamic programming may cause machine crash or severe paging of the operating system. Iterative Dynamic Programming (IDP), an extension of dynamic programming proposed by Kossmann and Stocker (2000), shows advantages in handling such problems. Unlike dynamic programming, IDP stops generating k -way joins ($k < n$, where n is the number of relations involved in a query) before system resources are exhausted. Instead, IDP selects a generated k -way join and deletes access paths and joins that involve at least one of the relations in the selected join. It then restarts dynamic programming with the selected k -way join as building block to complete the optimization. Figure 2-15 illustrates a sample process of a five-way join query with $k=3$. After the first three steps the memory is used up and IDP removes all the joins and access paths except the join τ and relations C and E which τ does not cover. A

new round of dynamic programming is launched with τ , C and E in Step 5 and the final plan, a three-way join, is obtained in Step 6. If both dynamic programming and IDP are viable in a system, IDP yields plans as good as the ones generated by dynamic programming. When dynamic programming consumes all system memory and causes system crash, IDP can still run and yield as-good-as possible plans.

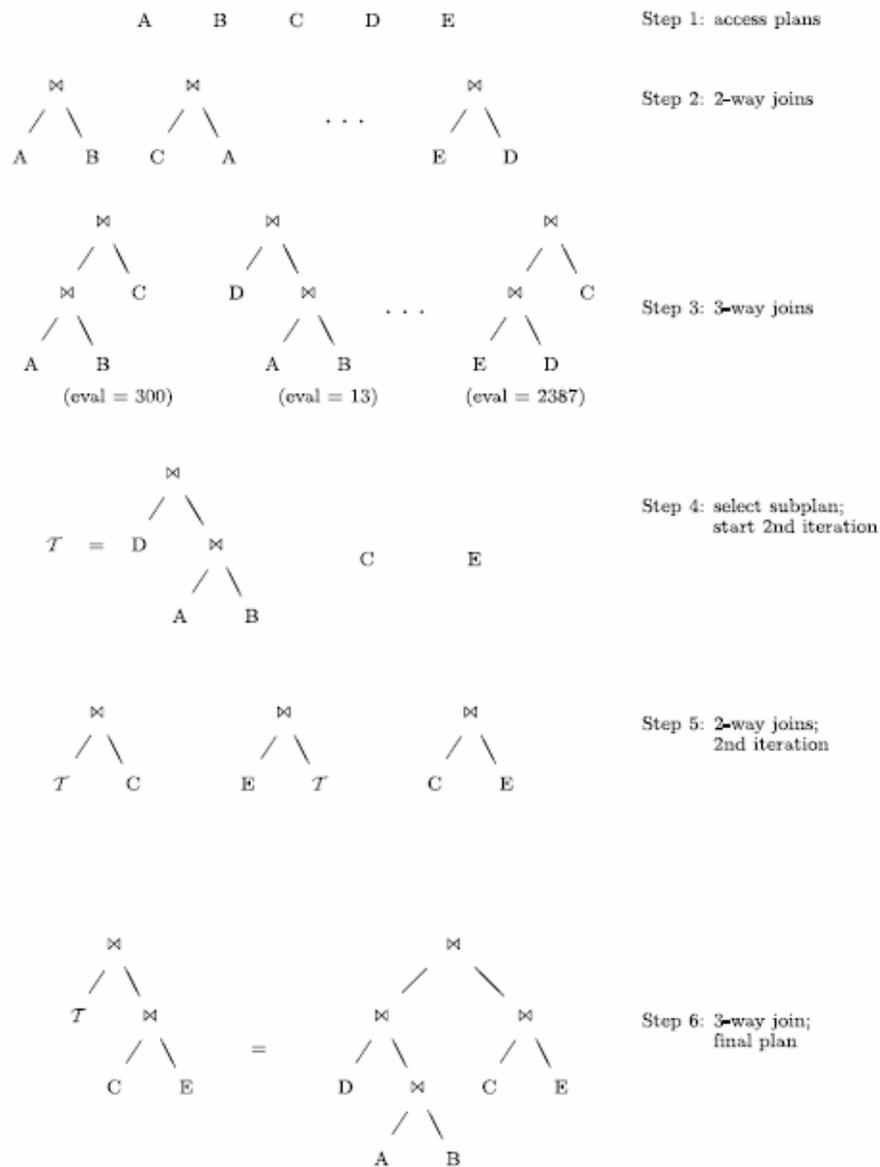


Figure 2-15. A sample optimizing process of IDP (Kossmann and Stocker 2000)

2.3.2. Randomized Algorithms

As an alternative to dynamic programming, a set of randomized algorithms has been developed (Ioannidis and Kang 1990; Swani 1989; Swani and Gupta 1988) to avoid the high cost in evaluating execution plans. In a randomized algorithm, points (candidate execution plans) in the search space are treated as points that are connected via edges: two points are connected if they can be transformed to each other by following one of the predefined moves. Figure 2-16 demonstrates allowable moves in a search space of bushy processing tree. A randomized algorithm looks for a point with minimum cost from a start point. Instead of checking cost at every neighbor (a plan within one move), a neighbor is selected randomly and checked. The process will continue from the neighbor node if its cost is lower than the start point, otherwise the algorithm turns to a new neighbor around the start point. A plan is considered local minimum if no neighbor with lower cost can be found in a number of tries. A randomized algorithm terminates after a predefined time interval or a specified number of start points has been tried. Among a set of local minimums, the one with the lowest cost is chosen as the final result. One advantage of randomized algorithms is the constant space requirement which is usually lower than dynamic programming for simple queries (Kossmann and Stocker 2000; Steinbrunn et al. 1997).

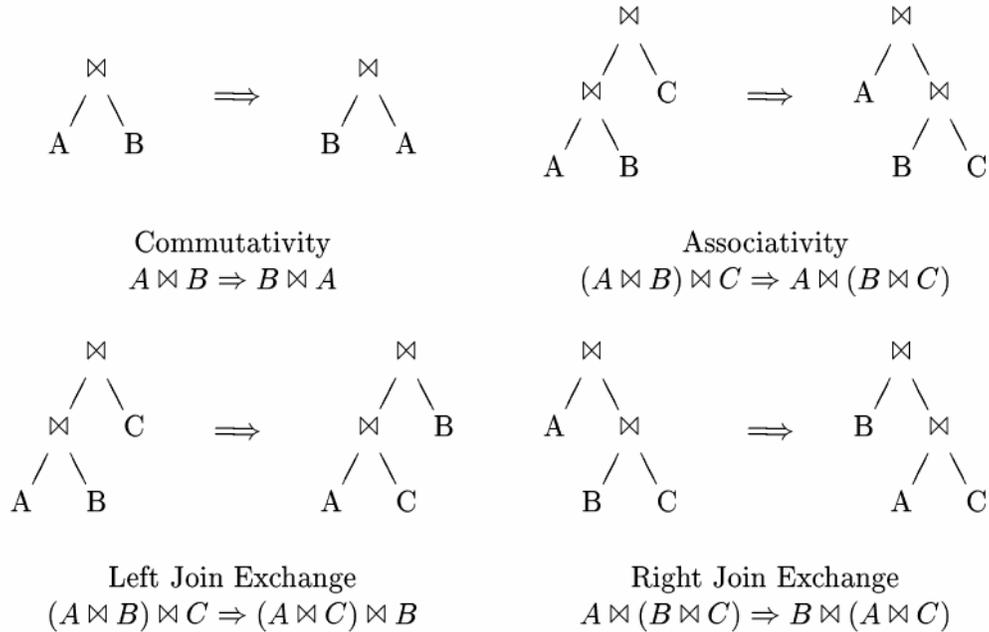


Figure 2-16. Moves defined for a search space of a bushy processing tree (Steinbrunn et al. 1997)

2.3.3. Dynamic Data Allocation

An alternative approach to improve distributed query processing is strategically allocating copies of data. In distributed database systems, multiple copies of data are usually placed according to predictions about usage of queries and network topology and remain in the same location through query executions. But for scenarios where query usage cannot be predicted, the dynamic data allocation approach is preferred. Two techniques, replication and caching, are explored in the dynamic data allocation approach and described in the following subsections.

Dynamic Replication

Replication is usually accomplished by duplicating data sets at multiple servers. A representative dynamic replication algorithm is the Adaptive Data Replication (ADR) by Wolfson et al. (1997),

which is designed to reduce communication costs by moving copies to servers close to clients that are likely to access the data. The authors observe that when the “read-one-write-all” replica protocol (Ozsu and Valduriez 1999) is adopted, a replication scheme for an object (the set of all servers at which the object is replicated) should be a connected subgraph so that minimum communication costs can be achieved. For instance, Figure 2-17 shows a replication scheme including Servers 5, 6 and 7 in a network with 9 servers. A replication scheme may expand or contract on the \bar{R} -neighbor (i.e., servers that belong to a scheme but have a neighbor that does not belong to that scheme) depending on the read/write pattern occurring in the network. Three tests are defined in ADR to control such behavior of replication scheme based on read/write statistics:

- Expansion test. For a neighbor j of an \bar{R} -neighbor (j does not belong to the scheme), if it sends more read requests than write requests, then add j to the scheme. In Figure 2-17, for example, if Server 2 sends more read requests to Server 5 (either from its own clients or from Server 1), it should be included in the scheme.
- Contraction test. For an \bar{R} -neighbor i , the copy of data will be deleted if it receives more write requests than read requests. For example, Server 5 should drop its copy if Servers 6, 7, 8 and 9 send more write requests than read requests propagated from Servers 1, 2, 3, 4 and 5.
- Switch test. In a network where the replication scheme only has one Server i , if the number of requests i receives from one of its neighbors, e.g., n , is more than the number of all other requests, i will send a copy of the object to n and discard its own copy.

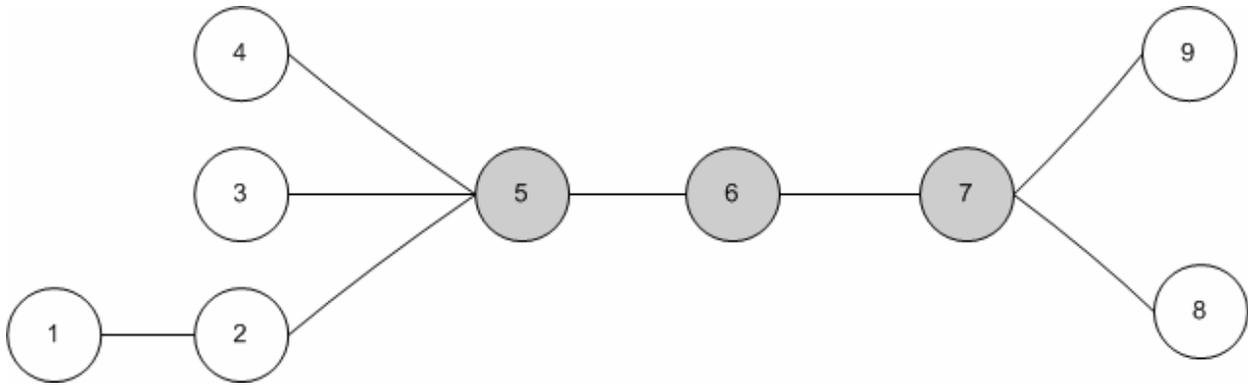


Figure 2-17. ADR diagram, adopted from Wolfson et al. (1997)

Wolfson et al. (1997) show that in tree-shaped networks, ADR converges the replication scheme to the optimal one when the read/write pattern of an object is regular. The communication cost in ADR is lower than the one with the optimal static replication scheme as the read/write pattern is randomized.

Cache Investment

Contrary to replication, caching tries to duplicate a table/index or part of it at a client to optimize query processing. Kossmann et al. (2000) suggest a cache investment method to conduct caching in a way that only copies of data that promise future benefits are kept at clients. Noticing that execution of a suboptimal plan can cause caching of some data at the client site, the authors argue that such execution can be a good investment if the cached data will be used in many queries afterwards. Two policies are proposed in finding investment candidates: reference-counting and profitable. The profitable investment policy tries to estimate the investment (the cost to cache an item) and Return Of Investment (ROI), i.e., the expected gain by caching an item, while the reference-counting policy selects candidates by their frequency without computing ROI and the investment. Nonetheless, both policies adapt to query patterns occurring

in a client according to history information on tables involved in queries. History information is represented in the form of a value calculated by the following equation:

$$V_t^c(q) = v_t^c(q) + \alpha * V_t^c(q-1) \quad 2.1$$

where $V_t^c(q)$ is the value of Table t at Client c after Query q is executed, $V_t^c(q-1)$ is the value of Table t before q is executed, α is a weight factor and $v_t^c(q)$ is a component that varies in the two policies. In the reference-counting policy, $v_t^c(q)$ in Equation (2-1) is set to 1 if Table t is involved in Query q or 0 otherwise. So $V_t^c(q)$ is a count of queries referring to Table t weighted by how recently they are used. The reference-counting policy selects tables with larger $V_t^c(q)$ as candidates for caching. The profitable policy computes the cost of investing in a Table t for Query q as the difference between the cost of best execution plan for q and the cost of a plan to bring pages of t to the client. ROI is estimated as $V_t^c(q)$ while $v_t^c(q)$ is set as the gain in q by caching t at Client c , i.e., the difference in the cost of q with and without caching t . Candidate tables are chosen based on three criteria:

- Query q involves Table t .
- ROI is higher than the investment.
- ROI minus investment is greater than the ROI of currently cached item(s) which would be replaced if the new item is cached.

The third criterion ensures that only most valuable items are kept in the client's cache.

Since updating a part or whole table makes caching less attractive, the value of $V_t^c(q)$ should be reduced when updates occur. If an invalidation-based cache consistency protocol is applied, $V_t^c(q)$ is calculated by the following equation:

$$V_t^c(q) := \frac{a-u}{a} * V_t^c(q) \quad 2.2$$

where a is number of pages of Table t that are cached at Client c after Query q is executed and u is number of pages of Table t that are updated before Query $q+1$ is executed. For propagation-based cache consistency protocols, $V_t^c(q)$ can be computed as:

$$V_t^c(q) := V_t^c(q) - m \quad 2.3$$

where m is the cost to send an update message to the client.

After conducting performance experiments on the two investment policies, the authors suggest that the profitable policy should be used in heterogeneous environments or where queries and updates are mixed. If execution overhead and ease of implementation are of concern, the reference-counting policy should be considered.

2.3.4. Economic Model

The mid-1980s saw a large body of research applying economic models to distributed computing (Ferguson et al. 1996). Similar to the free market mechanism functioning in capitalism, it is believed that in a distributed system, clients' needs would be satisfied if every server tries to maximize its profit by selling its services to clients. A representative distributed database system based on such a theory is Mariposa (Sidell et al. 1996; Stonebraker et al. 1996). The architecture of Mariposa is depicted in Figure 2-18 and its query processing procedure is briefly summarized below.

- A client generates a query and attaches a budget to it. The budget is determined by the importance of the query and its expected execution time. For instance, a client is willing to pay \$5 for completing a query Q in one minute but only \$2 if it is done in ten minutes.
- The query is parsed and optimized by a single-site optimizer, which ignores data distribution and prepares a plan by assuming that all data are in a single machine.
- A fragmenter decomposes the plan into a fragmented query plan by consulting a name server that holds the meta-data about each fragment.
- The plan is sent to a broker that will initiate an auction for carrying operations in the fragmented query plan. Servers that have parts of the data or want to conduct one or more operations in the plan are invited to participate in this auction and are asked to submit their bids in the form:

(Operator o , Price p , Running Time r , Expiration Date x)

- The broker gathers bids from servers and assigns operations to winning servers to execute. The broker keeps as its profit the rest of the budget after paying for the execution. For its own interest, the broker tries to maximize its profit by looking for the best offer in the bidding process. Using the example above, the broker can earn \$2 if it finds a server to evaluate Q in one minute for \$3. If no server can do this job in one minute, the broker will look for candidates that are capable to finish Q within ten minutes. If it happens that there is such a server with a bid of \$1, the broker will assign Q to that server and make a profit of \$1. In the case that there is no server qualified to carry Q within the specified time/budget limits, the broker will reject the query. The client has to revise the budget to make it acceptable.

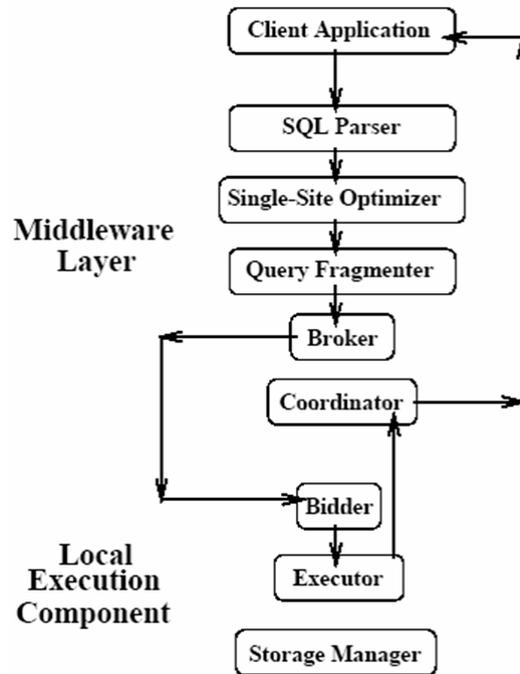


Figure 2-18. Architecture of Mariposa (Stonebraker et al. 1996)

Unlike cost-based optimization techniques, Mariposa reduces the complexity of estimating query execution costs by having servers bid on executing a query. Another benefit is that it offers a new way to realize dynamic data allocation. Sidell et al. (1996) present an extension of the economic model to support replica management which allow sites trade copies of fragments according to the changing pattern of activities.

2.4. Optimization Techniques in Grids

2.4.1. AppLeS and Condor

AppLeS (Berman 1999; Berman and Wolski 1997) is a high-performance scheduler developed at San Diego Supercomputing Center at University of California, San Diego. AppLeS is application-centric since it makes scheduling decisions according to the impact of various

resources on an application (Berman and Wolski 1997). For each application AppLeS assigns an AppLeS agent. The agent selects resources, chooses a performance-efficient schedule and implements that schedule. The organization of an AppLeS agent is depicted in Figure 2-19 which has four subsystems and an active agent called coordinator that harmonizes the activities among subsystems. The four subsystems are described below:

- Resource selector. AppLeS agent filters out inferior resource combinations in order to reduce the number of candidate schedules that are to be compared. Access right, resource capacities, and other constraints can be considered in filtering.
- Planner. Feasible resource configurations are passed to the planner to generate candidate schedules.
- Performance estimator. The estimator projects the performance for a candidate schedule with respect to the performance metric specified by the user.
- Actuator. The best candidate schedule is sent to the actuator that implements it at target resources.

Subsystems share the information pool that is composed of input from Network Weather Service (NWS) (Wolski 1997), User Interface (UI) and Models as shown in Figure 2-19. NWS provides dynamic information about current system status and forecasts the resource load at the time a task is scheduled. The user can specify, via the UI, the information about an application, including its structure, characteristics and constituent tasks, as well as execution constraints and criteria for performance. Default metacomputing application class models and application-specific models are stored in Models in order to estimate application performance.

In running AppLeS, the resource selector screens resources according to the information provided by the user through the UI. If such information is absent, suitable default values will be used. For each feasible resource configuration received from the resource selector, the planner generates a candidate schedule. The coordinator calls for the performance estimator to test each schedule according to the user's performance objective. The schedule that satisfies the objective best is selected and sent to the actuator to implement.

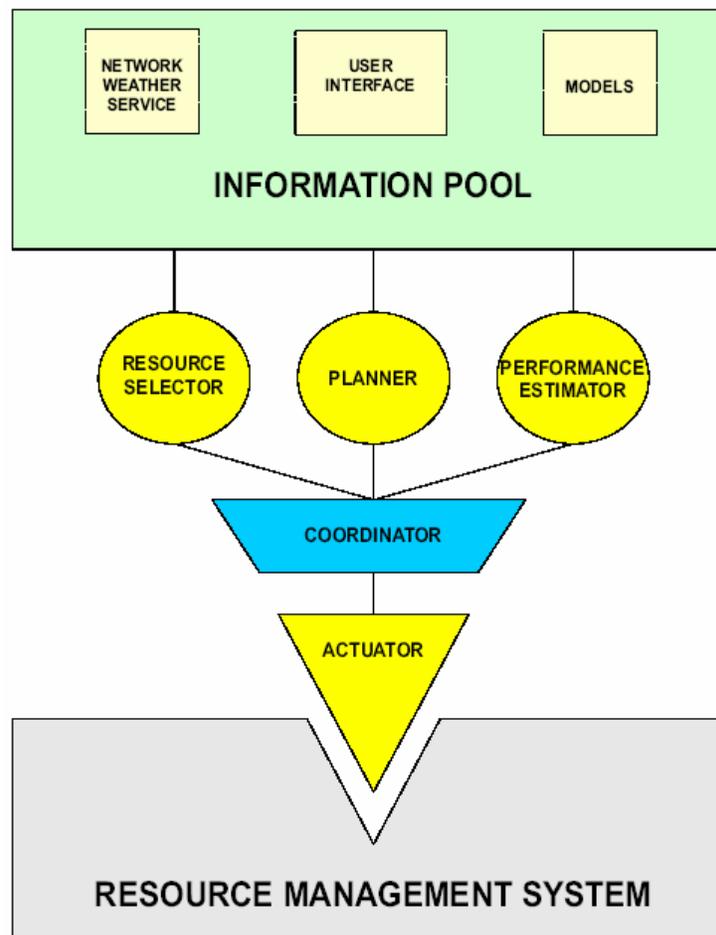


Figure 2-19. Organization of an AppLeS agent (Berman and Wolski 1997)

The Condor project (Raman et al. 1998; Thain et al. 2003) has been developed at the University of Wisconsin. A major functionality of Condor is to harness computing resources from participating machines for compute-intensive tasks. Figure 2-20 demonstrates the Condor kernel. A task requested by the user is analyzed by the problem solver that interprets the task into an internal representation such as a directed acyclic graph. This internal representation is passed to an agent. Agents and resources advertise themselves through the Matchmaker. When identifying a pair of agent and resource as compatible, the Matchmaker informs both parties of the pair. The agent is then responsible to contact the resource to ensure that the match is valid. After that, the agent starts a process in the Shadow that provides information to execute a task. The resource also initiates a process at Sandbox that creates a safe environment to run the task. Condor can make check points for certain types of tasks so that they can be recovered from the checkpoint file in case of failure. Checkpoints also allow a task to migrate to another machine. Another enabling mechanism for task migration is remote system calls. Remote system calls preserve the machine environment from where a task is submitted on a remote machine thus the task can be migrated during execution.

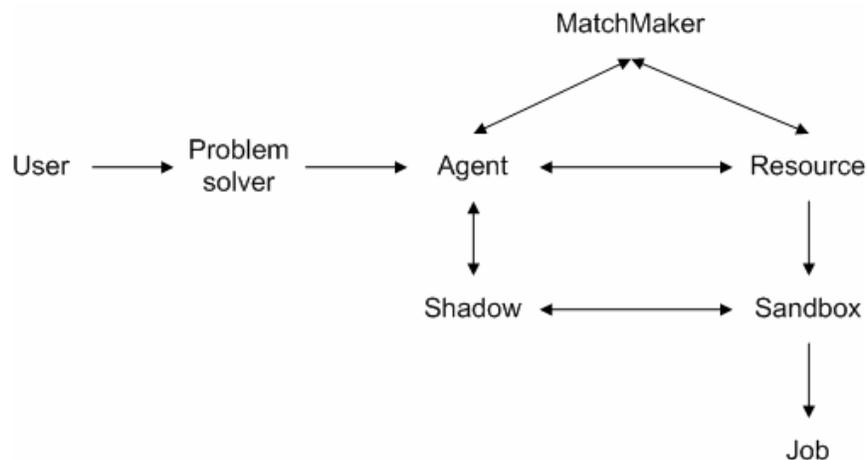


Figure 2-20. Condor kernel (Thain et al. 2003)

Scheduling in Condor is carried around the central Matchmaker. As stated before, agents and resources advertise their characteristics and requirements to the Matchmaker in the form of ClassAds (Raman et al. 1998). ClassAds is a semi-structured data model for resource description. Two sample ClassAds are shown in Table 2-7. After being informed by the Matchmaker, an agent and a resource start communicating for executing a task. When a task is submitted to the resource, it is up to the resource to decide when to initiate the task. In Condor-G (Frey et al. 2002), an adaptation of Condor for Globus, a scheduling strategy called “planning around scheduling” is deployed. Here “planning” refers to acquisition of resources by an agent and “scheduling” means management of a resource. This strategy requires that remote schedulers post their timetable or other scheduling information to the Matchmaker. Given such information, improved decisions regarding when and where to submit a task can be achieved. On the other hand, an opposite strategy, “scheduling around a plan”, can be applied as well. As Condor schedules parallel tasks on compute clusters (Wright 2001), until notified otherwise, the agent will assume that it has full control over the resource after contacting with the resource. Thus, after obtaining various resources by planning, the agent can set a schedule to execute tasks at these resources.

