Allocation of Class Fragments in Distributed Objectbase Systems

by

Subhrajyoti Bhar

A thesis
presented to the University of Manitoba
in partial fulfilment of the
requirements for the degree of
Doctor of Philosophy
in
Computer Science

Winnipeg, Manitoba, Canada, 1996

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

ISBN 0-612-12989-6

Canada

Name _____

Dissertation Abstracts International and *Masters Abstracts International* are arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation or thesis. Enter the corresponding four-digit code in the spaces provided.

COMPUTER SCIENCE
SUBJECT TERM

0984
SUBJECT CODE

UMI

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS
Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION
General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS
Language
- General 0679
- Ancient 0289
- Linguistics 0290
- Modern 0291

Literature
- General 0401
- Classical 0294
- Comparative 0295
- Medieval 0297
- Modern 0298
- African 0316
- American 0591
- Asian 0305
- Canadian (English) 0352
- Canadian (French) 0355
- English 0593
- Germanic 0311
- Latin American 0312
- Middle Eastern 0315
- Romance 0313
- Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY
Philosophy 0422
Religion
- General 0318
- Biblical Studies 0321
- Clergy 0319
- History of 0320
- Philosophy of 0322

Theology 0469

SOCIAL SCIENCES
American Studies 0323
Anthropology
- Archaeology 0324
- Cultural 0326
- Physical 0327

Business Administration
- General 0310
- Accounting 0272
- Banking 0770
- Management 0454
- Marketing 0338

Canadian Studies 0385
Economics
- General 0501
- Agricultural 0503
- Commerce-Business 0505
- Finance 0508
- History 0509
- Labor 0510
- Theory 0511

Folklore 0358
Geography 0366
Gerontology 0351
History
- General 0578
- Ancient 0579
- Medieval 0581
- Modern 0582
- Black 0328
- African 0331
- Asia, Australia and Oceania 0332
- Canadian 0334
- European 0335
- Latin American 0336
- Middle Eastern 0333
- United States 0337

History of Science 0585
Law 0398
Political Science
- General 0615
- International Law and Relations 0616
- Public Administration 0617

Recreation 0814
Social Work 0452
Sociology
- General 0626
- Criminology and Penology 0627
- Demography 0938
- Ethnic and Racial Studies 0631
- Individual and Family Studies 0628
- Industrial and Labor Relations 0629
- Public and Social Welfare 0630
- Social Structure and Development 0700
- Theory and Methods 0344

Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES
Agriculture
- General 0473
- Agronomy 0285
- Animal Culture and Nutrition 0475
- Animal Pathology 0476
- Food Science and Technology 0359
- Forestry and Wildlife 0478
- Plant Culture 0479
- Plant Pathology 0480
- Plant Physiology 0817
- Range Management 0777
- Wood Technology 0746

Biology
- General 0306
- Anatomy 0287
- Biostatistics 0308
- Botany 0309
- Cell 0379
- Ecology 0329
- Entomology 0353
- Genetics 0369
- Limnology 0793
- Microbiology 0410
- Molecular 0307
- Neuroscience 0317
- Oceanography 0416
- Physiology 0433
- Radiation 0821
- Veterinary Science 0778
- Zoology 0472

Biophysics
- General 0786
- Medical 0760

EARTH SCIENCES
Biogeochemistry 0425
Geochemistry 0996
Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES
Environmental Sciences 0768
Health Sciences
- General 0566
- Audiology 0300
- Chemotherapy 0992
- Dentistry 0567
- Education 0350
- Hospital Management 0769
- Human Development 0758
- Immunology 0982
- Medicine and Surgery 0564
- Mental Health 0347
- Nursing 0569
- Nutrition 0570
- Obstetrics and Gynecology 0380
- Occupational Health and Therapy 0354
- Ophthalmology 0381
- Pathology 0571
- Pharmacology 0419
- Pharmacy 0572
- Physical Therapy 0382
- Public Health 0573
- Radiology 0574
- Recreation 0575
- Speech Pathology 0460
- Toxicology 0383

Home Economics 0386

PHYSICAL SCIENCES
Pure Sciences
Chemistry
- General 0485
- Agricultural 0749
- Analytical 0486
- Biochemistry 0487
- Inorganic 0488
- Nuclear 0738
- Organic 0490
- Pharmaceutical 0491
- Physical 0494
- Polymer 0495
- Radiation 0754

Mathematics 0405
Physics
- General 0605
- Acoustics 0986
- Astronomy and Astrophysics 0606
- Atmospheric Science 0608
- Atomic 0748
- Electronics and Electricity 0607
- Elementary Particles and High Energy 0798
- Fluid and Plasma 0759
- Molecular 0609
- Nuclear 0610
- Optics 0752
- Radiation 0756
- Solid State 0611

Statistics 0463

Applied Sciences
Applied Mechanics 0346
Computer Science 0984

Engineering
- General 0537
- Aerospace 0538
- Agricultural 0539
- Automotive 0540
- Biomedical 0541
- Chemical 0542
- Civil 0543
- Electronics and Electrical 0544
- Heat and Thermodynamics 0348
- Hydraulic 0545
- Industrial 0546
- Marine 0547
- Materials Science 0794
- Mechanical 0548
- Metallurgy 0743
- Mining 0551
- Nuclear 0552
- Packaging 0549
- Petroleum 0765
- Sanitary and Municipal 0554
- System Science 0790

Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY
General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

THE UNIVERSITY OF MANITOBA

FACUTY OF GRADUATE STUDIES

COPYRIGHT PERMISSION

ALLOCATION OF CLASS FRAGMENTS IN DISTRIBUTED OBJECTBASE SYSTEMS

BY

SUBHRAJYOTI BHAR

A Thesis/Practicum submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Manitoba requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

Distributed Object-Based Systems (DOBSs) are an area of intense research in both the academic and applied research environments. A DOBS adds the benefit of distributed computing and the capability of object models for abstracting and manipulating information to provide a powerful environment for distributed processing. Application performance and processing cost in a DOBS are greatly influenced by communication overhead involved in accessing nonlocal data and making remote method invocations. Efficient distribution design is mandatory to ensure optimal performance of the distributed system at minimum cost. Distribution design in DOBS involves fragmentation of the objectbase and allocation of the resulting class fragments between the nodes of the network.

A top-down approach is adopted for the distribution design for a DOBS and it is assumed that the global conceptual schema is partitioned into a set of class fragments. The focus of this dissertation is the design of a scheme for allocating the fragments between the sites of the network subject to a set of constraints.

A general allocation taxonomy is defined and different DOBS allocation models are identified based on their data model, degree of redundancy, and design objective. A static nonredundant cost model for DOBS is formulated and two different approaches are presented for the proposed model. Allocation algorithms are designed based on the two approaches that use heuristic technique to obtain a near optimal allocation from an initial solution. The algorithms are implemented and test results are analyzed to obtain a comparative study of the performance of the two approaches and to obtain useful insights for DOBS allocation.

Acknowledgements

I am grateful to my supervisor Dr. Ken Barker for his relentless support and encouragement for the entire duration of this research work. He was easy and flexible to work with which gave me the freedom to pursue my research work in my own way. He was open to all kinds of ideas that had given me the opportunity to participate into many meaningful discussions with him at different times during this research work.

I express my sincere gratitude to my committee members Dr. Lawrence Saxton, Dr. A. S. Alfa and Dr. Mark Evans for their valuable insights, comments and suggestions and for their time and patience to read this thesis. Many thanks also go to the members of the research group at the Advanced Database Systems Laboratory of the University of Manitoba. The aggressive sessions of research group discussions provided several interesting suggestions and criticisms and were source of constant encouragement.

I am deeply indebted to my parents, who were constant source of encouragement from the other side of the globe. I am also thankful to my wife Tumpa for her love, understanding and patience.

I would also like to express my gratitude to the faculty and the support staffs of the department of computer science of the University of Manitoba for their co-operation and friendly assistance.

Finally, I thank my parents again for their love and sacrifice that gave me the strength to make this dream a reality. This thesis is dedicated to them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Optimal distribution of data and programs across a computer network is a major design problem that directly impacts performance. An optimal distribution design enhances application performance by reducing communication overhead due to references to non-local programs or data in a cost efficient way. Distribution design involves two logical steps: (1) fragmentation and (2) allocation. Fragmentation refers to clustering programs and data into groups or "fragments" based on *locality of access* exhibited by application access behavior. The resulting fragments are distributed across the network using an allocation scheme. The problem of allocation has been examined for distributed file systems (DFS) [LL83, GS90, Wah84, Chu69, DF82] and for distributed (relational) database systems (DDBS) [ÖV91, CP84, MR76, CY89, Ape88, CNW83]. The additional complexity in the distribution design introduced by the object model has only recently been investigated with no results appearing that address the specific problem of *allocation in Distributed Objectbase System.* Benefits obtained from an optimal allocation scheme in a distributed system include:

- Placing fragments "close to" the site where they are used results in significant reduction in communication cost due to intersite communication across the network. Minimizing network communication also reduces network delay which enhances system performance (ie. throughput and response time).

- Level of concurrency can be increased by allowing fragment replication which in turn improves throughput and response time. Further, copies of fragments placed at different sites improves fragment availability and system reliability.

- Using some load balancing techniques, utilization of shared resources (eg. processors, communication channels, I/O devices *etc.*) can be improved and congestion avoided or minimized by distributing the "load" across the network.

An *objectbase* is a collection of objects grouped into a finite number of classes. Objects with common attributes and methods belong to the the same class. Encapsulation bundles together attributes (data values) and the methods (procedures) that manipulate them. Inheritance allows reuse and incremental redefinition of new classes in terms of existing ones. A distributed objectbase system (DOBS) is a collection of local objectbases located at different local sites, interconnected by a communication network [EB94].

With the broad objective of providing an optimal distribution design for a persistent DOBS storage, this research contributes in the following ways:

1. Provides a classification of allocation models for different distributed systems based on an allocation taxonomy and identifies the characteristic features of the allocation models in DOBS.

2. Formulates a static nonredundant cost model for DOBS allocation and presents two different approaches for designing an optimal allocation in a DOBS. While the graphical approach is based on a heuristic graph partitioning technique, the other approach uses simulated annealing as the optimizing scheme.

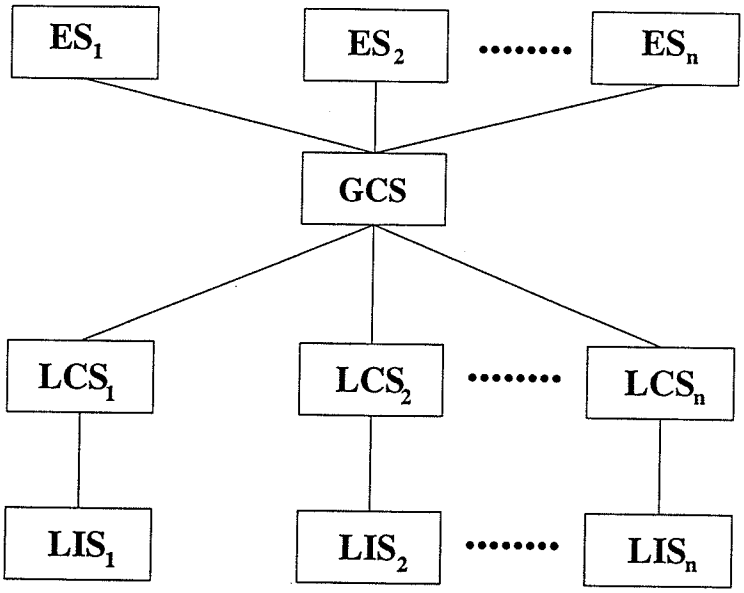

Figure 1.1: DOBS Architecture (Adapted from [OV91])

3. Provides algorithms for both the approaches and reports the results obtained by them.

4. Presents a comparative analysis of the two approaches based on the results obtained by application of the proposed algorithms on a simulated DOBS prototype.

1.2 The DOBS Architecture

This section outlines the distributed system architecture that defines the environment for which the DOBS allocation models are designed.

A distributed objectbase system (DOBS) is a collection of local objectbases, one at every site. The sites are interconnected by a communication network. At every site the *local conceptual schema (LCS)* defines the logical organization of class objects used by applications executing at local sites. Each *LCS* is mapped to a *logical internal schema (LIS)* which defines the physical organization of class objects stored at local site. The

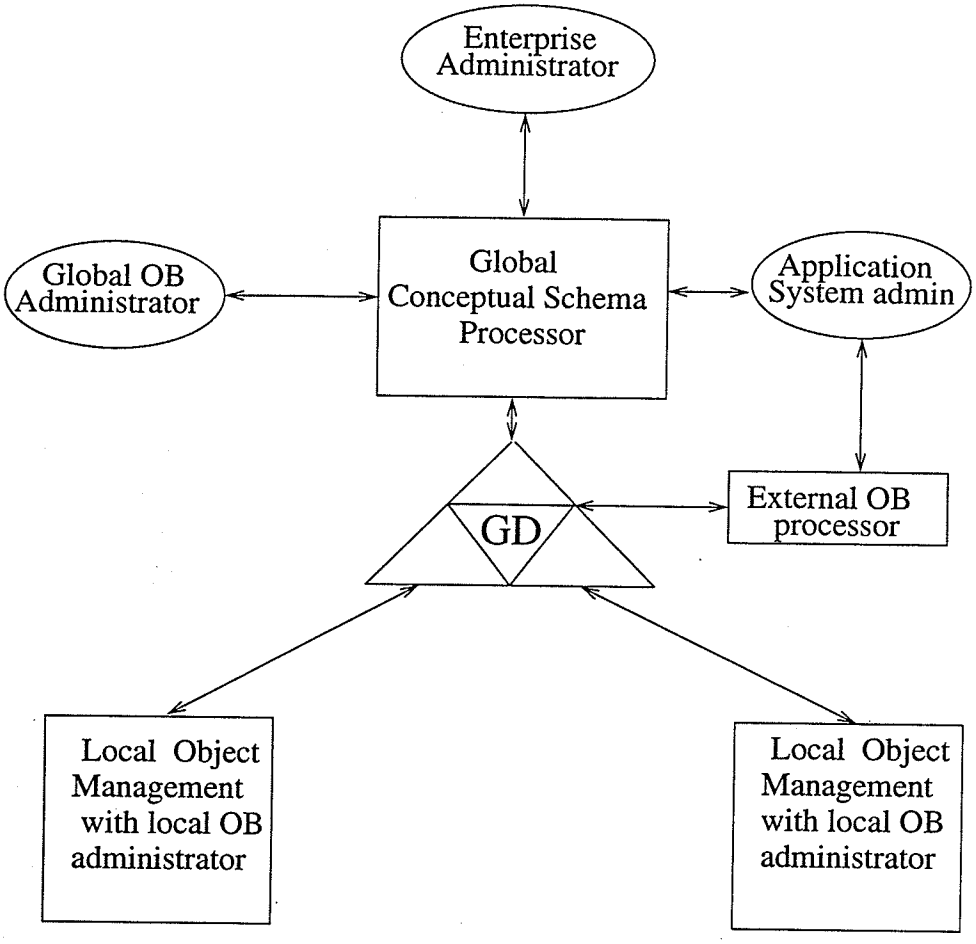

Figure 1.2: Functional Components of a DOBMS (Adapted from [OV91])

global objectbase, which is the aggregation of all local objectbases, is defined by a *global conceptual schema.* Applications executing at every local site view the organization of class objects as defined by the *external schema* at that site. The DOBS architecture is summarized in Figure 1.1.

A *global directory (GD)* provides global mapping between different organizational views of class objects distributed across the network and provides location transparency to applications. The primary functional components of the distributed objectbase management system (DOBMS) are shown in Figure 1.2 [EB94].

The following assumptions frame this research effort:

- The objectbase is partitioned into a finite set of class fragments. Each fragment has a finite size which is the sum of the sizes of its constituent objects/attributes/methods.

  The class fragments are generated by applying some fragmentation scheme to every class in the objectbase. The fragmentation scheme considers the application access pattern of the objectbase to cluster those object instances/attributes/methods together that exhibit high affinity with respect to application access. The proposed allocation algorithms are independent of fragmentation type (horizontal[1], vertical[2], or hybrid[3]).

- The fragmentation algorithm that generates class fragments is correct so the class fragments are disjoint, complete, and reconstructible [EB93].

- The distributed environment is homogeneous with respect to data models for the local objectbases.

- The network provides reliable and fault-free communication between sites by message passing. The network topology and the routing scheme do not change with time.

- The set of applications executing at different nodes of the distributed system does not change with time

- Applications do not migrate between sites and the frequency of executing any application at any particular site is constant.

- Application behavior does not change with time and therefore information obtained from the static analysis of the objectbase during the fragmentation stage of distribution design is valid input to the allocation process.

[1] Horizontal fragments are generated by partitioning the set of instance objects of a class.

[2] Vertical fragments are generated by partitioning the set of attributes and methods of a class.

[3] Hybrid fragments are the result of application of horizontal and vertical fragmentation in any order.

- A static objectbase environment is assumed. Therefore, class fragments or class objects do not migrate between classes or between sites and no new class is added or any existing class deleted.
- There exists a global object directory that locates objects referenced by user requests.

1.2.1 The Object Model

The objectbase is a collection of encapsulated objects. An object contains a set (possibly empty) of attributes (data values) that defines its state and a set (possibly empty) of methods[4] (procedures) to access and manipulate them. The attributes and methods are bundled together to form an encapsulated object. Any access to the attributes of an object is permitted only by executing any of the *public* methods belonging to that object. Objects are grouped into classes. Objects belonging to a class have in common all attributes and methods defined for that class. Inheritance allows object in "different" classes to share methods. Thus 'subclasses' are generated by inheriting attributes and/or methods from one or more (in the case of multiple inheritance) of other classes (called superclasses). However, a subclass may contain additional attributes and methods that are not inherited from any other class. The overall inheritance hierarchy of the objectbase is captured in a *class lattice* with a 'root class' as the ancestor of every other class. Each class has a class identifier *Cid* and every object has a globally unique object identifier *Oid*. A class is represented as a 4-tuple $< Cid, A, M, I >$, where Cid is the class identifier, A and M are the set of attributes and the set of methods defined for the class and I is the set of instance objects belonging to the class. A class object is represented as $< Cid, Oid, A, M >$, where Cid is the class identifier of the class containing the object, Oid is the globally unique object identifier, A is the set of instance attributes and M is the set of methods defined for the containing class. The set of methods are the only

[4] Methods are also known as behaviors.

means to access and manipulate the attribute instances[5].

The objectbase is partitioned into a set of class fragments by a fragmentation scheme. The fragmentation algorithm is applied to every class of the objectbase to generate "class fragments" for that class [EB93, EB94]. The fragments generated are correct with respect to disjointness, completeness, and reconstructibility. However, the type of fragmentation applied is orthogonal to the allocation problem.

Therefore, the research problem addressed in this dissertation can be stated as: Given a set of class fragments of finite size, a communication network with interconnected nodes, each having its own storage capacity and processing power and a set of applications executing at different sites, design an allocation scheme for optimally distributing the fragments between the nodes of the network so that the operating cost is minimum and performance requirements are met.

1.3 Motivation

The problem of allocation has been addressed for distributed file systems [LL83, GS90, Wah84, Chu69, DF82] and distributed relational database systems [ÖV91, CP84, MR76, CY89, Ape88, CNW83]. Previous research work demonstrating the difficulties in obtaining an optimal allocation leads to computational intractable solutions [Esw74]. The problem is formulated as a 0/1 integer programming problem which is NP Complete [GJ79]. "Near optimal" solutions are obtained in most cases by applying heuristic techniques based on the nature of the specific problem.

Distributed file system solutions are not applicable in distributed database environment because the data is more tightly integrated and is managed by a single system responsible for the entire database. File allocation means individual files are independent units of distribution. Applications do not see strong relationships between files so

[5]This definition is adapted from [EB94].

modeling the applications' behavior in a distributed file system is comparatively simple.

A file access in a distributed environment refers to either a remote access or a local read/write depending on the location of the file. Therefore, the allocation model only considers accesses to a single file so multi-file accesses required in a distributed database are ignored (consider, for example, a relational join). Therefore, files are allocated independently so only resource and performance constraints need to be considered in the model.

Database and object-oriented systems introduce additional complexity. Since relational database system transactions often involves operations on several relations, possibly located at different sites, the modeling of application behavior is more complex. Application behavior is typically modeled by determining the portions of the data accessed together. This behavior is reflected in the formation of *fragments* where data are closely affined[6]. Fragmentation can occur horizontally (collecting together groups of tuples accessed together), vertically (collecting together attributes that are accessed together), or hybrid (applying horizontal and vertical fragmentation) [ÖV91, EB93, EB94].

Object-oriented system are more complex because of the inherent relationships between objects. The object data model is capable of capturing relationships of arbitrary complexity so modeling their relationships and interactions is very difficult. Further, the problem lies in the class of NP hard[7] problems having runtime complexity analogous to the file allocation problem or the database allocation problem. Karlapalem, Navathe and Morsi [KNM92] identify the difference between the relational and the object-oriented environment in terms of interactions between data and methods. In a distributed database, the data and programs that access them are disjoint so the allocation scheme applies to data only. An object-oriented environment must account for the encapsulation of methods and the data referenced, so method invocation plays a major role in allocating objects.

[6] Affinity is essentially the likelihood and/or frequency that an application will reference two attributes or tuples.

[7] An optimization problem is NP hard if its corresponding decision problem is NP Complete.

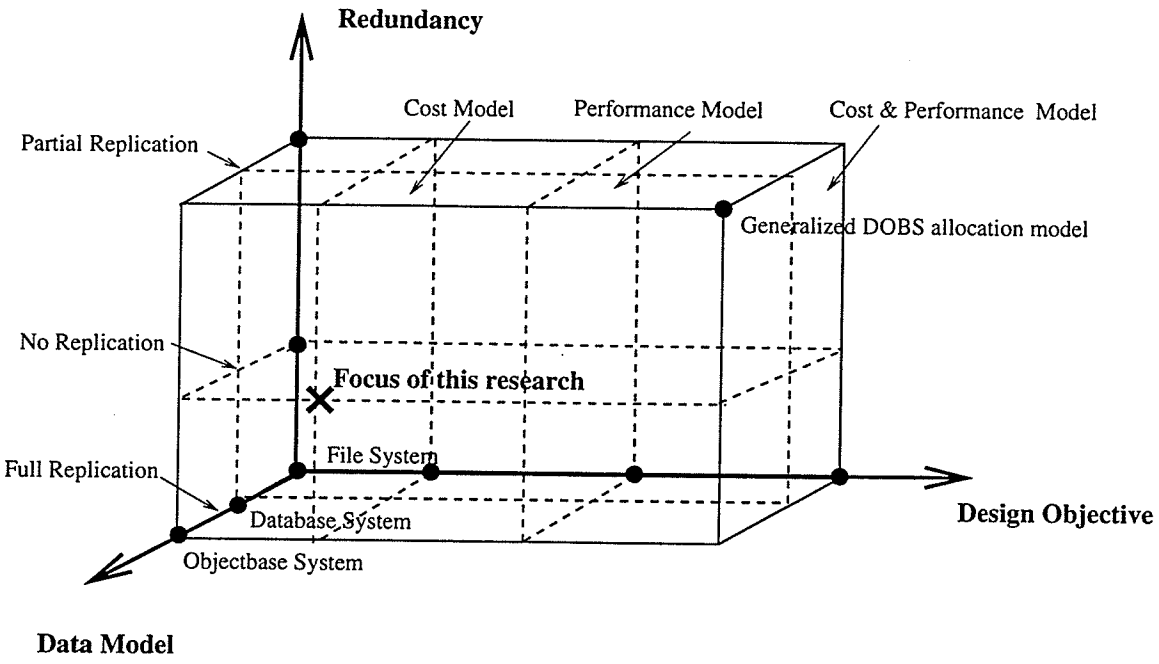

Figure 1.3: A Taxonomy for Allocation Models

Hence, the allocation problem needs to address the placement of data and method *together*. Moreover, as a method in one object can invoke another method in the same or a different object, the dependencies between methods must also be considered.