Table 2-7. Sample ClassAds (Raman et al. 1998)

A classAd for a machine	A classAd for a job
<pre>[Type = "Machine"; Activity = "Idle"; DayTime = 36107 // current time in seconds since midnight KeyboardIdle = 1432; // seconds Disk = 323496; // kbytes Memory = 64; // megabytes State = "Unclaimed"; LoadAvg = 0.042969; Mips = 104; Arch = "INTEL";</pre>	<pre>[Type = "Job"; QDate = 886799469; // Submit time secs. past 1/1/1970 CompletionDate = 0; Owner = "raman"; Cmd = "run_sim"; WantRemoteSyscalls = 1; WantCheckpoint = 1; lwd = "/usr/raman/sim2"; Args = "-Q 17 3200 10";</pre>

A classAd for a machine	A classAd for a job
<pre>OpSys = "SOLARIS251"; KFlops = 21893; Name = "leonardo.cs.wisc.edu"; ResearchGroup = { "raman", "miron", "solomon", "jbasney" }; Friends = { "tannenba", "wright" }; Untrusted = { "rival", "riffraff" }; Rank = member(other.Owner, ResearchGroup) * 10 + member(other.Owner, Friends); Constraint = !member(other.Owner, Untrusted) && Rank >= 10 ? true: Rank > 0 ? LoadAvg<0.3 && KeyboardIdle>15*60 : DayTime < 8*60*60 DayTime > 18*60*60;]</pre>	<pre>Memory = 31; Rank = KFlops/1E3 + other.Memory/32; Constraint = other.Type == "Machine" && Arch == "INTEL" && OpSys == "SOLARIS251" && Disk >= 10000 && other.Memory >= self.Memory;]</pre>

2.4.2. Geospatial Applications in Grids

Several research projects have applied grids to geospatial data processing. Hawick et al. (2003) proposed a framework that is focused on processing large-scale geographic data in the grid infrastructure. This framework uses Java and CORBA to build a middleware that glues services to access and manage multiple data sets. Based on the framework, a few applications have been developed (e. g., distributed processing of geospatial imagery, rainfall analysis and prediction). A sample scenario of a grid-enabled GIS application is provided by Shi et al. (2002), in which geographical models for a hypothetical watershed management project are hosted in an institute and operated on data sets maintained by different parties via grid services. Wang et al. (2002) implemented a grid-enabled teleimmersive spatial decision support system that utilizes computational grids through Grid-in-a-Box (GiB) testbed at National Computational Science Alliance to provide decision makers high-quality visualization information on desktop GIS. A quadtree-based domain decomposition and a static task scheduling algorithm are devised and evaluated for scalability by Wang and Armstrong (2003) in a computational grid. The authors

claim that for a uniform random distribution of datasets the scheduling algorithm scales well and the speedup is increased as additional resources are used.

3. Geoprocessing Optimization in Grids

3.1. Challenges

As GIS technology is being widely applied in numerous disciplines as a decision support tool, the demand for geoprocessing operations with large volumes of data increases. This makes centralized computing paradigm less efficient for geoprocessing. On the other hand, high-speed networks are paving the way for sharing geographically dispersed computing resources. With the help of fast network connections, harnessing distributed resources for heavy computation jobs becomes more feasible and attractive. To this end, grid technology is seen as an ideal platform to carry out data- and/or compute-intensive geoprocessing.

Although grid computing provides components to build distributed platforms for resource sharing, one major concern, i.e., performance, in geoprocessing, needs to be addressed. Not only is there a lack of application-level optimization services in grids, performance factors of grids also pose new challenges to optimizing queries. These factors include:

- Resource multiplicity. As various parties join in grids, it is likely that a certain resource, such as a data set, may be replicated and available at multiple hosts with different performances. So query optimizers should choose resources with minimum costs. In many distributed systems, optimal execution plans are determined through checking possible execution plans iteratively. This is a valid approach for cases where the search space is limited. However, as number of candidate resources increases, the search space grows; this is often the case in grid systems. A large search space leads to complexity in the exhaustive iteration approach and makes it inefficient and less appealing for grid-based query processing.

- Parallel execution. Considering the distribution and multiplicity of resources in grid systems, it is possible to parallelize query execution over different hosts to improve performance. A challenge in doing so is that attempting to check all possible parallelisms in a query would result in a much larger search space since each sequential candidate execution plan may be transformed into multiple parallel plans. This will increase the optimization complexity.
- Data transmission. Many grids consist of sites connected via wide-area networks or Internet in which network transmission is not reliable and delay may be significant and dominant. Such transmission delay or failure may significantly impact performances of data-intensive applications where large amounts of data may need to be transferred between sites. Thus unlike conventional tightly-coupled distributed systems, transmission becomes a major factor in optimizing grid-based query processing and needs to be taken into account by query optimizers.

These performance factors distinguish grid-based optimization from other optimization mechanisms employed in many other distributed systems and point to the major factors upon which optimizers for grids should be based. GOG is proposed to improve geoprocessing performance by addressing these performance factors.

3.2. Assumptions in GOG

Given that relational database management systems are widely used in grids, GOG assumes relational databases with interfaces to grid middleware. Therefore, GOG takes a query requesting a set of operations over specified relations as input. Of relational operations (e.g., selection, join

and projection), equi-join, which is a join that retrieves records with matching values in join fields, often contributes significantly to query execution cost and therefore is chosen to be one of the two types of operations considered in GOG. Furthermore, since a multi-relation equi-join (an equi-join that connects multiple relations) can be transformed into several two-way equi-joins (an equi-join that integrates two relations), GOG assumes two-way equi-joins in its input.

Besides two-way equi-join, GOG also supports geoprocessing operations (discussed in detail in Chapter 5). For the two types of geoprocessing, i.e., raster- and vector-based geoprocessing, they have pros and cons for different applications:

- Raster-based geoprocessing is primarily used in remote sensing and related fields. Some procedures are more efficient when implemented for raster data, such as overlay. Outcome of raster-based geoprocessing may not be precise enough in many situations, especially for cartographical applications.
- Vector-based geoprocessing is widely used in areas such as transportation, civil and environmental engineering. Unlike raster-based geoprocessing, vector-based geoprocessing can yield accurate results but may be more expensive. In many applications, raster data are used as a backdrop or context for result of vector-based operations. For example, buffers built around rivers and highways are displayed in the background of a satellite image classified into different areas (e.g., forest, residential area and open ground) so that emergency response personnel can make a better arrangement for disaster relief.

In addition, many latest relational database management system products (e.g., Oracle, IBM DB2, and PostgreSQL) are able to store vector data and support vector-based geoprocessing. Considering its numerous applications and wide support in DBMS, GOG is focused on optimizing vector-based geoprocessing. Optimization on raster-based geoprocessing is discussed in Section 6.3 as a topic for future research.

Since the focus of this research is optimization and there exist several query parsers that transform an SQL query into internal data structures (e.g., query tree or query graph), it is not necessary to build another query parser in GOG. Instead, GOG assumes that the query is represented in a tree structure called Abstract Query Tree (AQT). Each leaf node in AQT is either a geoprocessing operation or an equi-join with two base relations and an internal node is an operation on the results of leaf nodes. Sample AQT can be found in Figure 4-2.

It is assumed in GOG that operations and data sets in submitted AQT are in correct form and order. GOG does not perform syntactic or logical checking on AQT and processes relations in a query in the order specified by the user. Although re-ordering relations may yield additional execution plans, some with better performance, domain-specific operations may become expensive, or nonfunctional. Consider a query to find annual discharged amounts of a given pollutant species from factories inside a buffer (Query A in Figure 3-1). A spatial operation, “CONTAINS”, is applied to locate factories inside a given buffer and a join between the result of the “CONTAINS” operation and Relation *Discharge* is used to find the discharged amount of a pollutant species. In this case, Relations *Buffer* and *Factory* have a spatial field that can be related via the “CONTAINS” operation and *Factory* and *Discharge* have a common field,

factory ID, which relates factory records with corresponding records of discharged amount, but Relation *Discharge* does not have any spatial data. If the order of relations is changed as shown in Query B (Figure 3-1), the “CONTAINS” operation can not be conducted due to the lack of spatial information in *Discharge*. One way to process the query in the new order is to perform a Cartesian join between *Buffer* and *Discharge* to obtain all combinations of records from the two relations, followed by a “CONTAINS” operation (Query C in Figure 3-1). However, compared with the size of the intermediate result from the “CONTAINS” operation in Query A (i.e., a subset of Relation *Factory*), the result of the Cartesian join in Query C is larger and may take a significant time to transfer and process. Thus to avoid potential extra cost or failure in domain-specific operations the order of relations in AQT is kept intact.

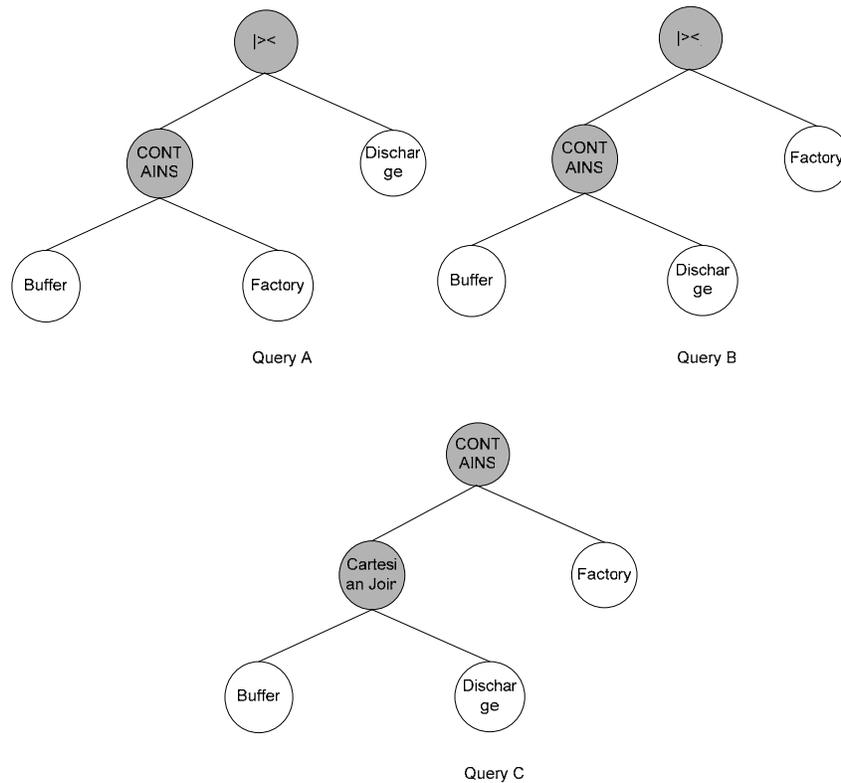


Figure 3-1. Orders of relations in AQT

3.3. Optimization Strategy

The optimization strategy in GOG is illustrated in Figure 3-2 with two major modules: resource selection and parallelism processing.

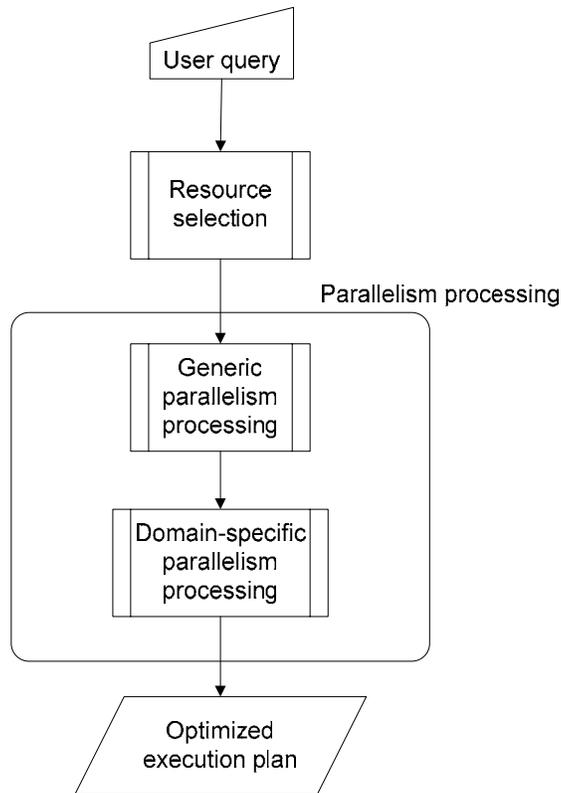


Figure 3-2. The proposed strategy to optimize computation in grids

Resource Selection

Optimization in databases can be seen as searching for an optimal execution plan in a search space composed by candidate plans (Steinbrunn et al. 1997). An exhaustive search throughout the search space can yield a plan that has the least cost, i.e., the best quality of all candidate plans. This is a valid and effective technique when the number of candidate plans is not large. However, as the number of hosts in a system grows, the number of candidate plans increases and

checking all plans becomes prohibitive. Furthermore, computing resources in grids are dynamic and the status of participating hosts changes over time, which makes searching all possible candidate impractical. Thus it is necessary to have a tradeoff between optimization cost and performance of optimized execution plans so that extra optimization costs can be avoided while performance of execution plans is maintained above a certain level.

The objective of optimization strategy in GOG is to improve geoprocessing performance with reasonable costs. GOG uses a resource selection process to limit the search space which would result in less optimization cost. The resource selection is conducted according to a ranking function which is discussed in Section 3.6. Resource providers are ranked by their costs for a specific operation and the providers with the highest rank are selected for carrying the operation. Resource selection helps GOG reduce optimization cost without losing potential superior computing resources.

Parallelism

GOG uses parallelism as an important means to speed up geoprocessing operations. Flynn and Rudd (1996) proposed four parallel architectures: single instruction single data stream (SISD), single instruction multiple data stream (SIMD), multiple instruction single data (MISD), and multiple instruction multiple data stream (MIMD). Based on Flynn and Rudd's work, Liu and Karimi (2004) suggest a classification on parallelism in grids according to the relationship between operation and input data (Figure 3-3): single operation single data (SOSD), single operation replicated data (SORD), multiple operations single data (MOSD), and multiple

operations multiple data (MOMD). The round boxes at the bottom are the corresponding implementations. Sample parallelism is illustrated in Figure 3-4.

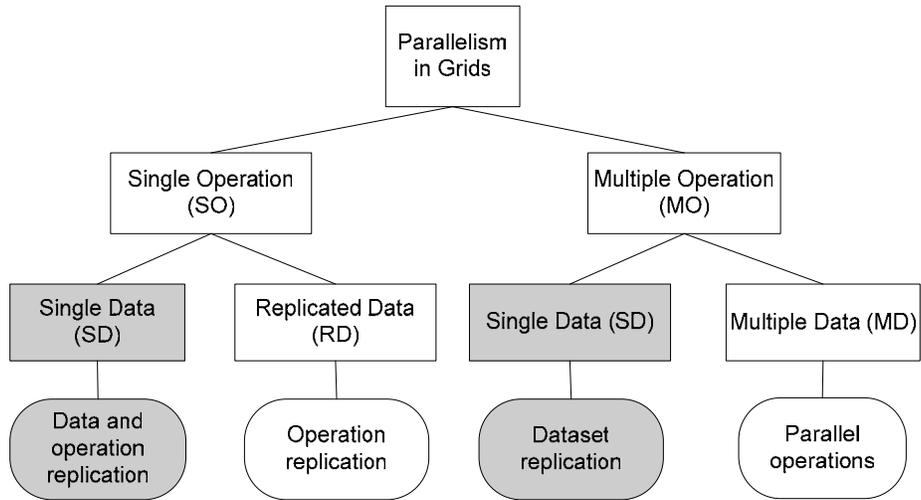


Figure 3-3. Classification of parallelism in grids

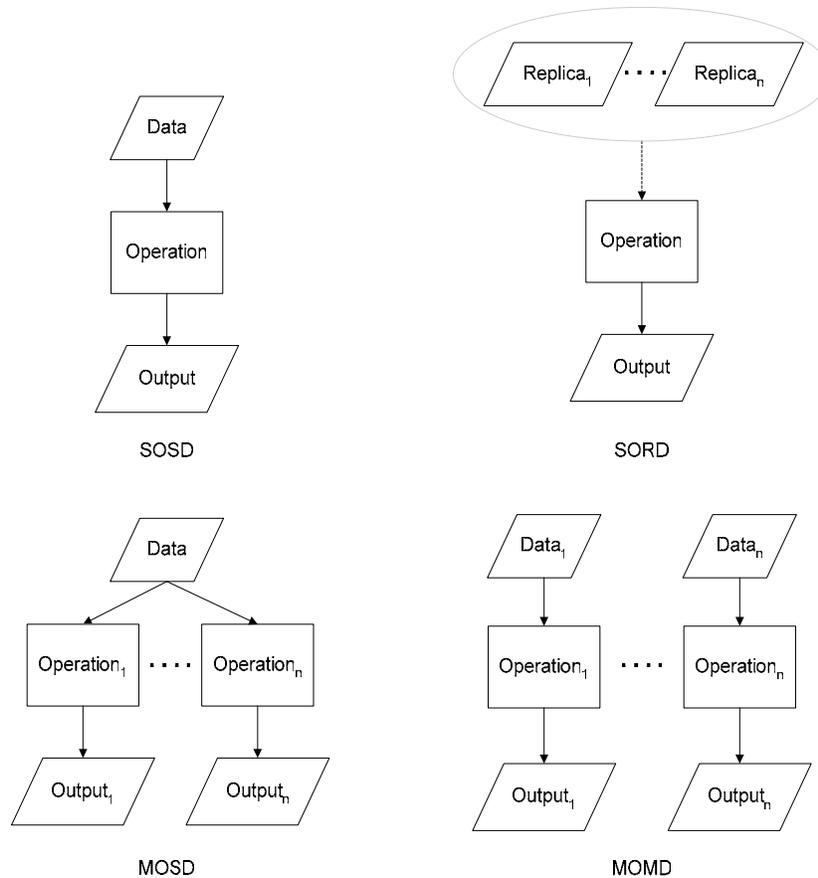


Figure 3-4. Sample parallelism in grids

As shown in Figure 3-3, implementing parallelism when there is a single data set (i.e., SOSD and MOSD) involves replicating that data set in one or more places. With large volumes of databases in grid systems (giga bytes or even tera bytes) and hosts connected via wide area networks, data replication could introduce much overhead that may outweigh the gain of parallelism. Thus in GOG these types of parallelisms are not considered and are presented as shaded rectangles in Figure 3-3. Since MOMD can be detected by checking data dependency between operations (without any domain-specific knowledge about these operations), it is also called generic parallelism. A module in GOG is devoted to processing generic parallelism for applications from different domains. On the other hand, SORD requires analyzing individual operations by using

domain-specific knowledge and thus is called domain-specific parallelism. The design of GOG allows modules from different application domains to be plugged in as a part of the optimization mechanism. A module to process parallelism in geoprocessing operation is built into GOG.

3.4. Architecture of Geoprocessing Optimization in Grids

GOG has two core modules, resource selection and parallelism processing, that would be executed in two separate phases utilizing four auxiliary services which provide both static and dynamic information about run-time computing environments. The architecture of GOG is shown in Figure 3-5.

The resource selection module prunes a set of hosts that maintain replicas for a base relation and selects one according to the ranking function. The selection process limits the search space so that exhaustive search is avoided. Based on the selected hosts, an optimized query execution plan will be built in the parallelism processing phase. The parallelism processing module checks data dependencies among operations involved in a query and recognizes parallelism, if any, in the execution plan. A service called “Geoprocessing Category Service” scans the query for geoprocessing and generate, if possible, parallel sub-plans specific for these operations. The output of the parallelism processing phase is an optimized query execution plan. Generated plans based on parallelization may be sub-optimal but are expected to perform better than randomly generated or statically optimized plans. The test of this assertion will be discussed in the next chapter. Optimized execution plans will be submitted to grid systems for execution.

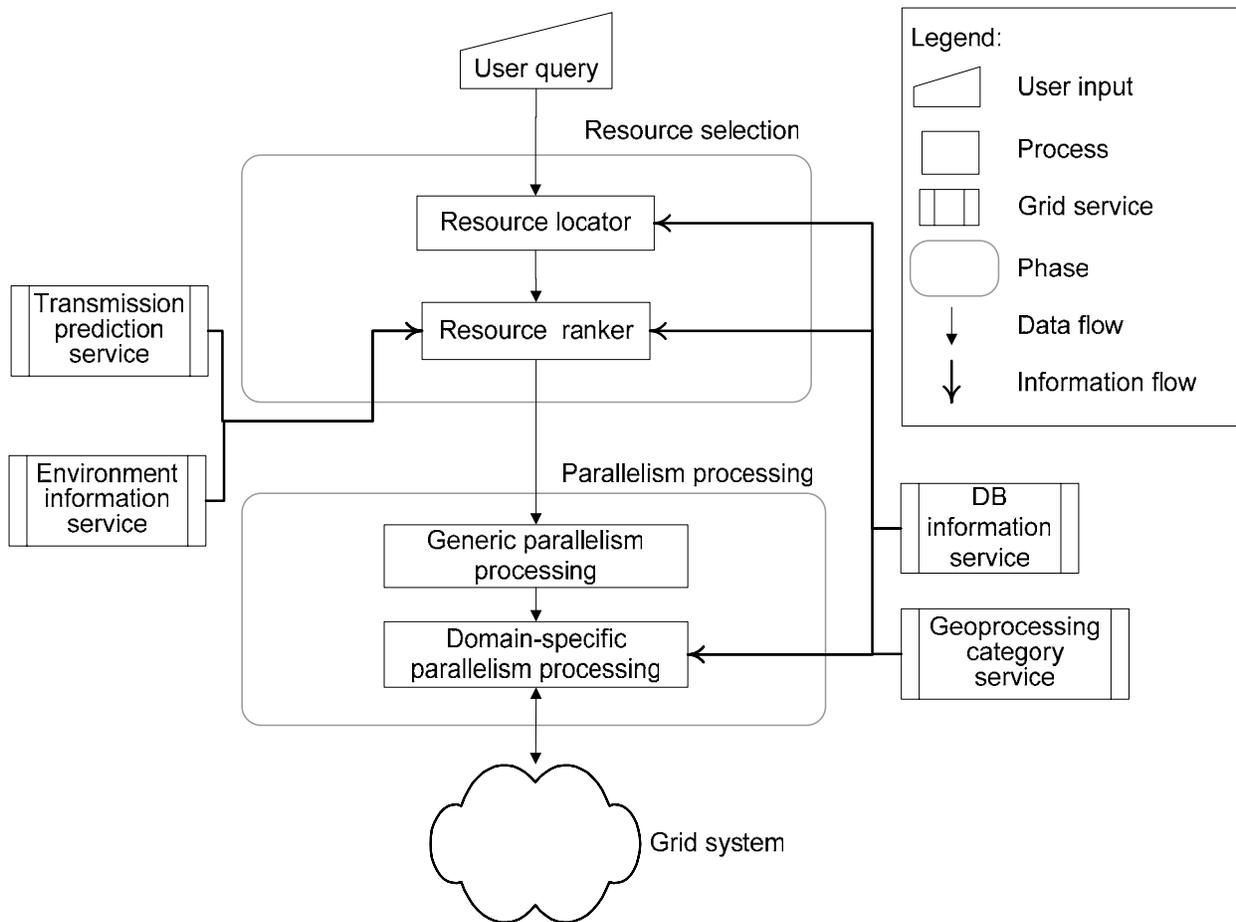


Figure 3-5. Architecture of GOG

3.5. Auxiliary Services

In order to provide run-time information for resource selection and parallelism processing modules, GOG includes the following four auxiliary services: Environment Information Service (EIS), Database Information Service (DIS), Transmission Prediction Service (TPS), and Geoprocessing Category Service (GCS). Given the heterogeneity in grid systems, the interfaces of the four services (i.e., the parameters they should provide to the core optimization modules and the data types of the parameters) are specified but no platform-specific implementation is required.

Environment Information Service (EIS)

EIS is responsible for providing both static and dynamic information about a given host, including:

- System workload: in a percentage rate, 0 means idle and 1 means that the host has no processing power left.
- Millions of Instructions Per Second (MIPS).
- RAM amount (MByte).

Database Information Service (DIS)

DIS manages a catalog of existing replicas of base relations in grids and retrieves them back to a client upon request. For a given relation, DIS can provide:

- Host: the host that maintains a replica of the relation.
- Relation size (MB).
- Number of records.
- Minimum and maximum ID.
- Number of disk blocks.

In order to estimate the join size, DIS also provides statistics about fields that may be used in joining relations:

- Field name.
- Number of distinct value of the field in the relation.
- Index height if there is any index built on the field.

Transmission Prediction Service (TPS)

For a relation in a given query, TPS is responsible for estimating a candidate host's transmission performance with respect to other hosts involved in that query. As stated before, transmission

time is a significant part of overall response time in grid-based query processing and should be taken into account when selecting hosts. One approach to estimate transmission performance is to calculate mean transmission latencies between hosts from historical data. A problem with this approach is that mean values can be significantly affected by the distribution of data: outliers with arbitrarily high or low values can greatly impact mean values, i.e., making mean values less reliable as an indication of overall transmission performance. Taking distribution factor into account, an index, Transmission Latency Reputation (TLR), is introduced. Rather than presenting transmission latencies, TLR represents the “reputation” of transmission latency for a host within a time period with respect to other hosts that might be involved in the same query. The larger the TLR of a host, the more inferior its transmission capacity with respect to other hosts.

Suppose that a query Q is to be executed during a time period t involves relations $R_1, R_2, \dots, R_i, \dots, R_N$ and relation R_i has M_i replicas that are located at hosts $H_{i1}, H_{i2}, \dots, H_{ij}, \dots, H_{iM_i}$, respectively. The TLR of Host H_{ij} during t can be computed as a weighted mean:

$$TLR_{ij} = \frac{\sum_{k=1}^N \sum_{l=1}^{M_k} s_{(ij,kl)}^2 * TL_{(ij,kl)}}{\sum_{k=1}^N \sum_{l=1}^{M_k} s_{(ij,kl)}^2} \quad (k \neq i) \quad 3.1$$

where $TL_{(ij,kl)}$ is the mean transmission latency between H_{ij} and H_{kl} during t and $s_{(ij,kl)}^2$ is the variance of transmission latencies between H_{ij} and H_{kl} . If H_{kl} and H_{ij} both have only a relation R_i that is involved in Q , H_{kl} should not be taken in computing TLR_{ij} . The reason for excluding such hosts from calculation is that there will be no transmission between that host and the host used in the computation while executing the query. They compete to be the provider of R_i . By

introducing weight, the mean transmission latency of a host is adjusted. The more outliers a host has in its latency data and the larger values of outliers, the larger will be the value of TLR.