The distributed object model must consider all relevant constraints to achieve a complete solution to the allocation problem.

1.4 A Taxonomy for Allocation

This section defines a taxonomy of allocation models for distributed systems that support persistence. The taxonomy illustrated in Figure 1.3 identifies the issues considered when analyzing allocation schemes in a distributed system and provides a guideline for classifying different allocation models. The characterizing parameters of an allocation model are: (1) type of data model, (2) degree of redundancy and (3) design objective. A brief

description of the scope of each of these parameters and their influence in the allocation design is presented below.

- Type of Data Model: A data model defines the data structures and the operations to be performed on them. Design of an allocation scheme is driven by the way data entities are structured and the way they are accessed by applications. While designing a distributed system, a particular data model is selected as the 'best choice' with respect to intended operations to be performed on data. Therefore, application behavior information is inherently captured in the selected data model. The goal of an allocation scheme is to distribute data and programs to facilitate application execution. Thus a strong relationship exists between an allocation model and the data model for which it is designed. Distributed systems can be classified based on the data model as (1) distributed file system (DFS), (2) distributed database (relational) system (DDBS) and (3) distributed objectbase system (DOBS). Accordingly, the allocation problems are classified as file allocation problem (FAP), database allocation problem (DAP) and object allocation problem (OAP), respectively. Due to differences in underlying data model, solutions for one class of allocation problems are not extensible to another class. This calls for designing different allocation models for each of the above types of distributed systems.

- Degree of Redundancy: Two important parameters for the distribution design are: (1) the distribution unit or granularity of distribution and (2) the degree of redundancy. The granularity of distribution is normally determined by the data model (in case of DFS) or by the fragmentation scheme that precedes allocation in the distribution design. Given the unit of distribution, the degree of redundancy has a significant influence in the allocation model design. Introducing additional copies or replicas of individual data units increases data availability and fault tolerance. On the other hand, multiple copies occupy more storage space which is a limited resource in a distributed system. Moreover, the cost of performing updates in-

creases linearly [CA80] with the number of copies maintained. A judicial trade-off is required to decide the optimal number of copies of each data unit to maintain. The complexity of the distribution design will depend on the degree of replication. While read-only queries can be served by the "nearest available" copy of the requested data thereby reducing response time, maintaining consistency between all copies introduces substantial overhead. For fully replicated systems, where every data item is replicated at every site, the solution to the distribution design problem is trivial. If we restrict to only a single copy of every data unit, the solution to the allocation problem is, though not trivial, simple compared to the same in a system with partial replication. Determining the optimal number of copies of each data becomes an additional issue in case of partially replicated allocation.

- Design Objective: Allocation design is a combinatorial optimization problem with a specific objective function that is subjected to maximization or minimization with respect to a set of constraints. The decision variables in the allocation problem correspond to allocation decisions made for each 'data unit of allocation'. Formulation of the objective function determines the nature of the optimization model. An allocation model can be formulated with the goal of obtaining a 'minimum cost' allocation. Alternatively, the allocation model may be designed to achieve optimal performance. Accordingly, the allocation models are classified as 'cost models' or 'performance models'. In cost models, the objective function is formulated with various cost factors (eg. communication cost, storage cost, processing cost *etc.*) involved in allocation and the overall cost is subjected to minimization with respect to a set of constraints, to obtain an optimal allocation scheme. In cost models, performance issues are either ignored or they are included in the model as constraints. The performance model performs optimization on a set of performance metrics (eg. throughput, response time, *etc.*) to arrive at an allocation scheme that meets the upper bound for cost. Ideally, an optimal allocation should meet both cost and performance requirements simultaneously. The corresponding allocation model is

referred to as 'cost & performance model' in the taxonomy illustrated in Figure 1.3.

1.5 Organization of The Document

The rest of the document is organized as follows: In Chapter 2 a review of background and related work on allocation in distributed systems is provided. Two alternative approaches are taken for solving the allocation problem in DOBS with a nonredundant cost model. Chapter 3 formally describes the DOBS allocation model and presents an allocation algorithm for it based on a graphical approach while solution technique using simulated annealing for the proposed model is described in Chapter 4. The results obtained by applying these algorithms on a carefully generated test data that represents a DOBS prototype are presented in Chapter 5 with a comparative analysis of the performances of the two approaches. Finally, Chapter 6 summarizes the contribution of this research and outlines future research directions.

Chapter 2

Related Work

This chapter reviews work on distribution design and allocation in different distributed environments including distributed file systems (DFS), distributed database systems (DDBS) and distributed objectbase systems (DOBS). Section 2.1 summarizes contributions made by researchers to the File Allocation Problem (FAP). Section 2.2 presents a review of the previous work on Database Allocation Problem (DAP) and Section 2.3 discusses related work in distribution design for distributed objectbase systems (DOBS). This chapter concludes with a brief overview of distribution design methodologies adopted in existing distributed systems.

2.1 Previous Work on File Allocation Problem (FAP)

This section reviews contributions to the file allocation problem. The problem of distributing data files across the nodes of a network is known as File Allocation Problem (FAP). The file allocation problem has been studied extensively in the literature. Several approaches have been proposed to obtain an optimal solution under different assumptions and with different objectives. Generally the problem is formulated as a combinatorial optimization problem that distributes files with respect to an objective function subject to

a set of constraints. The objective function and constraints are defined as functions of 'decision variables' associated with possible allocation of each file at any of the network sites. Standard optimization techniques (eg. integer programming, dynamic programming, linear programming, *etc.*) for solving problems in operations research are applied. The models proposed differ in their measure of optimality and the set of constraints applied.

Based on the type of objective function, they can be broadly classified using the proposed taxonomy as *FAP cost models* and *FAP performance models.* Researchers present various models in which the objective function is formulated as a "cost function" that includes different cost parameters such as storage cost, communication cost, query and update cost. The FAP is formulated as a combinatorial optimization problem that attempts to minimize the cost function. Alternatively, algorithms that find an optimal file assignment for optimizing certain performance measure (eg. file availability, delay, response time, throughput *etc.*) have been investigated. The problem is formulated as a performance optimization problem with a "performance model". Since it is possible theoretically to make the objective function as comprehensive as possible, optimal solutions to the allocation problems modeled as a combinatorial optimization problems, if obtained, would be very accurate. Unfortunately, previous attempts in this area demonstrate that solution techniques for these optimization problems with a large number of decision variables are prohibitively costly and computationally intractable for large problems. Therefore, heuristic approaches are adopted to obtain "near optimal" solutions.

This section first reviews the FAP/cost models and then details the FAP/performance models.

2.1.1 FAP/Cost Model

This section summarizes research where the file allocation problem is seen as a minimization of the operating cost represented by a cost function. The FAP/cost models proposed

differ in the formulation of the objective function and the set of constraints applied. Both redundant and nonredundant FAP/cost models are presented. An overview of nonredundant cost models for FAP is presented and the FAP/cost models that consider placement of multiple copies of each file over a distributed network are described in brief.

FAP Cost Model/Nonredundant Allocation

The nonredundant (single copy) file allocation problem has not been addressed explicitly in literature. Instead, the FAP is formulated as a general multiple copy file allocation problem thereby treating nonredundant allocation as a special case where the number of file copies is one.

FAP Cost Model/Replicated Allocation

Casey [Cas72] and Wah [Wah84] present algorithms for allocating multiple copies of a single file in a distributed network. It is assumed that the files are accessed independently and therefore the solution technique for single file allocation can be applied independently to each file in succession to solve the multiple file allocation problem.

Casey [Cas72] determines the optimal number of copies of a single file that should be placed in a network. The objective function includes storage and communication costs for queries and updates. A "cost graph" model is developed that applies efficient graph search techniques to find the globally optimal, least cost solution from the set of feasible solutions. If query and update communication costs are equal, this approach gives an upper bound for the number of file copies as a function of query and update traffic. The allocation model is not subjected to any constraint. Eswaran [Esw74] shows that Casey's formulation is NP-Complete. One important contribution of Casey's model is that it distinguishes query and update traffic.

Wah [Wah84] reasons that an optimal design must solve the allocation problem by considering query processing, concurrency control, and communication network design

since there is tight functional coupling between them. Wah shows an isomorphism between FAP with the warehouse location problem in operations research (OR) and suggests that OR solution techniques for solving the warehouse location problem can be applied to FAP models. A FAP model is developed that demonstrates the application of the "branch and bound method" to obtain an optimal allocation. The allocation problem is formulated as an optimization problem to minimize storage, update, and query cost. All feasible solutions are successively partitioned into smaller and smaller subsets. At every step, newly created subsets are rejected if their lower bounds lie above the least upper bound of all subsets created so far and branching continues only on promising subsets until an optimal solution is found.

Chu [Chu69] considers the problem of allocating multiple copies of each file in a fully connected network. The number of copies of each file must be known. Communication traffic between any two nodes of the network is modeled as an M/D/1 queuing process. The FAP is formulated as 0/1 integer programming problem where the decision variables account for possible placement decisions for each file. The objective is to minimize storage and communication costs subject to storage capacity and response time constraints. The resulting nonlinear problem is transformed to linear form by adding an additional constraint for each nonlinear term appearing in the objective function and in the initial constraints. Standard techniques for solving linear 0/1 programming problem are suggested to obtain optimal solution. Two contributions of this model are inclusion of file availability constraint and the response time constraint to account for system performance.

Levin and Morgan [LM77] present a FAP model that captures the interaction between the data files and programs that access them. Their model distinguishes between three different types of nodes: (1) the "user node", (2) the node originating file requests, and (3) the node containing the data file to be accessed. Query and update traffic are handled differently. A nonlinear 0/1 programming problem is formulated to minimize storage cost for data files and programs, and communication cost. Communication cost

accounts for interactions between users, programs originating file accesses, and data files. Instead of transforming it to a linear problem, the storage cost for programs are assumed to be negligible compared to that for the data files so the multiple file allocation problem is decomposed into a number of individual file minimization problems that can be solved for each file independently using an efficient graph search technique. A common drawback of all the file allocation models discussed so far is that each individual file allocation is considered independently and the impact of interactions between the files is ignored. Treating each file separately for allocation does not capture the complexity introduced by the interlinked files and therefore, does not lead to a globally optimal solution [CNW83].

Since solution to the FAP formulated as an integer programming problem is computationally cumbersome, Chandy [CH76] suggests an indirect approach to obtain a "near optimal" solution. The integer constraint corresponding to nondivisibility of file copies is dropped to formulate a linear programming problem (LPP) so standard LPP solution techniques may be applied. Solutions obtained from the LPP serves as a lower bound for the original FAP. An upper bound may be obtained by applying heuristics. An "add-drop method" is the heuristic employed where file copies are added and deleted in iterative steps to arrive at a least-cost allocation. It is argued that if the cost difference between the two bounds is insignificant, any further effort for a better solution may not be worthwhile.

Laning and Leonard [LL83] present a file allocation scheme for multiple copies of files in a store-and-forward network with adaptive routing. The proposed FAP cost model attempts to obtain an allocation for copies of files that minimizes storage and communication cost. The constraints are file availability and average message delay. The upper bound for the optimal number of copies for each file is determined based on information provided by the user. The allocation algorithm is a two step process. First, the set of initial candidate solutions, one for each file, is generated by solving the *p-median problem* for each file independently. A solution to the p-median problem gives optimal locations for p copies of a file in the network so that the sum of all "shortest distances" from the file locations to other nodes of the network is minimum. Solutions are obtained

for each possible value of 'p' (where p is the number of file copies) and the minimum cost solution is selected. The candidate solutions are then subjected to the constraints of file availability and network delay. If the file availability constraint is violated, the number of copies is changed to satisfy the constraint and the corresponding p-median solution is obtained again for the new p value and the process is repeated until the constraint is met for all the files. Due to nondeterministic factors such as flow control and adaptive routing, it is impossible to perform a closed form analysis on the candidate placements to estimate network delay. Discrete event simulation techniques are employed to ensure that the solution is acceptable with respect to the performance constraints.

Mahmoud and Riordon [MR76] present a cost model for optimal file allocation and capacity assignment of communication links in a distributed information network. The objective function subjected to minimization, includes communication cost and file storage cost. The constraints are based on required response time and file availability. Storage capacity at every site is ignored. A two-step solution technique is suggested. The "starting routine" finds the minimum number of file copies for each file that meets the availability constraint. For every file, a set of all possible allocations of its copies is constructed. The set of initial allocations is obtained by taking the cartesian product of all such sets for all files. Therefore, the set of initial allocations contains all possible distributions for the required number of file copies. The "optimizing routine" subjects the initial feasible solutions to an iterative process where file copies are added or deleted at different nodes of the network while preserving solution feasibility. The iteration terminates when no addition or deletion improves the value of the cost function. One "local optimal allocation" is obtained for each initial feasible solution by applying the optimizing routine. It is hypothesized that at least one of the local optimal solutions is close to a global optimal solution.

Ghosh, Murthy *et al.* [GMM92] present FAP/cost models that meet the performance requirements of the applications. They reason that the response time constraints are different for different types of queries. Therefore, communication delay must be analyzed

on a query by query basis and different acceptable levels of delay must be defined as constraints for each online query. Programs are assumed to be replicated at every site and queries are distinguished from updates. It is assumed that queries are processed online and the updates are processed in batch. Therefore, updates do not contribute to the "peak-hour network traffic". Two FAP/cost models are proposed. The cost functions for both the models include storage and communication costs. The cost function is subjected to minimization with respect to the constraints of communication delay, communication link capacity, and storage capacity at each individual site. The "worst case delay model" (FAP-WCD) attempts to find an allocation that meets the "worst case delay" constraint with minimum cost. A 0/1 integer programming problem is formulated and the branch and bound method is employed. Lagrangian relaxation method is used to obtain lower bounds on each candidate subproblem while upper bounds are obtained using heuristics. For a given allocation, the 'worst case delay' is constant between any pair of nodes, but the average delay is a function of traffic flow which in turn depends on the file distribution. The circularity of the problem makes the analysis of average delay in a distributed file system more difficult. A FAP/cost model for "average delay" (FAP-AD) is proposed. Their heuristic procedure has two components: the "optimizer" and the "simulator". The optimizer obtains the lower bounds on the optimal solution. The "simulator" simulates the operation of a packet switched network and, for a given query routing strategy, estimates the traffic on the network for each file access.

2.1.2 FAP/Performance Model

This section summarizes file allocation models whose objective functions seek to optimize some performance measure of the distributed system. First FAP/Performance models for nonredundant allocation are presented followed by models with multiple file copies.

FAP Performance Model/Single File copy

Ramamoorthy *et al.* [RC70], Arora [AG73], Hughes [HM73], Chen [Che73] and Kleinrock [Kle76] present schemes for nonredundant allocation of single copies of every file in a distributed system. Ramamoorthy *et al.* [RC70] shows that the file allocation problem can be solved for minimum file access time using known results from memory hierarchy research, under certain assumptions. Files are assumed to be divisible and access times linear functions of the number of blocks accessed while ignoring queuing delays and communication overheads. Only one node generates all file requests so other nodes behave as file stores analogous to memory units in a multilevel memory hierarchy system.

Arora [AG73] extends the model to include execution time, access time, and transfer time (in case of remote access) in the objective function. The assumption that a single node is the source of all file requests is also relaxed. The resulting linear optimization problem is solved using standard technique for solving transportation problems in Operations Research [Had63]. An alternative approach is suggested that attempts to find an allocation under a response time constraint using an iterative method.

Queuing network models are presented to account for queuing delays. The general approach adopted to solve the file allocation problems with queuing models is to divide the problem into two subproblems. First, the optimal distribution of file requests to all nodes are obtained analytically from the queuing model of the network. The second subproblem is to obtain an allocation that realizes the distribution.

Chen [Che73], Buzen [Buz71], Hughes [HM73] and Piepmeir [Pie75] present file allocation schemes using queuing models for a client/server network. A closed queuing model of a central server (star) network is considered by Buzen [Buz71] and Hughes [HM73] where a constant degree of multiprogramming ensures a constant number of file requests in the network. The objective function is system throughput which is subjected to maximization. Only general guidelines are provided to accomplish the file allocation heuristically.

Chen [Che73] formulates a queuing model for an open central server network. File request frequencies and execution times are assumed to be exponentially distributed and standard results for M/M/1 queue are applied to obtain file request distributions that minimizes average response time. While allocating the files to match the distribution exactly, files are assumed to be divisible and nodal capacity constraints are ignored. Buzen [BG74] and Piepmeir [Pie75] extend these results to an M/G/1 queuing model.

For general topological networks, both the problem of finding optimal file request distribution and the allocation scheme to realize it become nontrivial. This is because the file request distribution depends on the location of the files and vice versa, thereby introducing circularity into the problem.

Kleinrock [Kle76] suggests an iterative technique that finds near optimal file allocation for an open general topological network starting from a "guess allocation" used as an initial file assignment. For given link capacities and file request frequencies the objective is to minimize file response time.

If files cannot be split, exact matching of the request distribution poses additional difficulty and makes the problem nonlinear. The nonlinearity property implies that a heuristic technique must be employed to find a near-optimal solution. Several heuristic methods [FB76] and nonlinear programming techniques [FDA81, JFK78] are suggested.

The appearance of too many decision variables and the circular dependency between placement decisions and the distribution of request traffic at the nodes cause difficulty in formulating a closed form mathematical model for FAP based on queuing model. Therefore, this approach is not adopted in later work on FAP.

FAP/Performance Model/Multiple File copy

This section provides an overview of file allocation models with multiple copies of every file to achieve optimal performance based on a performance measure. Therefore, the

objective of these models is twofold: determine optimal number of copies of each file and seek an optimal assignment for them.

Jones [JFK78] presents a goal-programming[1]formulation of the allocation problem that minimizes operating and storage costs of all the file copies subject to a capacity constraint. Overhead involved in performing update operation with multiple copies is considered. Since the file access time for a file request depends on the volume of data to be accessed and the file assignment, the file request service time is dependent on file distribution itself. This circularity makes the analysis more complex. An iterative method is suggested that improves an initial allocation in each step.

2.2 Previous Work on Database Allocation Problem (DAP)

This section provides a summary of allocation models proposed for distributed relational database systems. While some of the models intend to minimize the operating cost, the objective of the performance models is to reduce response time and/or improve throughput.

2.2.1 DAP/Cost Model

This section reviews the cost models proposed for the database allocation problem (DAP) based on the taxonomy used to classify the FAP models. The DAP models are presented in the same order the FAP models are reviewed in the previous section. The degree of replication is a major issue in designing a DAP model. Multiple data copies would increase data availability at the cost of an increase in storage requirement and the overhead involved in maintaining copy consistency. Therefore, a trade-off is necessary. Based on

[1]The objective function of a goal-programming problem consists of several terms with weights corresponding to relative priorities associated with each of them.

the degree of redundancy, the DAP/cost models can be classified as the nonredundant allocation model and the redundant allocation model.

DAP/Cost Model/Nonredundant Allocation

Ceri and Pelagatti [CP84], Cornell and Yu [CY89], Gavish and Pirkul [Gav87, GP86], Ceri, Navathe *et al.* [CNW83] and Apers [Ape88] present DAP cost models for allocating single copies of every relation in a distributed relational database system.

Ceri and Pelagatti [CP84] show that due to complex application behavior in distributed database systems, solutions to the file allocation problem cannot be extended to the distributed database allocation problem. Instead of proposing any specific allocation model, they discuss the issues involved in the design of a DAP cost model and possible solution techniques. A "best-fit approach" is suggested for nonredundant allocation based on a measure of optimality for each possible distribution. A set of database fragments are obtained by fragmenting the relations. Mutual effects of placements of individual fragments are ignored. Redundant allocation is handled by progressively assigning replicated copies. Mutual effects of allocating different copies of any fragment are considered.

Ceri, Navathe *et al.* [CNW83] present a nonredundant DAP model where the logical schema is modeled as a directed graph with database relations as nodes and the logical relationships between them are represented by edges connecting the nodes. A fully connected network is considered and identical transmission cost for unit transmission between any two nodes is assumed. The objective function subjected to minimization includes access and transmission cost under the constraint of storage capacities at respective sites. The global schema is partitioned into disjoint subsets so that they can be independently optimized. Thus, the initial allocation problem is decomposed into a set of independent subproblems involving fewer variables. Each of the subproblems is solved using 0/1 linear programming technique and the solutions are merged to obtain a global allocation scheme. The solution for the nonredundant allocation model is extended to a partially

replicated allocation model using a greedy heuristic technique.

Sacca and Wiederhold [SW83] attempt to find an optimal non-redundant database partitioning and allocation over a closely connected cluster of processors. Each processor has its own processing power and input-output handling capacity. High channel capacity is assumed to handle the transaction traffic and communication overhead is assumed to be insignificant because of the close proximity of the processors. User applications that generate queries are not preassigned to any site. Instead, assignment of applications to different sites of the network is considered part of the design goal. Limitation of available resources such as processing power and I/O capability at every node and the limited capacity of the communication links connecting them are considered as constraints to formulate the allocation model. The objective of the allocation process is to design a scheme that distributes the transaction processing load evenly among the processors while minimizing the overall processing cost. The "cost evaluation function" that computes the overall transaction processing cost for a particular distribution is defined in terms of the number of processor cycles required, the number of I/O blocks accessed, and the volume of message traffic generated to execute the transactions. The distribution design process is divided into two steps: (1) fragmentation of the database relations, and (2) allocation of the fragments and the applications. A greedy heuristic procedure is adopted, where, starting from an initial "virtual network" with one "virtual node" for every fragment to be distributed, the number of nodes is reduced by one at every step by grouping fragments. The iteration continue until the number of virtual nodes is equal to the actual number of network nodes. The allocation scheme is realized by mapping each virtual node to one of the nodes of the network.

In a distributed database environment the amount of data transfer between data sites depends on the order in which the operations are executed on the set of database relations. Typically this order is decided by the query processor. An allocation scheme therefore must take the execution order of the operations into account. On the other hand, the query processor uses data allocation information to generate an optimal execution

schedule. Apers [Ape88] proposes a method that performs partial allocation of data in iterative steps based on scheduling information to arrive at a complete allocation that minimizes a cost function. Query and update traffic are treated differently and an optimization model is constructed that minimizes total communication and storage costs. The database relations are assumed to be partitioned into fragments based on application access pattern. A graphical approach is taken in making placement decisions for the fragments. Starting with an initial allocation where every transaction has its own copy of the fragments required for its execution, a "process scheduling graph" is generated by treating the set of fragments accessed by every transaction as a "virtual site". The nodes of the graph are the virtual sites and the edges between them represent the amount of data transfer between them for transaction operations. A cost model is formulated from this graphical representation that aims at minimizing the global data transfer. The algorithm maps the set of virtual sites of the initial allocation to the set of "physical sites" whose nodes are distributed. This is accomplished by "union" and "assignment" operations. A union operation merges two virtual sites to form a new virtual site and an assignment operation allocates a physical site for a virtual site. Two different techniques are suggested: (1) restricted search for optimal or "near optimal" solution using branch and bound algorithm and (2) heuristic technique to arrive at an "acceptable" solution. Allocation algorithms based on these techniques are presented for both static and dynamic environments. Static and dynamic environments correspond to adjusting or not adjusting the query processing schedules when computing the cost function for different allocations until a final solution is reached.

Gavish and Pirkul [GP86] assume that transactions exhibit strong "locality of reference" in making placement decisions for processors and databases in a distributed banking network. A distribution design model is formulated that attempts to find solutions for the design parameters such as the number of processors, their capacities and locations, the database partitions and their distribution. An integer programming problem is formulated with the decision variables accounting for the design parameters. Heuristic

procedure and branch and bound method are suggested as the solution technique.

Gavish [Gav87] adopts a system designer's view in designing an architecture for a distributed database system. The proposed model distinguishes the network nodes as "data source node", (where transactions originate), "database node", (where databases are stored), "report generator node", (that processes information and generates reports), and "user node", (where reports are sent). Data flows between different type of nodes for execution of every transaction and generation of reports is formulated as a function of the decision variables defined in the model. Different sets of decision variables account for the locations of the databases and the processors, originating location and execution location of each transaction, and generation and destination nodes of the reports produced from the execution of the transactions. The model leads to the formulation of an integer programming problem that minimizes the operating cost. Since the problem is NP-complete, methods are suggested for computing the lower bounds on the value of the optimal solution. Alternatively, a recursive procedure based on greedy heuristic is suggested to obtain a 'near optimal' solution as an acceptable configuration for distributed system. Based on a selection criteria, an initial set of "report generation sites" is determined. Database locations are determined with respect to the report generation sites. The set of database locations are then used to improve the set of report generation sites and the iterative process is repeated until no improvement takes place in two subsequent passes.