Geoprocessing Category Service (GCS)

To tune the optimization process for GIS applications, a catalog of parallel implementations of various geoprocessing is required. This catalog is maintained by GCS. Due to the fact that many parallel algorithms require partitioning in input data sets, partitioning policies are also necessary. Upon detecting a geoprocessing operation in a query, the optimization process calls GCS to check whether it is possible to parallelize the operation by utilizing available resources. GCS will compose a parallel execution plan if enough resources exist to permit parallelism. Based on the resources used in the parallel execution, the running time of the execution will be estimated and sent back to the optimization process together with specifications of the execution. Comparing the estimated running time of the parallel execution with the one of the non-parallel execution, the optimization process will determine whether to use a parallel or non-parallel execution.

3.6. Resource Selection

In the resource selection module (Figure 3-6), an AQT is recursively traversed in order to locate candidate hosts for each base relation in that AQT; this process is performed in a sub-module called resource locator. For each base relation, the resource locator contacts DIS which will return a list of candidate hosts that have the requested base relation. The returned list is passed to a sub-module called resource ranker. The resource ranker checks with DIS, TPS and EIS, respectively, to obtain both static and dynamic statistics about the candidate hosts. Based on the collected statistics, the resource ranker uses a ranking function to compute the rank of a given

candidate Host H_{ij} for a base Relation R_i . The ranking function is a linear combination of weighted and normalized values of five factors:

$$\begin{aligned}
 rank_{ij} = & \frac{(mips_{ij} - \min(mips)) \times w_{mips}}{\max(mips) - \min(mips)} + \frac{(ram_{ij} - \min(ram)) \times w_{ram}}{\max(ram) - \min(ram)} \\
 & + \frac{(count_{ij} - \min(count)) \times w_{count}}{\max(count) - \min(count)} \\
 & + \frac{w_{wk}}{(wk_{ij} - \min(wk)) / (\max(wk) - \min(wk)) + 1} \\
 & + \frac{w_{TLR}}{(TLR_{ij} - \min(TLR)) / (\max(TLR) - \min(TLR)) + 1}
 \end{aligned} \tag{3.2}$$

where:

- $mips_{ij}$: MIPS of H_{ij}
- w_{mips} : weight of MIPS
- ram_{ij} : currently available RAM amount at H_{ij} at the time period t (MB)
- w_{ram} : weight of RAM
- $count_{ij}$: number of relations that are involved in a query and maintained by H_{ij}
- w_{count} : weight of count
- wk_{ij} : current workload of H_{ij} (0 means idle and 1 means that H_{ij} is fully utilized)
- w_{wk} : weight of workload

The ranking function is adopted from the cost function proposed by Mackert and Lohman (1986) in which the total cost is split into components representing costs in CPU, I/O operation and data transmission, respectively. Before being multiplied by a weight, each parameter on the right side of Equation (3.2) is standardized as follows:

$$\frac{current_value - min_value}{max_value - min_value} \quad 3.3$$

Normalization makes combining factors together, each measured in a different unit, to represent overall capacity of a host possible. The weighted normalized values are summed up as the rank of a candidate host for the given base relation.

A new factor introduced into this ranking function is *count*, referring to the number of base relations that Host H_{ij} has in a given query. Taking this factor into consideration increases the chance of avoiding data transmission. If a host maintains multiple relations for a query, it should be considered as a good candidate to carry part of the query since joins of these relations can be performed locally.

In current GOG implementation the weights in Equation 3.2 are set to 1 since GOG is intended to be a generic query optimizer giving no preference for specific factors. For applications where certain factors become dominant or subordinate their weights can be adjusted accordingly. For instance, a relatively large value may be assigned to w_{mips} for an application with compute-intensive tasks. One method of determining weights is to use genetic algorithms (Goldberg 1989; Holland 1992) that employ evolution theory to find optimal solutions to certain problems. Genetic algorithms usually start with a population of initial solutions that have different values according to an objective function such as a cost function for query processing. A new generation of solution population is produced by applying a set of operations (e.g., reproduction and mutation) on the initial population. Similar to natural selection in evolution, components in solutions (such as a weight in Equation 3.2) that lead to desirable values of the objective function

are preserved and strengthened over generations. Thus solutions after a certain number of generations should be superior over their ancestors.

A scenario of using a genetic algorithm to determine values of the weights in Equation 3.2 is as follows. A solution is defined as an array of values, $\{w_j\}$ ($j \leq 5$), where w_j is the value assigned to the j^{th} weight in Equation 3.2. A pool of such n arrays, $\{w_{i,j}\}$ ($i \leq n$), where $w_{i,j}$ is the value of the j^{th} weight in Solution i , are built as the first generation of solution population. Based on these n arrays of weights, GOG generates execution plans of queries and compares them with optimal plans obtained from an exhaustive search algorithm (such as the one described in Section 4.2). The appropriateness of each solution is measured as the percentage of the difference between the execution costs of the two plans over the sum of the differences between the execution costs of all solutions and the optimal plan. The next generation of solution population is computed by using three operations, i.e., reproduction, crossover and mutation. In the reproduction operation, the value of a weight in the next generation is calculated as the weighted sum of corresponding weights from most appropriate solutions in the current generation. Contributions by each current weight to the sum are determined by their appropriateness. For example, Equations 3.4 and 3.5 can be applied in such a reproduction process:

$$w'_{i,j} = \sum_{k=1}^l (\alpha_k * w_{k,j}) \quad (l \leq n) \quad 3.4$$

$$\alpha_k = \frac{c_k - c_{min}}{\sum_{m=1}^l (c_m - c_{min})} \quad 3.5$$

where $w'_{i,j}$ is the j^{th} weight of Solution i in the next generation, l is the number of most appropriate solutions chosen for reproduction, α_k is the appropriateness of Solution k , c_k is the

execution cost of a plan generated using Solution k , and c_{min} is the execution cost of the optimal plan. In the crossover operation, new solutions are composed by coupling two randomly selected solutions in current population, for instance, $\{w_{1,j}\}$ and $\{w_{3,j}\}$ ($j \leq 5$). A random number, $t \in [1, n-1]$, is generated to divide each of the two arrays into two sub-arrays, e.g., $\{w_{1,j}\}$ ($1 \leq j \leq t$) and $\{w_{1,j}\}$ ($t < j \leq n$) for the first solution. Two new solutions are then composed by switching sub-arrays, e.g., $\{w_{1,j}\}$ and $\{w_{3,j}\}$ ($1 \leq j \leq t$), in the two solutions. New solutions can also be obtained by using the mutation operation that can be implemented by altering values of weights with random numbers. The solutions computed by the three operations form a new generation of solution population which is used to generate new execution plans. Such “optimization-generation-optimization” cycle is repeated until a preset criterion, e.g., a threshold for cost difference between generated execution plans and the optimal plan, is met.

Machine learning or pattern recognition algorithms (Duda et al. 2000) can also be applied to determine values of weights in Equation 3.2. Considering that application requirements as well as the status of running environments significantly impact the choice of weight values, research is needed to optimize weight values for each application.

After computing ranks of all candidate hosts for a base relation, the host with the highest rank is chosen as the provider for the base relation. As stated in the beginning of this chapter, the factors in the ranking function are those that will greatly impact the performance of using a base relation in executing a query. Thus it can be argued that ranking reflects the fitness of a host as a candidate for a base relation in a given query. The higher rank a host receives, the higher chance that it is a better choice to be used as the provider of the relation. Although the host with the

highest rank may not be the best candidate, it will have a better performance than a randomly selected host. Furthermore, compared to exhaustive optimization algorithms, the ranking function in GOG is easy to implement.

Selecting one host for one relation may eliminate potential parallel executions for a query since there will be limited choices available to the parallelism processing module. It can be argued, however, that due to the following reasons sacrificing parallelism for a smaller search space is advantageous, especially for certain types of queries.

1. Although including resources with lower ranks may yield more parallel executions, such executions tend to be less effective due to inferiority of these resources. Furthermore, if these resources are assigned with a significant amount of computation, they may delay or block the whole process.
2. Costs of composing and comparing parallel executions without resource selection can be high. As stated previously, there should be a tradeoff between optimization cost and performance of optimized execution plans. For queries targeted by GOG, i.e., data- and/or compute-intensive queries with multiple replicas, cost of considering all possible parallel execution plans could potentially be very high or prohibitive.

As shown in experiments presented in Chapters 4 and 5, by keeping a balance between optimization cost and query execution performance, GOG achieves better overall query processing performance with low cost.

When ranking hosts for all base relations for a query is completed, the selected hosts for relations are added to the AQT which is transformed into a Physical Query Tree (PQT). A PQT has

information about where to locate a base relation in a grid system but no parallel execution information. Parallel executions are checked in the parallelism processing module.

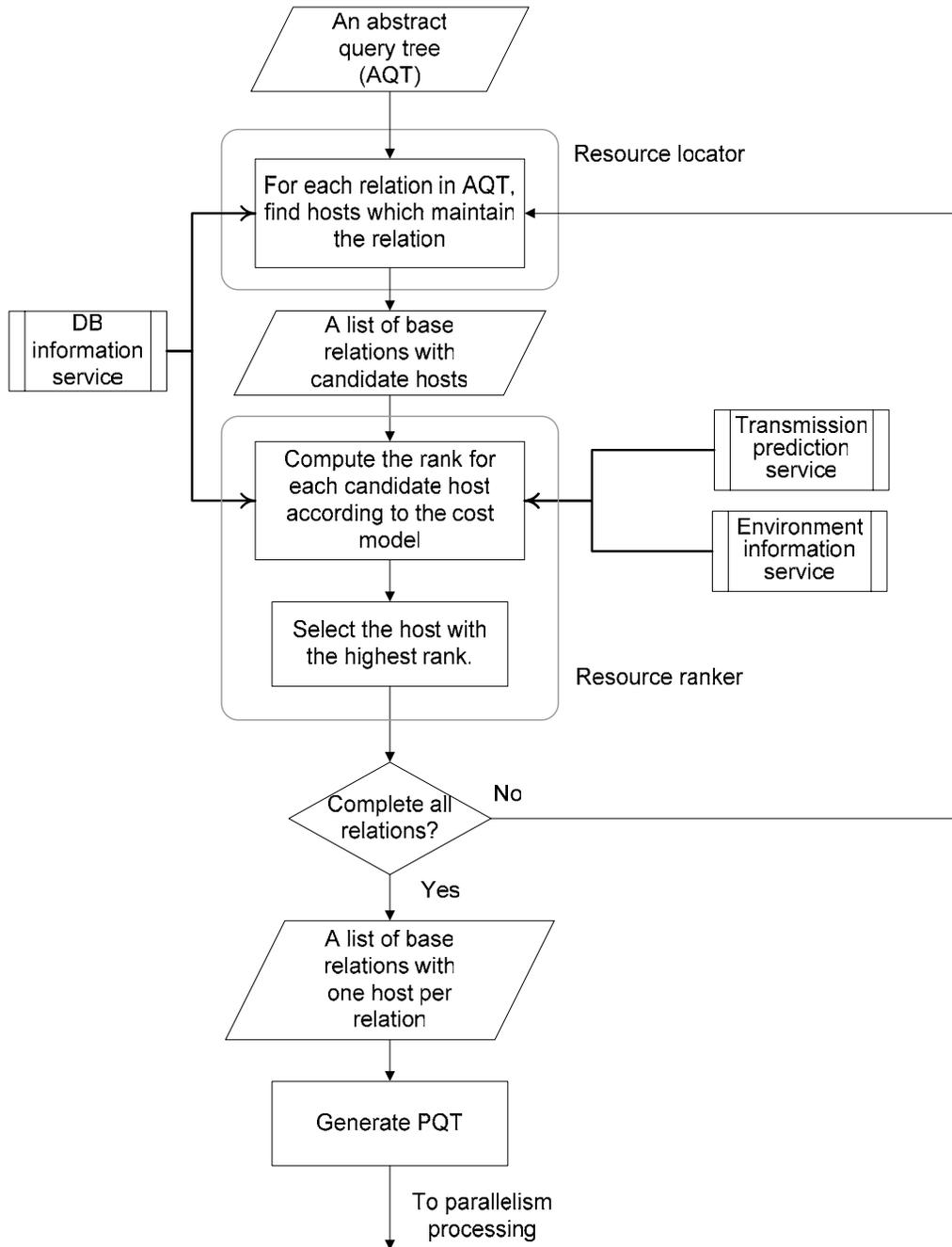


Figure 3-6. Workflow in the resource selection module

3.7. Parallelism Processing

The workflow in the parallelism processing module is shown in Figure 3-7.

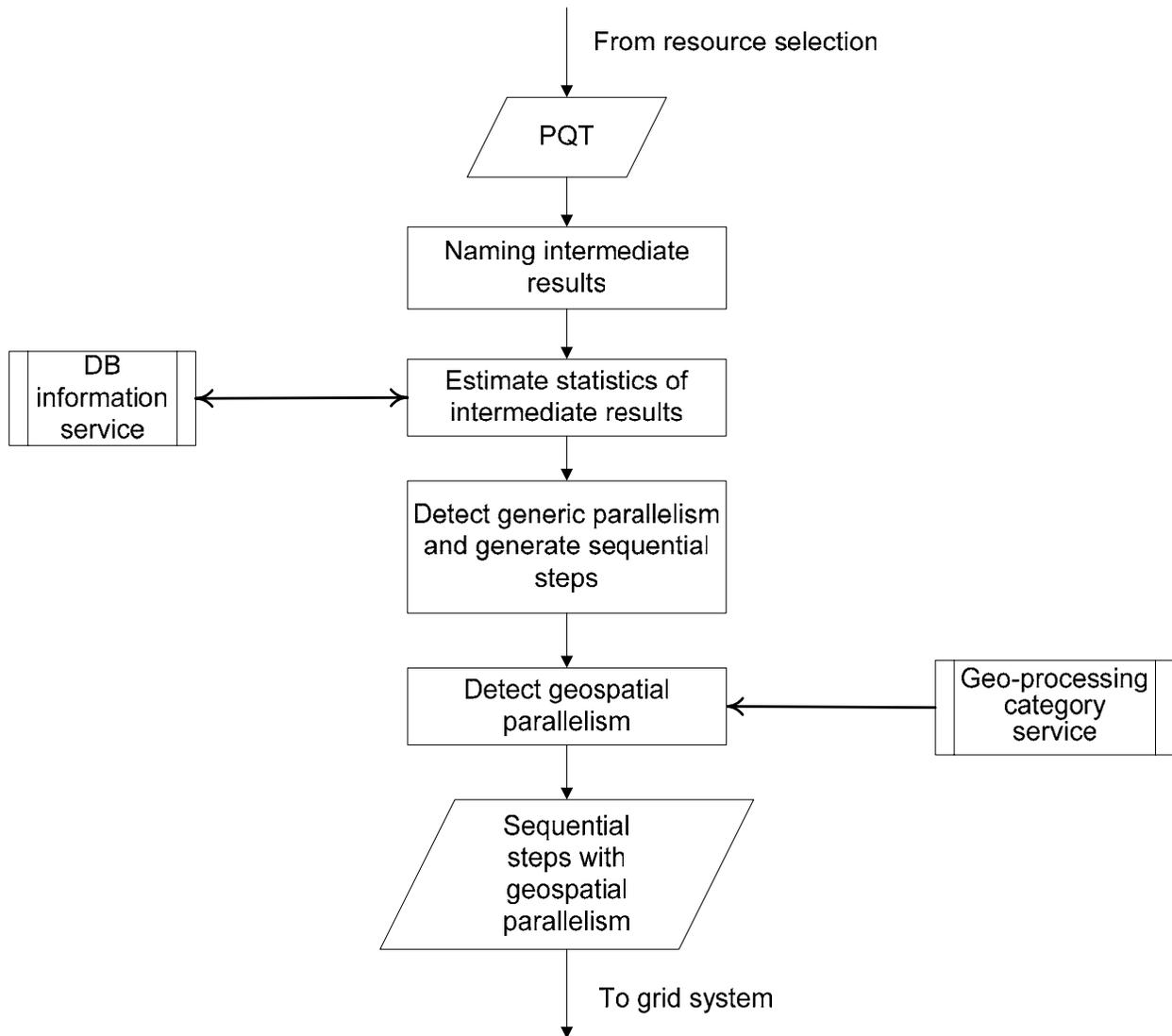


Figure 3-7. Workflow in the parallelism processing module

Intermediate Results

Before checking parallelism in a PQT generated by the resource selection module, the information about intermediate results (i.e., the result of a join or geoprocessing operation that is part of the input to another join or geoprocessing operation) needs to be known. This is due to two reasons:

- Determining data transfers between operations. Suppose Join J_2 uses the result of Join J_1 as an input relation. In this case, GOG needs to name the result of J_1 and record the location where the result is stored in order to determine if a data transfer is needed before conducting J_2 .
- Determining if a domain-specific parallel execution is superior over sequential execution. Although a parallel execution may reduce running time, it introduces overhead, e.g., data transfer. So whether to use a parallel or sequential execution depends on the comparison of their total costs. In order to estimate total costs, certain statistics on relations in a join, such as size and number of distinct values of the join field, should be known.

In GOG, an intermediate result of an operation (either a join or geoprocessing operation) is assumed to be stored in the host where the operation is conducted. GOG estimates statistics on the following parameters:

- Number of records. The function proposed by Silberschatz et al. (2002) to estimate join size is adopted in GOG:

$$\#_records = \min\left(\frac{n_r * n_s}{V(A, r)}, \frac{n_r * n_s}{V(A, s)}\right) \quad 3.6$$

where n_r and n_s are number of records in join relations r and s , respectively, $V(A,r)$ is the number of distinct values in the join field A of Relation r and $V(A,s)$ is the number of distinct values in the join field A of relation s .

- Size. The average record size of the intermediate result can be estimated as the average of average record sizes of relations r and s . Thus the size of the intermediate result can be calculated as:

$$size = \#_records * avg(avg(record_size_r), avg(record_size_s)) \quad 3.7$$

- Number of blocks. The number of blocks can be estimated as the average of the two join relations:

$$\#_block = avg(\#_block_r, \#_block_s) \quad 3.8$$

- Number of distinct values and index height of the join field. The number of distinct values and the index height of the join field are estimated as the largest value among the two join relations.

These statistics are temporarily added to the information repositories in DIS and will be removed once an execution plan for a query is determined.

Detect Generic Parallelism

In GOG, a “parallelism-inside-sequential-step” structure (Figure 3-8) is introduced to represent the output of optimization (i.e., an execution plan). In this structure, an execution plan is composed of a series of steps that are to be run sequentially. Each step in the plan, called a sequential step, includes a set of operations that is scheduled to run in parallel. Operations in one sequential step have to wait for the operations in previous steps to be completed so that all their

input data become available. If no parallelism is found in a query, for instance, a left-deep join, then a sequential step only has one operation (e.g., a join).

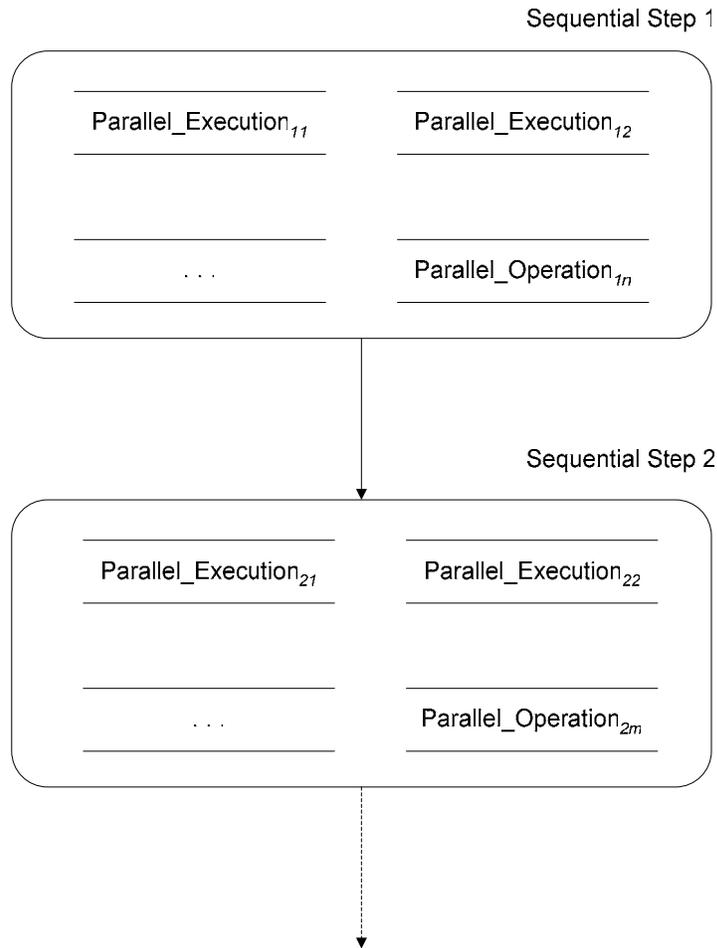


Figure 3-8. Structure of execution plan optimized by GOG.

The algorithm to detect generic parallelism is demonstrated in Figure 3-9. According to the definition of generic parallelism in Section 3.3, generic parallelism exists when two or more joins do not have data dependency. Thus it is inferred that generic parallelism can only be found between operations in the leaf nodes of a PQT since all operations in the internal nodes have data dependency on one or more leaf nodes. So the algorithm first collects all leaf nodes in a PQT

into Set N . For Node n in N , if it does not have data dependency with any other nodes in N , it is removed from the PQT and added to the current sequential step. This process is repeated until all nodes in the PQT are processed. The outcome is an ordered set of sequential steps.

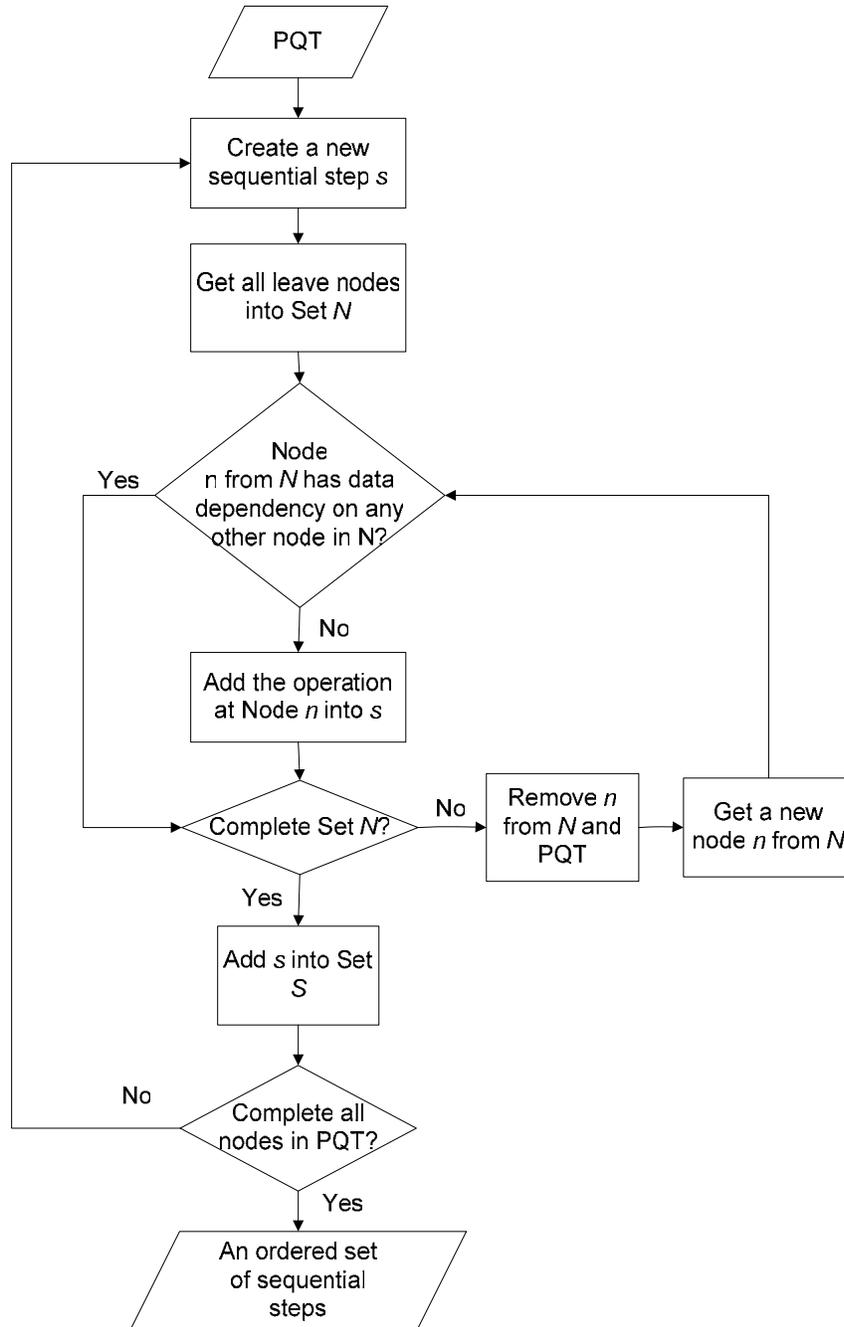


Figure 3-9. Algorithm to detect generic parallelism

Detect Domain-Specific Parallelism

As stated in Section 3.3, GOG will search for domain-specific parallelism in a query if a corresponding domain-specific category service is available. The logic in detecting domain-specific parallelism is as follows. If an operation in a step is a domain-specific operation (e.g., spatial join), the corresponding category service would be called to provide a parallel execution and the estimated cost for it. By comparing this cost with the cost of the non-parallel execution in the current step, the one with less cost will be chosen. This algorithm is shown in Figure 3-10.

GCS in current implementation of GOG can provide parallel execution plans for two spatial operations (“WITHIN_DISTANCE” and “CONTAINS”). This module is discussed in detail in Chapter 5.

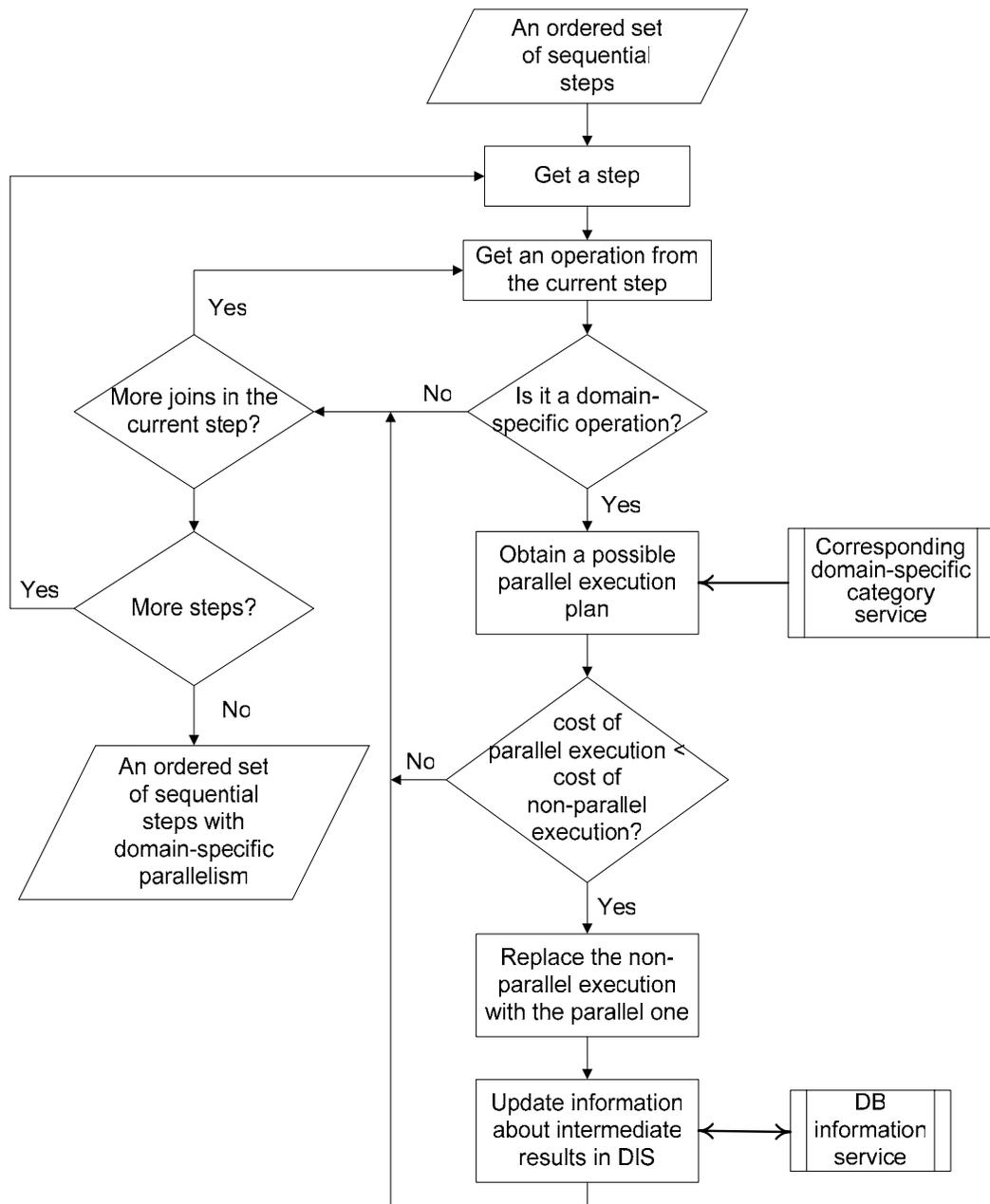


Figure 3-10. Algorithm to detect domain-specific parallelism

4. Experiment on Geoprocessing Optimization in Grids

4.1. Stages in Query Processing

Query processing can be divided into two stages (Figure 4-1): query optimization and query execution. Query optimization stage includes query parsing, optimization, code generation and other processes that help generate execution plans. In query execution stage, plans from query optimization stage are executed and results are sent back to the client. Correspondingly, overall Query Processing Time (QPT) is composed of two parts: time to generate query execution plans and time to run plans. These two parts are called Query Optimization Time (QOT) and Query Execution Time (QET), respectively. QPT is computed as the sum of QOT and QET.

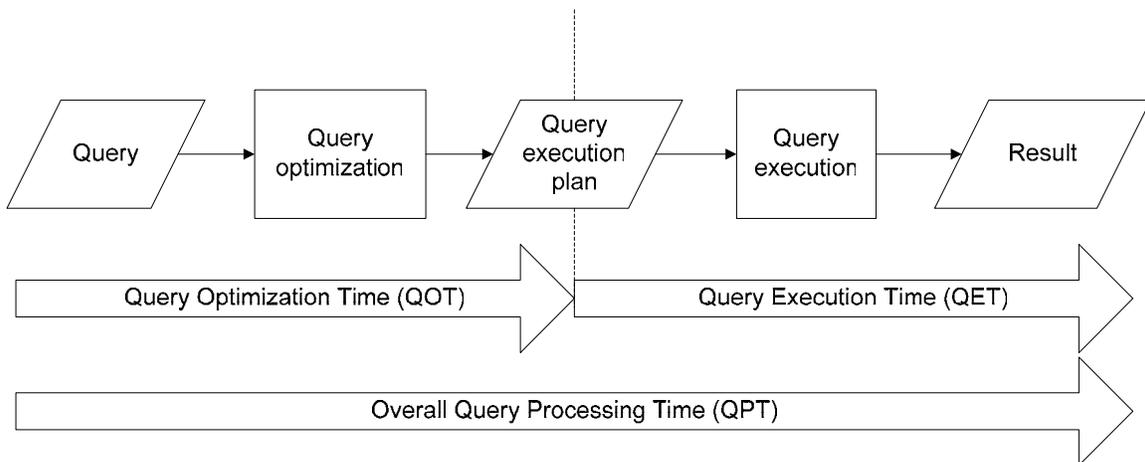


Figure 4-1. Stages in query processing

QPT reflects the appropriateness of optimization techniques that are designed to reduce overall system response time. A good optimization technique should not only take less QOT, it should also assure less QET. In this dissertation an experiment is conducted to examine the goodness of GOG by comparing GOG with two other optimizers, complete iteration optimizer (CIO) and randomized optimizer (RO). QOT is measured as running time of an optimizer to generate execution plans. QET is estimated as cost of a query execution plan which is discussed in next

section. QOT, QET and QPT of the three optimization techniques on four queries are recorded and results are discussed at the end of the chapter.

4.2. Comparing Optimization Techniques

GOG is compared and contrasted with two optimization techniques: CIO and RO. CIO is based on an exhaustive searching algorithm that finds an optimal execution plan among all candidate plans. It provides an optimal solution with additional cost. RO is a fast algorithm that randomly generates execution plans. Due to its randomness in building execution plans, it is believed that plans generated by RO represent the average quality of all candidate plans.

Complete Iteration Optimizer

The first step in CIO is to build the search space by generating all possible PQT. Each PQT is transformed into an ordered set of sequential steps by using the algorithm presented in Figure 3-9. Since operations in the same step run in parallel, the largest execution cost among all operations is the execution cost of this step. Because steps are executed sequentially, the execution cost of a set of sequential steps (i.e., an execution plan) is the sum of the costs of each step:

$$cost_{plan} = \sum_{i=1}^N \max(cost_{op_{i1}}, \dots, cost_{op_{ij}}, \dots) \quad 4.1$$

where $cost_{op_{ij}}$ is the execution cost of operation j in step i and N is number of steps in the execution plan.

Consider the scenario for carrying an operation at a host in a distributed system. Relations first have to be transferred to and made ready in the destination host, which involves transferring relations to the destination host (if the destination host is different than the host where a relation is stored) and reading relations from storage (usually hard drive) into memory. After all relations are ready in the destination host, the operation can start. Thus the execution cost of an operation op_{ij} can be estimated as a linear combination of the following components: cost to transfer each relation to the destination host, cost to read each relation, and cost to execute the operation.

$$cost_{op_{ij}} = \sum_{k=1}^{\#_relation} [tcost_k + w_{IO} * (\#_IO_k)] + \sum_{l=1}^{\#_join} w_{CPU} * (\#_insts_l) \quad 4.2$$

where w_{CPU} and w_{IO} are defined as Mackert and Lohman (1986) proposed in the R* system, as number of milliseconds per CPU instruction and the sum of the seek, latency and transfer times for a block of data, respectively. The transfer cost, $tcost_k$, of relation k is predicted as follows:

$$tcost_k = \frac{size_k}{b_{ho,hd}} = \frac{size_k}{avg(size_{sample_file} / tl_{ho,hd})} \quad 4.3$$

where $size_k$ is the size of Relation k , $b_{ho,hd}$ is the estimated bit rate from Host ho to hd , origin and destination, respectively. Equation 4.3 also shows that $b_{ho,hd}$ can be calculated from the historic transmission latency data as the mean of transmission latencies between ho and hd at the query submission day divided by the size of the sample file used in measuring transmission latency. Number of I/O operations is estimated as the size of a relation divided by the block size at a host. Without any knowledge about indexing in any base relation, number of CPU instructions is measured as the number of tuples in the Cartesian-product of two joined relations. Costs of all execution plans will be computed and the one with the smallest cost will be selected as the optimal execution plan.

To be consistent and comparable, running costs of execution plans generated by RO (discussed below) and GOG are also estimated by Equations 4.1 and 4.2.

Randomized Optimizer

Unlike CIO and GOG, RO randomly selects hosts for each base relation from all candidate hosts and no parallelism is carried. In randomly selecting hosts, RO uses a pseudorandom number (e.g., generated by *Random* class in Java library) which in turn employs a linear congruential random number generator (Knuth 1997). RO uses sequential steps to represent generated execution plans. Each step only has one operation and the order of steps is the same as the one in AQT.

4.3. Experiment Hypotheses

Four major hypotheses are set to test the appropriateness of GOG:

H_{01} : QOT of GOG is less than QOT of CIO.

H_{02} : QET of GOG is less than QET of RO.

H_{03} : QPT, i.e., the sum of QOT and QET, of GOG is less than QPT of RO.

H_{04} : QPT of GOG is less than QPT of CIO.

Hypothesis H_{01} tests if GOG reduces optimization cost in terms of QOT, i.e., GOG composes an execution plan in less time than the time it takes to determine an optimal execution plan. The optimal execution plan for a query is defined as the one with the least cost among all candidate plans in the search space, i.e., the one determined by CIO. Since GOG uses the ranking function to limit the search space, it is expected that GOG should run faster than CIO.

Hypothesis H_{02} checks performance of execution plans generated by GOG in terms of QET. Considering the randomness in selecting hosts, performance of execution plans composed by RO are chosen to represent the average case. Due to the tradeoff between optimization cost and performance of execution plans, discussed in Section 3.3, execution plans generated by GOG may not be optimal. However, with the resource selection and parallelism processing procedures, these plans should be better than the average case.

As discussed in Section 4.1, the overall performance of an optimizer in processing queries, including optimization cost and performance of generated query execution plans, can be tested by checking its QPT. GOG is expected to have less QPT than RO since it not only avoids extra optimization cost but also employs better resources as well as parallelism in query execution (Hypothesis H_{03}). Considering that GOG reduces search space at the cost of suboptimal performance of execution plans, it is also expected to outperform CIO in terms of QPT for queries whose optimization costs are expensive (Hypothesis H_{04}).

In addition to the four major hypotheses, effects of “count” and TLR factors in the ranking function are tested in this experiment. Among the five factors considered in the ranking function, CPU speed, host workload and RAM amount are widely used in cost models of different database management systems, such as IBM System R (Mackert and Lohman 1986), and proven to be valid and important in measuring query execution costs. In this research “count” and TLR factors are taken into account for realizing the impact of resource multiplicity and data transmission on grid-based query processing, as discussed in Chapter 3. The effect of these two

factors is tested by removing one or both from the ranking function and comparing the resultant performance differences (in terms of QPT) (see Section 4.5.4).

4.4. Experiment Design

The three optimization techniques, GOG, CIO, and RO, are implemented in Java Standard Edition (Version 1.4.2) and tested in a workstation configured with RedHat Linux (Release 9.0). The auxiliary services in GOG use MySQL (version 4.1.7) as a backend data repository.

4.4.1. Simulated Grid Environment

A grid environment is simulated based on the data from the PlanetLab (PlanetLab 2005). The simulated grid has 12 hosts that are distributed in North America, Europe and Asia. Workload and transmission latency data were collected at the 12 hosts several times daily over the period from May 2nd to June 23rd in 2004. Four databases are simulated with multiple relations (Table 4-1). Each database has several replicas distributed in the 12 hosts (Table 4-2). Since the information of hardware configurations of the 12 hosts is unavailable at the time the thesis was written, it is simulated as shown in Table 4-3. Considering that the three optimizers use the same configuration information, however, choices on values of configuration factors (e.g., RAM amount) will not affect the comparison result.

Table 4-1. Simulated databases and relations

Database	Relation	#_Records	Size (MB)
DB1	R1	550000	200.4
DB2	R2	100000	301
DB1	R3	3000000	1300
DB3	R4	50000	181.4

Database	Relation	#_Records	Size (MB)
DB4	R5	300000	241.6

Table 4-2. Simulated database replicas

Database	Replica Host
DB1	bu
DB1	cuhk
DB1	diku
DB1	ucla
DB2	duke
DB2	msu
DB2	rochester
DB2	ucl
DB3	ucla
DB3	umd
DB3	unipassau
DB4	bu
DB4	cuhk
DB4	unipassau
DB4	virginia
DB4	wide

Table 4-3. Simulated host configuration

Host	MIPS	RAM (MB)	Block Size (kb)
bu	654	512	512
cuhk	409	128	512
diku	409	256	512
duke	837	1024	512

Host	MIPS	RAM (MB)	Block Size (kb)
msu	613	512	512
rochester	654	1024	512
ucl	327	128	512
ucla	1674	1024	512
umd	1674	2048	512
unipassau	736	1024	512
virginia	837	1024	512
wide	613	512	512

4.4.2. Query Testing

Four queries represented in AQT (Figure 4-2) are used as inputs to the three optimization techniques. Considering that query structure may affect building parallel executions (e.g., balanced structures can lead to more number of parallel executions while unbalanced structures such as a left-deep tree leaves little opportunity for parallelism), different structures are used in representing the four queries to reduce the bias in selection over query structures. Similarly, each query is designed to have different relations so that the bias towards specific relations or hosts is reduced.

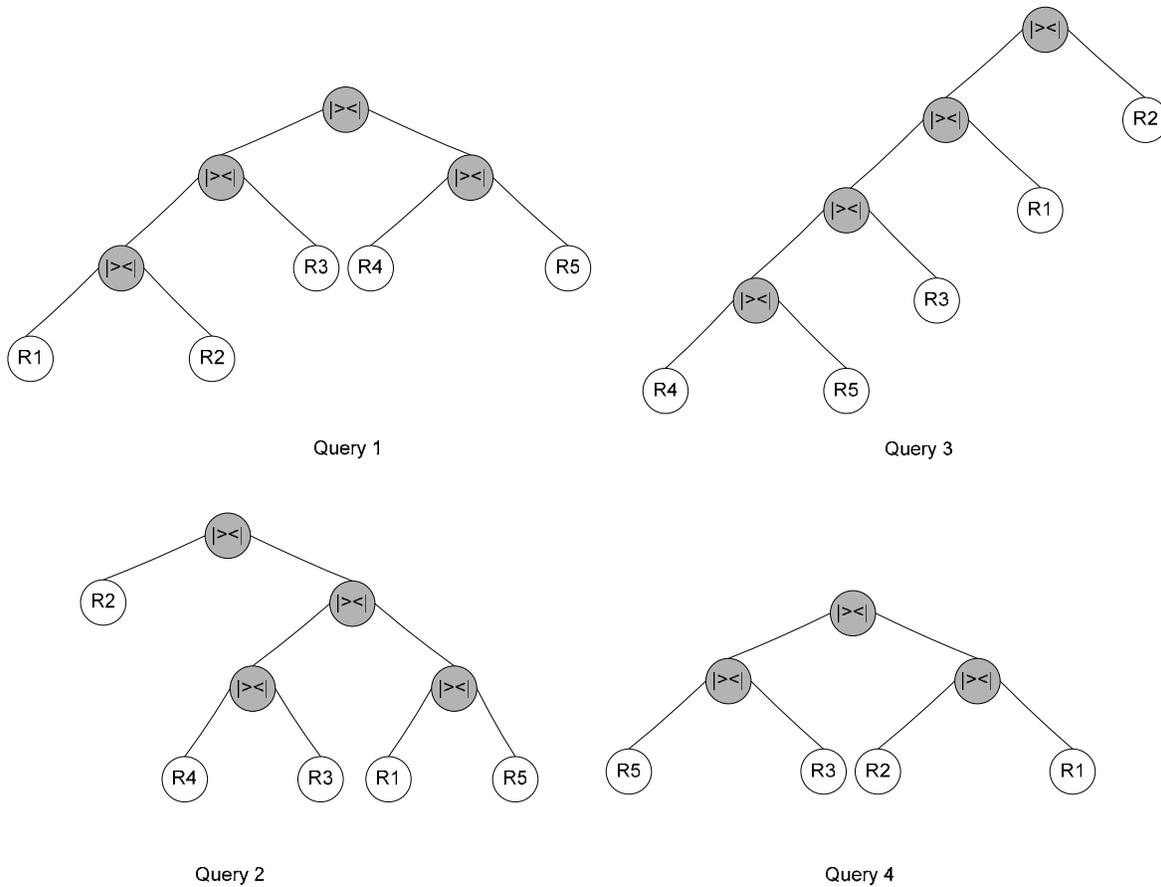


Figure 4-2. Queries represented as AQT

4.4.3. Experiment Scenario

Considering possible daily fluctuation pattern in workload and network traffic, the three optimizers, plus three variants of GOG, i.e., GOG without “count” factor (GOG_c), GOG without TLR factor (GOG_T), and GOG without “count” and TLR factors (GOG_c_T), are tested in each day of a week: each optimizer processes every query three times per day. In total there are $6[\text{optimizers}] * 4[\text{queries}] * 3[\text{times}] * 7[\text{days}] = 504$ optimization processes in the experiment. The optimizers’ running times, estimated running times of generated execution plans, and the sums of the two times are recorded (Appendix A) for analysis.

4.5. Experiment Results and Analysis

4.5.1. QOT Analysis

Figure 4-3 presents average QOT in the four queries. It is observed that mean QOT of GOG are in the middle of the three QOT series in all four queries, which confirms Hypothesis H_{01} .

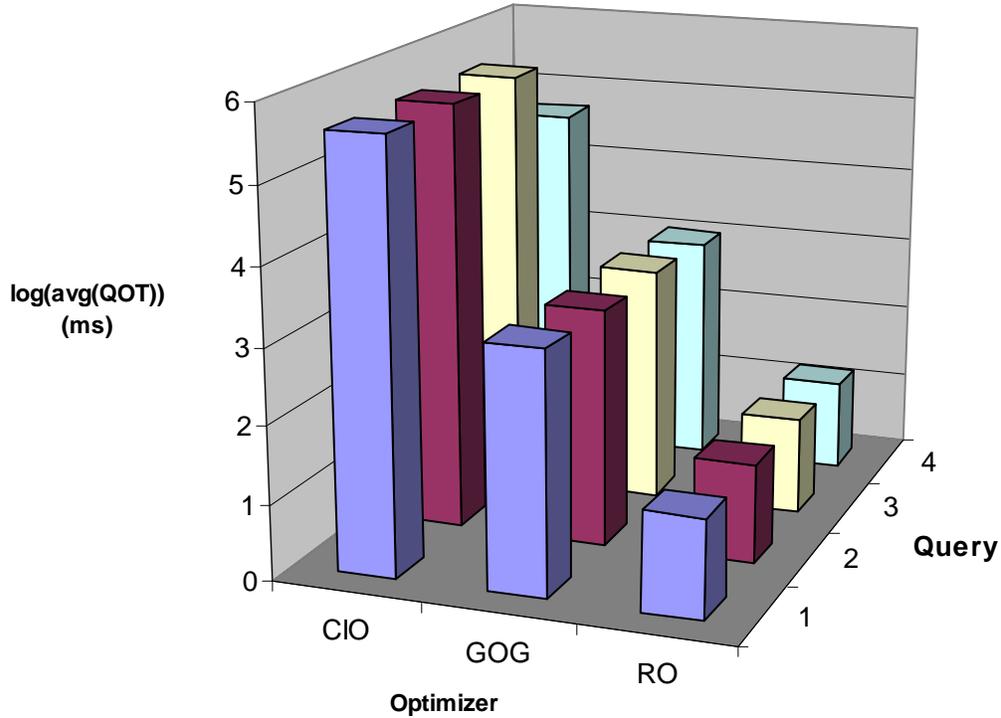


Figure 4-3. Average QOT of GOG, CIO, and RO

A three-factor ANOVA (classes: “query”, “optimizer” and “day”) is applied to check the main effects on QOT of the three optimizers. It is found that the interactions between each pair of the three classes are significant (p -value less than 0.0001), which requires further analyses on individual query-day combinations.

One-factor ANOVA with Tukey multiple comparison (class: “optimizer”) is next conducted for each pair of query-day combination (28 tests in total). All tests show that for the four queries,

QOT of GOG are less than QOT of CIO but longer than these of RO. Thus it can be inferred that for the four testing queries, GOG runs faster than CIO but is slower than RO, i.e., hypothesis H_{01} holds.

4.5.2. QET Analysis

Figure 4-4 shows average QET of each optimizer in the four queries. It is observed that execution plans generated by CIO tend to run faster than those by RO while plans built by GOG stand in between. This observation is confirmed by the following statistical tests.

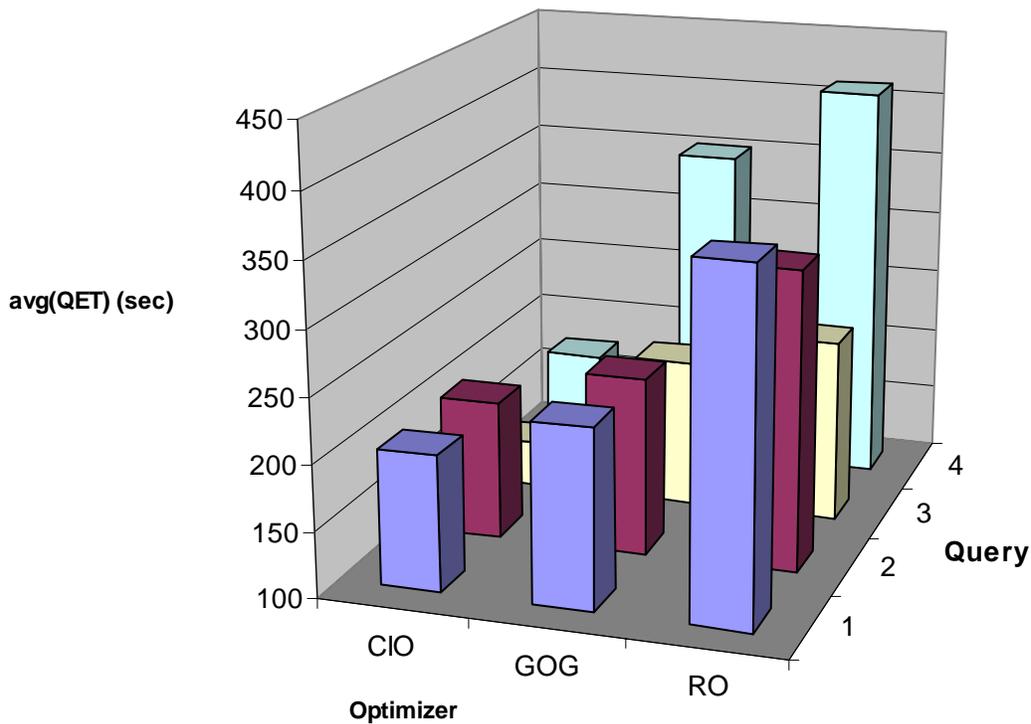


Figure 4-4. Average QET of GOG, CIO, and RO

A three-factor ANOVA (classes: “query”, “optimizer” and “day”) is first conducted. The analysis result shows that there is a significant interaction between class “optimizer” and “query”

(p -value less than 0.0001). It also shows that class “day” does not seem to have influence on QET (p -values equals to 0.1331). Thus QET (i.e., estimated execution costs of execution plans) collected in the seven days can be pooled in following tests.