Cornell and Yu [CY89] identify interdependency between access plan selection[2] and the assignment of database relations to different sites of a distributed database system. Relations are not fragmented and are assigned to a site as a whole. A fully connected network with equal communication capacity on each link is assumed. An integer programming problem is formulated with the communication cost as the objective function which is subjected to minimization. The constraints are processing capacity, disk I/O ca-

[2]Typically access plan selection is performed by query processor [Ape88].

pacity and storage capacity at each site, and the communication capacity of the network links connecting the sites.

Yu, Siu *et al.* [YSLC83] present an adaptive algorithm for allocation of database files in a star network. Update costs are ignored as only one copy of each file is considered. File access pattern is assumed to remain unchanged over time. The cost function includes communication cost due to queries. Starting from an arbitrary initial file distribution, the adaptive process computes online the "gain" of placing a file at any particular site. For each query being processed, the "gain" is evaluated for all possible file allocation of the files being accessed by the query. The final allocation is obtained by placing every file at a site where the corresponding gain is maximum. The nonredundant DAP model is extended for replicated allocation by including update costs in the cost function.

In a distributed system, network topology and distribution of data are closely related. Design of a distributed system should exploit the coupling between data distribution and program and network topology to arrive at a design that provides optimal system performance at minimum cost. Irani and Khabbaz [IK82] and Chen and Akoka [CA80] address the integrated problem of design of communication network and distribution of database files. They reason that separate consideration of these two subproblems can lead to suboptimal design and the DAP cost models proposed by them capture the strong relationship between network topology and database allocation in a distributed database system.

Irani and Khabbaz [IK82] assume that the database is partitioned into a set of files and the transactions are logically broken into subtransactions so each refers to only one file. This eliminates interactions between the files and reduce the database allocation problem to classical file allocation problem (FAP). Network topologies are modeled by graphs and are classified based on number of nodes (n), connectivity (k), and the diameter (d) of the network. "Connectivity" is defined as the minimum number of disjoint paths between every pair of sites and the maximum path length between any pair of sites is called

the "diameter" of the network. Read-only queries are distinguished from updates and poisson arrival of database request (queries and updates) and exponential distribution of message lengths are assumed. An iterative procedure is suggested for an optimal design which minimizes storage and communication cost with respect to the constraints of network reliability, file availability, and communication delay. Files are allocated one at a time. A lower bound on the number of copies of each file is obtained analytically in terms of file availability, network reliability, and site failure probability. A two step iterative algorithm is presented. Each iteration tries to obtain an optimal allocation of file copies and a network channel capacity assignment for a given network topology. Files are allocated using a greedy heuristic followed by an interchange heuristic that attempts to improve the solution by reallocating copies of the file for a possible improvement in the value of the cost function. In the second iterative step optimal assignments for all possible network topologies are compared to arrive at a globally optimal design.

Chen and Akoka [CA80] consider simultaneously the distribution of processing power, programs and databases, and assignment of communication channel capacities in a distributed system. No fixed topology is assumed and dependencies between data and programs are considered. A DAP cost model is formulated and the problem is framed as an integer programming problem. The cost function subjected to minimization includes equipment (processor, storage, communication link, *etc.*) cost, communication cost, and storage costs for programs and databases. The solution technique suggested is a variation of the branch and bound method. First, the problem is solved without the integer constraint to make the domain of the decision variables unrestricted. This reduces the complexity of the problem and allows 'linear programming' techniques to be applicable for obtaining an optimal solution. However, only feasible values for the decision variables would give a feasible allocation. The initial solution is then modified by replacing noninteger values of the decision variables with feasible values in steps and solving the resultant subproblem at each step. A modified version of the branch and bound algorithm is employed.

DAP/Cost Model/Replicated Allocation

Fisher and Hochbaum [FH80] and Chieu and Raghavendra [CR90] address the problem of allocation of multiple copies of a database in a distributed network.

Fisher and Hochbaum [FH80] present a DAP model that makes placement decisions for copies of databases by trading off the access cost, which is reduced by additional copies, against the cost of storing and updating the additional copies. The problem is formulated as a mixed integer programming problem[3] with the cost function consisting of storage costs and communication costs for transmitting queries and updates from the "user" to the database programs and from database programs to database locations. A lower bound and an upper bound on the number of database copies are set as a constraint. An initial solution is obtained using heuristics. Two heuristic methods are employed. The "greedy heuristic" constructs an initial allocation by successively adding an additional copy of databases where it would produce the greatest reduction in the value of the objective function. Copies are added until an additional copy does not improve the solution. An 'interchange heuristic' is then applied to improve it by deleting one copy and adding another. The Lagrangian relaxation method is used to obtain lower bounds on the cost of an optimal allocation. The initial solution is then subjected to branch and bound method for a least cost solution.

Chieu and Raghavendra [CR90] address the problem of finding the optimal number of database copies and their locations to optimize the communication cost due to remote accesses between pairs of network sites. It is assumed that the local storage capacity at each site is large enough to hold a copy of the entire database and there are at least three copies of the database present in the network. The "triple module redundancy (TMR)" scheme is assumed for reliability reasons. The DAP is formulated as a 0/1 integer programming problem with decision variables corresponding to every possible

[3] In mixed integer programming problems, some of the decision variables are allowed to assume noninteger values.

allocation of copies of the database. The objective function is the communication cost due to remote accesses. A lower bound on the optimal value of the objective function is obtained by Lagrangian relaxation technique. This bound is used to prune branches in a branch and bound algorithm to arrive at a "near optimal" solution.

Özsu and Valduriez [ÖV91] discuss the mathematical formulation of a DAP/cost model for redundant allocation. The cost function includes communication costs for queries and updates, and storage cost. The relevant constraints are response time, storage capacity, and processing power at individual site. They enumerate the input information required for the allocation process and how they can be represented in the mathematical formulation of the DAP model. No solution technique is suggested. Instead it is demonstrated that the analysis becomes increasingly difficult and mathematically cumbersome as the model is made more rigorous by including all relevant issues and constraints.

2.2.2 DAP/Performance Model

This section reviews DAP models that allocate database fragments to optimize system performance with respect to certain performance metrics. Nonredundant DAP/performance models are reviewed first followed by a review of the DAP/performance models for replicated allocation.

DAP/Performance Model/Nonredundant Allocation

Cornell and Yu [CY89] present a nonredundant allocation model for a distributed database system. The approach taken applies heuristic techniques to minimize average response time under the constraints of available storage and processing power at every node and the communication link capacities. An open queuing network model is formulated for a fully connected network. Each node is modeled as an M/M/1 queue with single CPU and multiple disks. The average response time is obtained analytically as the sum of processing time, communication delay, and disk I/O time. Exponential distribution of

message lengths is assumed in computing communication delays. Average response time is expressed as a function of the decision variables corresponding to possible allocation alternatives and is subjected to minimization using an optimization model. Simulated annealing technique is suggested as a heuristic alternative method for solving problems of large size.

DAP/Performance Model/Redundant Allocation

Ciciani *et al.* [CDY90] present a model to examine the performance tradeoffs introduced in data replication in distributed database system with respect to communication overhead, network delay, and response time constraints. The database is assumed to be partitioned into a set of fragments and an equal number of copies of each fragments are distributed across the network. An analytical model is presented that compares the performances of different concurrency control protocols. Queuing models are used to estimate the resource contention effects by discrete event simulation. Simulation results are obtained for updates and read-only queries. The results obtained shows that if communication overhead is ignored, the response time increases with increases in the degree of replication for pessimistic, semi-optimistic, and optimistic concurrency control protocols. While the pessimistic (two-phase locking) protocol outperforms optimistic and semi-optimistic protocols in the absence of any replication, the optimistic protocol results in better performance as replication increases.

2.3 Research in Distribution Design for Distributed Objectbase Systems

This section presents a review of work done on distribution design in Distributed Objectbase Systems (DOBS).

Karlapalem *et al.* [KNM92] discuss various issues involved in distribution design of

object-oriented database systems. They identify the characteristic features of the object-oriented model that make the problem of distribution design more complex in comparison to its peer problem in relational database system.

In relational database systems, transactions that manipulate data, are not considered an integral part of the database. Therefore, distribution design in relational systems is reduced to partitioning the relations based on the locality of access of the constituent tuples and allocating the resulting relation fragments. In the object-oriented environment, execution of a transaction translates to a sequence of method executions on the classes of the objectbase. Due to method encapsulation within the class, methods that execute on the attributes of the same class are inseparable from the objectbase. Hence, distribution design of an object-oriented system must consider distribution of both attributes and methods. Resolving the issue of overlapping sets of attributes accessed by two different methods becomes part of the fragmentation problem. Distribution of methods introduces another level of complexity into the allocation problem. The type of mechanism ("Remote Procedure Call" or "Prefetch and Caching") used by the methods to access objects and/or the attributes has a direct impact on the cost and/or performance model for method executions, which in turn will govern the allocation decisions. Location transparency is also an important issue to be considered. They argue that the critical issues for the distribution design for an object-oriented system are : (1) level of transparency, (2) method invocation strategy, (3) application processing semantics and (4) structural and behavioral properties of the objectbase. They identify the goals of a distribution design in an object-oriented system as: (1) to reduce irrelevant data access by application and (2) to reduce the amount of inter-site data transfer.

Ghandeharizadeh *et al.* [GW94, GWLZ93] present a greedy algorithm for the placement of objects in a distributed parallel objectbase system. A query migration paradigm is assumed for processing queries that access nonlocal objects. The objectbase is viewed as a directed graph with the nodes as the objects and the edges joining them representing semantic relationship between the objects. The object placement algorithm is

designed as a greedy graph partitioning algorithm that reduces the number of "internode traversals" of queries and maximizes the benefits of parallelism by evenly distributing processing workload across all sites. Navigational queries in object-oriented databases are distinguished from the queries in relational databases which are associative in nature. A skewed distribution of the object access frequencies is assumed and the objectbase is partitioned into $m + 1$ fragments where m is the number of sites. Each of the first m fragments is assigned to a site and the objects in the last fragment are redistributed across the m sites. The distribution is subjected to a workload constraint at every local site.

Ezeife and Barker [EB93, EB94] present algorithms for obtaining horizontal and vertical fragments by optimally partitioning the global objectbase. The fragmentation schemes are designed to achieve improved performance by reducing irrelevant data access, minimizing data transfer, increasing concurrency, and reducing irrelevant data replication. They assume that the distributed database consists of local databases at individual sites each having its own external and internal schema. Based on application access frequency and application execution pattern, fragmentation schemes are suggested that reduce processing costs by minimizing communication overhead and irrelevant data access. A taxonomy is presented that classifies different models based on method type and the type of the attributes accessed. The different models suggested are: (1) simple attribute/simple method, (2) simple attribute/complex method, (3) complex attribute/simple method and, (4) complex attribute/complex method.

Horizontal fragments for the first "class model" (simple attribute/simple method) are obtained by partitioning the set of instance objects of a class in a way that supports locality of reference. A *Link Graph* is generated to capture the instance hierarchy in the class lattice. For each class, primary horizontal fragments are generated based on application access patterns on the attributes. Derived horizontal fragments are formed by propagating horizontal fragmentation of the subclasses to their superclass. Finally, reconciliation is done between the primary and the derived fragments to arrive at the set of horizontal fragments for every class in the class lattice. The scheme is extended for

other class models.

The vertical fragmentation algorithm is presented for simple attribute/simple method model that evaluates "affinity" between attributes of a class and between any two methods of any class in the objectbase. The affinity between two entities is measured as the frequency of accessing them together by the set of applications. Inherited attributes and methods are assumed to be stored with the superclass and the subclasses have pointers to them. Access frequencies of a class method from its subclasses are taken into account while computing the method affinity matrix. The Method Affinity Matrix captures the information about the frequency of accessing any two methods belonging to the class under consideration. The Bond Energy Algorithm [ÖV91] is then applied to obtain partitions based on a cluster affinity matrix that attempts to place those methods that are invoked together in one fragment. In the second step, the attributes accessed by the methods are placed in the same fragment where the corresponding methods that use them are placed. Any conflict regarding placement of an attribute referred to by two or more methods from different fragments is resolved by selecting the candidate fragment that has the highest affinity with the attribute under consideration.

2.3.1 Fragment Allocation in Distributed Object System

The problem of allocation has not been investigated extensively in the object oriented environment. Although a few articles have appeared in literature that discuss various issues and constraints associated with the problem, no algorithm or solution strategy has been suggested. Karlapalem, Navathe and Morsi [KNM92] identify the importance of fragmentation and allocation of an objectbase for minimizing irrelevant data access and nonlocal data references as a part of distribution design. Characteristics of object oriented systems that distinguish it from a relational system are listed and their impact on distribution design are discussed.

2.4 Overview of Existing Distributed Systems

This section reviews the existing distributed systems and analyses how the problem of distribution design is handled in them. An overview of distributed file systems (DFS) are given followed by a summary of distributed database systems (DDBS).

2.4.1 Distributed File Systems

This section presents an overview of the existing distributed file systems (DFS). The DFS's reviewed are Andrew, Apollo Domain, Coda, SUN-NFS, Helix, Sprite, LOCUS, IBIS and ROE.

Andrew is a distributed file system designed jointly by Carnegie Mellon University and IBM. It is designed on a client/server architecture. *Vice* is a collection of dedicated file servers composed exclusively of files. It acts as the information sharing backbone. A client process, *Venus*, running on every workstations finds the required file from Vice and caches it into local disk. Caching at local sites exploits the advantage of temporal reference of file access. The same principle is adopted in the Apollo Domain system and in the Coda system which is a descendant of Andrew.

Helix [FO84, FO81] is a distributed file system for a client/server type architecture. Files in Helix are stored in local disks of the file servers independent of user access pattern. No replication or migration of files between servers are permitted. Files are assigned permanently at the site where it is first created. The two-phase commit protocol is used to ensure consistency for transactions that use files at multiple servers. Xerox Distributed File System (XDFS) [MD82] also takes the identical approach with a more extensive facilities for transaction management.

Sun Microsystem's NFS has independent local file system at every node of the network. A nonlocal file system may be mounted in the local file directory and can be accessed by the user. For reference to a remote file, the local kernel maintains a cache of

blocks read to reduce data transfer overhead between sites. Data transfer between sites is done in large blocks. Both server and client modules reside in the UNIX kernel of each computer. File requests to a nonlocal file are translated by the client modules to NFS protocol operations and passed to the NFS server module at the computer holding the requested file.

Sprite, designed at University of California, Berkeley, supports distributed file system in a diskless workstation environment. A few machines with disks act as servers which store files with remote links for the client process at other nodes to access. Clients do not have to import files from servers explicitly. The Sprite file system is organized as a collection of domains. Domains are file subsystems obtained by partitioning the global file system such that all the files of one domain are stored at a single server node. Consistency and concurrency is maintained by controlled caching between the client and the server node.

The File system in LOCUS is organized by "logical filegroups" which can be mounted as individual file systems. Corresponding to every logical filegroup, there is a set of nodes where files of the logical filegroup can be mounted in the local file system. Replications are allowed and updates are propagated to all file copies.

Another approach is adopted by IBIS file system that allows files to be replicated. At any point only one of the many copies of a file is treated as "primary copy" and the rest are "secondary". When a primary copy is updated, other copies can either be updated simultaneously by propagating the update or they can be invalidated to deny access to inconsistent data. If the primary copy is unavailable due to network partition, a temporary primary is elected from the secondary copies. The temporary and actual primaries are merged again when communication is reestablished. Migration of primary copies is allowed but it is not automatic.

ROE [Flo89] maintains a distributed globally accessible, shared file system across the network. Replicated distribution of files is accessed with the help of global file direc-

tory which is also replicated at every site. Consistency among file copies is maintained by "weighted voting". Heuristics are employed for both initial file allocation and file migration algorithms. The static initial file allocation scheme distributes files across the network based on available storage, system load, and network topology. The objective of allocation is to minimize file access delay and maximize file availability. A "replication factor" is selected at the time of file creation. Each file copy is independently allocated. Out of several candidate nodes for assigning a file, the one exhibiting minimum current delay is selected provided it has required storage. Information provided by the user who creates the file also influence file placement decision. File migration is done based on resource utilization information collected from recent past history of the system and the network. ROE supports "archival" file migration where file size, file type, and time since last reference are used as predicting factor for anticipating next reference. Short term file migration strategies are adopted to exploit temporal locality of file reference. Three different file migration strategies are developed: demand based, anticipatory, and compensatory. Compensatory file migration is done to adjust for changes in the load distribution in the distributed system.

2.4.2 Existing Distributed Database Systems

This section reviews some of the existing database system that are either distributed or based on client/server architecture. The systems reviewed are ITASCA, ORION, ENCORE, GOBLIN and THOR.

ITASCA and ORION-2 are designed on multiclient/multiserver architecture where private and shared databases are distributed across the network. Distribution of databases is controlled by the servers. Schema is replicated at every site but not the individual data items. Distribution of data may be changed due its migration either by the user or due to requirement of the system. ENCORE is another object oriented database model where data are located in the same node with the server and processes executing at other nodes

access data through server using a remote procedure call (RPC) mechanism. GOBLIN is a network of workstations having memory-resident databases. Partitioning and distribution information are maintained by the Redistribution Administration Table (RAT). In Thor, clients are distinctly separated from the set of distributed servers although a single node can act as both a server and a client. Data is stored at some of the server nodes.

Chapter 3

Static Allocation: Graphical Approach

This chapter presents algorithms for static allocation in DOBS. The algorithms are based on the nonredundant cost model for DOBS defined in the allocation taxonomy in Figure 1.3 presented in Chapter 1. The allocation problem is found to be NP hard for distributed file system (DFS) and for distributed relational database system (DDBS) since it can be formulated as a 0/1 integer programming problem, which is NP Complete [GJ79]. With the additional complexity involved in modeling the problem for a DOBS, the problem still remains NP hard[1]. Therefore, it is highly unlikely that an optimal, polynomial time solution to the allocation problem for a DOBS could be obtained. This indicates that heuristic approaches need to be adopted to obtain a "near optimal" solution in a reasonable time. Therefore, the algorithms proposed in this dissertation are based on heuristics. Two different heuristic approaches are taken : (1) a *graphical approach* and (2) a *simulated annealing approach.* An initial allocation is obtained as a starting solution which is then subjected to optimization based on the heuristic techniques

[1] If the interactions between class fragments are ignored then the DOBS allocation problem reduces to FAP which is NP hard.

adopted by the two approaches. Section 3.1 formally describes the static nonredundant allocation model for DOBS with assumptions explicitly stated and definitions used in the model clearly presented. Section 3.2 presents an algorithm for generating an initial allocation which can be used as a starting solution for the optimization algorithms based on either approach. Since the DOBS allocation model considered in this dissertation is a *cost model*, the central issue for its design is the *cost of allocation*. Section 3.3 presents the method for estimating interfragment reference costs based on the initial allocation that leads to the formulation of the allocation problem's cost model where the cost function is an optimization objective function. Section 3.5 presents a heuristic optimization algorithm based on a graphical approach. An alternative approach using simulated annealing is presented in Chapter 4.

3.1 The Allocation Model

This section formally describes the static, nonredundant cost model for distributing class fragments in a DOBS. The allocation problem is then formally defined based on this model. The assumptions made and the definitions required to formally present the model are described next.

Several issues influence the allocation design in DOBS. Considering all these issues simultaneously would make the allocation model complex and difficult to analyze. Therefore, the model is restricted to the following assumptions to ensure the problem is manageable:

- Communication cost is the dominating factor so it is subjected to minimization. Therefore, only the communication costs due to nonlocal references between class fragments and between applications and class fragments are included in the 'cost function'.

- Cost of communication between any two sites is proportional to the length (in number of network hops) of the communication link along the shortest path joining the two sites.
- The objectbase is nonredundant. In other words, no class objects or class fragments are replicated and there is one and only one copy of each of them in the distributed objectbase.
- Number of class fragments to be distributed (m) is large compared to number of network sites (n).
- Local processing power at every site is sufficient to handle transaction processing load.
- The communication links connecting the sites have sufficient capacity to support the transaction traffic.

3.1.1 The Problem

The formal description of the problem requires a precise definition of several terms.

Definition 3.1.1 An *objectbase* $\mathcal{OB} = \{f_1, f_2, \ldots, f_m\}$ is a set of m class fragments generated by some fragmentation algorithm. Each fragment f_i has a finite size v_i. ■

Definition 3.1.2 *A class fragment consists of a subset (or the entire set) of objects belonging to the class where each object* is either partially (in vertical fragments) or fully (in horizontal fragments) contained by the fragment. ■

Therefore, the j^{th} class fragment f_i^j of class $C_i = \{Cid_i, A_i, M_i, I_i\}$ contains a set $A_i' \subseteq A_i$ of attributes and a (possibly empty) set $M_i' \subseteq M_i$ of methods for every instance object in the set $I_i' \subseteq I_i$ that are part of the fragment f_i^j. Hence, f_i^j can be represented as $\{Cid_i, A_i', M_i', I_i'\}$.

Definition 3.1.3 An *allocation* $\mathcal{A} = \{F_1, F_2, \ldots, F_n\}$ is a partitioned set of all class fragments constituting the objectbase such that:

1. $\forall_i\ F_i,\ \exists j, \sum_{k|f_k \in F_i} v_k \leq d_j$, where d_j is the storage capacity at site s_j. This is the condition for *feasibility* of an *allocation.*
2. if $f_k \in F_i$ then $f_k \notin F_j, \forall i, \forall j, i \neq j$. This is *disjointness* property of *allocation.*
3. $\forall k, \exists i, f_k \in F_i$. This the *completeness* property of an *allocation.* ■

In the following definitions the j^{th} method of class C_i is denoted as m_j^i.

Definition 3.1.4 *Method Attribute Usage* $MAU(m_j^i, a_k^i)$ *of a method* m_j^i *of a class* C_i *for an attribute* a_k^i *in the same class* C_i is the average number of accesses made to attribute a_k^i by every invocation of method m_j^i . ■

Definition 3.1.5 *Method Method Usage* $MMU(m_j^i, m_k^l)$ *of a method* m_j^i *of a class* C_i *for another method* m_k^l *in any class* C_l is the average number of accesses (invocations) made to method m_k^l by every invocation of method m_j^i . ■

An application q_k accessing the distributed objectbase is a sequence of method invocations on an object or on a set of objects. The invocation of method j of class C_i is denoted by m_j^i.

Definition 3.1.6 The *Method Invocation Sequence MIS(q_k) of application* q_k is represented by $\{m_{j1}^{i1}, m_{j2}^{i2}, \ldots, m_{jn}^{in}\}$, where each m_j^i refers to a method invocation by q_k. ■

Definition 3.1.7 *Application Site Affinity* $ASA(q_i, s_j)$ *between an application* q_i *and a site* s_j is the number of times the application q_i is executed at site s_j. ■

Definition 3.1.8 *Application Fragment Affinity* $AFA(q_i, f_j)$ *between an application* q_i *and a fragment* f_j is the average number of references made by every execution of the application q_i to fragment f_j. ■

The following definitions are adapted from [EB93, EB94] :

Definition 3.1.9 *Method Attribute Reference* $MAR(m_j^i)$ *of a method* m_j^i *of a class* C_i is the set of all attributes of the class C_i referenced by the method m_j^i. ■

Definition 3.1.10 *Method Method Reference* $MMR(m_j^i)$ *of a method* m_j^i *of a class* C_i is the set of all methods of any class referenced by the method m_j^i. ■

Since method code may have conditional statements based on predicates on the attribute values, the exact set of attributes and/or methods accessed by any method execution is not known until runtime. A conservative approach is adopted in considering all possible sets of attributes that the method may access during its execution (for all possible values of the predicates appearing in the conditional statements) and the union of all such sets of attributes is taken as the set of attributes (MAR) accessed by the method. The set of all possible methods (MMR) is similarly obtained.