Four one-factor ANOVA (class: “optimizer”) tests (with Tukey multiple comparison) on each of the four queries are conducted. The results show that QET of the three optimizers are significantly different from each other (p -value less than 0.0001). The results of multiple comparison indicate that for the four queries, QET of GOG are less than QET of RO but greater than these of CIO, meaning that hypothesis H_{02} stated in Section 4.3 holds.

4.5.3. QPT Analysis

QPT are calculated by adding QOT and QET of an optimizer in processing a given query. Average QPT of CIO, GOG, and RO in the four testing queries are depicted in Figure 4-5. It is observed that in Queries 1, 2 and 3, mean QPT of GOG are less than mean QPT of CIO and RO. In Query 4, mean QPT of GOG is greater than mean QPT of CIO but less than those of RO.

A three-factor ANOVA (classes: “query”, “optimizer” and “day”) is applied to analyze QPT of the three optimizers. The result indicates that the interaction between classes “optimizer” and “query” is significant (p -value less than 0.0001) and the “day” class does not seem to impact QPT. Thus observations on one optimizer throughout a week are pooled in following tests.

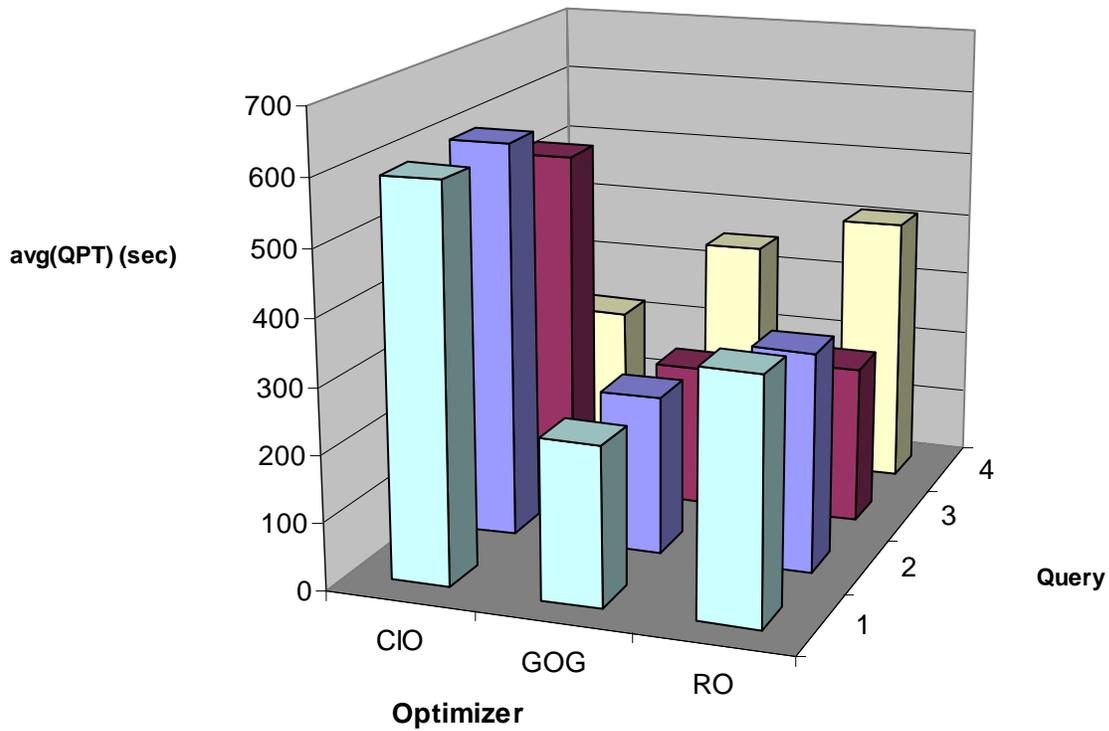


Figure 4-5. Average QPT of CIO, GOG, and RO

For each of the four testing queries, a one-factor ANOVA (class: “optimizer”) (with Tukey multiple comparison) is used to check the differences between QPT of the three optimizers. It is found that GOG has significantly less QPT than CIO and RO in Queries 1, 2 and 3. In Query 4, QPT of GOG are greater than QPT of CIO but less than QPT of RO. Average QPT gains in GOG are calculated and summarized in Table 4-4 (the negative value in the right-bottom cell is due to the larger QPT of GOG in Query 4).

Table 4-4. QPT gains in GOG

Query	GOG vs. RO (%)	GOG vs. CIO (%)
1	34.56	59.71
2	27.65	60.17
3	10.36	59.42
4	14.35	-58.05

GOG outperforms CIO and RO in Queries 1, 2 and 3 which can be explained by the tradeoff between QOT and QET in GOG. GOG tends to avoid extra optimization cost through resource selection, at the risk of eliminating superior resources and potential parallel executions, as discussed in Section 3.6. Replicas and operations involved in Queries 1, 2 and 3 are more than those in Query 4, which leads to a quite large search space in the first three queries (about 15,000 candidate plans) and a relatively small search space for Query 4 (2560 candidate plans). Thus by reducing QOT via resource selection, GOG significantly brings down QPT in Queries 1, 2 and 3 below QPT of CIO. As for RO, although it also has small QOT in the three queries, GOG employs superior resources as well as parallelism to speed up query execution and therefore makes its QPT less than QPT of RO. On the other hand, due to the small search space in Query 4, gain in QOT of GOG via resource selection cannot compensate its relatively large QET compared to QET of CIO, as seen in Figure 4-3 and Figure 4-4. Thus CIO has less QPT than GOG in Query 4. In other words, exhaustive searching can be effective when search space is relatively small. For RO, its QPT in Query 4 is greater than QPT of GOG due to resource selection and parallel execution employed in GOG.

Based on this analysis, it is inferred that for queries with large search space such as Queries 1, 2 and 3, the overall performance of GOG is better than CIO and RO, i.e., Hypotheses H_{03} and H_{04} hold. For queries with small search space, e.g., Query 4, GOG is less effective than CIO but still better than RO.

4.5.4. Analysis on Factors in the Ranking Function

In order to check the impact of “count” and TLR factors in the ranking function, three variants of GOG, i.e., GOG without “count”, GOG without TLR, and GOG without “count” and TLR, are tested using the four testing queries. Average QPT of GOG and the three variants are shown in Figure 4-6. A one-factor ANOVA (class “optimizer”) with Tukey multiple comparisons is used to check QPT in the four queries. It is found that QPT of GOG is consistently better than QPT of the three variants in Queries 1, 2 and especially 3. There is no significant difference between QPT of the three variants in Queries 1, 2 and 3. For Query 4, no significant difference is detected between QPT of GOG and the three variants.

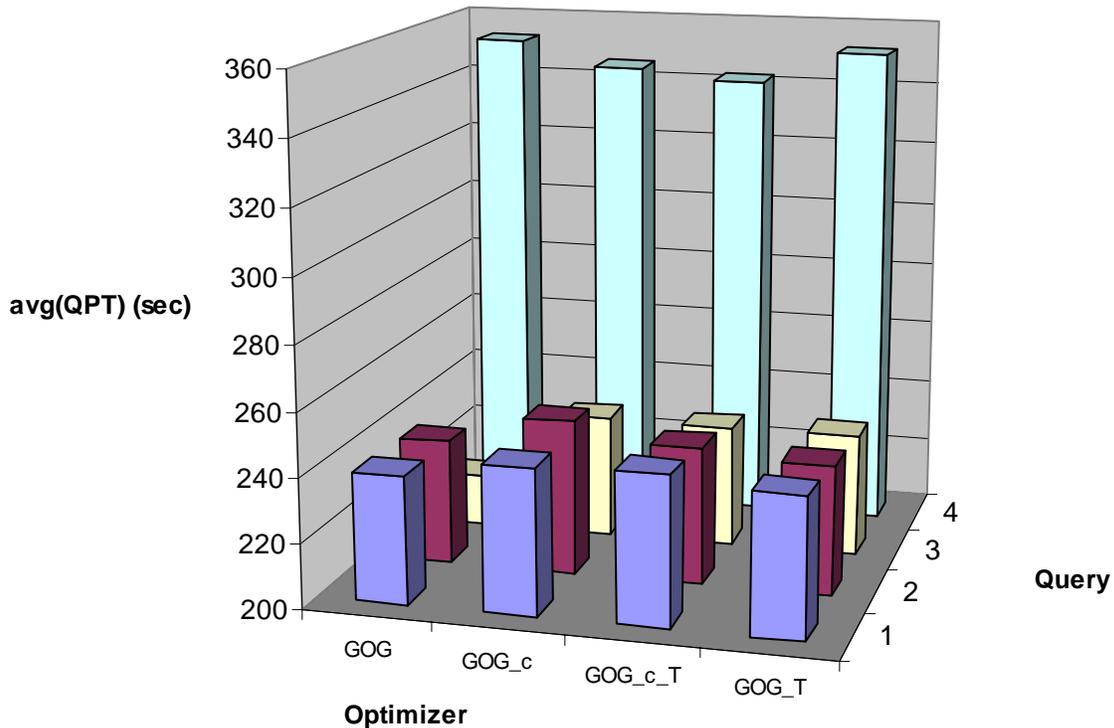


Figure 4-6. Average QPT of GOG and three variants

Statistical test results show that QPT of GOG in Queries 1, 2 and 3 are significantly less than QPT of GOG_c. Compared to GOG_c, average QPT of GOG are reduced by 2.22%, 3.71% and 9.29% in the three queries, respectively. This indicates that using the “count” factor helps reduce overall response times of the three queries. As for the TLR factor, QPT of GOG is significantly different from QPT of GOG_T in Query 3 (average QPT decreased by 9.25%) but is not significantly different from QPT of GOG_T in Queries 1 and 2 (average QPT decreased by 1.52% and 0.62%, respectively). The insignificant difference of QPT in Queries 1 and 2 can be explained by parallelism utilized in query execution. Due to its left-bushy tree structure, all joins in Query 3 have to be executed in sequential. Thus all reduced data transmission costs by using TLR in the ranking function are counted in QET and thus in QPT. On the other hand, Queries 1 and 2 have two joins in leaf nodes that can run in parallel. Since only the maximum cost of a series of parallel operations is counted in QET (see Equation 4.1), part of the reduced transmission costs are overlooked in Queries 1 and 2. For instance, suppose the join of $R1$ and $R2$ ($J1$) and the join of $R4$ and $R5$ ($J2$) in Query 1 are planned to run in parallel. If the running time of $J1$ is larger than $J2$, the running time of $J1$ will be counted in QPT but the running time of $J2$ (including data transmission time) is omitted due to the parallel execution. Considering the method of measuring running time in parallel execution, differences between QPT of GOG_T and GOG in Queries 1 and 2 may not be significant.

The insignificant difference between QPT of GOG and the three variants in Query 4 is not unexpected, considering less relations and data transmissions involved in the query. A significant difference between QPT of GOG and the three variants is expected to be found if more relations and joins are used in Query 4.

4.6. Summary

From the statistical analyses, it can be inferred that for the four testing queries:

1. GOG runs faster than CIO (in order of magnitude) but is slower than RO;
2. GOG tends to generate execution plans with less than average costs.
3. Overall query processing times of GOG are better than the ones of RO.
4. GOG has better overall query processing times than CIO in queries that have large search space.
5. Employing factors “count” and TLR in the ranking function helps improve query execution performance.

These conclusions indicate that the design objective of GOG, i.e., improving query processing in grids with less optimization cost, is achieved. The quality of optimization results of GOG can be further improved by utilizing domain-specific parallelism, which is discussed in Chapter 5.

5. Proof of Concept

An advantage of GOG is that domain-specific parallelism can be included in the optimization strategy. In the current implementation of GOG, GCS is built to support parallelism for geoprocessing operations. To test the validity of GOG with respect to geoprocessing parallelism, an air quality control application built in a grid testbed as a proof of concept is developed and tested. In this chapter, the grid testbed is introduced first, followed by the design of GCS and a description of the application. Test results of the application are presented and analyzed.

5.1. Grid Testbed

A grid testbed was built in the Geoinformatics Laboratory at the University of Pittsburgh. In building the testbed, various hardware and software configurations were used in order to emulate, as much as possible, a grid environment. The testbed is composed of four (hosts) PC workstations (*gis21*, *gis22*, *gis23* and *gis16*), each with a different hardware configuration, connected in a local area network. The workstations are configured with RedHat Linux (Version 9.0) and a grid middleware, Globus (Version 3.2). Three of the workstations (*gis22*, *gis23*, and *gis16*) are configured with a database management system. To emulate a heterogeneous environment, two DBMS, Oracle (installed in *gis22* and *gis23*) and PostgreSQL (installed in *gis16*), are selected. Host *gis21* acts as the access point to the testbed. GOG and CIO run in *gis21* to receive user queries and submit execution plans to hosts in the testbed. Configuration of the testbed is shown in Table 5-1 (WD and CNT are discussed in next section).

Table 5-1. Testbed configuration

Workstation	DBMS	Copy of Air Emission Database	Support to WD	Support to CNT
gis22	Oracle 10g	yes	yes	yes
gis23	Oracle 10g	yes	yes	yes
gis16	PostgreSQL (Release 8.0)	yes	no	no
gis21	N/A	N/A	N/A	N/A

A grid-based open source software package, OGSA-DAI (Version 4.0), is configured in *gis22*. OGSA-DAI is an extension of OGSA (discussed in Chapter 2) that incorporates data resources into OGSA. It complies to Grid Data Service Specification proposed by the Global Grid Forum (GGF) Database Access and Integration Services (DAIS) Working Group (WG) (Krause et al. 2002). To integrate a database into a Globus-based grid, the database should be registered to one or more instances of OGSA-DAI. OGSA-DAI probes functionalities of the database and implements grid services corresponding to these functionalities with the help of grid middleware, i.e., Globus. The resultant grid services are published in and accessed via Globus. Figure 5-1 demonstrates the OGSA-DAI configuration in the testbed and a sample scenario (presented in dashed arrows with sequence number). As seen in this diagram, all three databases are registered to the OGSA-DAI instance in *gis22*. When the user submits a query to the query optimizer in *gis22* (Arrow 1), the optimizer generates an execution plan based on current status of computing resources. For each database involved in the execution plan, Globus is contacted (Arrow 2) in order to activate the corresponding grid factory service (Arrow 3). The grid factory service in turn creates an instance of grid service (Arrow 4) that will provide access to the database (Arrows 5 and 6).

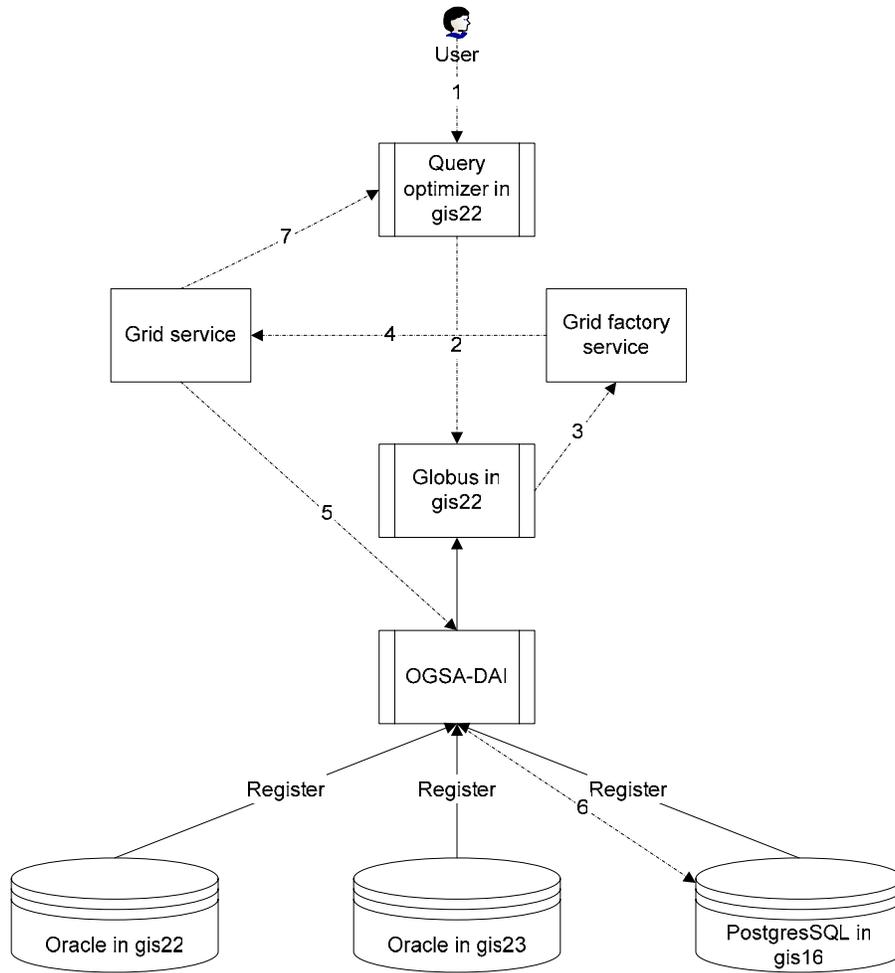


Figure 5-1. OGSA-DAI configuration in the testbed

Since statistical analyses in Section 4.5 show that execution plans generated by RO perform worse than those by GOG and CIO and the primary purpose of the proof of concept is to compare performances of execution plans, RO is not considered in the proof of concept.

5.2. Design of GCS

The responsibility of GCS is to provide a parallel execution for geoprocessing operations if the grid has appropriate resources (e.g., a certain number of hosts that can conduct such operations). GCS also estimates running times of suggested parallel executions in order to help the optimization process determine whether to use parallel execution or non-parallel execution.

As a proof of concept, current implementation of GOG is built to support two spatial operations, “within_distance” and “contains” (abbreviated as WD and CNT operations respectively afterwards). The two operations have the following form:

WITHIN_DISTANCE (spatial_column1, spatial_column2, distance, unit)

CONTAINS (spatial_column1, spatial_column2)

WD operation checks if spatial objects defined by *spatial_column1* and *spatial_column2* are within the specified distance of a certain unit. The distance between two objects is defined as the minimum distance between any points in the two objects. WD operation computes the distance between two spatial objects in a nested loop, every object in *spatial_column2* is read, spatially indexed, and evaluated against all the objects in *spatial_column1*. Thus WD operation could become computationally expensive when the number of objects in *spatial_column2* is more than one.

CNT operation determines whether a spatial object in *spatial_column2* is contained by an object in *spatial_column1*. “contains” spatial relationship exists for two objects when the boundary and interior of one object is entirely inside of another (Egenhofer and Franzosa 1991). Similar to

WD, CNT operation uses nested loop when the number of objects in *spatial_column2* is greater than one.

As for many compute-intensive operations, one way to improve executions of WD and CNT is partitioning input relations along spatial columns and running a number of parallel partitioned operations. There are two approaches to partition data for WD and CNT operations:

- I. Divide one of the spatial relations into several partitions. Each partition is paired with the other input relation and a WD or CNT operation is carried on that pair. This method requires at least two hosts that support the given spatial operation.
- II. Divide both spatial relations into different partitions. Combine partitions by crossing over the partitions from the two relations and conduct WD or CNT operation on each of the combinations. This method requires at least $2*2=4$ hosts to support the given operation.

Given the limited number of hosts in the testbed, Method I is adopted in current implementation of GCS.

The design of GCS is shown in Figure 5-2. After determining that an operation is a spatial operation, GCS searches the grid for all hosts which support that operation. If there are enough hosts (more than 1), it proceeds to generate a partition plan for parallel execution. Among the two relations involved in a geoprocessing operation, the larger one is chosen to be partitioned. This decision is based on the expectation that when the number of partitions is large enough, both the partitioned relation and another join relation can fit into the main memory to reduce the

number of I/O operations. An equal number of records from the partitioned relation are assigned to each host. Based on the partition plan, a set of data staging steps are generated to transfer relations to each host for parallel execution and collect intermediate results after operations are completed.

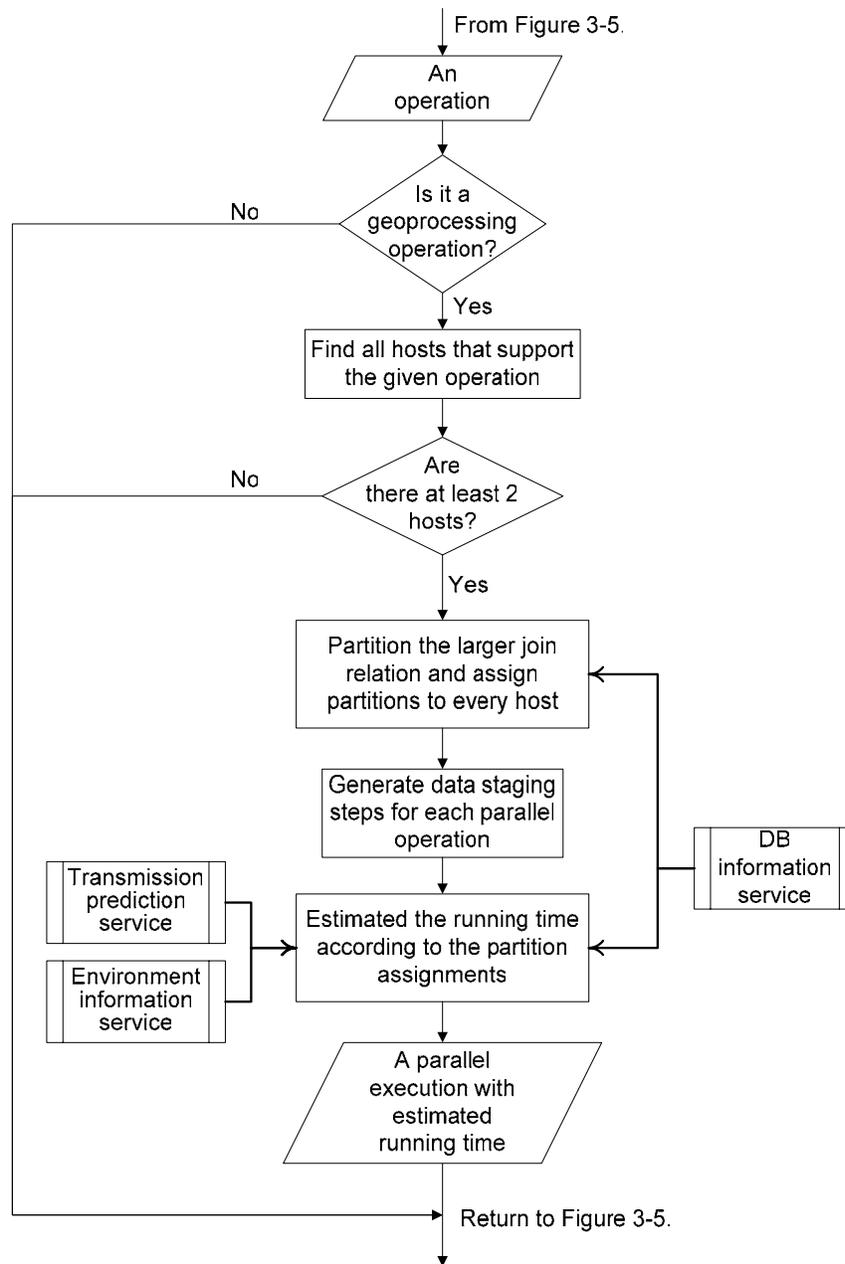


Figure 5-2. Design of GCS

The total running time of a parallelized spatial operation i is estimated as the maximum running time of operations on each partition:

$$t_i = \max(t_{ij}), j = 1, 2, \dots, n \quad 5.1$$

where t_{ij} is running time of Operation i on Partition j . t_{ij} can be estimated as a linear combination of three components:

$$t_{ij} = t_{ij,op} + t_{ij,partition} + t_{ij,result} \quad 5.2$$

where $t_{ij,op}$ is the estimated running time of a given operation on Partition j , $t_{ij,partition}$ is the estimated time to transfer the partition to Host j if that host does not have a replica of the partitioned relation, $t_{ij,result}$ is the estimated time to transfer the result to the host that will assemble and use the results as input. $t_{ij,partition}$ and $t_{ij,result}$ can be estimated in the same way as the transfer cost in Equation 4.2. Running time of join is often estimated based on the number of IO operations, but this method is not suitable to predict running time for WD and CNT operations since CPU cost constitutes a significant part to the running time of these operations besides IO cost. Without knowledge on specific implementation at a host, one way to predict $t_{ij,WD}$ is to model it using observed statistical data. Given that both WD and CNT employ nested loops to evaluate the spatial relationship between two objects, the running time can be represented as a linear function of the size of a partition. Linear regressions are conducted on the hosts in the testbed that support the two operations (Appendix C) and show that running times in these hosts have good linear relationships with the sizes of partition (p -values are less than 0.0001 and values of R^2 are greater than 0.96). Thus $t_{ij,op}$ is estimated using the derived linear function for a given operation at each host. These linear functions are also used to estimate the running time of WD and CNT operations without any partition, i.e., non-parallel execution.

The generated parallel execution together with the estimated running time is returned to the parallel processing module which will compare the running times of parallel and non-parallel executions and select the smaller as a part of the final execution plan.

5.3. Design of the Proof of Concept

One area where geoprocessing is intensively used is air quality control in environmental engineering. A problem environmental scientists and engineers often encounter is to locate pollution sources that may cause abnormal concentrations of pollutant species at a location of a specific time. A solution to locate suspect pollution sources can be expressed as follows:

- I. Find the location where the concentration of a pollutant species is abnormal at a specified time.
- II. Compute the backward wind trajectory that passes the location with high concentration of a pollutant species.
- III. Use either WD or CNT operation to determine pollution sources that are within a certain range of the wind trajectory, for instance, 20 km:
 - a) Using WD operation. Check to see if any pollution source is within 20 km of the trajectory.
 - b) Using CNT operation. Build a buffer with 20 km radius around the trajectory. Check to see if any pollution source is contained by the buffer.

In this research, the above application is used as the proof of concept to demonstrate the validity and efficiency of GOG. Components of the proof of concepts are discussed in following sections.

In addition, performances of the two geoprocessing operations, WD and CNT, are recorded and analyzed.

5.3.1. Air Emission Database

An air emission database is built to test the validity and efficiency of GOG in optimizing geospatial queries. The air emission database has locations and other attribute data (e.g., name, discharged pollutant species) of pollution sources in the United States. It also maintains concentrations of pollutant species collected in a grid map. Table 5-2 shows a summary of major relations in the database and the entity-relation diagram is included in Appendix D.

Table 5-2. Summary of major relations in the air emission database

Information Relations	Description	Size (MB)
<i>pt_source</i>	Information of point pollution sources, e.g., geographical coordinates, discharged pollutant species.	293
<i>pollution_source</i>	Names of pollution sources, states and counties where sources are located.	~ 10
<i>concentration</i>	Concentrations of pollutant species in a grid map.	74 (per day)
<i>trj_tmp</i>	Back wind trajectories at specific times.	< 2
<i>trj_tmp_buf</i>	Buffers built around trajectories in ' <i>trj_tmp</i> '.	< 2

In computing wind trajectories, a location with an abnormal concentration of a specific pollutant species at a given time is first located by querying Relation '*concentration*'. The location and time are then used as input to the HYSPLIT model (HYSPLIT 2005) to compute the backward wind trajectory. Based on the trajectory, a buffer is generated and imported into the database together with the trajectory. Such a process to build a backward wind trajectory buffer is demonstrated in Figure 5-3. Points in the map are pollution sources in the continent states (the strip pattern in some mid-west and western states is caused by missing or misplaced coordinate

data in these locations). A maximum concentration of ozone gas (O_3) is found at Location A in Georgia on August 13, 2000 as identified by the star symbol in Figure 5-3. According to the time and location, a backward wind trajectory is calculated by HYSPLIT and a 20 km buffer, Buffer A, around the trajectory is built. The direction of the wind trajectory is indicated by the bold arrow.

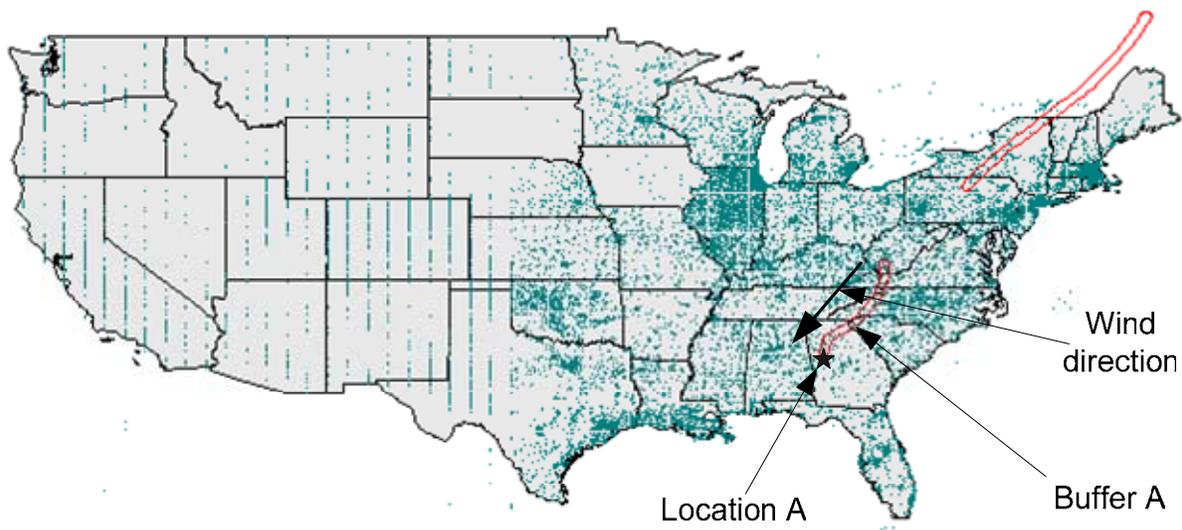


Figure 5-3. Pollution sources and wind trajectory buffers

Building buffers around backward wind trajectories can help identify sources that may cause certain environmental pollutions, which is an important application in environmental monitoring and management. As Figure 5-3 shows, there are some pollution sources along the upwind direction in Buffer A that may be potential cause for the high ozone gas concentration found at Location A. Since airstreams tend to spread along their trajectories, an improvement can be made by using variable width buffer rather than constant width buffer in order to obtain more accurate area affected by a wind. For example, the width of Buffer A can be gradually increased along the

wind direction by an amount according to a certain function, such as 20% of the current distance from the start of the trajectory. Further considerations can be taken into account when computing variable width buffers, e.g., terrain elevations, which are beyond the scope of this research. Considering that buffering in most existing GIS packages and spatial DBMS (including the ones installed in the grid testbed) is implemented using constant width, a constant width (20 km) is used to generate buffers in the air emission database around wind trajectories. Future research is needed to explore utilizing variable width buffer in locating pollution sources.

5.3.2. Query Testing

The process to locate suspect pollution sources using WD operation is transformed into a query called Q_WD in AQT (Figure 5-4). Relation “*trj_tmp*” stores trajectory information. Relation “*pt_source*” has the spatial data of point pollution sources and relation “*pollution_source*” has attribute data of pollution sources (e.g., name, state and county). “*pt_source*” has foreign key “*ID*” on “*pollution_source*” that is the unique identification number for pollution sources. A WD operation is first conducted on spatial columns of “*trj_tmp*” and “*pt_source*” to locate all pollution sources that are within 20 km range of the trajectories in “*trj_tmp*”. An equi-join is next carried on the resultant set and “*pollution_source*” to obtain the names of suspect pollution sources.

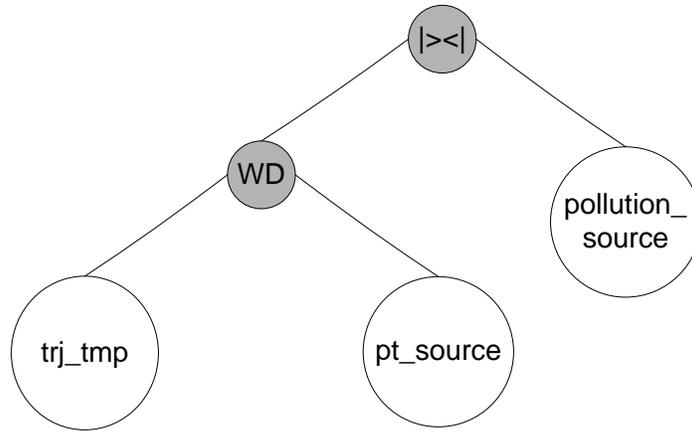


Figure 5-4. Query Q_WD

A query Q_CNT using CNT operation to locate pollution sources is depicted in Figure 5-5. In this query, CNT operation takes a buffer object from relation “*trj_tmp_buf*” that is built around a wind trajectory and checks if any point from “*pt_source*” is contained by the buffer. The result set of the CNT operation is next joined with relation “*pollution_source*” to obtain names of the selected pollution sources.

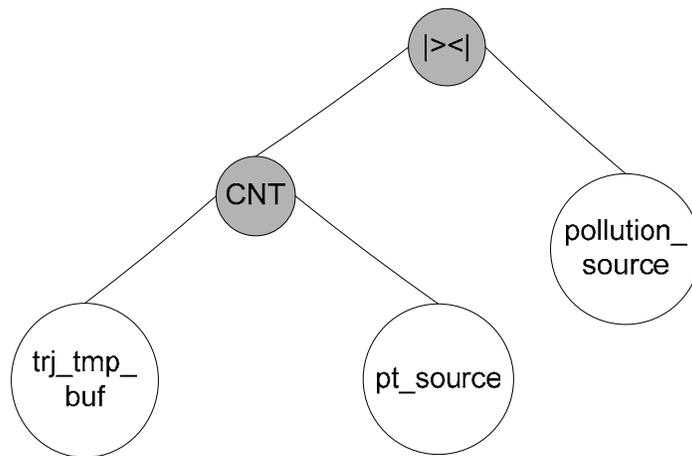


Figure 5-5. Query Q_CNT

5.4. Experiment Results and Analysis

Each of Q_WD and Q_CNT queries is evaluated by CIO and GOG for 5 runs. Optimization and execution times of each run are included in Appendix B. Some sample execution plans generated by GOG and CIO are shown in Appendix E. All runs of Q_WD and Q_CNT report an identical set of 27 pollution sources for a wind trajectory, which shows the validity of GOG with respect to geoprocessing parallelism. A two-factor ANOVA (classes: “query” and “optimizer”) is used to check the main effects on query execution times of the two optimizers. The result shows that the interaction between query and optimizer is significant (p -value less than 0.05), which requires further analyses on query execution times of individual queries.

5.4.1. Q_WD Analysis

Figure 5-6 and Figure 5-7 show QOT and QET in Q_WD by GOG and CIO. QPT in Q_WD are depicted in Figure 5-8. It is observed that GOG takes less time to yield execution plans than CIO due to limited search space in GOG. Furthermore, since CIO does not have support for parallel geoprocessing, it always instructs a single workstation to complete the query. On the other hand, GOG determines that the cost to run parallel WD operations is less than running non-parallel WD, thus it partitions the “*pt_source*” relation and has *gis22* and *gis23* to run a partitioned WD operation. Parallelizing WD by partitioning the “*pt_source*” relation reduces the query execution time almost by half.

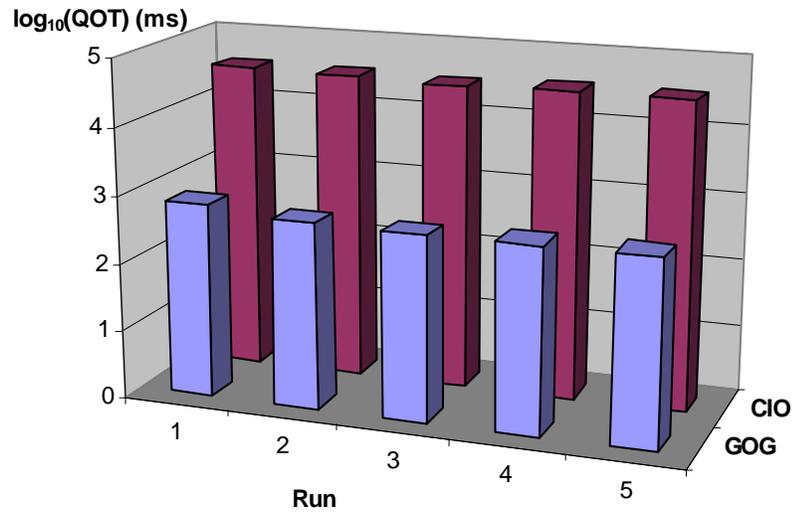


Figure 5-6. Query optimization times of Q_WD

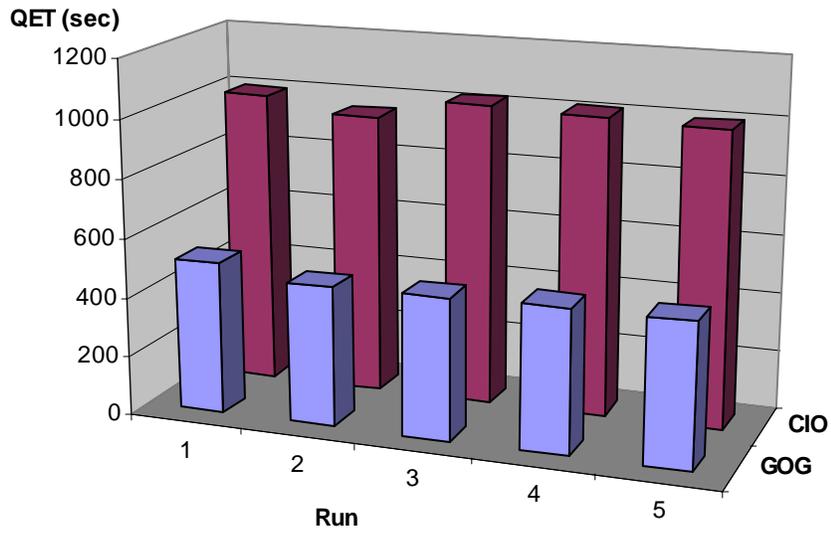


Figure 5-7. Query execution times of Q_WD

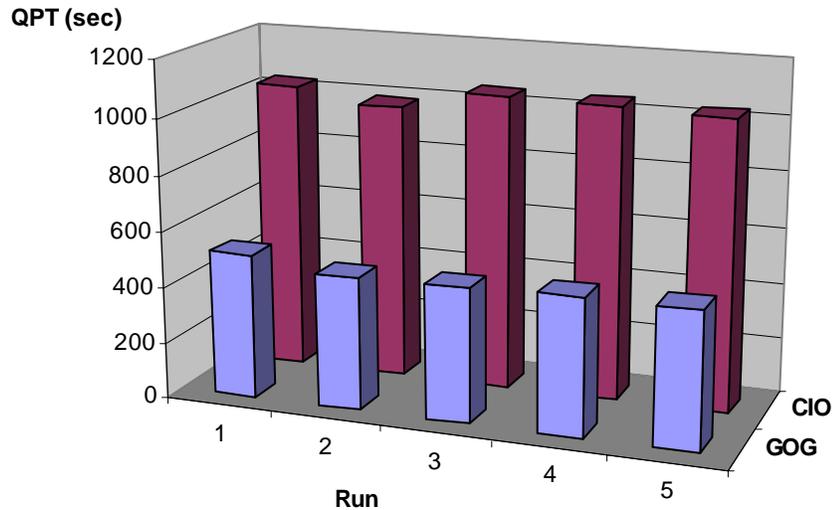


Figure 5-8. QPT of Q_WD

A two-sample *t*-test is applied to QPT. The result shows that there exists a significant difference between QPT of GOG and CIO (*p*-value less than 0.0001). The 95% confidence interval of differences between QPT of GOG and CIO is (541.17, 572.62) seconds. In Query Q_WD, GOG reduces QPT by 49% on average. Such significant improvement is achieved primarily via data partitioning and parallelization of the WD operation.

5.4.2. Q_CNT Analysis

QOT and QET of Q_CNT by GOG and CIO are shown in Figure 5-9 and Figure 5-10, respectively. QPT in Q_CNT are displayed in Figure 5-11. Similar to Query Q_WD, GOG optimizes Q_CNT faster than CIO since GOG has smaller search space. It is also indicated in Figure 5-10 that execution plans generated by GOG use less time to complete than those by CIO. This is explained by the utilization of parallelism in GOG. Since there are two hosts (*gis22* and *gis23*) available for CNT operation, GOG divides “*pt_source*” relation into two parts and assigns

one to each of the two hosts for parallel execution. Thus execution times in GOG are about a half of the ones in CIO.

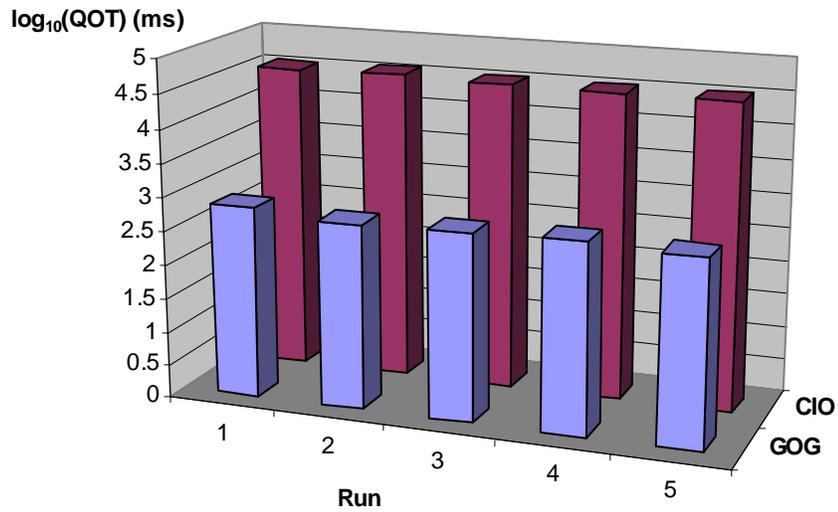


Figure 5-9. Query optimization times of Q_CNT

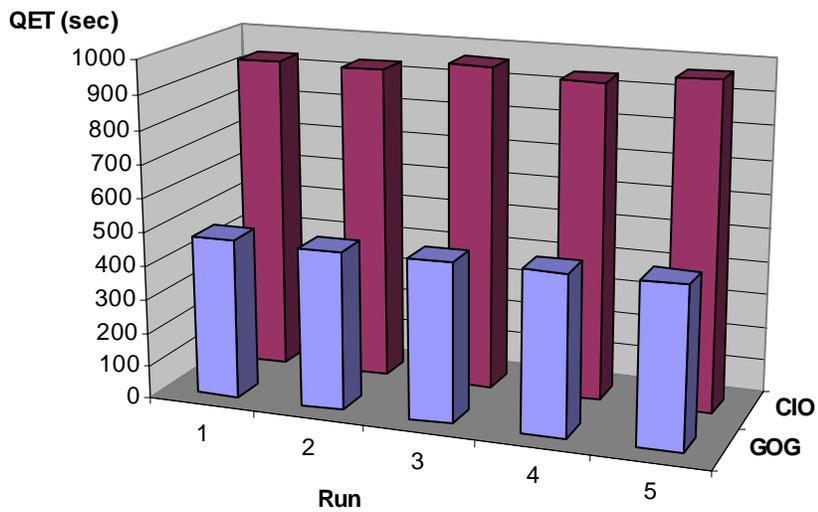


Figure 5-10. Query execution times of Q_CNT

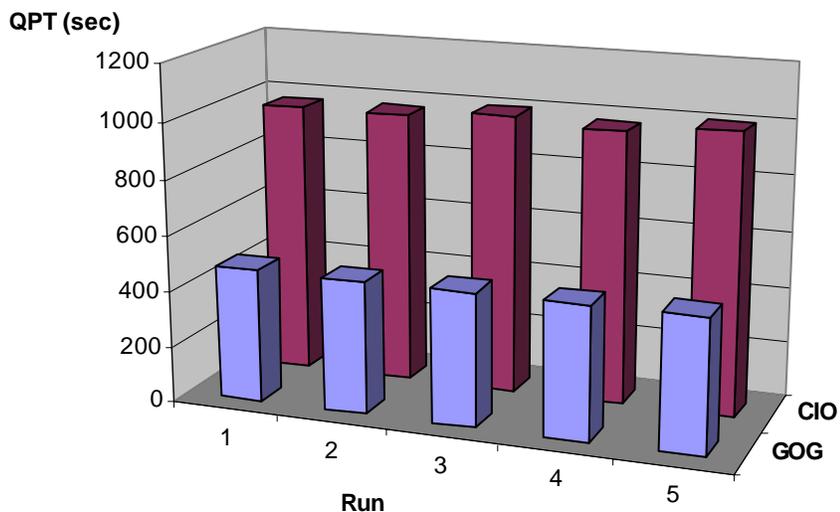


Figure 5-11. QPT of Q_CNT

A two-sample *t*-test is conducted to examine the difference of QPT of query Q_CNT in GOG and CIO. Analysis shows that a significant difference is found (*p*-value less than 0.0001). The 95% confidence interval of the differences between QPT of the two optimizers is (507.21, 522.98) seconds. On average, QPT of GOG is 53% less QPT of CIO on average. Therefore it can be argued that GOG significantly improves the execution of query Q_CNT.

5.4.3. Performance Comparison between WD and CNT

Although both WD and CNT operations are used to test whether a geographical object falls within a buffer, their inputs and implementations are different, which may result in different performance. In the proof of concept, running times of WD and CNT in a workstation are collected and analyzed (Figure 5-12). A two sample *t*-test is applied to check the differences between the running times of the two operations. The analysis shows that running times of CNT

are significantly smaller than those of WD (p -value equals to 0.0195). Thus it can be concluded that CNT runs faster than WD.

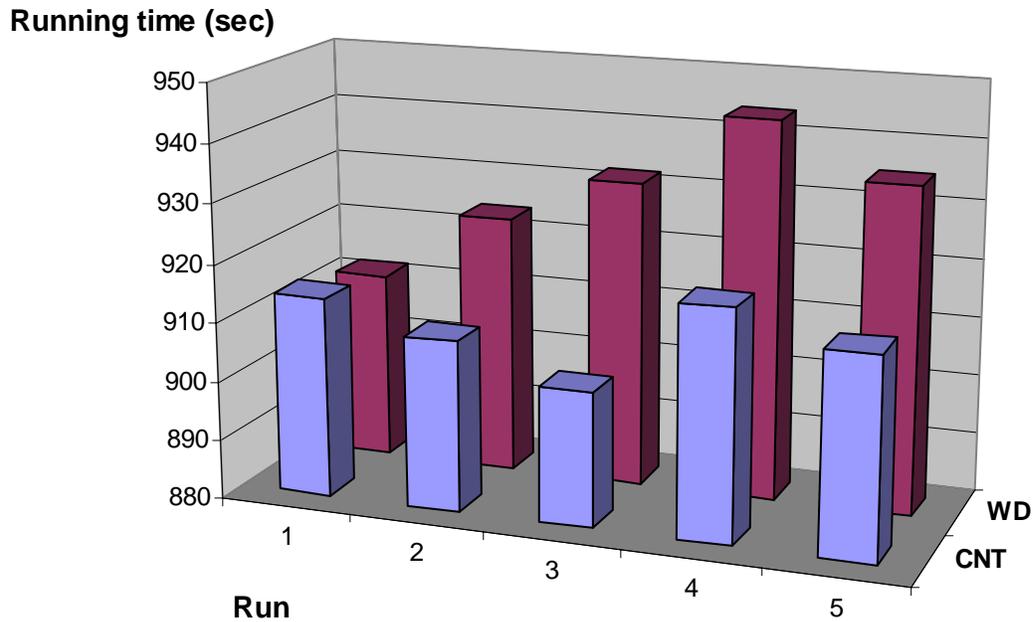


Figure 5-12. Performances of WD and CNT operations

5.5. Summary

By applying GOG in an environmental application, the proof of concept demonstrates the validity of GOG in optimizing spatial queries. For the two spatial queries, Q_WD and Q_CNT, query results obtained by GOG are consistent with the ones obtained by CIO. It is also shown in the proof of concept that on average, GOG improves executions of the two queries by about 50% in terms of overall query processing time compared to CIO. This performance gain is mainly resulted from less query execution time in GOG. With the help of GCS, GOG is able to detect and parallelize geoprocessing operations in the two queries, which leads to less query execution time. On the other hand, CIO does not have support for parallelization of geoprocessing and uses non-parallel executions for WD and CNT operations which is why it takes longer to complete.

6. Conclusions and Future Research

6.1. Summary of the Research

In this research a new technique, GOG, for optimizing geoprocessing in grid systems is developed. It is observed that for complex and dynamic distributed systems like grids, conventional optimization techniques, using exhaustive search, become expensive and impractical. GOG employs an optimization strategy that limits the search space and reduces optimization cost. To assure an acceptable level of performance of query execution while reducing cost, a ranking function is introduced into GOG to prune inferior resources. The ranking function ranks candidate computing resources by taking into account several important performance factors and selects the ones that incur less costs. GOG also exploits parallelism to speed up query execution. Two types of parallelisms, generic and domain-specific, are identified and evaluated in optimization. Compared with exhaustive search and randomized optimizations, GOG optimizes queries with less cost, running faster than exhaustive search optimization and generating execution plans with better-than-average performance. In a proof of concept, it was demonstrated that GOG supports parallel geoprocessing and generates query execution plans which run faster than the ones determined by exhaustive search optimization.

The contributions of this research are:

- A new technique for optimizing geoprocessing in grids.
- A ranking function that determines the overall capacity of hosts.
- A new index that measures transmission capacity of a host in the context of a query.
- A module for processing parallel geoprocessing operations.

6.2. Conclusions

Compared with other optimization techniques and tested in a proof of concept, GOG shows a good balance between low cost and high quality. Several conclusions can be drawn from this research:

- It is feasible to reduce optimization cost while maintaining quality at a certain level.
- Query execution can be significantly improved by taking into account certain factors.
- As an important means to speed up query execution in distributed systems, parallelism can be handled at different levels of granularity, which can lead to flexible design and better adaptability of an optimization technique.
- Data transmission can be a dominant factor in the cost of query execution in distributed systems connected via wide area networks. Given the uncertainty in selecting computing resources during optimization and the dynamism of grids, measuring transmission capacity of a host in the context of a query can provide estimations with reasonable quality.

6.3. Future Research

Grid computing is still a new area for research, so is optimization in grid systems. Many problems in this field remain to be studied. In light of this research, some topics that need further investigation are highlighted below:

- Ecology in optimization. Most current optimization mechanisms in grids are focused on improving execution performance for individual queries. While this approach is valid from the perspective of single executions, such an optimization style may not be appropriate since it could deteriorate performances of other queries running at the same

time or later. It is important for query processing mechanisms to treat the running environment as an ecosystem, i.e., utilizing resources while preserving them for other tasks. New optimization strategies and algorithms need to be developed to achieve this objective.