Definition 3.1.11 *Access frequency of an application* is the average number of accesses an application makes to "data". If $Q = \{q_1, q_2, \ldots, q_q\}$ is the set of applications accessing the objectbase, $acc(q_i, d_j)$ indicates the access frequency of application q_i on data item d_j where "data" item d_j can be a fragment, an object, or an attribute or method of an object. ■

Definition 3.1.12 *Fragment Fragment Reference* $FFR(f_i^j, f_l^k)$ *between any two class fragments* f_i^j *of class* C_i *and* f_l^k *of class* C_l is the sum of the references made by any method in fragment f_i^j to any attribute and/or method in fragment f_l^k and the references made by any method in fragment f_l^k to any attribute and/or method in fragment f_i^j. ■

Therefore constructing Fragment Fragment Reference Matrix ($FFRM$) where $FFRM(i,j) = FFR(f_i, f_j) = FFRM(j,i)$ is possible.

The problem of allocating object fragments in a distributed system can be formally stated as follows:

> **Given** a set of n network sites S $= \{s_1, s_2, \ldots, s_n\}$ with respective storage capacities D $= \{d_1, d_2, \ldots, d_n\}$ and a set of m fragments F $= \{f_1, f_2, \ldots, f_m\}$ where each fragment f_i is of size v_i and $m \gg n$, the **goal** is to find a mapping from F to S such that
>
> (i) The sum of the storage requirements of all the fragments assigned to a particular site must not exceed the total available storage at that site.
>
> (ii) The total number of non-local references is minimized.

To keep the complexity of the problem manageable, the issues of load balancing, network delay, and system reliability are not considered.

3.2 Generating a Starting Solution

This section presents an algorithm for generating an initial feasible allocation for the model described in the previous section with respect to the storage constraints at every site. The initial allocation is used as the starting solution by the optimization procedures described in Section 3.4 and Section 4.3.3.

The following condition must hold for any feasible solution: The total available storage at all sites taken together must be large enough to hold all the fragments. In other words, $\sum_{i|d_i \in D} d_i \geq \sum_{i|f_i \in F} v_i$, where v_i is the size of fragment f_i.

This condition is necessary but not sufficient. At every site, the assigned fragments may not produce an "exact fit" in the local storage so some amount of storage is left

unused. The total unused storage at all individual sites may be large enough to hold another fragment, but since the available storage is not contiguous, the fragment cannot be allocated without splitting, thereby requiring additional storage due to internal fragmentation of local storage spaces. Since a "top-down" design approach is used, "reasonable" estimates of the total storage requirement to obtain a feasible allocation for a given set of fragments required.

An estimate[2] of total storage, large enough for a feasible allocation, is available as input to Algorithm 3.1 which generates an initial feasible allocation and also to the optimization algorithms.

An initial allocation is obtained by applying an algorithm that makes placement decisions for the fragments based on the available storage at every site and the applications that reference objects in the fragments.

The inputs to the fragment allocation problem consists of:

- The set of n sites S $= \{s_1, s_2, \ldots, s_n\}$ with corresponding storage capacities D $= \{d_1, d_2, \ldots, d_n\}$
- The set of fragments F $= \{f_1, f_2, \ldots, f_m\}$; where $m \gg n$
- The Application Site Affinity matrix (ASA)
- The Application Fragment Affinity matrix (AFA)
- The network topology matrix (N) which holds the distance and connectivity information among sites. An entry in N, e_{ij} is 0 if $i = j$ or the shortest distance between s_i and s_j, if $i \neq j$.

[2] An estimate is obtained from a large number of feasible allocations by finding the storage required for each of them. The individual allocations are obtained by random placements of the fragments at different sites.

- $NDiam$ = the diameter of the network[3].

The output is a partition of the set F into n subsets $\{F_1, \ldots, F_n\}$, such that $\sum_{k|f_k \in F_i} v_k \leq d_i$; $\forall i = 1, \ldots, n$; where v_k is the size of fragment f_k and d_i is the storage capacity of site s_i and $\bigcup_i F_i = F$.

A brief sketch of the algorithm for initial allocation appears in Figure 3.1. A fragment is first placed at a site that refers to it (lines 3 -11). If this is impossible because there is insufficient storage space at any such site, another nearby one is selected. Since this will introduce communication overhead because of nonlocal references a site which is at the shortest distance from any of the sites that refer to it is selected (lines 12 - 21).

The fragments are sorted in descending order of their sizes so that the allocation scheme tries to find a place for the larger fragments first (line 1). The rationale behind this is that it is less difficult to allocate larger fragments at the beginning when there is sufficient large spaces to hold them at the individual sites while near the end of the initial placement process smaller fragments are more likely to fit into any of the small "holes" in storage spaces created from earlier placements. A "best-fit" placement strategy is used to reduce the amount of unused storage at individual sites. To obtain better performance, a list of sites is maintained sorted in ascending order of their *remaining* storage capacities (lines 2 and 20).

3.3 Estimation of Fragment Reference Cost

This section describes a method for estimating the communication cost involved in interfragment references between class fragments in a DOBS. A good estimate for the communication cost associated with each interfragment reference is necessary to obtain

[3] Diameter of a network is the maximum distance between any two nodes measured along the shortest path connecting them in the network.

Algorithm 3.1 *Initial Allocation*

begin algorithm Initial
input: S: The list of n sites
D: The set of available storages at every local site
F: The set of class fragments
V: The set of fragment sizes
ASA: The application site affinity matrix
AFA: The application fragment affinity matrix
N: The network topology matrix
$NDiam$: The diameter of the network
output: $\{F_1, F_2, \ldots, F_n\}$: Initial Allocation
var: *allocated*: an array of boolean flags for allocation status of every fragment

Step(1) :
Sort the set F into F' in descending order of the fragment sizes (1)
Sort the list S into S' in ascending order of the available storage (2)
Step(2) :
for every fragment f_i in the ordered set F' **do** (3)
    allocated$[i] \leftarrow$ **False** (4)
    Form sublist (S'') of sites with applications referencing the fragment (5)
Step(3) :
    for every site $s_j \in S''$ taken in order **do** (6)
Step(4) :
        if $(d_j \geq f_i)$ **then** (7)
            allocate fragment f_i in site s_j (8)
            $d_j \leftarrow d_j - v_i$ (9)
            allocated$[i] \leftarrow$ **True** (10)
            exit Step(4) (11)
        end if
    end for
{There is no space in any of the site that make reference to the fragment so look for space anywhere in the network to place the fragment.
We shall first try the sites that are nearest to those which refer the fragment.}
Step(5) :
    if (**not** *allocated*) **then** (12)
        for every site $s_p \in S$ taken in order **do** (13)
            for $k = 1$ to $NDiam$ **do** (14)
                Form sublist S''' that are a distance k from sites in S'' (15)
                for every site $s_q \in S'''$ taken in order **do** (16)
                    if $(d_q \geq v_i)$ **then** (17)
                        allocate fragment f_i in site s_q (18)
                        $d_q \leftarrow d_q - v_i$ (19)
                        update list S' to preserve the sorted order (20)
                        exit Step(5) (21)
                    end if
                end for
            end for
        end for
    end if
end for
end algorithm Initial

Figure 3.1: Initial Allocation Algorithm

an allocation scheme that minimizes overall communication overhead arising from all intersite data traffic.

Allocation decisions are closely related to "method execution plan". When a method executes, either the nonlocal attributes accessed by the method are "data shipped" to the *method site* or the method itself is "function shipped" to one of its nonlocal *data sites.* In this dissertation it is assumed that the method execution plan for all class methods is always based on the "data shipping" strategy and the goal of this research is to design an allocation scheme for the class fragments that minimizes the overall communication overhead involved in transferring attributes and/or objects between sites for all method executions.

Therefore, an estimate of the interfragment communication cost for every pair of class fragments is required. It is assumed that the methods are executed at the respective sites of their containing fragments as assigned by the initial allocation and all nonlocal attribute references involve data shipping the attributes from their respective sites to the method site. Heuristic optimization algorithms described in Sections 3.4.2 and 4.3.3 are then employed to minimize nonlocal interfragment communication costs. The allocation scheme is applied to a static objectbase so the sizes of the attributes and that of the methods[4] remain constant. Further the number and class membership (and hence the size) of the method invoking parameters are constant for every invocation of any class method. Also, the class membership of the results returned by any class method is assumed to be constant[5].

The following definitions are adopted:

Definition 3.3.1 *The cardinality of a homogeneous collection C of data entities, denoted as card(C)* is the number of data entities in that collection. The data entity can be an attribute, an object, or a collection of either of them. ■

[4]The size of a method is the size of its executable code.

[5]This information can either be obtained directly from the compiler or from static analysis.

Definition 3.3.2 *The size of an objectbase entity x, denoted as size(x)* is the number of bytes contained by x. The entity x can be a class fragment, an object, an attribute or a method or any collection of them. The size of a complex entity is the sum of the sizes of all of its constituent entities. ■

For attributes of fixed size data types (eg. integer or real), their sizes can be obtained from the 'type' information at compile time, but the exact size of an attribute of dynamic data types (*eg*, dynamic array, B-tree) is not known until runtime. An estimate of the size of such a dynamic attribute can be made as follows. Any dynamic size attribute can be treated as a set or collection of fixed size attributes (*eg*, nodes of a B-tree or a link list which can be of type integer or real) constituting the dynamic attribute[6]. In reality, the cardinality of the collection of a dynamic attribute may change with time. However, since a static environment for DOBS is considered in this dissertation, the cardinality of such collections associated with the dynamic size attributes is assumed to remain constant over time.

The *effective size* of a dynamic size attribute with respect to any method will depend on its 'selectivity'. The *selectivity* of a method on a dynamic size attribute is the proportion of the collection of the constituent attributes being referenced by the method. Therefore, as far as the method is concerned, the size of the dynamic size attribute can be estimated as the product of the selectivity factor, the cardinality of the collection, and the size of the fixed size attribute constituting the collection. *Selectivity factor* is evaluated from a static analysis as the average number of constituent attributes of a dynamic size attribute accessed by a method. Note that since the selectivity on a dynamic size attribute (and hence the effective size) can be different for different class methods and even for different executions of the same class method. However, the selectivity of a method on a dynamic size attribute is computed based on average number of constituent

[6] Assuming the collection is homogeneous. An attribute with heterogeneous data value collection can be viewed as a collection of homogeneous "collections" and the above definition apply recursively.

elements being accessed by all invocations of the method. Therefore, *selectivity* of any method on a dynamic attribute is considered to be a constant in this dissertation.

Definition 3.3.3 *The parset(m_i^j) of a method m_i^j in class C_i* is the set of parameters that needs to be passed as invoking arguments to m_i^j by the invoker at the time of its invocation. ■

Definition 3.3.4 *The result res(m_i^j) returned from the execution of method m_i^j of class C_i* is the data value(s) returned by the method to its invoker. ■

The elements in the parameter set and the result returned from execution of a class method can be an attribute of an object, an object, or a collection of objects and can either be of fixed or dynamic size. Their sizes can be obtained either at compile time (for static data type) or can be estimated from their dynamic data types.

Definition 3.3.5 *The location of a method m_i^j of class C_i, represented as siteof(m_i^j)* is the site $s_i \in S$ that is assigned the class fragments containing the method. ■

Definition 3.3.6 *The distance dist(s_i, s_j) between any two sites s_i and s_j, $s_i, s_j \in S$* is the shortest "data path" along the network links connecting s_i and s_j in the network topology. ■

Interfragment references can be classified as (1) *Method-Attribute Reference*, where a method in one fragment accesses one or more attributes in another fragment; and (2) *Method-Method Reference*, where a method in one fragment, in the process of its execution, invokes one or more methods in another. Interfragment data communication ($IFDC$) arising from Method-Attribute reference for method m_l in fragment f_i accessing attribute a_k in fragment f_j is given by

$$IFDC_{MA}(m_l^{f_i}, a_k^{f_j}) = \sum_{i|q_i \in Q} acc(q_i, m_l) * MAU(m_l, a_k) * size(a_k)$$

Therefore, assuming that the data communication cost is directly proportional to the distance between the origin and destination site, the interfragment communication cost ($IFCC$) due to Method-Attribute reference between method $m_l \in f_i$ and all attributes $a_k \in f_j$ and $a_k \in MAR(m_l)$ is

$$IFCC_{MA}(m_l^{f_i}, f_j) = \sum_{k|a_k \in f_j} IFDC_{MA}(m_l^{f_i}, a_k^{f_j}) * dist(siteof(f_i), siteof(f_j))$$

Interfragment data communication ($IFDC$) due to Method-Method reference can be viewed as the sum of (1) the data traffic due to method invocation process ($IFDC_{Minv}$) and (2) the data traffic from the result transfer of method executions to the invoker ($IFDC_{Rtrn}$). Therefore,

$$IFDC_{MM}(m_l, m_n) = \sum_{i|q_i \in Q} (IFDC_{Minv} + IFDC_{Rtrn}) * acc(q_i, m_l) * MMU(m_l, m_n)$$

$$IFDC_{Minv}(m_l^{f_i}, m_n^{f_j}) = \sum_{i|p_i \in parset(m_n)} size(p_i)$$

and

$$IFDC_{Rtrn}(m_l, m_n) = size(res(m_n))$$

The communication cost for Method-Method reference for $m_l \in f_i$ invoking $m_n \in f_j$ is given by

$$IFCC_{MM}(m_l^{f_i}, m_n^{f_j}) = (IFDC_{Minv} + IFDC_{Rtrn}) * dist(siteof(m_l^{f_i}), siteof(m_n^{f_j}))$$

Definition 3.3.7 *Fragment Reference Cost $FRC(i,j)$ from fragment f_i to fragment f_j* is the total cost of communication due to all references made by any method in fragment f_i to any attribute or method in fragment f_j for all possible application executions. ■

Therefore, *Fragment Reference Cost* $FRC(i,j)$ *due to references from fragment* f_i *to fragment* f_j is the sum of all interfragment Method-Attribute references and interfragment Method-Method references from f_i to f_j.

$$FRC(i,j) = \sum_{l|m_l \in f_i} (IFCC_{MA}(m_l, f_j) + \sum_{n|m_n \in f_j \wedge m_n \in MMR(m_l)} IFCC_{MM}(m_l, m_n))$$

The *Interfragment Reference Cost (IFRC)* between any two class fragments is the sum of all *fragment reference costs (FRCs)* for references in both directions between them.

Definition 3.3.8 *Interfragment Reference Cost* $IFRC(i,j)$ *between fragment* f_i *and fragment* f_j is the sum of the fragment reference costs(FRC) from fragment f_i to fragment f_j and that from fragment f_j to fragment f_i. $IFRC(i,j) = FRC(i,j) + FRC(j,i)$ ■

The *Inter Fragment Reference Cost Matrix* $(IFRC)$ whose $(i,j)^{th}$ element is $IFRC(i,j)$ can then be constructed.

Due to object encapsulation, applications can access any class fragment only by invoking one of the methods contained by the fragment. Therefore any reference to a class fragment by an application can be treated identically to *Interfragment Method-Method Reference*. Hence, the data communication cost due to *Application Fragment Reference Cost* between application q_k and fragment f_i can be written as

$$AFCC(q_k, f_i) = \sum_{l|m_l \in f_i \wedge m_l \in MIS(q_k)} acc(q_k, m_l) * (AMDC_{Minv}(q_k, m_l) + AMDC_{Rtrn}) * dist(siteof(q_k), siteof(f_i)) \quad (3.1)$$

where

$$AMDC_{Minv}(q_k, m_l) = \sum_{j|p_j \in parset(m_l)} size(p_j)$$

and

$$AMDC_{Rtrn}(q_k, m_l) = size(res(m_l))$$

Definition 3.3.9 *Site Affinity* $saff(f_i, s_j)$ *of a class fragment* f_i *at site* s_j is the total communication cost due to all references to fragment f_i made by all local applications executing at site s_j.

$$saff(f_i, s_j) = \sum_{k|q_k \in Q \wedge ASA(q_k, s_j) \neq 0} AFCC(q_k, f_i)$$

■

3.4 Heuristic Optimization: The Graphical Approach

In this section we sketch the heuristic procedure used to obtain a "near optimal" allocation from the initial allocation obtained using Algorithm 3.1 above. The design of the optimization algorithm is based on work in graph theory by Kernighan and Lin [KL70]. A brief outline of their work with its analogy to the allocation problem in DOBS is presented in Section 3.4.1. The optimization algorithm for the proposed allocation model is presented in Section 3.4.2.

3.4.1 The Kernighan-Lin Approach

Kernighan and Lin [KL70] address the graph partitioning problem of dividing an undirected weighted graph into a given number of partitions so the sum of the weights associated with the edges between the partitions is minimal.

In the model they present a graph representing a set S of $2n$ nodes with an associated weight matrix $W = [w_{ij}]$, $i, j = 1, \ldots, 2n$ that is to be partitioned into a given number of components. Every element w_{ij} refers to the weight associated with the edge between nodes i and j. The authors suggest a two-way partitioning scheme that divides the set S into two equal subsets each having n nodes so the sum of the weights corresponding to the edges joining the nodes in one subset to the nodes in the other subset is minimized.

The method is heuristic in nature and attempts to find a near optimal two-way partition through an interchange process.

Starting from an arbitrary initial partition[7], where S is split into S_1 and S_2, the interchange process swaps nodes between the two initial partitions to minimize the sum of the weights associated with the edges between the two sets. The process is carried on iteratively, giving a sequence of interchanges, each reducing the total weight of all the edges joining the two partitions. Local optimality is reached when no further reduction is possible through the interchange process. Repeated application of the two-way partitioning scheme to pairs of initial partitions is suggested as an extension of the proposed scheme to multiple-way partitioning problems.

This graph partitioning problem relates closely to the problem of allocation of class fragments in a DOBS. In the graphical formulation of the object allocation problem, the set of class fragments can be considered analogous to the set of nodes and the interfragment references and their associated costs are analogous to the weighted edges between the nodes. Therefore associating a reference cost (IFRC) with every pair of fragments gives a measure of communication cost due to references between the class fragments. Accordingly, an *Inter Fragment Reference Cost (IFRC)* matrix, analogous to the weight matrix W, can be constructed. The objective of the DOBS allocation scheme can then be restated as *divide the set of class fragments into 'n' partitions, where n is the number of sites, so that the total cost of all nonlocal interfragment references is minimum.* This analogy suggests that the graph partitioning technique proposed by Kernighan and Lin may be adapted, with the necessary modifications, for the design of an allocation scheme for DOBS. The following differences are identified between the graph partitioning model of Kernighan and Lin and the DOBS allocation model that require modifications to the graph partitioning algorithm:

[7] Kernighan and Lin do not specify how the initial partition is obtained. This is the major drawback of their proposal.

1. The KL algorithm[8] does not have any size constraints on the partitions generated and therefore the partitions can be of any arbitrary size. In the DOBS allocation problem, each of the network sites has a finite amount of available storage and the sum of the sizes of the assigned fragments to any particular site must not exceed the available storage of that site, otherwise the solution becomes infeasible. Therefore, size constraints are applied independently on each site for the DOBS allocation problem and an exchange and/or move of any fragment between site pairs is only allowed by the algorithm provided the resulting allocation does not violate the storage constraints at either site.

2. The KL algorithm only deals with equal size partitions having equal number of elements and the elements are always exchanged between any two partitions so that the number of elements in each partition remains constant. In DOBS allocation, the initial allocation could result in an allocation where the numbers of fragments in different sites are not necessarily equal. To handle this problem, "dummy" fragments are added to the "real" fragments to make the number of fragments in each of the sites in the pair equal. The graph partitioning algorithm is then applied on the sites with equal number of fragments. Any exchange of a "real" fragment with a "dummy" one is treated as a "move" of the real fragment to the site of the dummy fragment.

3. The KL algorithm, when applied to a pair of arbitrary partitions, guarantees convergence because the gains from the individual exchanges of element pairs, involving all the elements of the partitions, would sum to zero. This is because when all the elements in one partition are exchanged with all the elements of the other, we get back the original partitions. This does not hold true for DOBS allocation algorithm. Since the fragments may have different site affinities for the two sites participating

[8] In rest of this dissertation the graph partitioning algorithm proposed by Kernighan and Lin will be called the KL algorithm.

in the fragment exchange process, even if all fragments of one site are exchanged with all fragments of another, the resulting partitions would correspond to a different value of the cost function. Since the convergence of pairwise optimality is not ensured under the conditions of the DOBS allocation problem, the fragment interchange process between any site pair is only carried out until the gain becomes zero or negative, or all the fragments at either or both the sites are employed in the exchange process.

4. The problem of DOBS allocation, when formulated as a graph partitioning problem, results in a multiple-way partitioning problem. The algorithm presented by [KL70] is intended for obtaining a near optimal solution for the two-way partitioning problem. Repeated application of the two-way partitioning scheme to pairs of the starting partitions is suggested as an extension of the proposed scheme for multiple-way partitioning problems. No guarantee is given about the convergence of the process. The suggested technique, when applied to DOBS allocation to obtain a n-way (where n is the number of network sites) partition of the objectbase, has been found experimentally to fail because of this problem. When the same site is involved in multiple site pairs on which the two-way partitioning process is applied independently, any change in the set of assigned fragments to the site may destroy the pairwise optimality obtained between other site pairs previously considered. This mutual destructive interference of pairwise optimality between sites may lead to a combinatorial problem in converging in the search for a "truly pairwise optimal" allocation. To guarantee convergence in reasonable time, the following trade off is made. After each iteration the cost of the resulting allocation is compared with its previous value. The process is terminated if the current cost is higher and the solution obtained in the previous iteration is taken as the final solution.

The modified graph partitioning algorithm is then applied to $IFRC$ to obtain the desired allocation from an initial starting solution. In Section 3.2 an algorithm that

generates one such initial solution bounded by the storage capacity constraints and the network topology is presented. This initial solution becomes the input to the heuristic optimization scheme based on the Kernighan-Lin approach.

3.4.2 A Heuristic Algorithm Based on Graphical Approach

This section describes the optimization algorithm based on the heuristic approach adopted by Kernighan and Lin in their graph partition problem. The algorithm attempts to improve the initial distribution of fragments between a pair of sites by interchanging fragments to obtain a distribution that is locally optimal with respect to the nonlocal references between them. The algorithm is applied repeatedly on every pair of sites in S to obtain an allocation that is as close as possible to a pairwise optimal solution.

Based on the $IFRC$, the access relationships between the fragments are represented as an undirected, weighted graph with nodes for each fragment and the weights on the edges determined by the total reference cost between the pair of fragments. This graph is called the *Fragment Reference Graph (FRG).*

The fragment allocation problem can be reformulated as a partitioning problem that divides the FRG into n subgraphs, where n is the number of sites in the network, such that the cost of references between the sites is minimized and the site capacities are respected. Each of the partitions can then be mapped on to a network site to obtain an allocation that minimizes the total cost of all nonlocal interfragment references. Since the communication cost due to the nonlocal references dominates the overall cost of allocation, the allocation obtained can be treated as a "near optimal" solution to the allocation problem in DOBS.

The approach taken is as follows. Starting from an initial allocation (obtained by Algorithm 3.1 or any other algorithm that generates an initial feasible allocation), fragments are interchanged between every pair of sites to reduce the cost due to nonlocal interfragment references. However, when a fragment changes site, its site affinity changes

as well. A decrease in site affinity would lead to an increase in the cost due to remote invocations of any method, contained by the fragment, accessed by an application directly[9]. Therefore, as far as site affinity is concerned, a "gain" results only if the fragment is moved to a site with a higher site affinity for that fragment. Only those fragments are interchanged that produce a "net gain" taking interfragment references and site affinities into account. Net gain corresponds to the total change in the 'cost function' due to changes in interfragment reference cost($IFRC$) and site affinity ($saff$). Consider an arbitrary pair of sites (A, B), $A, B \in S$. Let the initial allocation at site A and site B be F_A and F_B, respectively. For each fragment $f_a \in F_A$, an *external reference cost* (ERC_a) is computed which is the sum of the communication costs of all nonlocal references made to f_a from all the fragments in site B and an *internal reference cost* (IRC_a) is also computed which is the sum of the cost of all references made to it from within site A. Therefore,

$$IRC_a = \sum_{k|f_k \in F_A, f_k \neq f_a} IFRC(f_a, f_k)$$

and

$$ERC_a = \sum_{k|f_k \in F_B} IFRC(f_a, f_k)$$

Next ERC_b and IRC_b is calculated for every fragment $f_b \in B$.