- Adaptive query execution. Many queries submitted to grid systems are expected to be data- and/or compute-intensive and may take long time to complete (hours or even days). Due to the dynamism in grids, part or even the whole execution plan may become inefficient or invalid during execution. A monitoring mechanism that can dynamically adjust query execution is needed.
- Optimization of raster-based spatial processing. Unlike vector-based operations, raster-based spatial processing is based on raster data that are difficult to be maintained and queried in relational databases. Thus there is a need to develop new approaches to optimizing raster-based geoprocessing in grids.

APPENDIX A

Experiment Results of Simulated Grid Environment

Optimizer	Date	Query	QOT (ms)	QET (ms)	Overall (ms)
CIO	Sun	1	396566	161995	558561
GQO	Sun	1	1766	233030	234796
GQO_c	Sun	1	1629	236433	238062
GQO_T	Sun	1	1609	233030	234639
RO	Sun	1	31	537746	537777
CIO	Sun	1	392483	161995	554478
GQO	Sun	1	1231	236861	238092
GQO_c	Sun	1	1201	233877	235078
GQO_T	Sun	1	1404	228975	230379
RO	Sun	1	19	259965	259984
CIO	Sun	1	390984	161995	552979
GQO	Sun	1	1196	231959	233155
GQO_c	Sun	1	1182	233877	235059
GQO_T	Sun	1	1185	228975	230160
RO	Sun	1	20	279885	279905
CIO	Mon	1	389910	167184	557094
GQO	Mon	1	2095	231159	233254
GQO_c	Mon	1	1246	245414	246660
GQO_T	Mon	1	1249	245251	246500
RO	Mon	1	18	279182	279200
CIO	Mon	1	389365	167184	556549
GQO	Mon	1	1266	231159	232425
GQO_c	Mon	1	1257	249627	250884
GQO_T	Mon	1	1252	236829	238081
RO	Mon	1	18	277446	277464
CIO	Mon	1	388490	167184	555674
GQO	Mon	1	1266	245251	246517
GQO_c	Mon	1	1262	239871	241133
GQO_T	Mon	1	1291	243961	245252
RO	Mon	1	20	398279	398299
CIO	Tue	1	388691	169743	558434
GQO	Tue	1	1869	232107	233976
GQO_c	Tue	1	1246	250270	251516
GQO_T	Tue	1	1259	238132	239391
RO	Tue	1	19	308360	308379
CIO	Tue	1	389550	169743	559293

Optimizer	Date	Query	QOT (ms)	QET (ms)	Overall (ms)
GQO	Tue	1	1265	232107	233372
GQO_c	Tue	1	1247	239597	240844
GQO_T	Tue	1	1299	245998	247297
RO	Tue	1	19	220857	220876
CIO	Tue	1	388946	169743	558689
GQO	Tue	1	1255	238132	239387
GQO_c	Tue	1	1274	239597	240871
GQO_T	Tue	1	1245	241025	242270
RO	Tue	1	23	552687	552710
CIO	Wed	1	391172	233606	624778
GQO	Wed	1	2032	241279	243311
GQO_c	Wed	1	1242	246464	247706
GQO_T	Wed	1	1229	246220	247449
RO	Wed	1	18	472438	472456
CIO	Wed	1	391338	233606	624944
GQO	Wed	1	1227	246464	247691
GQO_c	Wed	1	1202	245978	247180
GQO_T	Wed	1	1204	250919	252123
RO	Wed	1	19	288591	288610
CIO	Wed	1	390717	233606	624323
GQO	Wed	1	1211	241764	242975
GQO_c	Wed	1	1210	246464	247674
GQO_T	Wed	1	1204	241279	242483
RO	Wed	1	18	266825	266843
CIO	Thu	1	389746	231366	621112
GQO	Thu	1	1808	242074	243882
GQO_c	Thu	1	1187	246840	248027
GQO_T	Thu	1	1177	249796	250973
RO	Thu	1	21	255489	255510
CIO	Thu	1	390695	231366	622061
GQO	Thu	1	1189	242074	243263
GQO_c	Thu	1	1177	249796	250973
GQO_T	Thu	1	1179	245644	246823
RO	Thu	1	19	315488	315507
CIO	Thu	1	390017	231366	621383
GQO	Thu	1	1188	242074	243262
GQO_c	Thu	1	1173	246840	248013
GQO_T	Thu	1	1179	246835	248014
RO	Thu	1	20	279240	279260
CIO	Fri	1	388638	236585	625223
GQO	Fri	1	1742	245534	247276
GQO_c	Fri	1	1192	249736	250928
GQO_T	Fri	1	1220	247380	248600
RO	Fri	1	18	707988	708006
CIO	Fri	1	388830	236585	625415
GQO	Fri	1	1203	241052	242255

Optimizer	Date	Query	QOT (ms)	QET (ms)	Overall (ms)
GQO_c	Fri	1	1234	245254	246488
GQO_T	Fri	1	1190	245534	246724
RO	Fri	1	18	249830	249848
CIO	Fri	1	388745	236585	625330
GQO	Fri	1	1236	241052	242288
GQO_c	Fri	1	1187	245159	246346
GQO_T	Fri	1	1195	245254	246449
RO	Fri	1	18	327685	327703
CIO	Sat	1	390373	231546	621919
GQO	Sat	1	1752	238218	239970
GQO_c	Sat	1	1149	243739	244888
GQO_T	Sat	1	1159	235022	236181
RO	Sat	1	18	796279	796297
CIO	Sat	1	390625	231546	622171
GQO	Sat	1	1169	235349	236518
GQO_c	Sat	1	1152	243739	244891
GQO_T	Sat	1	1161	244806	245967
RO	Sat	1	20	276283	276303
CIO	Sat	1	392016	231546	623562
GQO	Sat	1	1171	235349	236520
GQO_c	Sat	1	1156	241610	242766
GQO_T	Sat	1	1152	244806	245958
RO	Sat	1	19	342342	342361
CIO	Sun	2	394880	164325	559205
GQO	Sun	2	1757	231452	233209
GQO_c	Sun	2	1627	234008	235635
GQO_T	Sun	2	1591	235507	237098
RO	Sun	2	31	318777	318808
CIO	Sun	2	394895	164325	559220
GQO	Sun	2	1288	234436	235724
GQO_c	Sun	2	1274	233031	234305
GQO_T	Sun	2	1178	231452	232630
RO	Sun	2	19	256413	256432
CIO	Sun	2	394164	164325	558489
GQO	Sun	2	1196	234436	235632
GQO_c	Sun	2	1182	240294	241476
GQO_T	Sun	2	1169	231452	232621
RO	Sun	2	19	551850	551869
CIO	Mon	2	393376	169362	562738
GQO	Mon	2	1363	214231	215594
GQO_c	Mon	2	1246	251802	253048
GQO_T	Mon	2	1247	228323	229570
RO	Mon	2	20	284856	284876
CIO	Mon	2	394387	169362	563749
GQO	Mon	2	1262	214231	215493
GQO_c	Mon	2	1282	241923	243205

Optimizer	Date	Query	QOT (ms)	QET (ms)	Overall (ms)
GQO_T	Mon	2	1238	214231	215469
RO	Mon	2	19	561023	561042
CIO	Mon	2	393058	169362	562420
GQO	Mon	2	1254	214231	215485
GQO_c	Mon	2	1240	241923	243163
GQO_T	Mon	2	1251	214231	215482
RO	Mon	2	19	304301	304320
CIO	Tue	2	393943	171420	565363
GQO	Tue	2	1241	219016	220257
GQO_c	Tue	2	1222	253840	255062
GQO_T	Tue	2	1220	251289	252509
RO	Tue	2	23	206532	206555
CIO	Tue	2	393819	171420	565239
GQO	Tue	2	1240	219016	220256
GQO_c	Tue	2	1220	242243	243463
GQO_T	Tue	2	1271	219016	220287
RO	Tue	2	18	259937	259955
CIO	Tue	2	393676	171420	565096
GQO	Tue	2	1236	251289	252525
GQO_c	Tue	2	1228	242243	243471
GQO_T	Tue	2	1220	227935	229155
RO	Tue	2	19	292998	293017
CIO	Wed	2	393450	235283	628733
GQO	Wed	2	1197	247448	248645
GQO_c	Wed	2	1237	254315	255552
GQO_T	Wed	2	1202	247448	248650
RO	Wed	2	18	267074	267092
CIO	Wed	2	394036	235283	629319
GQO	Wed	2	1226	247448	248674
GQO_c	Wed	2	1207	253830	255037
GQO_T	Wed	2	1200	247448	248648
RO	Wed	2	20	302379	302399
CIO	Wed	2	392875	235283	628158
GQO	Wed	2	1211	247933	249144
GQO_c	Wed	2	1204	247933	249137
GQO_T	Wed	2	1195	247448	248643
RO	Wed	2	19	282666	282685
CIO	Thu	2	393640	233043	626683
GQO	Thu	2	1185	247996	249181
GQO_c	Thu	2	1185	252561	253746
GQO_T	Thu	2	1177	247996	249173
RO	Thu	2	20	324756	324776
CIO	Thu	2	394617	233043	627660
GQO	Thu	2	1191	247996	249187
GQO_c	Thu	2	1179	252561	253740
GQO_T	Thu	2	1174	247996	249170

Optimizer	Date	Query	QOT (ms)	QET (ms)	Overall (ms)
RO	Thu	2	19	338539	338558
CIO	Thu	2	392621	233043	625664
GQO	Thu	2	1188	247996	249184
GQO_c	Thu	2	1207	249064	250271
GQO_T	Thu	2	1172	247996	249168
RO	Thu	2	19	465467	465486
CIO	Fri	2	393749	238261	632010
GQO	Fri	2	1209	247221	248430
GQO_c	Fri	2	1225	251935	253160
GQO_T	Fri	2	1202	247221	248423
RO	Fri	2	19	303090	303109
CIO	Fri	2	393437	238261	631698
GQO	Fri	2	1218	247221	248439
GQO_c	Fri	2	1228	251935	253163
GQO_T	Fri	2	1197	247221	248418
RO	Fri	2	20	378845	378865
CIO	Fri	2	393864	238261	632125
GQO	Fri	2	1206	247221	248427
GQO_c	Fri	2	1198	251839	253037
GQO_T	Fri	2	1191	258639	259830
RO	Fri	2	20	305741	305761
CIO	Sat	2	392673	233222	625895
GQO	Sat	2	1166	243788	244954
GQO_c	Sat	2	1151	248623	249774
GQO_T	Sat	2	1166	243788	244954
RO	Sat	2	20	277391	277411
CIO	Sat	2	393750	233222	626972
GQO	Sat	2	1170	244115	245285
GQO_c	Sat	2	1160	245229	246389
GQO_T	Sat	2	1150	243788	244938
RO	Sat	2	19	301873	301892
CIO	Sat	2	391793	233222	625015
GQO	Sat	2	1166	243788	244954
GQO_c	Sat	2	1157	245229	246386
GQO_T	Sat	2	1154	243788	244942
RO	Sat	2	19	351631	351650
CIO	Sun	3	399017	135545	534562
GQO	Sun	3	1264	208594	209858
GQO_c	Sun	3	1235	231890	233125
GQO_T	Sun	3	1242	238474	239716
RO	Sun	3	19	232030	232049
CIO	Sun	3	397724	135545	533269
GQO	Sun	3	1241	206038	207279
GQO_c	Sun	3	1239	228975	230214
GQO_T	Sun	3	1231	234422	235653
RO	Sun	3	20	242425	242445

Optimizer	Date	Query	QOT (ms)	QET (ms)	Overall (ms)
CIO	Sun	3	398019	135545	533564
GQO	Sun	3	1241	208594	209835
GQO_c	Sun	3	1236	231230	232466
GQO_T	Sun	3	1233	233763	234996
RO	Sun	3	18	240280	240298
CIO	Mon	3	398003	106847	504850
GQO	Mon	3	1325	209933	211258
GQO_c	Mon	3	1754	233331	235085
GQO_T	Mon	3	1335	239284	240619
RO	Mon	3	19	239189	239208
CIO	Mon	3	397466	106847	504313
GQO	Mon	3	1311	209933	211244
GQO_c	Mon	3	1351	233239	234590
GQO_T	Mon	3	1338	238938	240276
RO	Mon	3	19	245253	245272
CIO	Mon	3	399399	106847	506246
GQO	Mon	3	1345	200055	201400
GQO_c	Mon	3	1304	237109	238413
GQO_T	Mon	3	1324	239375	240699
RO	Mon	3	19	219895	219914
CIO	Tue	3	397061	114446	511507
GQO	Tue	3	1348	211295	212643
GQO_c	Tue	3	1296	235895	237191
GQO_T	Tue	3	1294	237502	238796
RO	Tue	3	19	258411	258430
CIO	Tue	3	398727	114446	513173
GQO	Tue	3	1304	202376	203680
GQO_c	Tue	3	1320	231846	233166
GQO_T	Tue	3	1297	243442	244739
RO	Tue	3	19	236858	236877
CIO	Tue	3	397678	114446	512124
GQO	Tue	3	1335	211295	212630
GQO_c	Tue	3	1292	231846	233138
GQO_T	Tue	3	1289	232233	233522
RO	Tue	3	18	229433	229451
CIO	Wed	3	396887	148053	544940
GQO	Wed	3	1261	219049	220310
GQO_c	Wed	3	1252	240385	241637
GQO_T	Wed	3	1253	242830	244083
RO	Wed	3	19	223203	223222
CIO	Wed	3	397036	148053	545089
GQO	Wed	3	1299	219049	220348
GQO_c	Wed	3	1253	236903	238156
GQO_T	Wed	3	1259	235803	237062
RO	Wed	3	20	230253	230273
CIO	Wed	3	398284	148053	546337

Optimizer	Date	Query	QOT (ms)	QET (ms)	Overall (ms)
GQO	Wed	3	1317	236903	238220
GQO_c	Wed	3	1265	240385	241650
GQO_T	Wed	3	1253	235803	237056
RO	Wed	3	18	249594	249612
CIO	Thu	3	396893	148509	545402
GQO	Thu	3	1281	219797	221078
GQO_c	Thu	3	1231	246977	248208
GQO_T	Thu	3	1236	251626	252862
RO	Thu	3	19	266860	266879
CIO	Thu	3	397211	148509	545720
GQO	Thu	3	1289	219797	221086
GQO_c	Thu	3	1231	248200	249431
GQO_T	Thu	3	1240	251626	252866
RO	Thu	3	18	251315	251333
CIO	Thu	3	397907	148509	546416
GQO	Thu	3	1242	219797	221039
GQO_c	Thu	3	1298	248750	250048
GQO_T	Thu	3	1238	222240	223478
RO	Thu	3	19	244435	244454
CIO	Fri	3	396874	147986	544860
GQO	Fri	3	1268	218372	219640
GQO_c	Fri	3	1288	235647	236935
GQO_T	Fri	3	1247	245748	246995
RO	Fri	3	19	241933	241952
CIO	Fri	3	397106	147986	545092
GQO	Fri	3	1265	235015	236280
GQO_c	Fri	3	1281	239540	240821
GQO_T	Fri	3	1255	218372	219627
RO	Fri	3	18	243549	243567
CIO	Fri	3	397916	147986	545902
GQO	Fri	3	1267	218276	219543
GQO_c	Fri	3	1257	235015	236272
GQO_T	Fri	3	1279	225238	226517
RO	Fri	3	18	262778	262796
CIO	Sat	3	396641	147706	544347
GQO	Sat	3	1230	214304	215534
GQO_c	Sat	3	1259	239255	240514
GQO_T	Sat	3	1211	235651	236862
RO	Sat	3	18	225198	225216
CIO	Sat	3	396589	147706	544295
GQO	Sat	3	1223	214304	215527
GQO_c	Sat	3	1220	239255	240475
GQO_T	Sat	3	1252	236283	237535
RO	Sat	3	19	249142	249161
CIO	Sat	3	397800	147706	545506
GQO	Sat	3	1227	213977	215204

Optimizer	Date	Query	QOT (ms)	QET (ms)	Overall (ms)
GQO_c	Sat	3	1216	236307	237523
GQO_T	Sat	3	1217	241762	242979
RO	Sat	3	20	236424	236444
CIO	Sun	4	49301	142167	191468
GQO	Sun	4	999	355185	356184
GQO_c	Sun	4	999	329965	330964
GQO_T	Sun	4	997	338594	339591
RO	Sun	4	14	335733	335747
CIO	Sun	4	50567	142167	192734
GQO	Sun	4	1033	316164	317197
GQO_c	Sun	4	994	316164	317158
GQO_T	Sun	4	991	316164	317155
RO	Sun	4	16	361895	361911
CIO	Sun	4	49360	142167	191527
GQO	Sun	4	1055	355185	356240
GQO_c	Sun	4	992	316164	317156
GQO_T	Sun	4	988	351045	352033
RO	Sun	4	14	378501	378515
CIO	Mon	4	49251	137693	186944
GQO	Mon	4	1064	364811	365875
GQO_c	Mon	4	1057	344082	345139
GQO_T	Mon	4	1056	287404	288460
RO	Mon	4	16	387010	387026
CIO	Mon	4	49403	137693	187096
GQO	Mon	4	1066	299505	300571
GQO_c	Mon	4	1057	287404	288461
GQO_T	Mon	4	1067	308765	309832
RO	Mon	4	15	387114	387129
CIO	Mon	4	49386	137693	187079
GQO	Mon	4	1056	364811	365867
GQO_c	Mon	4	1094	287404	288498
GQO_T	Mon	4	1055	308765	309820
RO	Mon	4	14	395494	395508
CIO	Tue	4	50205	142087	192292
GQO	Tue	4	1046	296734	297780
GQO_c	Tue	4	1044	354932	355976
GQO_T	Tue	4	1042	314371	315413
RO	Tue	4	15	704622	704637
CIO	Tue	4	49427	142087	191514
GQO	Tue	4	1050	357716	358766
GQO_c	Tue	4	1042	371315	372357
GQO_T	Tue	4	1070	314371	315441
RO	Tue	4	15	301778	301793
CIO	Tue	4	49284	142087	191371
GQO	Tue	4	1103	296734	297837
GQO_c	Tue	4	1041	354932	355973

Optimizer	Date	Query	QOT (ms)	QET (ms)	Overall (ms)
GQO_T	Tue	4	1039	314371	315410
RO	Tue	4	14	492881	492895
CIO	Wed	4	49309	199275	248584
GQO	Wed	4	1013	362452	363465
GQO_c	Wed	4	1009	363735	364744
GQO_T	Wed	4	1007	366719	367726
RO	Wed	4	15	366719	366734
CIO	Wed	4	49321	199275	248596
GQO	Wed	4	1011	363735	364746
GQO_c	Wed	4	1010	366719	367729
GQO_T	Wed	4	1012	590396	591408
RO	Wed	4	19	710049	710068
CIO	Wed	4	49298	199275	248573
GQO	Wed	4	1018	363735	364753
GQO_c	Wed	4	1046	370592	371638
GQO_T	Wed	4	1006	363735	364741
RO	Wed	4	14	371917	371931
CIO	Thu	4	50199	197091	247290
GQO	Thu	4	996	361566	362562
GQO_c	Thu	4	991	361566	362557
GQO_T	Thu	4	992	361566	362558
RO	Thu	4	14	385630	385644
CIO	Thu	4	49406	197091	246497
GQO	Thu	4	999	425984	426983
GQO_c	Thu	4	995	361566	362561
GQO_T	Thu	4	993	361566	362559
RO	Thu	4	15	387870	387885
CIO	Thu	4	49405	197091	246496
GQO	Thu	4	994	361566	362560
GQO_c	Thu	4	993	363972	364965
GQO_T	Thu	4	1024	370758	371782
RO	Thu	4	23	347960	347983
CIO	Fri	4	49254	200668	249922
GQO	Fri	4	1014	366922	367936
GQO_c	Fri	4	1002	353406	354408
GQO_T	Fri	4	1002	353197	354199
RO	Fri	4	16	366658	366674
CIO	Fri	4	49376	200668	250044
GQO	Fri	4	1015	353406	354421
GQO_c	Fri	4	1005	370891	371896
GQO_T	Fri	4	1009	353197	354206
RO	Fri	4	15	398291	398306
CIO	Fri	4	49941	200668	250609
GQO	Fri	4	1002	375150	376152
GQO_c	Fri	4	1002	353406	354408
GQO_T	Fri	4	1006	353406	354412

Optimizer	Date	Query	QOT (ms)	QET (ms)	Overall (ms)
RO	Fri	4	14	391347	391361
CIO	Sat	4	49577	195889	245466
GQO	Sat	4	975	353559	354534
GQO_c	Sat	4	980	333064	334044
GQO_T	Sat	4	968	362756	363724
RO	Sat	4	14	342976	342990
CIO	Sat	4	49342	195889	245231
GQO	Sat	4	984	353559	354543
GQO_c	Sat	4	981	333064	334045
GQO_T	Sat	4	974	333064	334038
RO	Sat	4	15	485164	485179
CIO	Sat	4	49305	195889	245194
GQO	Sat	4	980	333978	334958
GQO_c	Sat	4	979	333978	334957
GQO_T	Sat	4	981	353559	354540
RO	Sat	4	14	344891	344905

APPENDIX B

Query Optimization and Execution Times of Q_WD

Optimizer	Run	Query Optimization Time (ms)	Query Execution Time (sec)
CIO	1	36375	998
CIO	2	34062	952
CIO	3	31265	1020
CIO	4	34875	1011
CIO	5	34203	997
GOG	1	734	515
GOG	2	563	473
GOG	3	532	478
GOG	4	547	487
GOG	5	562	487

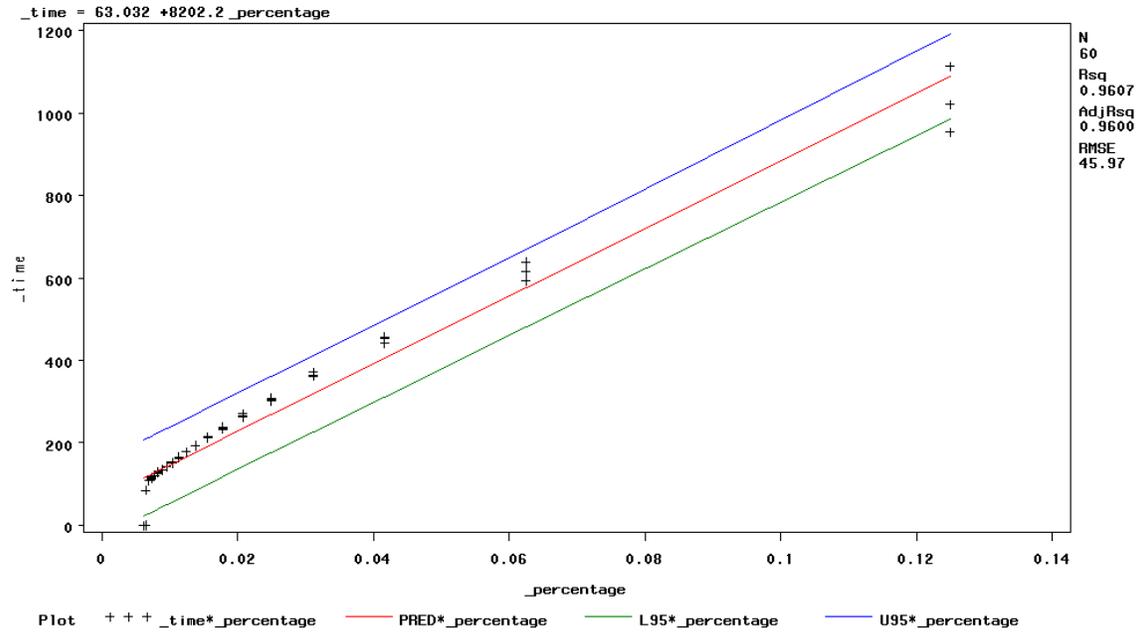
Query Optimization and Execution Times of Q_CNT

Optimizer	Run	Query Optimization Time (ms)	Query Execution Time (sec)
GOG	1	688	475
GOG	2	532	470
GOG	3	578	475
GOG	4	657	475
GOG	5	563	478
CIO	1	35390	936
CIO	2	37813	933
CIO	3	34390	962
CIO	4	33390	939
CIO	s5	33109	968