The site affinities are evaluated as follows:

$$\left.\begin{array}{l} saff(f_a, A) = \sum_{k|q_k \in Q \wedge ASA(q_k, A) \neq 0} AFCC(q_k, f_a) \\ saff(f_a, B) = \sum_{k|q_k \in Q \wedge ASA(q_k, A) \neq 0} AFCC(q_k, f_a) \end{array}\right\} \forall f_a \in F_A$$

and

$$\left.\begin{array}{l} saff(f_b, B) = \sum_{k|q_k \in Q \wedge ASA(q_k, B) \neq 0} AFCC(q_k, f_b) \\ saff(f_b, A) = \sum_{k|q_k \in Q \wedge ASA(q_k, B) \neq 0} AFCC(q_k, f_b) \end{array}\right\} \forall f_b \in F_B$$

[9] An application accesses a class fragment directly if it invokes a method in that fragment.

When a fragment f_a in node A is interchanged with a fragment f_b in node B, all local references becomes nonlocal and vice versa for both f_a and f_b. From [KL70], for any $f_a \in F_A$ and $f_b \in F_B$, if f_a and f_b are interchanged between sites A and B, the reduction in communication cost due to nonlocal interfragment references is $D_a + D_b - 2r_{ab}$, where $D_a = ERC_a - IRC_a, D_b = ERC_b - IRC_b$ and r_{ab} = reference cost between f_a and f_b. The increase in site affinity due to the interchange for fragment f_a is $\Delta_a = saff(f_a, B) - saff(f_a, A)$, and that for fragment f_b is $\Delta_b = saff(f_b, A) - saff(f_b, B)$. Therefore the total *gain* in the interchange of fragments f_a and f_b is given by $gain = D_a + D_b - 2r_{ab} + \Delta_a + \Delta_b$.

The interchange between a *dummy* and a *real* fragment essentially leads to moving the *real* fragment to the site of the *dummy* fragment. The contribution to the corresponding *gain* comes only from the *real* fragment since the *D value* and *site affinities* of a *dummy* fragment is zero. Since the two-way partitioning scheme applies only to sites with an equal number of fragments, two *dummy* fragments are added to the site from which the *real* fragment is removed. Physical movements of fragments between sites are effected only if the interchange is permitted by the storage constraints. The interchange of fragments between sites will also cause a change in the *D values* of the remaining fragments at both the sites as follows [KL70]:

$$D'_x = D_x + 2r_{xa} - 2r_{xb}, \ \forall x \in A - \{f_a\}$$

$$D'_y = D_y + 2r_{yb} - 2r_{ya}, \ \forall y \in B - \{f_b\}$$

The algorithm for fragment exchange is presented in Figure 3.3.

Fragments are interchanged between pairs of sites in iterative steps. One pair of sites is selected at a time for fragment interchange. At every step, a pair of fragments, one from each site, is selected that produces maximum positive gain if they are interchanged between the two sites.

The process is then repeated until there are no fragment pairs producing any additional gain on interchange. When no more interchange produces any positive gain, the fragment distribution with respect to the pair of sites is optimal.

The algorithm for pairwise optimal allocation between a pair of sites is formally presented in Figure 3.2. Four logical steps are required:

Step One

The *local optimality* is reset to *false* for the site pair (Figure 3.2, line 5). *Dummy* fragments are added, if required to make the number of fragments at both sites equal (Figure 3.2, lines 6-9). The *site affinity* measures are computed for every fragment in A and B with respect to both the sites (Figure 3.2, lines 10-11). Step 2 through Step 4 are repeated until a pairwise optimal fragment allocation between the two selected sites is obtained.

Step Two

The *cumulative gain* which gives the total gain or reduction in cost due to all fragment exchange made so far is initialized to zero (Figure 3.2, line 13). D_a and D_b values are computed for every fragments at site A and site B (Figure 3.2, line 14).

Step Three

For every possible pair of fragments in site A and site B, potential *gain* in interchanging them between the two sites is calculated (Figure 3.2, line 18). The pair that has the highest potential positive gain is selected for interchange. The fragments are interchange using Algorithm 3.3. The selected fragment pair is removed from the set of fragments at each site. The gain corresponding to the pair selected is added to cumulative gain and the iteration is continued until either of the site runs out of fragments (for pairing with another fragment of the other site) or no pair can be formed that has a positive potential gain on interchange (Figure 3.2, lines 18-24).

Step Four

If there is no fragment pair found that contributes any positive gain in any iteration, then the allocation of fragments between the two sites is pairwise optimal. Local optimal flag

Algorithm 3.2 *Graphical Approach - Pairwise Optimization*

begin algorithm PairOptimize
input: (A, B):The pair of sites selected for optimization
var: *locally optimal*: boolean flag indicates pairwise optimality of allocation
gain: Gain corresponding to the fragment pairs selected for interchange
cumulative gain: Total gain for all chosen fragment pairs for interchange
$\mathcal{F}_A, \mathcal{F'}_A$: set of fragments holding copies of allocation at site A
$\mathcal{F}_B, \mathcal{F'}_B$: set of fragments holding copies of allocation at site B

$\mathcal{F}_A \leftarrow F_A^c$ (1)
$\mathcal{F}_B \leftarrow F_B^c$ (2)
if ($\mathcal{F}_A = \emptyset$ **and** $\mathcal{F}_B = \emptyset$) **then** (3)
return (4)
Step(1) : *locally optimal* $\leftarrow$ **False** (5)
if (No. of fragments in $\mathcal{F}_A >$ No. of fragments in $\mathcal{F}_B$) **then** (6)
Add *dummy* fragments to $\mathcal{F}_B$ to make no. of frags in $\mathcal{F}_A$ and $\mathcal{F}_B$ equal (7)
else if (No. of fragments in $\mathcal{F}_B >$ No. of fragments in $\mathcal{F}_A$) **then** (8)
Add *dummy* fragments to $\mathcal{F}_A$ to make no. of frags in $\mathcal{F}_A$ and $\mathcal{F}_B$ equal (9)
endif
Compute $saff(f_a, A)$ and $saff(f_a, B)$ $\forall f_a \in \mathcal{F}_A$ using ASA and AFA (10)
Compute $saff(f_b, B)$ and $saff(f_b, A)$ $\forall f_b \in \mathcal{F}_B$ using ASA and AFA (11)
repeat step(2) through step(4) **until** (**not** *locally optimal*) (12)
Step(2) : Initialize cumulative gain $\leftarrow 0$. (13)
Compute D_a for all $f_a \in \mathcal{F}_A$ and D_b for all $f_b \in \mathcal{F}_B$ using $IFRC$. (14)
$\mathcal{F}'_A \leftarrow \mathcal{F}_A$ (15)
$\mathcal{F}'_B \leftarrow \mathcal{F}_B$ (16)
repeat Step(3) **until** (($\mathcal{F}'_A = \emptyset$) **or** ($\mathcal{F}'_B = \emptyset$) **or** ($gain \leq 0$)) (17)
Step(3) : Select $f_a \in \mathcal{F}'_A$ and $f_b \in \mathcal{F}'_B$ such that
$gain = D_a + D_b - 2r_{ab} + \Delta_a + \Delta_b$ is maximum (18)
if ($gain \leq 0$) **then** (19)
$cgain \leftarrow 0$ (20)
exit Step(3) (21)
else
XchangeFragment(f_a, f_b); (22)
$\mathcal{F}'_A \leftarrow \mathcal{F}'_A - f_a$ (23)
$\mathcal{F}'_B \leftarrow \mathcal{F}'_B - f_b$ (24)
end repeat
Step(4) : **if** (cumulative gain = 0) **then** (25)
locally optimal $\leftarrow$ **True** (26)
exit Step(4) (27)
endif
end repeat
$F_A^c \leftarrow \mathcal{F}_A$ (28)
$F_B^c \leftarrow \mathcal{F}_B$ (29)
end for
end algorithm PairOptimize

Figure 3.2: Pairwise Optimal Allocation Algorithm

Algorithm 3.3 *Fragment Pair Exchange*

```
begin algorithm XchangeFragment
Input:          (f_a ∈ F_A, f_b ∈ F_B): The fragment pair selected for exchange
if (f_a = dummy fragment) then                                              (1)
      move f_b to site A subject to d_A                                     (2)
      if (move is successful)                                               (3)
            update all D_a ∈ F'_A and all D_b ∈ F'_B                         (4)
            add 2 dummy fragments to F'_B to make its size equal to F'_A    (5)
            cumulative gain ← cumulative gain + gain                        (6)
      endif
else if (f_b = dummy fragment) then                                         (7)
      move f_a to site B subject to d_B                                     (8)
      if (move is successful)                                               (9)
            update all D_a ∈ F'_A and all D_b ∈ F'_B                         (10)
            add 2 dummy fragments to F'_A to make its size equal to F'_B    (11)
            cumulative gain ← cumulative gain + gain                        (12)
      endif
else exchange f_a and f_b between F_A and F_B subject to d_A and d_B         (13)
      if (exchange is successful) then                                      (14)
            update all D_a ∈ F'_A and all D_b ∈ F'_B                         (15)
            cumulative gain ← cumulative gain + gain                        (16)
      endif
endif
end algorithm XchangeFragment
```

Figure 3.3: Fragment Exchange Algorithm

Algorithm 3.4 *Evaluating cost of an allocation*

```
begin algorithm CostEval
input:          Allocation={F_1, F_2, ..., F_n}:
output:         Cost: Cost of allocation
Cost ← 0 /* Initialize Cost */                                              (1)
for every pair of fragment(f_i, f_j), i ≠ j do                              (2)
     if ((f_i ∈ F_p and f_j ∈ F_q) and (p ≠ q)) then                        (3)
          Cost=Cost+IFRC(i,j) * distance(s_p, s_q)                          (4)
endfor
return(Cost)                                                                (5)
end algorithm CostEval
```

Figure 3.4: Allocation cost evaluation algorithm

Algorithm 3.5 *Heuristic Optimization - Graphical Approach*

begin algorithm GlobalOptimize
input: **IFRM:**The Interfragment reference matrix
ASA: The Application Site Affinity matrix
AFA: The Application Fragment Affinity matrix
$D = \{d_1, d_2, \ldots, d_n\}$: The set of available storage at every site
$V = \{v_1, v_2, \ldots, v_m\}$: The set of fragment sizes of all class fragments
$IAlloc = \{F_1^i, F_2^i, \ldots, F_n^i\}$: Initial allocation
output: $FAlloc = \{F_1^f, F_2^f, \ldots, F_n^f\}$: The final allocation after optimization
FCost: The final cost of the optimized allocation
var: *oldcost*: cost of allocation in previous iteration
newcost: cost of current allocation
globally optimal: Flag to indicate global optimality
chgalloc: Boolean flag, one for every site, to indicate change of allocation
$CAlloc = \{F_1^c, F_2^c, \ldots, F_n^c\}$: Current allocation
$BestAlloc = \{F_1^b, F_2^b, \ldots, F_n^b\}$: Best Allocation obtained so far

```
Initialize globally optimal ← False /* global optimal flag is set to FALSE */   (1)
oldcost ← CostEval(IAlloc);                                                     (2)
CAlloc = IAlloc                                                                 (3)
BestAlloc = IAlloc                                                              (4)
while (not globally optimal) do                                                 (5)
    globally optimal ← True                                                     (6)
    Initialize chgalloc[i] ← False, ∀s_i ∈ S                                    (7)
    for every pair of sites (A, B), A, B ∈ S do                                 (8)
        PairOptimize(A, B); /* call procedure PairOptimize for site pair (A,B) */ (9)
    endfor
    newcost ← CostEval(CAlloc)                                                  (10)
    for every s_i ∈ S do                                                        (11)
        if (chgalloc[i]=True)                                                   (12)
            globally optimal ← False                                            (13)
    endfor
    if (newcost ≤ oldcost) then                                                 (14)
        oldcost ← newcost                                                       (15)
        BestAlloc ← CAlloc                                                      (16)
    else
        globally optimal ← True                                                 (17)
    endif
endwhile
Fcost ← oldcost                                                                 (18)
FAlloc ← BestAlloc                                                              (19)
return(FAlloc,Fcost)                                                            (20)
end algorithm GlobalOptimize
```

Figure 3.5: The Global Optimization Algorithm

is set *true* and the algorithm terminates (Figure 3.2, lines 25-29).

Algorithm 3.5 applies Algorithm 3.2 repeatedly to every pair of sites in S to arrive at an allocation that is "as close as possible" to *pairwise optimal* across all pairs of sites.

The cost of allocation is evaluated using Algorithm 3.4 after each iteration and the minimum cost allocation obtained so far is recorded. The Algorithm 3.4 uses interfragment reference cost matrix and the network topology matrix for the evaluation of cost of any allocation. For every nonlocal reference, the corresponding cost is weighted by the length of the data path between the origin and destination site and the weighted sum of all nonlocal interfragment references is taken as the cost of the allocation.

A set of boolean flags are used to indicate the sites whose allocation is changed by the current iteration (Figure 3.5, lines 1,7). If the allocation of all sites are unchanged in any iteration, then a pairwise optimal allocation is obtained and the algorithm terminates (Figure 3.5, line 13). Alternatively, the cost of current allocation is compared with that of the best solution seen and the algorithm terminates if the current cost is higher than that of the best solution. In that case, the best solution is taken as the final allocation (Figure 3.5, lines 14-23).

An allocation algorithm based on graphical approach is presented in this Chapter. The heuristic optimization algorithm is designed to improve an initial allocation using graph partitioning technique. The results obtained by applying this algorithm on a set of test data representing a DOBS prototype are presented in Tables 5.1-5.4. An alternative approach for DOBS allocation design using simulated annealing as the heuristic principle is described in Chapter 4.

Chapter 4

Static Allocation: Simulated Annealing Approach

This chapter describes the simulated annealing approach to the allocation problem in DOBSs. Section 4.1 gives a brief overview of the fundamental concepts and general principles of simulated annealing. The simulated annealing algorithm is described in general in Section 4.2. Section 4.3 presents how this technique can be applied to the DOBS allocation problem followed by a discussion on implementation issues of the algorithm in Section 4.4. The results obtained by the algorithm based on simulated annealing are presented in Tables 5.1 to 5.4.

4.1 Simulated Annealing: Fundamental Concepts

This section presents the fundamental concepts and principles behind the simulated annealing process and its various applications.

Simulated Annealing is a general purpose combinatorial optimization technique based on stochastic principles for solving discrete optimization problems. The goal of

simulated annealing is to identify an 'optimal' or 'near optimal' solution from the solution space of the problem. This heuristic approach addresses optimization problems where the solution spaces are so large that any analytical method or enumerative search method (eg. branch and bound method) leads to computational intractability.

The simulated annealing methodology is used to develop efficient search techniques for an optimization problem to find a solution that corresponds to the optimal value of a function of many independent variables. This function, usually called the *cost or objective function*, represents a quantitative measure of 'goodness' of the 'state' or configuration of some complex system. The method relies on probabilistically accepting intermediate 'inferior' solutions through a set of user-controlled parameters. At each step a 'state'[1] is selected at random from the solution space that represents a feasible solution. The cost function for the state is evaluated and is tested against a set of rules that decides the acceptability of the solution. As the algorithm progresses, it eliminates certain feasible solutions from its search path, thereby reducing the intractable problem to one solvable in finite time. A control variable, commonly known as *temperature*[2] is used to control the probability of accepting *up-hill moves* and gives direction to the search for successful convergence. It is expressed in the same units as the objective function. An algorithm based on greedy heuristic monotonically selects only improved solutions which often leads to the algorithm being trapped in a poor local optima. In contrast, the simulated annealing process accepts, with certain probability, a solution that is "inferior" relative to the last accepted solution, thereby introducing some "chance" of coming out of the trap of a poor local optima. Although there is no guarantee that the solution obtained by simulated annealing would be optimal or even near optimal, results obtained by application of the technique in various fields (eg., VLSI circuit design problem [WM94], traveling salesman problem, database partitioning problem [WDIY88] etc.) indicate promising potential of

[1] Each state corresponds to a solution in the solution space and is defined by a set of state variables.

[2] In keeping with the terminology used by Kirkpatrick *et al.* [KGV83] this thesis also refers to this control parameter as *temperature*.

the methodology in solving computationally intractable problems in a reasonable time.

4.2 Simulated Annealing Algorithm

The simulated annealing algorithm is a generic combinatorial optimization algorithm based on the same annealing process. However, construction of the cost function, the solution acceptance criteria, and the procedure to obtain the initial and subsequent states depend on the nature and characteristics of the specific problem. The algorithm selects a feasible solution from the solution space of the problem at random and tests for its acceptance as a candidate solution for optimality. Given a random solution to the problem the solution is accepted as a candidate solution for optimality based on the following *acceptance rules*:

Acceptance Rule 4.2.1 The current solution is always accepted if it is *better* compared to its preceding solution in the search sequence.

Acceptance Rule 4.2.2 An *inferior* current solution is accepted based on an exponentially decaying probability (a function of temperature known as *acceptance function* [WM94] or *acceptance probability* [HRSV86]) whose value decreases as more solutions from the solution space are considered, leading to the convergence of the annealing process.

The first solution acceptance criteria performs the optimization by continually choosing the best solution. The second criteria allows the algorithm to work on a temporary inferior solution that may allow it to reach a better solution in future. This is required because continually selecting the best solution (like a greedy algorithm) may lead to a poor local minima. Solutions are accepted in iterative blocks. Each block is identified by a specific value of the annealing parameter *temperature*. Solutions are accepted in a sequence until an *equilibrium condition* is reached when the *temperature* is reduced by a

predefined factor, known as the *cooling rate* and another block of iteration with the new temperature is started. The converging condition of the annealing process is defined by the *freezing condition* that determines the precision of the final result. Starting from an *Initial temperature*, the simulated annealing proceeds to search for the "best possible" solution guided by the *annealing schedule*. An *annealing schedule* is therefore, defined by (1) *initial temperature*, (2) *cooling rate*, (3) *equilibrium condition*, and (4) *freezing condition*.

A brief sketch of the generic algorithm based on simulated annealing is given in Figure 4.1.

4.3 Application of Simulated Annealing to DOBS Allocation

This section reformulates the allocation problem in DOBS for the application of the simulated annealing as the heuristic optimization technique. DOBS allocation is a combinatorial optimization problem involving a large set of decision variables each determining the placement of a class fragment across the sites of the network. This indicates that obtaining a solution for the DOBS allocation problem by classical methods would be too costly and impractical, if not impossible. Therefore, heuristic methods are employed. A heuristic method based on graphical approach is described in Chapter 3. An alternative approach using simulated annealing as the heuristic technique is presented here.

The central concept of simulated annealing is the formulation of the *cost function* that provides direction to the annealing process. Formulation of the *cost function* for DOBS allocation problem is described next followed by the method of selecting the *annealing parameters*. The simulated annealing algorithm for DOBS allocation is formally presented in Section 4.3.3 and its implementation issues are discussed in Section 4.4.

Algorithm 4.1 *Simulated Annealing Algorithm-Generic*

begin algorithm SimAnnGen

input: $\mathcal{S}$: set of initial feasible solutions
$\mathcal{T}$: initial temperature
r: cooling rate
$\mathcal{E}cond$: condition for equilibrium
$\mathcal{F}cond$: condition for freezing
$\mathcal{C}$: Cost Function
output: optimal feasible solution
var: *temperature*: control variable
C_{curr}: cost of current solution
C_{last}: cost of last accepted solution
Δ: cost difference

```
temperature ← T /* Set temperature to T */                                  (1)
Select at random an initial feasible solution s0 ∈ S                       (2)
C_last ← C(s0)                                                             (3)
while (not Fcond) do                                                       (4)
    while (not Econd) do                                                   (5)
        Select at random a solution from the set S of feasible solutions   (6)
        Compute the value C_curr of the cost function C for that solution  (6)
        Δ ← C_curr − C_last                                                (8)
        if (Δ < 0) then                                                    (9)
            accept the current solution as a candidate solution            (10)
            C_last ← C_curr                                                (11)
        else
            if (random() < e^(−Δ/T)) then                                  (12)
                accept current solution                                    (13)
                C_last ← C_curr                                            (14)
            end if
        end if
    end while
    temperature ← T * r                                                    (15)
end while
```

The last successfully accepted solution is the final solution.
end algorithm Simulated-Annealing

Figure 4.1: Generic Simulated Annealing Algorithm

4.3.1 Formulation of the Cost Function

For the allocation problem in DOBS the following factors are considered in the cost function based on the assumptions made in the model. A fragment has two types of affinities: (1) *Fragment Affinity* which is its affinity with other fragments and (2) *Site Affinity* which is its affinity with the local applications at any site.

Fragment Affinity is measured by the *interfragment reference costs (IFRC).* Assuming that cost due to affinity with local fragments can be ignored, only the nonlocal references between class fragments are used to compute the *cost* of placing a fragment at a particular site. Therefore, the "cost term"($C_{f_i}^{FA}$) for *Fragment Affinity* of fragment f_i can be represented as:

$$C_{f_i}^{FA} = \frac{1}{2}\sum_{i=1}^{m} \sum_{j|siteof(f_j)\neq siteof(f_i)} IFRC(i,j)$$

The "site affinity" of any fragment for a particular site is measured by the cost of communication for all local applications at that site accessing the fragment. Therefore, site affinity of fragment f_i at site s_j can be written as (recall Definition given in Chapter 3):

$$saff(f_i, s_j) = \sum_{k|q_k \in Q \wedge ASA(q_k,s_j)\neq 0} AFCC(q_k, f_i)$$

Therefore, a *site affinity matrix (SAM)* is constructed whose $(i,j)^{th}$ term is $saff(f_i, s_j)$. Summing up the costs due to all references made by local applications on the fragment computes the contribution of *site affinity* to the cost function. The "cost term" for *Site Affinity* of a fragment f_i at site s_j is given by (recall definition given in Chapter 3)

$$C_{f_i}^{SA} = \sum_{j|s_j\neq siteof(f_i)} saff(f_i, s_j)$$

The *cost function* C evaluates the total cost of placements of all fragments for a particular allocation as a sum of the cost of allocating individual fragments taking its

fragment affinity and site affinity into consideration. Therefore, the *cost function* can be expressed as :

$$C = \sum_{i=1}^{m} (C_{f_i}^{FA} + C_{f_i}^{SA})$$

4.3.2 Selection of Control Parameters

The parameters defining the *annealing schedule* that guides the annealing process are: (1) the *initial temperature*, (2) the *cooling rate*, (3) the *equilibrium condition*, and (4) *freezing* or *terminating condition.*

Selection of appropriate values for these control parameters is a nontrivial but critical task. The control parameters determine the way the solution space is traversed by the algorithm. How good an approximation to the optimal solution obtainable depends on the choice of the values of the control parameters. The choice of *initial temperature* and the *cooling rate* determine the size of the solutions space which is searched by the algorithm for an optimal solution. Further the *cooling rate* also dictates how quickly the *freezing condition* is reached from the *initial temperature.*

One of the major issues of simulated annealing is its long computation time. All other things being equal, a reduction in the *initial temperature* usually results in a decrease in computation time [WM94]. However, a sufficient number of solutions must be tested for a good approximation of the optimal solution and a low starting temperature would fail to accept many intermediate solutions that have potentials to lead the search process to a "near optimal" solution.

In a related work by Wolf *et al.* [WDIY88] simulated annealing is used to obtain an appropriate architecture for a hybrid data sharing-data partitioning architecture for transaction processing. Based on the analysis presented by them, the *initial temperature*

for the annealing process to find the optimal allocation in DOBS is set to 10.0. The unit of the control parameter (*temperature*) is the same as that of the cost function.

Implementation of the *acceptance function* must ensure that its range is the real domain $(0.00, 1.00)$ so that the value of the function is comparable to the value generated from a random number generator. However, the *acceptance function* involves Δ representing the difference between the cost of current allocation and that of the last accepted allocation in a sequence of solution acceptance. Some normalization is therefore required in the exponent $-\frac{\Delta}{T}$ so that $0.00 \leq p = e^{-\frac{\Delta}{\mathcal{K}T}} \leq 1.00$ is satisfied for all values of *temperature*. $\mathcal{K}$ is introduced as the *normalization factor* which is evaluated adopting the method presented by Walsh *et al.* [WM94] as follows:

A high probability (0.99) of accepting *inferior* solutions is desirable at initial (high) temperatures. The costs of 2000 random allocations, generated from the initial allocation are evaluated and their differences are employed to obtain a measure for the average value of Δ at the initial stages of the annealing process. The value of the normalizing constant $\mathcal{K}$ is then evaluated from the equation

$$\mathcal{K} = \frac{-\Delta}{T * \log(0.99)}$$

by employing the average value of Δ and the initial temperature (T) as 10.0. This value of $\mathcal{K}$ is taken as the upper bound for the cost of an inferior solution to be accepted at the starting temperature with a very high probability (0.99). The value for $\mathcal{K}$ used in the Algorithm 4.2 is 14790700.

To ensure that the process converges in reasonable time, appropriate cooling rate must be chosen to reduce the temperature. A very fast "cooling" would result in searching the solution space sparsely. On the other hand, a very slow 'cooling' would cause the goal of obtaining an approximate optimal solution in reasonable time infeasible. A —em cooling rate of 0.9 is used by Wolf *et al.* [WDIY88] in their work on hybrid data sharing-data partitioning architecture. Using their selection of the annealing parameter as a

guideline, a set of annealing schedules are tried for DOBS allocation with *cooling rates* $0.75, 0.8, 0.85, 0.9 and 0.95$. The value of 0.8 is selected as the "best choice".

One important criterion for the simulated annealing technique to produce "good" solution is that *equilibrium* condition must be attained by the annealing schedule for every *temperature*. The equilibrium condition at any temperature is defined as the state in the annealing sequence that does not allow acceptance of any poorer solution at that *temperature*. However, the implementation of the *equilibrium* condition depends on the characteristics of the specific problem. For DOBS allocation problem, the approach taken by Wolf *et al.* [WDIY88] is adopted and the *equilibrium* condition is defined as the state when sufficient number of solutions in the same neighborhood are considered from the solution space. Since the size of the solution space is proportional to the number of fragments and the number of sites, the *equilibrium condition* is set as a function of the above two parameters. Accordingly, the *equilibrium* condition for the DOBS allocation problem defined for the implementation presented in this thesis is as follows:

An *equilibrium* is reached at any *temperature* in the annealing process when either of the following occurs:

1. 1000 feasible solutions are accepted by the annealing process
2. 5000 feasible solutions are rejected as they do not meet the acceptance criteria.

The values of 1000 and 5000 are chosen based on the values employed for the number of fragments and the number of sites used in the test data set, using the selection strategy adopted by Wolf *et al.* [WDIY88].

The *Freezing condition* determines the condition for termination of the annealing process. Ideally, it must be defined such that any premature convergence is prevented and the effect of annealing is utilized to the fullest. Following the approach taken by Wolf *et al.* [WDIY88] in the design of the simulated annealing algorithm for hybrid data

partitioning-data sharing architecture, the *freezing condition* for the annealing schedule to be applied to DOBS allocation problem is defined as follows:

A *freezing condition* is reached when the *annealing process* fails to accept any new solution for all iteration blocks corresponding to three successive *temperatures* in the annealing schedule.

Selection of the control parameters defining the *Initial temperature*, the *cooling rate*, the *equilibrium condition* and the *freezing condition* for the design of an annealing schedule is inherently dependent on the the characteristics of the problem and therefore, can not be generalized. Nevertheless, they are crucial implementation issues for the success of the method to produce "good" solution. This dissertation is primarily focused on the design of allocation schemes for DOBS. Therefore, it is beyond the scope of this research to find the optimal set of control parameters that produce the *best* solution for DOBS allocation. However, the formulation of the DOBS allocation problem to find a solution using simulated annealing technique leaves scope for future research to find an optimal annealing schedule for the problem.

4.3.3 The Algorithm for Optimal Allocation

The algorithm is formally described in Figure 4.2. It executes in four steps:

Step One - Initialization

The control variable *temperature* is set to its initial high value and the allocation cost of last accepted allocation (*Lcost*) is set to the cost of Initial allocation obtained from Algorithm 3.1 (lines 1-2). The counter for checking the *freezing condition* is also reset to zero (line 3).

Step Two - Acceptance of Solutions

It is assumed that a procedure that generates a random allocation is available to this algorithm. A feasible allocation is generated when this procedure is called and it

is taken as the current allocation (*CAlloc*) (line 7). The cost associated with *CAlloc* is evaluated as *Ccost* and is compared with *Lcost* (lines 8-9). If the cost of the new allocation (*Ccost*) is found to be less compared to *Lcost* the current allocation is accepted (lines 15-16). Otherwise, the difference of costs between *Ccost* and *Lcost* is used to find the value of the *acceptance function.* The value of *acceptance function* is then compared with an externally generated random number to ascertain the probability of accepting the current allocation as a candidate solution (lines 15-16). If the current solution is accepted then the count for the number of successful acceptances is incremented and the cost of last accepted allocation(*Lcost*) is set equal to the cost of current allocation (lines 12 and 17). If the current solution is not acceptable then the count for the number of failed attempts (*NFailure*) is incremented (line 20). This step is repeated until the *equilibrium condition* is attained for that *temperature.*

When *equilibrium* is reached for a particular temperature the *temperature* is reduced by applying the *cooling rate* (line 24). If no solutions are accepted in any of the iterations with the previous *temperature Fail-Temp* is incremented by 1 (line 22). The boolean variable *All-Reject* is used to indicate if no solutions are accepted for a particular *temperature* and is set to *true* or *false* based on the acceptance of a new allocation (lines 14 and 19).

Step Three - Termination of Annealing Process

As *temperature* is gradually reduced the probability of a *poorer* solution to be accepted gradually decreases. The algorithm terminates when no new solution is accepted for three successive blocks of iterations. This is indicated by the value of *Fail-Temp* being equal to 3 (line 4). The last accepted allocation in the annealing schedule is considered as the optimized allocation generated by the algorithm (line 25).

Algorithm 4.2 *Optimization using simulated annealing*

begin algorithm SimAnnOpt

input: **IFRM**, **SAM**:Interfragment Ref. matrix, Site Affinity matrix
$\mathcal{C}$: the cost function
$\mathcal{T}$: initial Temperature
δ: cooling rate
NSuccess: maximum number of *successes* (initialized to zero)
NFailure: maximum number of *failures* (initialized to zero)
IAlloc: Initial Feasible Allocation
output: **OAlloc**: Optimal Allocation
var: *CAlloc*, *LAlloc*: current allocation, last accepted allocation
Ccost, *Lcost*: cost of current allocation, cost of last accepted allocation
Δ: difference of *Ccost* and *Lcost*
Temp: current temperature
Fail-Temp: counts for temp. in which no soln. is accepted
All-reject: flag to indicate no soln. is accepted

```
Temp ← T /* Set Temp to its initial high (T) */                                  (1)
Lcost ← C(IAlloc) /* Initialize Lcost to the cost of Initial Allocation */      (2)
Fail-Temp ← 0                                                                    (3)
repeat until (Fail-Temp=3)                                                       (4)
    All-reject ← True                                                            (5)
    repeat until (NSuccess=1000 or NFailure=5000)                                (6)
        CAlloc ← a randomly chosen allocation                                    (7)
        compute Ccost ← C(CAlloc)                                                (8)
        Δ ← Ccost - Lcost                                                        (9)
        if (Δ ≤ 0) then                                                          (10)
            accept CAlloc as a candidate solution                                (11)
            NSuccess ← NSuccess +1                                               (12)
            Lcost ← Ccost                                                        (13)
            All-Reject ← False                                                   (14)
        else if (rand() < e^(-Δ/(K*Temp))) then                                  (15)
            accept CAlloc as a candidate solution                                (16)
            NSuccess ← NSuccess +1                                               (17)
            Lcost ← Ccost                                                        (18)
            All-Reject ← False                                                   (19)
            else NFailure ← NFailure +1 /* solution rejected */                  (20)
    end repeat
    if (All-Reject) then                                                         (21)
        Fail-Temp ← Fail-Temp +1                                                 (22)
    else Fail-Temp ← 0                                                           (23)
    Temp ← T * δ                                                                (24)
end repeat
OAlloc ← The last accepted feasible allocation                                   (25)
end algorithm SimAnnOpt
```

Figure 4.2: Simulated Annealing Algorithm for Optimal Allocation

4.4 Implementation Issues

This section presents a brief discussion on the issues involved in the implementation of the Algorithm 4.2.

One of the input to the algorithm is an initial allocation which could be treated as the starting solution to the annealing process. Solution generated by Algorithm 3.1 described in Chapter 3 is employed for this purpose. Further, a time efficient procedure is required to generate new feasible allocations (preferably different allocation at each invocation) from the starting solution. The new feasible allocations are obtained from the previous allocation by either moving some fragments from one site to another or by exchanging fragments between sites subject to site storage constraints. A procedure is employed that generates "random moves" to obtain a new feasible allocation. A "random move" either *moves* a randomly chosen fragment from its current site to another randomly selected site or *interchanges* it with a fragment at another site subject to the storage constraints at the respective sites. *Moving* a fragment is tried first and *interchange* of fragment pair between two sites is done only when no fragment move to any site is feasible due to the storage constraints at every site. To keep the implementation simple, random moves are restricted to only 'single fragment move's and 'single pair interchange's. The implementation could be generalized to perform *moves* and interchanges involving multiple fragments at a time.

Instead of rearranging the fragments between the sites physically, a log of the changes in fragment distribution is created for every feasible allocation generated by *random moves*. This makes the "undo" operation, required to get back the last accepted allocation when a solution is rejected, simple and minimizes overhead. Further, the corresponding change in the *cost function* is evaluated from the log instead of computing the cost of allocation due to every fragment.

A pseudo random number generator is employed to produce the sequence of random numbers in the real domain $(0.0, 1.0)$. These are used to compare with the value of the

acceptance function for solution acceptance at all temperature.

Chapter 5

Analysis of Allocation Algorithms

This chapter summarizes the results obtained by applying the allocation algorithms described in Chapter 3 and 4 and presents a comparative analysis of the performances of the algorithms based on the results. Section 5.1 describes the mechanism and assumptions used to generate the set of test data for a DOBS prototype. This data set is employed to verify the efficiency and performance of the proposed allocation schemes. Section 5.2 describes the validation schemes for the test data set to ensure its integrity and consistency with the conditions and constraints of the proposed DOBS model. Section 5.3 describes the implementation environment and the test plan for the allocation algorithms and Section 5.4 presents the test results and analyses the performance of the optimization algorithms. The important insights obtained from the analysis of the results that can be usefully applied for a DOBS allocation are summarized in Section 5.5. This chapter concludes with a brief discussion on the complexity and runtime of the allocation algorithms.

5.1 Generation of Test Data

This section describes the mechanism used in this dissertation for generating test data sets employed for analyzing the allocation algorithms to verify their efficiency and correctness. Ideally, the values of the input parameters used in the algorithms presented in Chapter 3 and 4 are obtained either directly from system statistics (eg. network topology, local storage capacity at every site, number of applications accessing the objectbase *etc.*), or they are derived from static analysis based on compile time information about the objectbase. However, in absence of real data, the set of input parameters required by the algorithms are generated based on the following assumptions :

1. **Fragment size:** Fragment sizes of all class fragments are assumed to be normally distributed. The rationale is that fragments of very large or very small size are few and there are approximately equal number of fragments of size smaller and larger than the 'mean fragment size'. Therefore, a normal distribution fits as a 'close guess'. Fragment sizes are therefore generated as a normal variate with mean of 5000 bytes and standard deviation of 500.

2. **Local storage:** Available storage at every site is assumed to be equal. Though this may not represent the distribution of local storages at the sites of a real life DOBS, this assumption makes the analysis tractable. Total available storage at all sites is ensured to meet the feasibility criterion for DOBS allocation.

3. **Interfragment Reference Count :** In a typical distributed objectbase environment it is expected that there will be a small number of fragments which will be referenced extensively while a few will be rarely accessed. Therefore, the distribution of interfragment reference counts is modeled as an exponential distribution with a reasonably high mean to obtain the desired "skewness" in the distribution. The mean of the distribution is taken to be 20.

4. **Application Site Affinity:** Generally speaking, applications in a distributed environment tend to be specific with respect to their "usual" location of execution. In other words, any application is most frequently executed from a specific site or group of sites and quite rarely from other sites of the network. Hence, site affinities of the applications are modeled as exponential variates. The mean of the distribution is taken as 4. It is assumed that there are 10 applications accessing the DOBS.

5. **Application Fragment Affinity:** In a typical objectbase system, each application exhibits "locality of reference" in accessing class fragments. Therefore, for each application, there is a set of class fragments which is referenced quite frequently compared to other fragments which are accessed rarely. An exponential distribution is considered to be the "best fit" in simulating the access behavior of the applications on the class fragments. Mean of the distribution is assumed to be 6.

6. **Network topology:** The allocation algorithms presented are generic with respect to network topology so they are applicable to a DOBS with any arbitrary network topology. The network topology used in this dissertation to analyze the algorithms is generated as a connected random graph with a given number of vertices[1]. A connection probability of 0.4 is assumed for generating the random graph.

For every input parameter, the set of feasible values employed to test the algorithms, are generated as stochastic random variable obeying certain distribution functions, based on the above assumptions. Standard library functions provided in *smpl* [Mac89] are used.

The granularity of the proposed DOBS allocation model considers each attribute and method of a class individually to estimate the reference costs. However, for the sake of testing the efficiency and correctness of the allocation algorithms, the collective effect of all the attributes and/or methods of a class fragment is considered. Therefore,

[1] Each vertex in the graph corresponds to a network site and each edge a network link.

instead of generating test data for each attribute or method in a fragment, only a single value is used as the reference count between any pair of fragments without any loss of generality. This makes the implementation and testing of the algorithms simple. It is to be noted that no pertinent information about the elements (attributes and/or methods) constituting the class fragment is lost because any reference to or from a class fragment is the sum of all individual references to or from the attributes and methods contained by the fragment.

5.2 Validation of Test Data

Before the allocation algorithms are applied to the set of input data, it is necessary to validate the test data against a set of feasibility constraints to ensure that the algorithms produce feasible allocation in DOBS. The conditions that must hold for the input data to be valid input to the proposed allocation schemes are :

- Every fragment size is a nonzero positive integer denoting the number of bytes the fragment contains.
- Storage capacity at every site is a nonnegative integer denoting the number of bytes it can hold.
- The total available storage at all sites must be large enough to contain all the class fragments without splitting (considering unused storage due to storage fragmentation across the network).
- The storage of at least one site must be able to contain the largest class fragment.
- Each element in the Interfragment Reference Matrix (IFR) is a nonnegative integer.
- Elements of the matrices capturing information about network topology, application site affinity, and application fragment affinity contain feasible (nonnegative integers) values.

5.3 Implementation of The Allocation Algorithms

The problem of DOBS allocation involves a large number of parameters, each having its own influence in the final solution obtained by applying either of the two approaches. This makes the analysis of the allocation algorithms a difficult and nontrivial task. Three parameters are selected as design variables while other design parameters are kept unchanged. The design variables selected for the analysis of the algorithms are (1) number of sites, (2) number of class fragments, and (3) percentage of available storage in excess of the size of the global objectbase. The last of the three parameters does not appear explicitly in the DOBS allocation model described in Section 1.3 but plays an important role in the performance of the allocation algorithms. Allocation schemes based on both the approaches are applied on the test data sets obtained by assigning different values to these design variables. Since both the allocation algorithms are designed to improve the initial allocation by moving or exchanging fragments between sites, the success of each of these move or exchange would depend on the amount of "free" storage space available at every site. Generally speaking, it is expected that the more free space, the higher is the percentage of the successful exchanges. Therefore, the capabilities of the allocation algorithms to improve the cost of allocation depends on the successful fragment moves or exchanges made which in turn depends on the amount of free space available at different network sites.

The percentage improvement in cost achieved by the optimized allocation over the cost of the initial allocation is taken as the yardstick to measure the relative merits of the two algorithms. The allocation algorithms are implemented in C and C++ and are tested on the sets of test data on SUN SPARCstations running SUNOS version 4.1.3.

The algorithm based on the graphical approach adopted by Kernighan and Lin [KL70] is implemented with the required modifications that are necessary to adapt the two-way graph partitioning algorithm for the DOBS allocation problem as described in Section 3.4.1.

Both the algorithms are run on the sets of test data that are generated by varying one of the three design variables at a time while the other parameters held constant. For a given number of fragments and a given amount of excess global storage space, the number of sites is varied to observe the behavior of the allocation algorithms. A pair of graphs, one from the results produced by each of the allocation algorithms, is thus obtained by plotting the reduction percentage in the allocation cost from that of the initial allocation against the number of sites in the network. A set of such pair of graphs are obtained by repeating the above observations for different number of class fragments constituting the objectbase and for different amount of excess available storage. The results obtained by the allocation algorithms for different sets of test data are presented in Tables 5.1 to 5.4. The corresponding graphs showing the comparative performance of the algorithms are plotted in Figures 5.1 to 5.16.

No. of sites	*100 Frags*		*200 Frags*		*300 Frgas*		*400 Frags*	
	KL	*S.A.*	*KL*	*S.A.*	*KL*	*S.A.*	*KL*	*S.A.*
2	13.03	7.98	11.07	7.58	8.25	5.79	6.90	4.31
3	9.77	8.26	8.65	6.75	7.87	6.53	6.25	5.06
4	0.74	9.28	4.64	14.30	4.64	8.30	3.51	6.91
6	2.36	13.89	3.31	13.54	2.38	14.84	2.88	13.70
8	0.00	12.97	0.29	10.50	0.06	12.83	0.00	10.31
10	3.38	11.68	1.96	11.61	3.20	13.61	2.28	12.13
15	1.57	7.38	0.96	7.82	1.27	8.62	1.10	7.52
20	2.38	6.42	1.15	5.98	1.09	5.52	0.67	5.13

Table 5.1: Percentage improvement by optimization with 15% excess storage

No. of sites	*100 Frags*		*200 Frags*		*300 Frgas*		*400 Frags*	
	KL	*S.A.*	*KL*	*S.A.*	*KL*	*S.A.*	*KL*	*S.A.*
2	13.11	7.28	11.80	7.99	8.72	6.24	7.04	4.85
3	10.24	8.36	7.80	6.99	8.22	8.55	6.08	6.71
4	4.15	10.38	5.51	17.02	5.99	8.20	4.20	6.86
6	4.19	15.20	3.44	15.03	1.68	16.45	1.97	15.99
8	0.00	15.08	3.79	14.96	0.08	12.02	11.09	12.18
10	3.20	14.49	3.56	14.42	1.50	16.43	1.98	14.77
15	0.91	7.04	0.88	7.88	1.02	9.22	1.21	8.62
20	1.58	6.26	1.07	6.01	0.53	5.44	0.70	5.44

Table 5.2: Percentage improvement by optimization with 20% excess storage

No. of sites	*100 Frags*		*200 Frags*		*300 Frgas*		*400 Frags*	
	KL	*S.A.*	*KL*	*S.A.*	*KL*	*S.A.*	*KL*	*S.A.*
2	11.34	7.39	11.47	8.36	8.30	5.976	7.06	5.83
3	11.40	9.39	8.10	7.80	7.12	10.15	5.85	8.57
4	6.71	8.51	5.00	21.36	5.06	8.12	4.24	6.56
6	3.55	16.30	4.24	14.65	4.21	17.01	2.32	15.54
8	0.00	14.23	5.98	21.79	0.00	13.08	0.00	7.90
10	2.12	15.86	2.99	16.49	1.07	17.61	0.83	16.68
15	1.79	8.33	0.00	9.02	1.11	10.16	0.00	10.06
20	1.12	6.84	1.32	6.34	0.50	5.97	1.04	6.00

Table 5.3: Percentage cost improvements by optimization with 25% excess storage

No. of sites	100 Frags		200 Frags		300 Frgas		400 Frags	
	KL	S.A.	KL	S.A.	KL	S.A.	KL	S.A.
2	10.69	5.60	10.74	8.42	8.14	8.05	8.17	6.21
3	7.45	10.01	5.61	9.10	7.96	18.27	5.85	16.39
4	9.26	6.48	4.78	26.19	6.68	4.16	5.92	3.55
6	4.38	17.56	4.39	13.92	3.43	18.91	1.69	15.34
8	1.89	14.13	2.13	28.84	2.98	16.22	2.01	18.37
10	0.00	16.19	2.18	16.74	1.81	17.09	1.52	16.26
15	2.17	10.64	1.70	11.09	1.65	11.13	2.17	11.70
20	1.81	7.13	1.35	7.55	1.76	6.73	0.70	7.40

Table 5.4: Percentage improvement by optimization with 40% excess storage

5.4 Analysis of Results

Since simulated annealing is a nondeterministic process, the result produced by the allocation algorithm based on simulated annealing will be different for different executions of the same set of input parameters. Therefore, the average of the results obtained from 100 executions with the same set of parameters is considered in the comparative analysis.

For a given number of class fragments and a given amount of storage in excess of the size of the objectbase, the number of sites is varied and the performance of the allocation algorithms are plotted as the percentage improvement in cost over the cost of the initial starting solution. For example in Figure 5.1 in page 90, the allocation algorithms are run with 100 fragments and 15% excess storage, while the number of network sites is set to 2, 3, 4, 6, 8, 10, 15 and 20. The observation is repeated with 200, 300 and 400 fragments and the results are plotted in Figures 5.2, 5.3, and 5.4 respectively. The entire set of observation is repeated for 20%, 25% and 40% excess storages and the graphs presented in Figures 5.5 to 5.16 are similarly obtained.