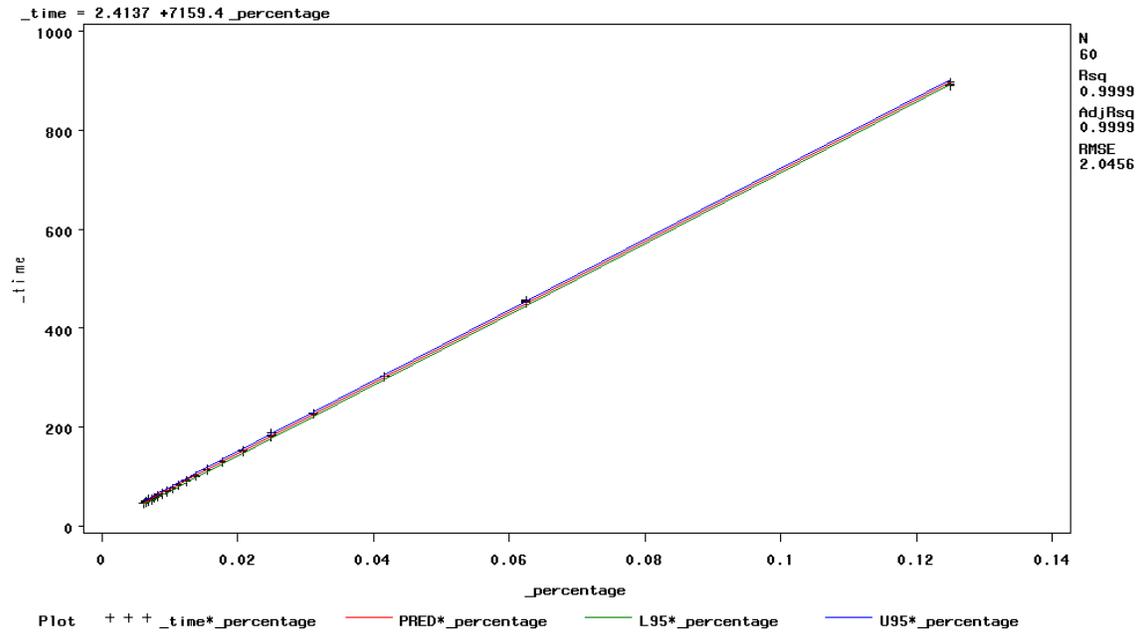
APPENDIX C

Linear Regressions on WD Running Times in hosts GIS22 and GIS23

GIS22

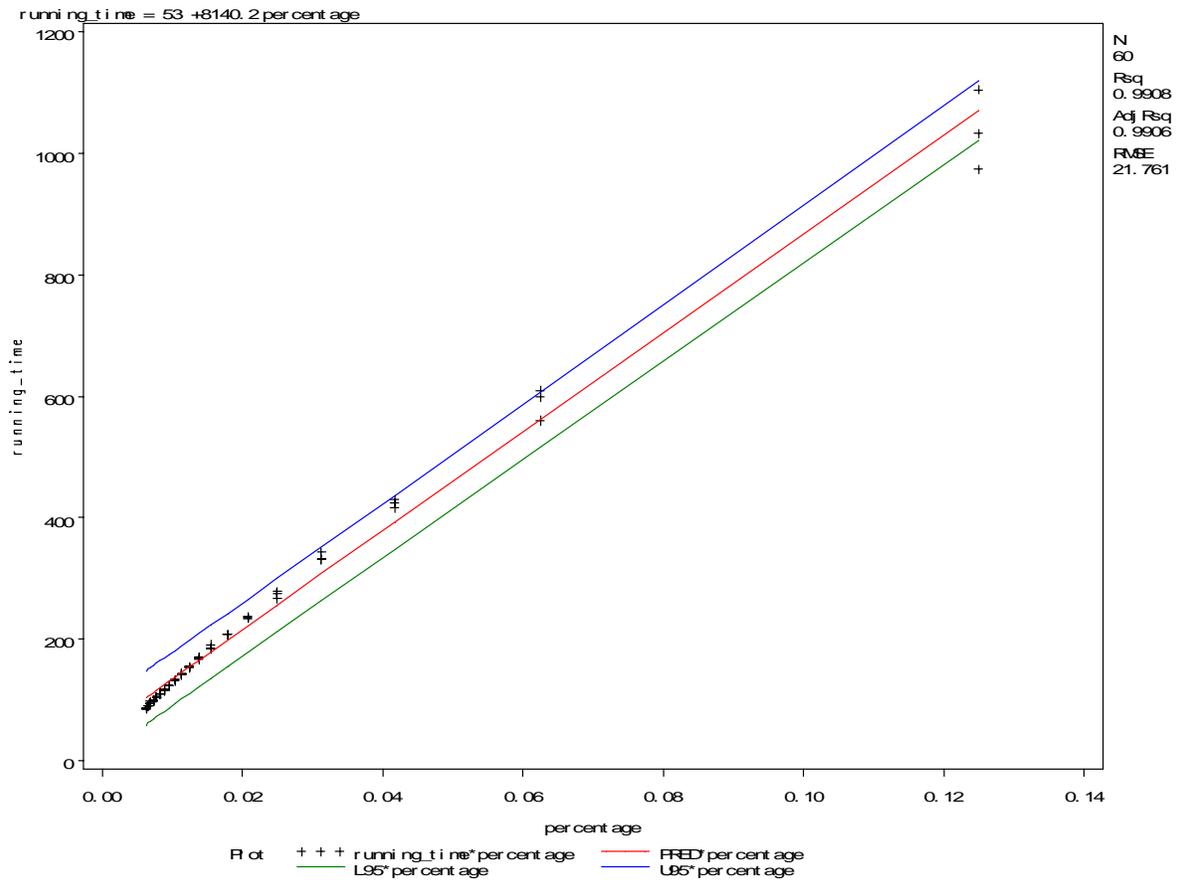


GIS23

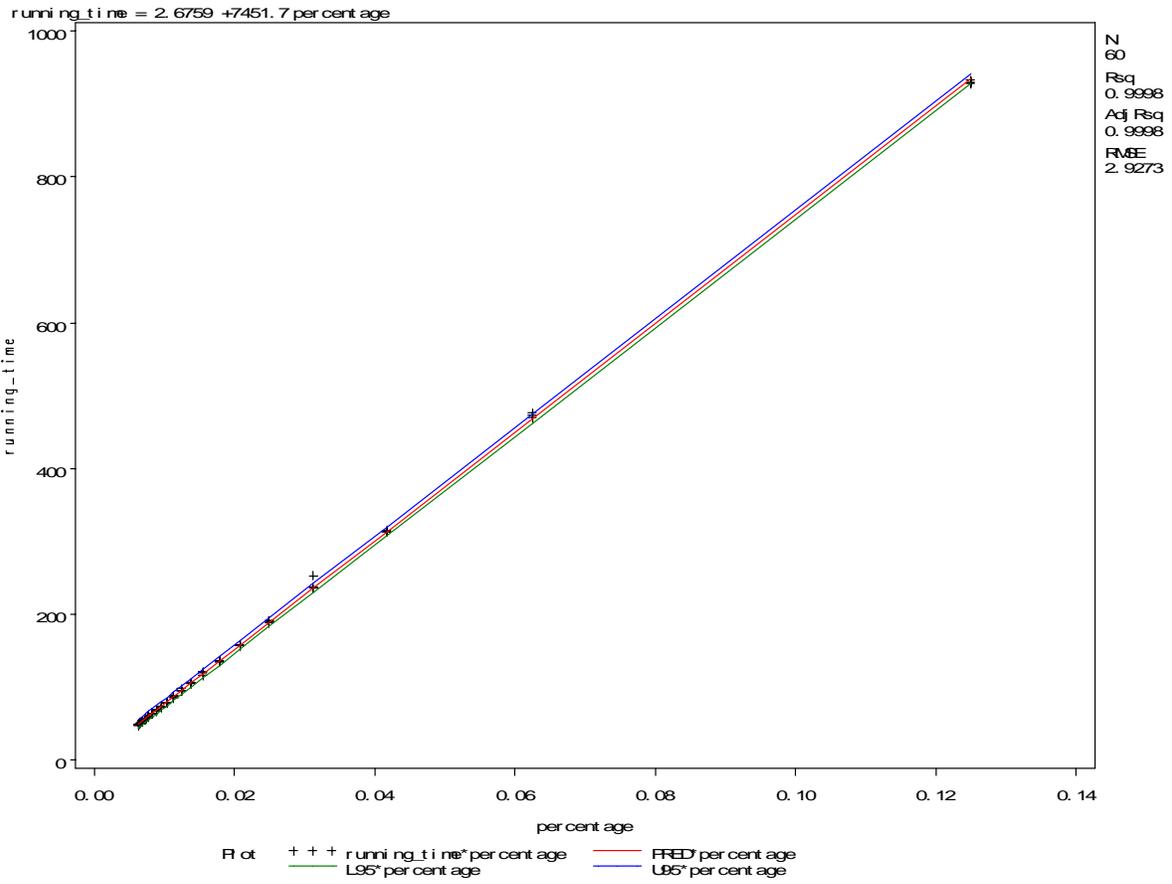


Linear Regressions on CNT Running Times in hosts GIS22 and GIS23

GIS22

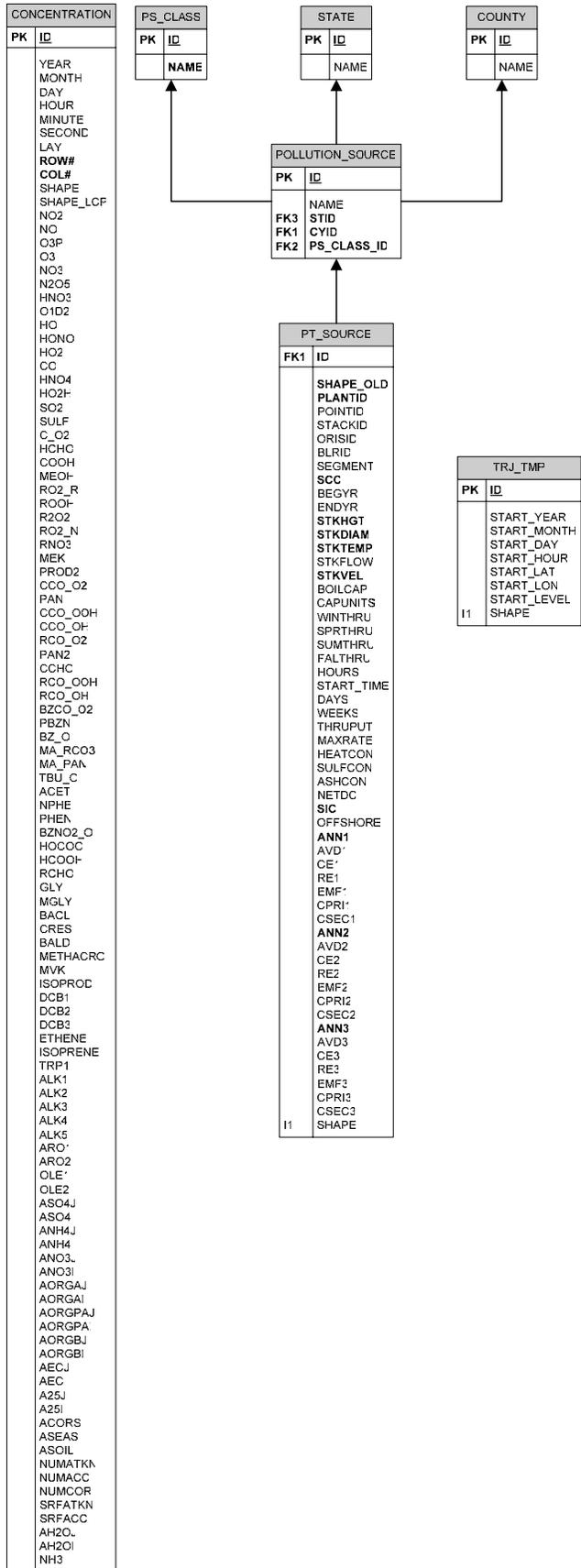


GIS23



APPENDIX D

Entity-Relationship Diagram of Air Emission Database



APPENDIX E

Sample Execution Plans for Q_WD

An execution plan generated by GOG:

```
Step 1:
Parallel spatial joins:
Join 1:
Destination host: gis22, table to be partitioned: pt_source, spatial
operation: SDO_WITHIN_DISTANCE, resultant table: tmp_tab_0.
Partition 1: trj_tmp@gis23, pt_source(id from 1 to 261515)@gis23->
pt_source_tmp_part@(joining host)gis23, resultant relation: tmp_tab_0_part_0
Partition 2: trj_tmp@gis22, pt_source(id from 261516 to 523031)@gis22->
pt_source_tmp_part@(joining host)gis22, resultant relation: tmp_tab_0_part_1

Step 2:
Regular 2-way joins:
Joint 1: (tmp_tab_0@gis22^pollution_source@gis22->tmp_tab_1@gis22)
```

An execution plan generated by CIO:

```
Step 0: (pt_source@gis22^trj_tmp@gis22->tmp_tab_0@gis22)
Step 1: (pollution_source@gis22^tmp_tab_0@gis22->tmp_tab_1@gis22)
```

BIBLIOGRAPHY

- Berman, F. (1999). "High-Performance Schedulers." *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, eds., Morgan Kaufmann Publishers, Inc., San Francisco, CA, 279-309.
- Berman, F., and Wolski, R. (1997). "The AppLeS Project: A Status Report." *8th NEC Research Symposium*, 1997, Berlin, Germany.
- Birrell, A., and Nelson, B. (1984). "Implementing Remote Procedure Calls." *ACM Transaction on Computer System*, 2(1), 39-59.
- Booth, D., et al. (2003). "Web Services Architecture, W3C Working Draft 8 August 2003." World Wide Web Consortium (W3C).
- Burrough, P. A., and McDonnell, R. A. (1998). *Principles of Geographical Information Systems*, Oxford University Press, Oxford, U.K.
- Butler, R., et al. (2000). "Design and Deployment of a National-Scale Authentication Infrastructure." *IEEE Computer*, 33(12), 60-66.
- Callaghan, B. (2000). *NFS Illustrated*, Addison-Wesley, Reading, MA.
- Casanova, H. (2002). "Distributed Computing Research Issues for Grid Computing." *Quarterly Newsletter for the ACM Special Interest Group on Algorithms and Computation Theory (SIGACT News)*, 33(2).
- Chapin, S. J., et al. (1999). "Resource Management in Legion." *Future Generation Computer Systems*, 15(5-6), 583-594.

- Crockett, T. W. (1998). "Digital Earth: A New Framework for Geo-referenced Data." *Institute for Computer Applications in Science and Engineering Research Quarterly*, 7(4).
- Czajkowski, K., et al. (2001). "Grid Information Services for Distributed Resource Sharing." *the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), August 2001*, 2001.
- Czajkowski, K., et al. (1998). "A Resource Management Architecture for Metacomputing Systems." *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998.*, 1998, pg. 62-82.
- Dierks, T., and Allen, C. (1999). "The TLS Protocol Version 1.0, IETF, RFC 2246." <http://www.ietf.org/rfc/rfc2246.txt>.
- Dijkstra, E. W. (1959). "A Note on Two Problems in Connexion with Graphs." *Numerische Mathematik*, 1, 269-271.
- Duda, R. O., et al. (2000). *Pattern Classification*.
- e-Science. (2004). "The e-Science Project." <http://www.escience-grid.org.uk>.
- Egenhofer, M., and Franzosa, R. (1991). "Point-Set Topological Spatial Relations." *International Journal of Geographical Information Systems*, 5(2), 161-174.
- EROS. (2003). "EROS: Earth Remote Observation System." Canada Centre of Remote Sensing. <http://www.ccrs.nrcan.gc.ca/ccrs/data/satsens/eros/erosteke.html>.
- Erwin, D., et al. (2002). "UNICORE Plus Final Report - Uniform Interface to Computing Resources." UNICORE Forum.
- Erwin, D. W., and Snelling, D. F. (2001). "UNICORE: A Grid Computing Environment." *Lecture Notes in Computer Science*, 2150, 825-834.
- Ferguson, D. F., et al. (1996). "Economic Models for Allocating Resources in Computer Systems." *Market-Based Control: A Paradigm for Distributed Resource Allocation*, S. Clearwater, ed., World Scientific, Hong Kong.

- Flynn, M. J., and Rudd, K. W. (1996). "Parallel Architectures." *ACM Computing Surveys (CSUR)*, 28(1), 67-70.
- Foster, I. (2002). "The Grid: A New Infrastructure for 21st Century Science." *Physics Today*, 55(2), 42, February, 2002.
- Foster, I., et al. (2003). "The Open Grid Services Architecture Platform." Global Grid Forum.
- Foster, I., and Kesselman, C. (1998a). "The Globus Project: A Status Report." *Heterogeneous Computing Workshop*, 1998a.
- Foster, I., and Kesselman, C. (1998b). *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, San Francisco, CA.
- Foster, I., et al. (2002). "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration." *Open Grid Service Infrastructure WG, Global Grid Forum*, 2002.
- Foster, I., et al. (1998). "A Security Architecture for Computational Grids." *ACM Conference on Computers and Security*, 1998, 83-91.
- Foster, I., et al. (2001). "The Anatomy of the Grid-Enabling Scalable Virtual Organizations." *International J. Supercomputer Applications*, 15(3).
- Frey, J., et al. (2002). "Condor-G: A Computation Management Agent for Multi-Institutional Grids." *Cluster Computing*, 5(3), 237-246.
- GGF. (2004). "The Global Grid Forum." <http://www.gridforum.org>.
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley.
- Graham, S., et al. (2002). *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*, SAMS Publishing.

- Grimshaw, A. S., et al. (2002). "From Legion to Avaki: The Persistence of Vision." *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman et al., eds., John Wiley and Sons Ltd., 265-298.
- Grimshaw, A. S., and Wulf, W. A. (1997). "The Legion Vision of A World-wide Virtual Computer." *Communications of the ACM*, 40(1), 39-45.
- Guttman, A. (1984). "R-trees: A Dynamic Index Structure for Spatial Searching." *ACM SIGMOD*, 1984.
- Hawick, K. A., et al. (2003). "Distributed Frameworks and Parallel Algorithms for Processing Large-scale Geographic Data." *Parallel Computing (Special issue: High performance computing with geographical data)*, 29(10), 1297 - 1333.
- Haynos, M. (2004). "Perspectives on Grid: Grid Computing -- Next-generation Distributed Computing." IBM. <ftp://www6.software.ibm.com/software/developer/library/gr-heritage.pdf>.
- Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, The MIT Press.
- Hovestadt, M. K., O., et al. (2003). "Scheduling in HPC Resource Management Systems: Queuing vs. Planning." *the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, 2003, 1–20.
- HYSPLIT. (2005). "NOAA ARL HYSPLIT Model." National Oceanic and Atmospheric Administration. <http://www.arl.noaa.gov/ready/hysplit4.html>.
- Ioannidis, Y. E., and Kang, Y. C. (1990). "Randomized Algorithms for Optimizing Large Join Queries." *the 1990 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 1990, Atlantic City, NJ, U.S.A, 312–321.
- Jensen, J. R. (1996). *Introductory Digital Image Processing*, Prentice Hall, Upper Saddle River, N.J.

- Karimi, H. A., et al. (2004). "Evaluation of Uncertainties Associated with Geocoding Techniques." *Computer-Aided Civil and Infrastructure Engineering*, 19(3), 170-185.
- Karimi, H. A., and Liu, S. (2004). "Developing an Automated Procedure for Extraction of Road Data from High-Resolution Satellite Images for Geospatial Information Systems." *Journal of Transportation Engineering, American Society of Civil Engineering (ASCE)*, 130(5), 621-631.
- Kistler, J. J. (1996). *Disconnected Operation in a Distributed File System, 1002*, Springer-Verlag, Berlin.
- Kistler, J. J., and Satyanarayanan, M. (1992). "Disconnected Operation in the Coda File System." *ACM Transactions on Computer Systems (TOCS)*, 10(1), 3-25.
- Knuth, D. E. (1997). *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Third Edition, Addison-Wesley, Reading, Massachusetts.
- Kossmann, D., et al. (2000). "Cache Investment: Integrating Query Optimization and Distributed Data Placement." *ACM Transactions on Database Systems (TODS)*, 25(4), 517-558.
- Kossmann, D., and Stocker, K. (2000). "Iterative Dynamic Programming: A New Class of Query Optimization Algorithms." *ACM Transactions on Database Systems (TODS)*, 25(1), 43-82.
- Krause, A., et al. (2002). "Grid Database Service Specification (version 0.2)." Global Grid Forum. <http://www.cs.man.ac.uk/grid-db/papers/DAIS:StatementSpec.pdf>.
- Kreveld, M. v. (1997). "Algorithms for Triangulated Terrains." *Conference on Current Trends in Theory and Practice of Informatics*, 1997, 19-36.
- Leymann, F. (2001). "Web Services Flow Language (WSFL 1.0)." IBM Software Group.
- Liu, S., and Karimi, H. (2004). "GeoPlan: A Novel Approach to Optimize Service-Oriented Distributed Geospatial Data Processing." *International Conference on Parallel and*

- Distributed Processing Techniques and Applications (PDPTA'04)*, 2004, Las Vegas, Nevada.
- Lo, C. P., and Yeung, A. K. W. (2002). *Concepts and Techniques of Geographic Information Systems*, Prentice Hall, Upper Saddle River, New Jersey.
- Mackert, L. F., and Lohman, G. M. (1986). "R* Optimizer Validation and Performance Evaluation for Distributed Queries." *the Twelfth International Conference on Very Large Data Bases*, 1986, Kyoto.
- McRobbie, M. A., et al. (2001). "A Global Terabit Research Network."
<http://www.gtrn.net/global.pdf>.
- Mitchell, W. B., et al. (1977). "GIRAS - a Geographic Information Retrieval and Analysis System for Handling Land Use and Land Cover Data." U.S. Geological Survey.
- Muntz, R. R., et al. (2003). *IT Roadmap to a Geospatial Future*, National Academies Press, Washington, D.C.
- Murray, C., et al. (2003). "Oracle Spatial User's Guide and Reference, 10g Release 1 (10.1)." Oracle Corporation.
- OGSA-WG. (2004). "The Open Grid Service Architecture Working Group."
<http://www.gridforum.org/ogsa-wg/>.
- Ozsu, M. T., and Valduriez, P. (1999). *Principles of Distributed Database Systems*, 2nd edition, Prentice-Hall, Inc., Upper Saddle River, New Jersey, USA.
- Pawloski, B., et al. (1994). "NFS Version 3: Design and Implementation." *USENIX Association Conference Proceedings*, 1994, 137-151.
- PlanetLab. (2005). "PlanetLab Platform." <http://www.planet-lab.org/>.
- Raman, R., et al. (1998). "Matchmaking: Distributed Resource Management for High Throughput Computing." *the Seventh IEEE International Symposium on High Performance Distributed Computing*, 1998.

- Rigaux, P., et al. (2002). *Spatial Databases with Applications to GIS*, Academic Press, San Diego, CA.
- Roure, D. D., et al. (2003). "The Evolution of the Grid." *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman et al., eds., John Wiley and Sons Ltd.
- Sandberg, R., et al. (1985). "Design and Implementation of The Sun Network File System." *Summer Technology Conference USENIX*, 1985, 119-130.
- Satyanarayanan, M. (1990). "Scalable, Secure, and Highly Available Distributed File Access." *Computer*, 23(5), 9 - 18, 20-21.
- Selinger, P. G., et al. (1979). "Access Path Selection in A Relational Database Management System." *ACM SIGMOD Conference on Management of Data*, 1979, Boston, MA, U.S.A, 23-24.
- SETI@Home. (2004). "The SETI@Home Project." <http://setiathome.ssl.berkeley.edu/>.
- Shi, Y., et al. (2002). "Grid Computing for Real Time Distributed Collaborative Geoprocessing." *International Symposium on GeoSpatial Theory, Processing and Applications*, 2002, Ottawa, Ontario, Canada.
- Sidell, J., et al. (1996). "Data Replication in Mariposa." *the 12th International Conference on Data Engineering*, 1996, New Orleans, LA, U.S.A, 485-494.
- Silberschatz, A., et al. (2002). *Database System Concepts*, 4th edition, McGraw-Hill.
- Smarr, L., and Catlett, C. (1992). "Metacomputing." *Communications of the ACM*, 35(6), 44-52.
- Steinbrunn, M., et al. (1997). "Heuristic and Randomized Optimization for the Join Ordering Problem." *VLDB Journal: Very Large Data Bases*, 6(3), 191--208.
- Stix, G. (2001). "The Triumph of the Light." *Scientific American*, 284(1), 80-85, January, 2001.
- Stonebraker, M., et al. (1996). "Mariposa: A Wide-Area Distributed Database System." *The VLDB Journal*, 5(1), 48-63.

- Streit, A., et al. (2005). "UNICORE - From Project Results to Production Grids." Grid Computing and New Frontiers of High Performance Processing, L. Grandinetti, ed., Elsevier.
- Swani, A. (1989). "Optimization of Large Join Queries: Combining Heuristics and Combinational Techniques." *the ACM Conference on Management of Data (SIGMOD)*, 1989, Portland, OR, U.S.A, 367–376.
- Swani, A., and Gupta, A. (1988). "Optimization of Large Join Queries." *the ACM SIGMOD Conference on Management of Data (SIGMOD)*, 1988, Chicago, IL, U.S.A, 8–17.
- Tanenbaum, S. A., and Steen, V. M. (2002). *Distributed Systems: Principles and Paradigms*, Pearson Education, Inc., Singapore.
- TeraGrid. (2004). "The TeraGrid Project." <http://www.teragrid.org>.
- Thain, D., et al. (2003). "Condor and the Grid." Grid Computing: Making The Global Infrastructure a Reality, F. Berman et al., eds., John Wiley, 299-335.
- Thatte, S. (2001). "XLANG: Web Services for Business Process Design." Microsoft Corporation.
- Tomlinson, R. (1998). "The Canada Geographic Information System." History of Geographic Information Systems, T. W. Foresman, ed., Prentice-Hall, 21-32.
- Wang, S., and Armstrong, M. P. (2003). "A Quadtree Approach to Domain Decomposition for Spatial Interpolation in Grid Computing Environments." *Parallel Computing Journal*, 29(10), 1481-1504.
- Wang, S., et al. (2002). "Using Grid-Enabled Teleimmersive Spatial Decision Support Systems (TIDSS) to Visualize Uncertainty for Water Quality Protection in Agroecosystems." *International Conference of Geoinformatics (2002)*, 2002, Nanjing, P.R. China.
- Wolfson, O., et al. (1997). "An Adaptive Data Replication Algorithm." *ACM Transactions on Database Systems*, 22(2), 255-314.

Wolski, R. (1997). "Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service." *6th IEEE Symp. on High Performance Distributed Computing*, 1997, Los Alamitos, CA, 316--325.

Wright, D. (2001). "Cheap Cycles from the Desktop to the Dedicated Cluster: Combining Opportunistic and Dedicated Scheduling with Condor." *Linux Clusters: The HPC Revolution*, 2001, Champaign-Urbana, IL, U.S.A.