Despite the irregular nature of the graphs, the following common characteristics are exhibited by all of them:

1. The algorithm based on the graphical approach produces best solution for a network of two sites. The performance of the algorithm then sharply falls with the increase in the number of network sites. For example, in Table 5.1 in page 85, one can find there is 13.03% improvement achieved by the graphical approach for a network of 2 sites and only 2.38% when there are 20 sites in the network. The same observation can also be made from the graph in Figure 5.1. This is expected because the graphical approach is based on the two-way graph partitioning algorithm proposed by Kernighan and Lin [KL70] that attempts to arrive at a solution that is as close as possible to pairwise optimal. Therefore, the algorithm is best suited for finding a near optimal distribution of fragments between two network sites. When the two-way partitioning technique is applied iteratively on all site pairs, reaching a

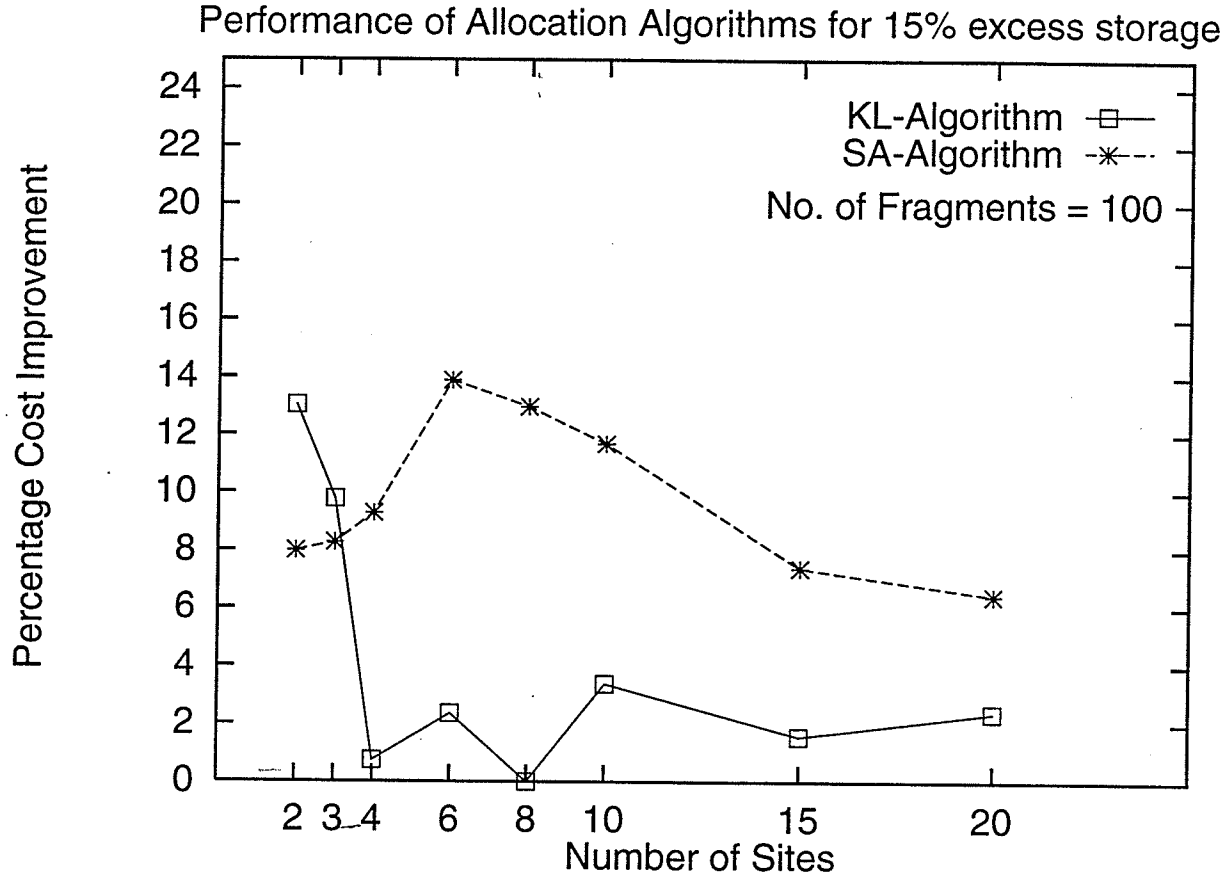

Figure 5.1: Comparative Performance of the Allocation Algorithms for 15% excess storage and 100 fragments

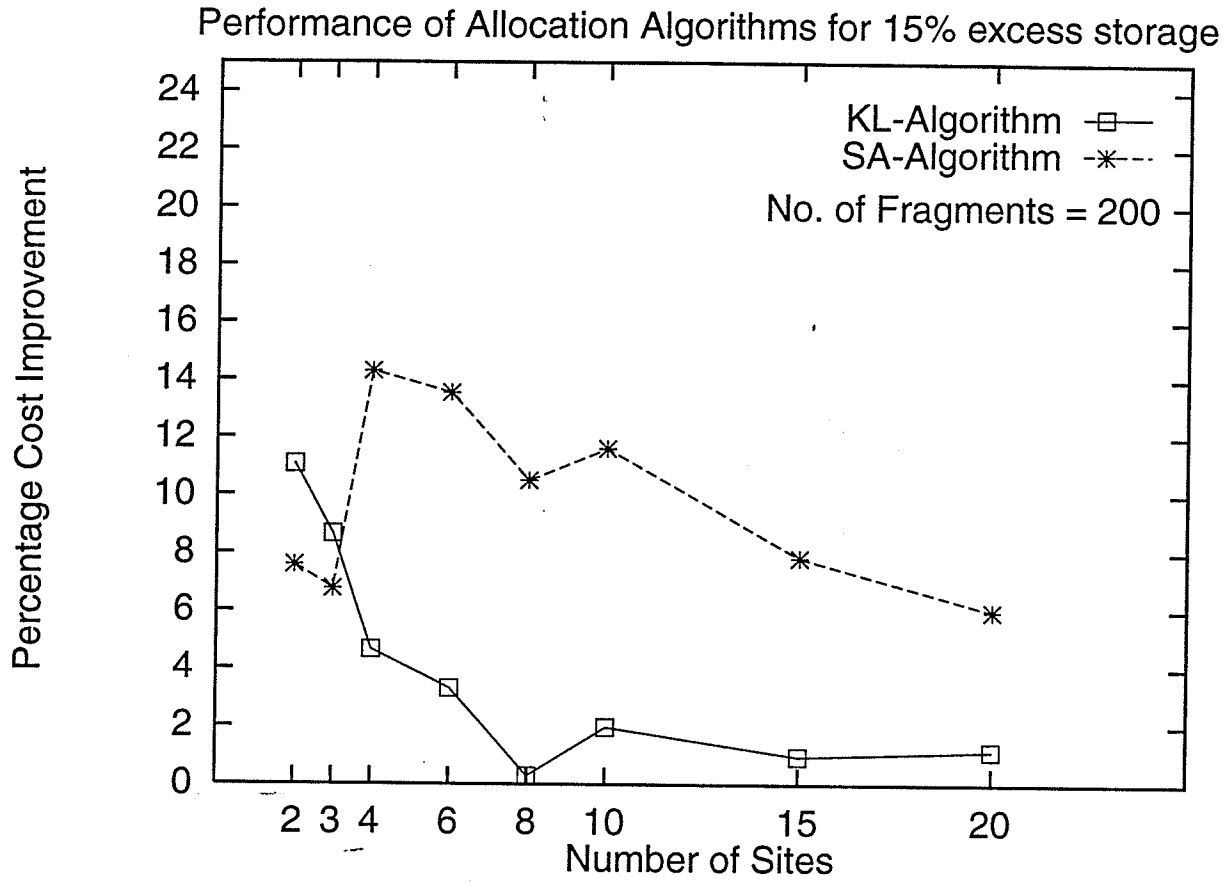

Figure 5.2: Comparative Performance of the Allocation Algorithms for 15% excess storage and 200 fragments

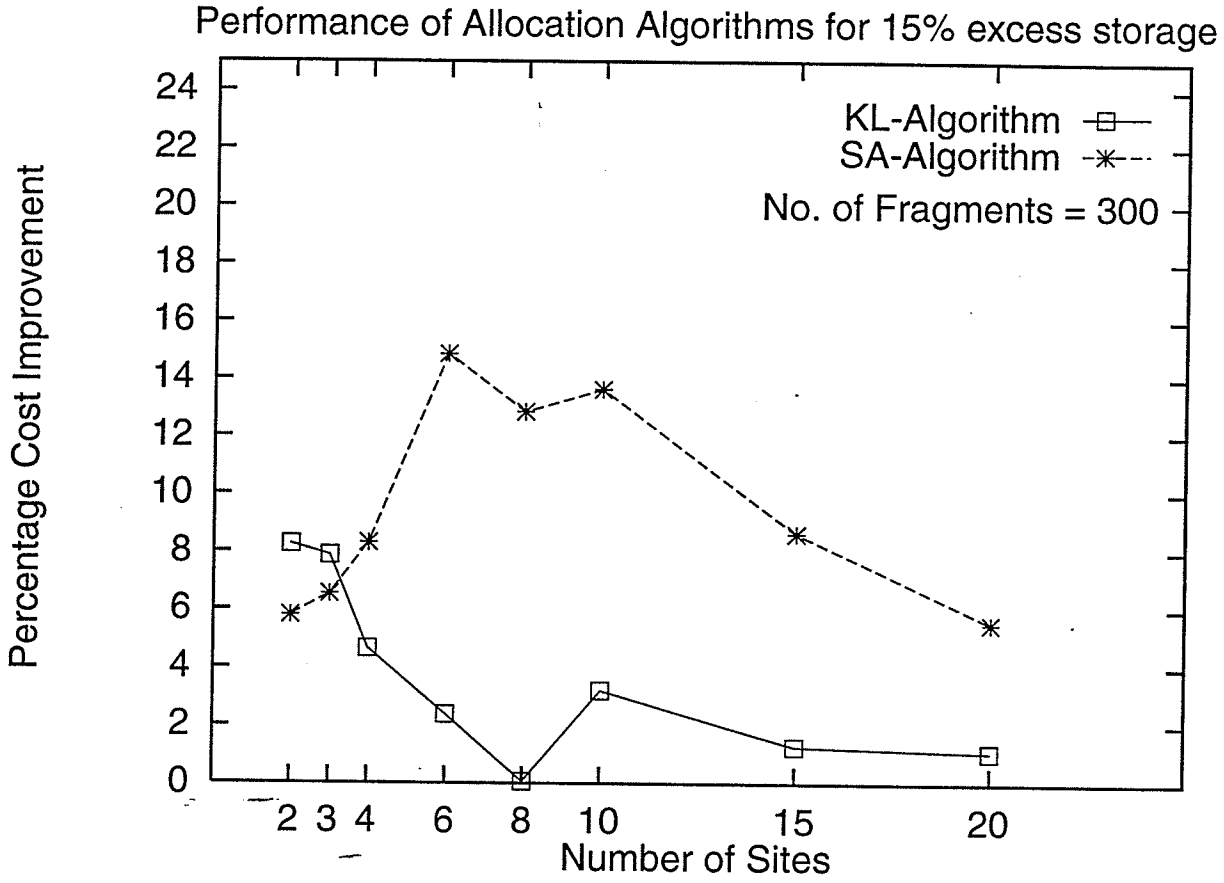

Figure 5.3: Comparative Performance of the Allocation Algorithms for 15% excess storage and 300 fragments

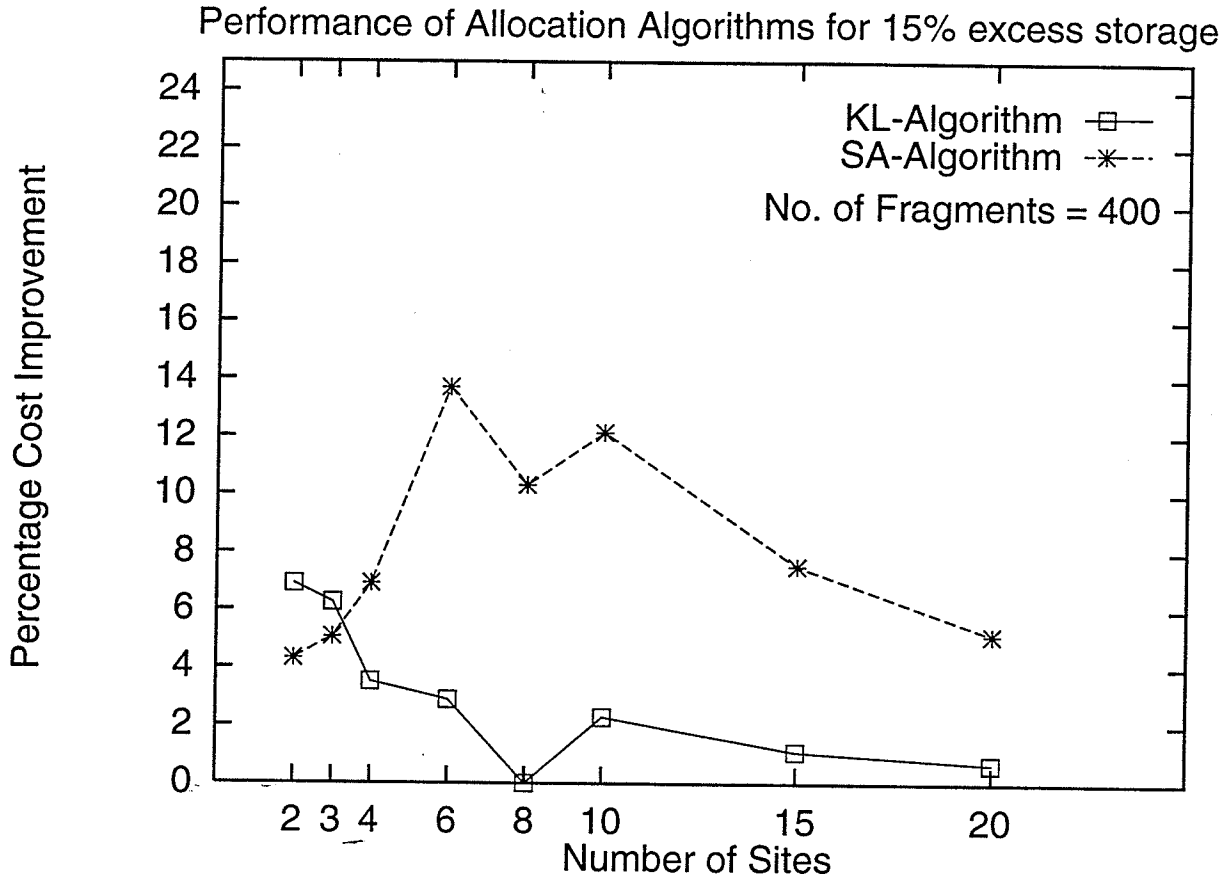

Figure 5.4: Comparative Performance of the Allocation Algorithms for 15% excess storage and 400 fragments

near optimal solution becomes more unlikely due to the mutual dependencies of pairwise optimality between the site pairs. Further, pairwise optimality does not guarantee global optimality. Also, since the distances between the network sites are not necessarily uniform, the local gain obtained in the process of arriving at pairwise optimal allocation between different site pairs does not contribute the same weight to the global gain. Therefore, as the number of network sites increases, it is less likely for a pairwise optimal solution to become a globally optimal solution. Further, as the number of network sites increases, it becomes increasingly difficult to obtain pairwise optimality since fragments at the common site of multiple site pairs would cause mutual interference in achieving pairwise optimality.

2. On the contrary, since simulated annealing is based on stochastic principles, the allocation algorithm based on simulated annealing is expected to perform better when the solution space is reasonably large. Therefore, it is not surprising that the simulated annealing algorithm outperforms its graphical counterpart by a significant margin when the number of sites is greater than 4. Table 5.1 shows that the improvements obtained by the simulated annealing method are between 6.42% (for 20 sites) and 13.89% (for 6 sites) when 100 fragments are distributed in a DOBS with 15% excess storage. In comparison, the allocation algorithm based on graphical approach, though gives higher improvements (13.03% for 2 sites and 9.77% for 3 sites) but manages to get only a marginal($0 - 3.38\%$) improvement for more than 4 sites. The efficiency of the simulated annealing approach grows rapidly to about $15 - 16\%$ for number of sites between 6 and 15. However, as the number of sites becomes greater than 15 there are too many nonlocal interfragment references to optimize the cost of allocation to a reasonably small value. Therefore, the performance of the simulated annealing process gradually decreases when the number of network sites is greater than 15. This observation can be verified from the results reported in Tables 5.1 - 5.4. Similar results are reported in [WDIY88] where simulated annealing technique is applied to the load balancing problem in a

partitioned hybrid distributed database system. However, for the load balancing problem, the simulated annealing technique does not produce any feasible solution when the number of network sites grows larger than four because of excessive number of remote procedure calls.

3. Both the algorithms exhibit some apparent inconsistent behavior for certain values of the data set. For example, in Figure 5.1 in page 90, the performance of the graphical algorithm dropped abruptly from 13.03% (for 2 sites) to 0.74% (for 4 sites) and again to 0% (for 8 sites) with 100 fragments in the objectbase (Table 5.1). Similar irregularities are also found in the performance of the simulated annealing algorithm. For example, in Figures 5.2, 5.3, and 5.4 in pages 91-93, a sudden drop in the performance of the simulated annealing algorithm occurs for a network of 8 sites. A closer look at the solution procedure provides the following insights to these anomalies:

   Since the initial allocation distributes the class fragments based on a "best fit" strategy[2], the fragments are not uniformly distributed among the sites. While some sites are "tightly" packed with fragments, leaving very little free storage space, other sites are sparsely populated and contain large amounts of available storage. Since both the allocation algorithms attempt to improve the initial allocation by moving and/or exchanging fragments between the sites, the efficiency of either of these algorithms would depend on how many of the attempted *moves* and/or *exchanges* between the site pairs are successfully performed by the algorithms. Further, the success of such *moves* and *exchanges* depends on the amount of free storage space available at the sites and the relative sizes of the fragments involved in the move and/or exchange. It is found that for certain specific instances of the input data, the distribution of available storage space at the network sites after initial allocation is so "skewed" that a large number of move and/or swap attempts of the fragments

[2]The best-fit strategy is chosen to minimize the waste of global storage space due to storage fragmentation across the sites.

between the sites are not successful due to the limited storage space at either or both the sites involved in the process. As a result, the performance of either or both the algorithms drops rather abruptly, as found in the graphs in Figures 5.1 to 5.16. For example, the amount of available storage space at the 8 sites with 100 fragments and 15% excess storage (Figure 5.1) after the initial allocation is found to be 4, 1648, 0, 1817, 16, 2246, 37511 and 37623 bytes. Since we have considered fragment sizes of a mean 5000 bytes and of standard deviation of 500 bytes, and they are normally distributed, the most likely (95% confidence interval) range for fragment sizes is between 4000 and 6000 bytes. This leaves only two of the eight sites with the potential to participate in any fragment move from any other site. Also, sites with available free space of 0, 4 and 16 bytes will only allow exchanges of fragments of very small size differences, which does not occur in our test data very often as the distribution of fragment sizes has a standard deviation of 500 bytes.

Similar situations occur whenever the distribution of available free storage distribution at the sites is heavily skewed as observed in Figures 5.1- 5.16. Strangely enough the occurrence of such poor performances by the allocation algorithms are found to appear with some periodicity. A close observation of the Tables 5.1- 5.4 and the Figures 5.1- 5.16 reveals that, for 8 sites with 100 or 300 fragments and for 8 or15 sites with 200 or 400 fragments the allocation algorithms produce poor solutions for any amount of excess storage. Following explanations help to understand these results: The test data generation process determines available storage at every site by equally dividing the size of the global objectbase after adding the excess storage required for a feasible allocation to it. Therefore, the amount of available storage at each site would be proportional to the ratio of the number of fragments and the number of network sites. Further, the starting solution generated by the initial allocation algorithm would depend on the amount of available storage at each site. Therefore, it is not unreasonable to expect that for the sets of test data with number of fragments and number of sites in the same (or nearly same) proportion, the

"best-fit" allocation strategy used by the initial allocation algorithm would result in an allocation that leaves nearly the same kind of distribution of free storage at the network sites. Hence, the distribution of free storage space at the sites after initial allocation for all of the above mentioned cases would be similar to that obtained for 8 sites and 100 fragments with 15% excess storage.

However, note that the distribution of available storage space at individual sites after initial allocation would affect the graphical algorithm and the simulated annealing algorithm differently. Since the algorithm based on the graphical approach attempts to improve upon the initial allocation by considering one pair at a time, the different fragment moves/swaps tried by the algorithm is limited compared to the number of fragment move and swaps tried by the simulated annealing algorithm which always aims at global optimality. Therefore, when the initial allocation leaves the network with a "skewed" distribution of local available storages, the simulated annealing can still manage to make more fragment moves and swaps compared to its graphical counterpart. This explains why the inconsistent results obtained from the simulated annealing algorithm are better than the corresponding inconsistent results obtained by the graphical algorithm. However, there is one instance (Figure 5.8, no. of sites=8 in page 101) in our set of test results, where the observed result is extremely uncharacteristic. Validating this claim would require a detailed trace which is very difficult, if not impossible, because the random sequence can only be regenerated if the seed could be reproduced consistently.

Finally, due to complex relationships between the parameters themselves, it is not trivial to predict in advance if a particular set of input parameter values would lead to an inconsistent result. Hence, such results cannot be avoided so we need to draw conclusions based on anticipated results because the process is inherently random. Further, the test data values are generated by drawing random samples from standard statistical distributions (eg. fragment sizes are generated as a normal variate, interfragment references are generated as an exponential variate, *etc.*). Therefore,

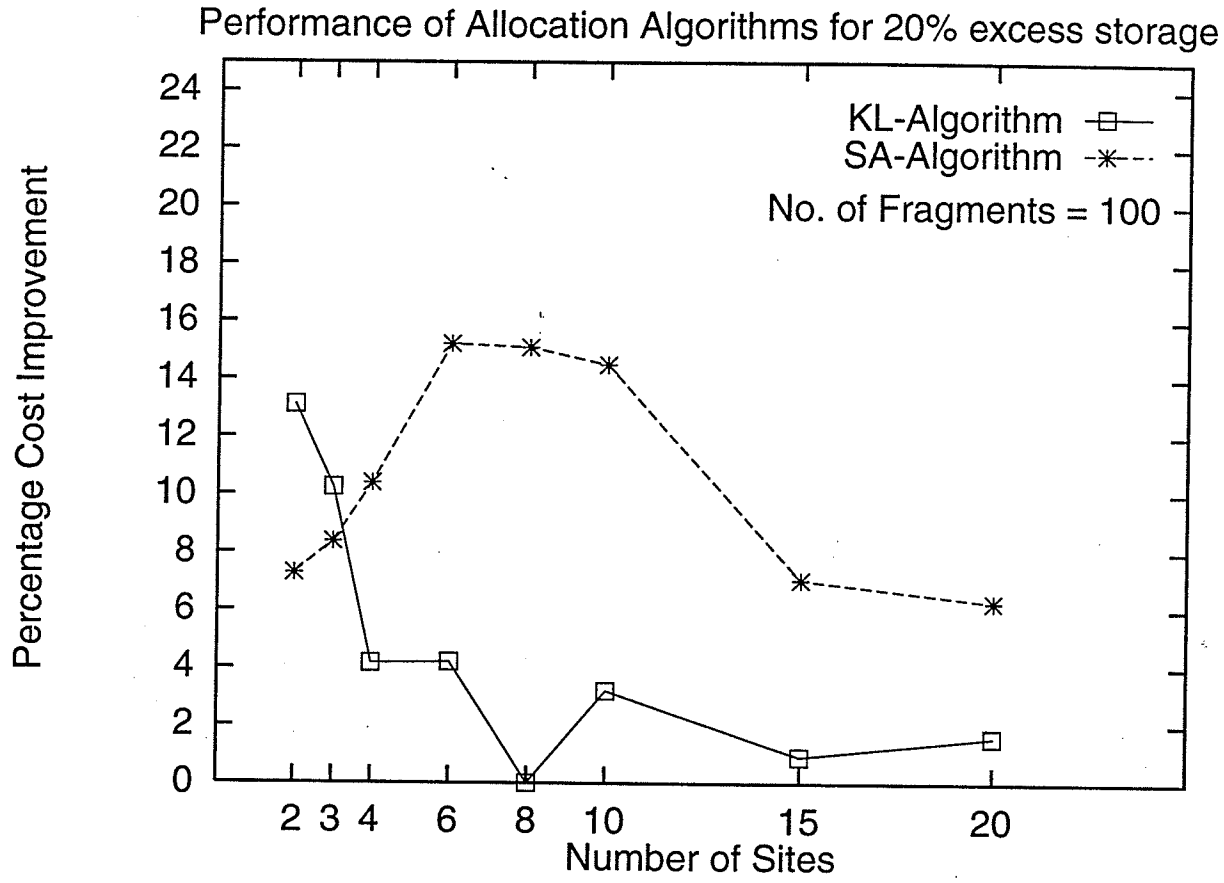

Figure 5.5: Comparative Performance of the Allocation Algorithms for 20% excess storage and 100 fragments

the set of input data used for analysis would not be identical in every execution of the algorithms. This uncontrolled variation of the input data makes the analysis of the observed results more difficult.

5.5 Insights for The Design of DOBS Allocation

This section summarizes the important observations from the test results and the expectations derived from them which can act as a guideline for the design of DOBS allocation.

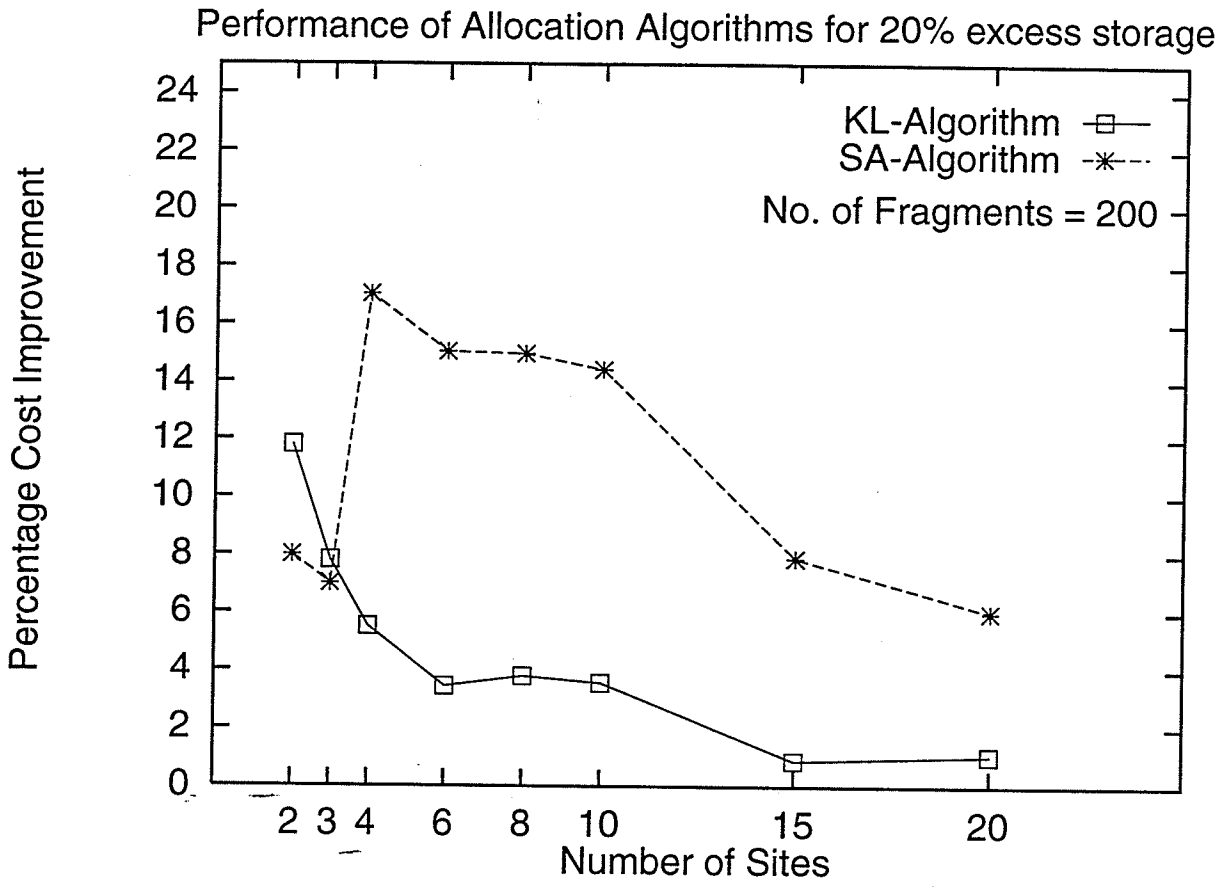

Figure 5.6: Comparative Performance of the Allocation Algorithms for 20% excess storage and 200 fragments

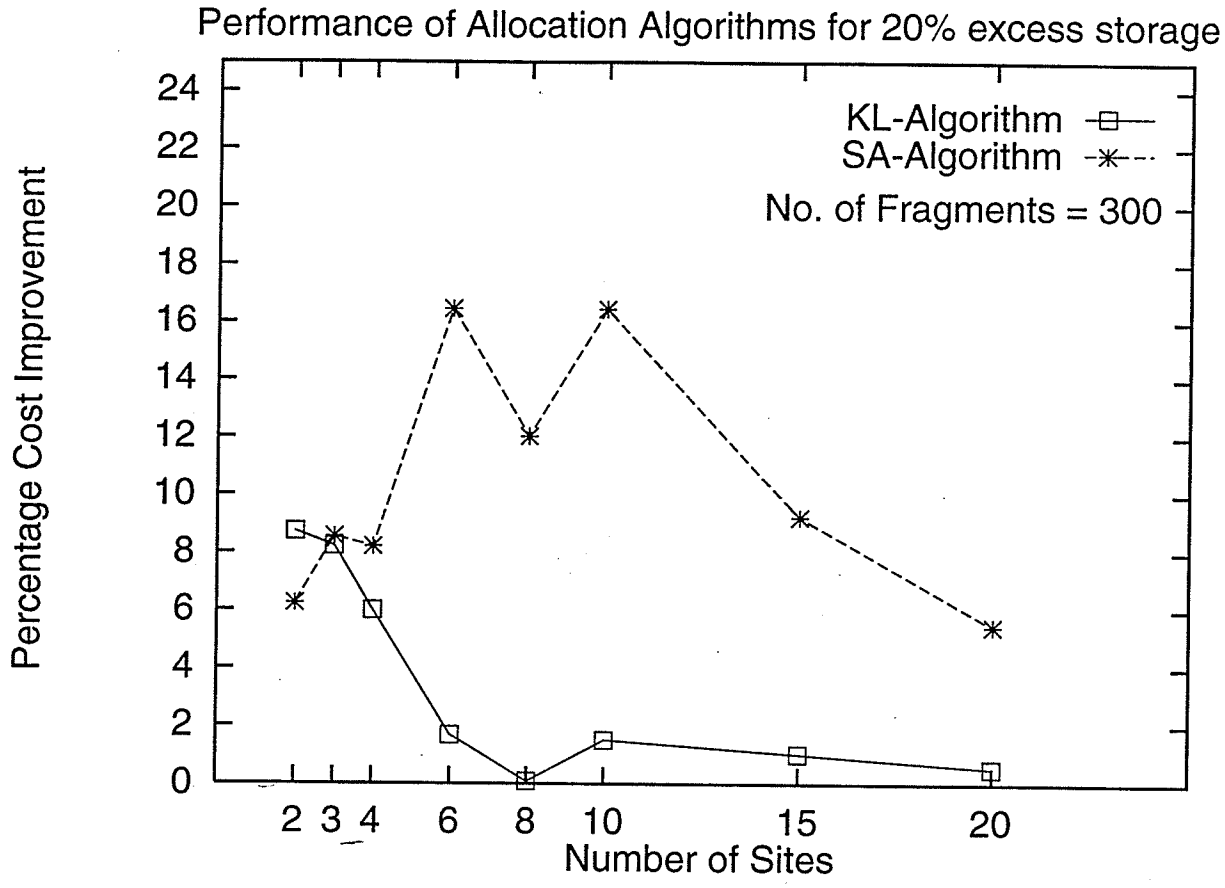

Figure 5.7: Comparative Performance of the Allocation Algorithms for 20% excess storage and 300 fragments

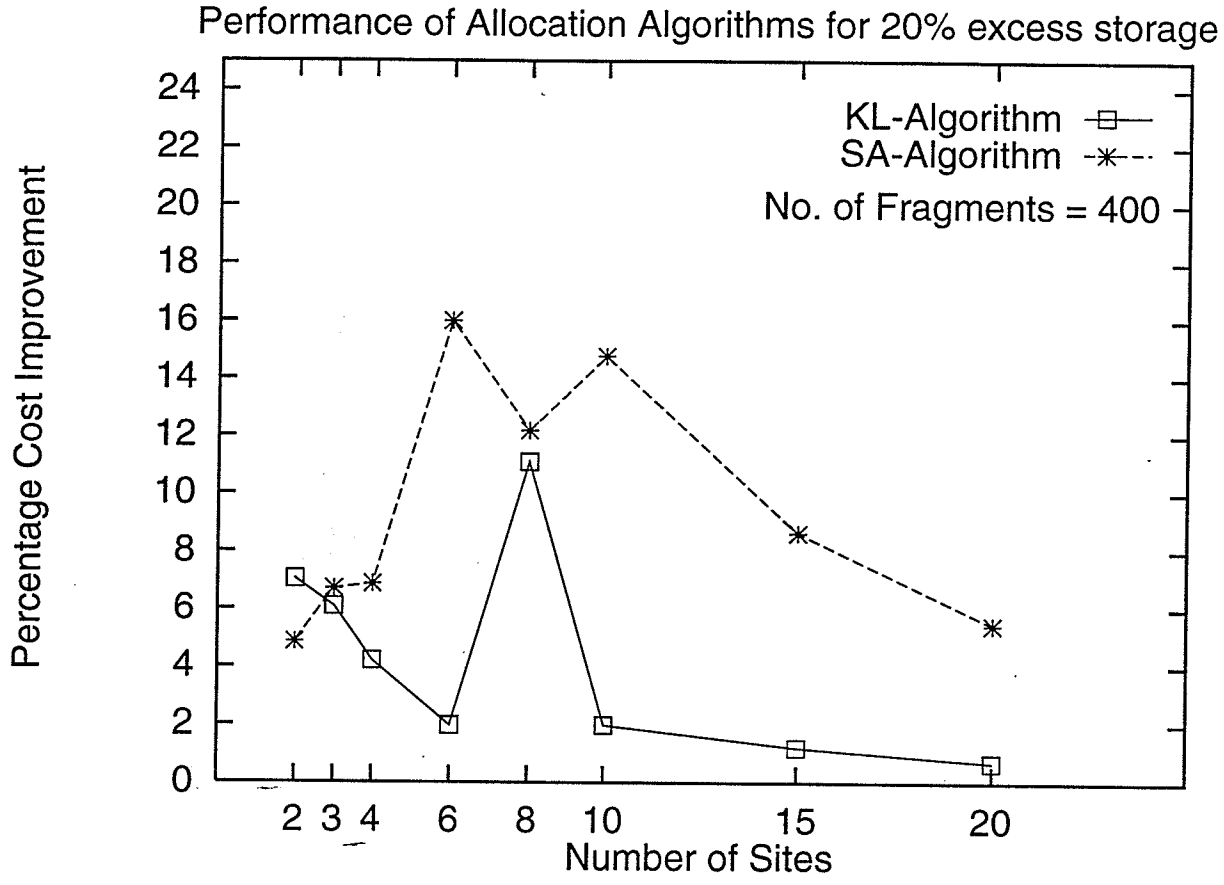

Figure 5.8: Comparative Performance of the Allocation Algorithms for 20% excess storage and 400 fragments

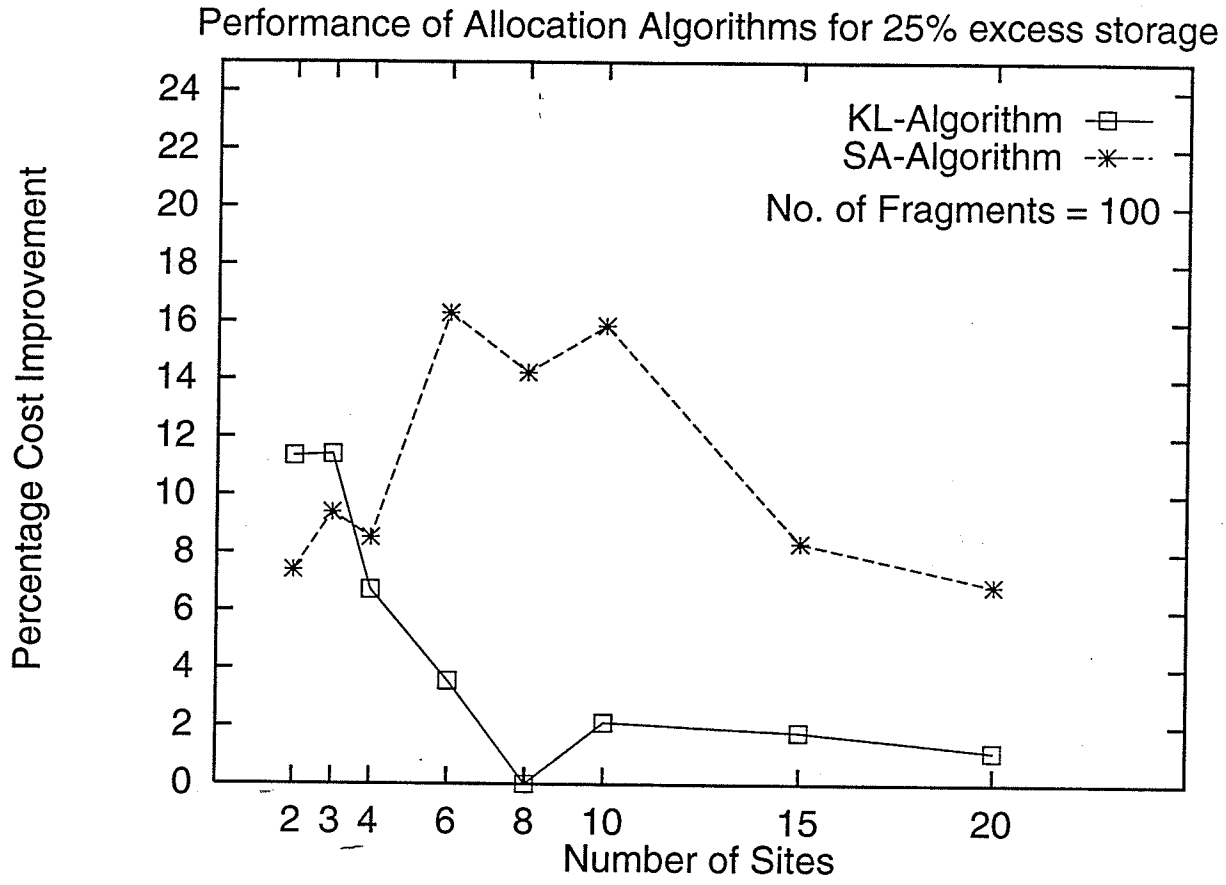

Figure 5.9: Comparative Performance of the Allocation Algorithms for 25% excess storage and 100 fragments

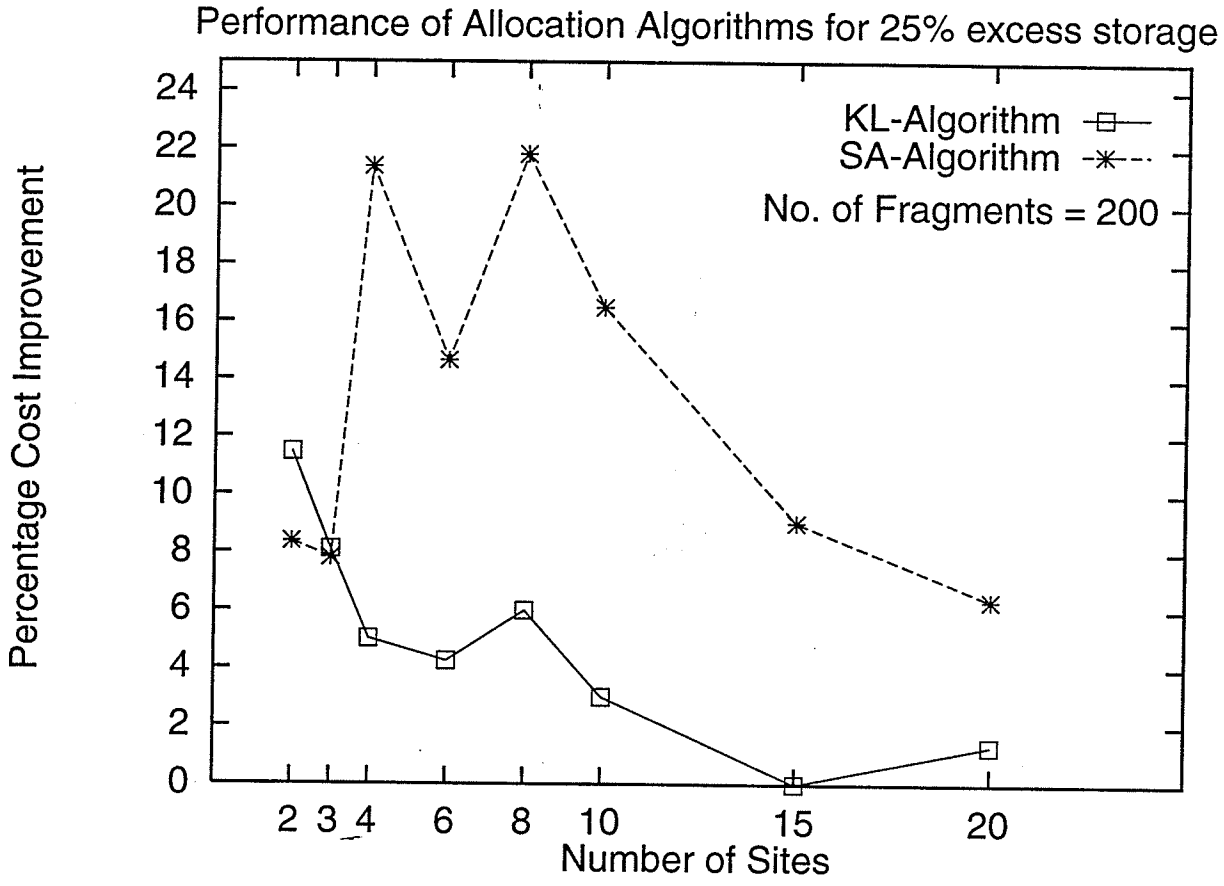

Figure 5.10: Comparative Performance of the Allocation Algorithms for 25% excess storage and 200 fragments

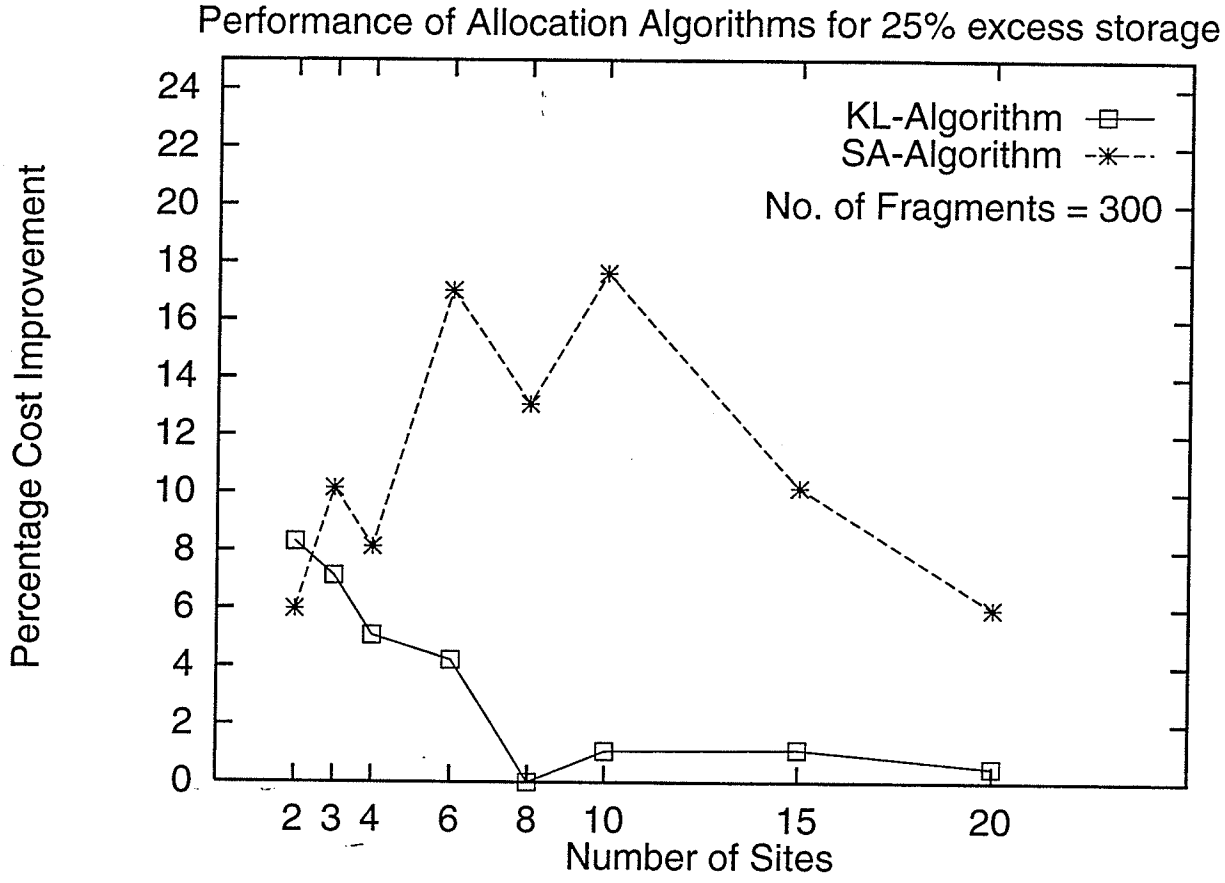

Figure 5.11: Comparative Performance of the Allocation Algorithms for 25% excess storage and 300 fragments

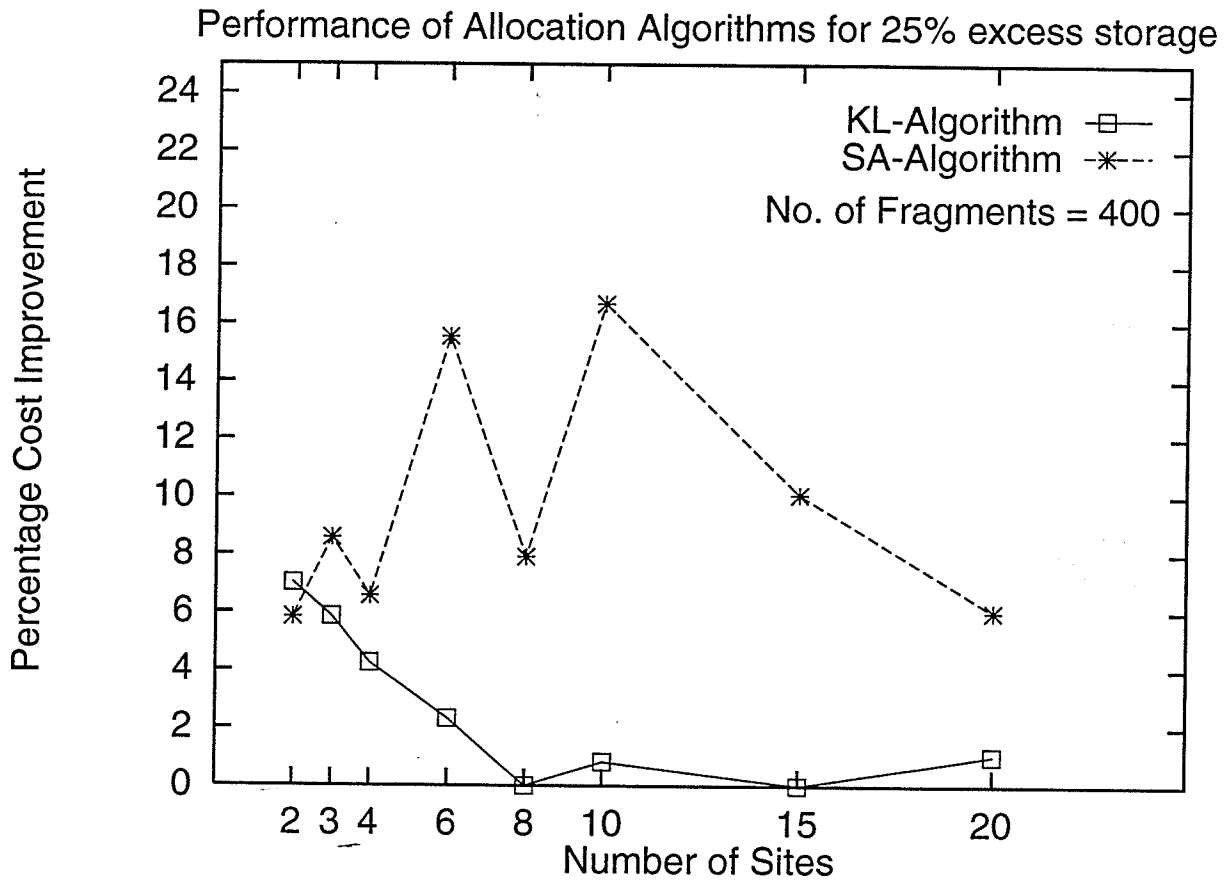

Figure 5.12: Comparative Performance of the Allocation Algorithms for 25% excess storage and 400 fragments

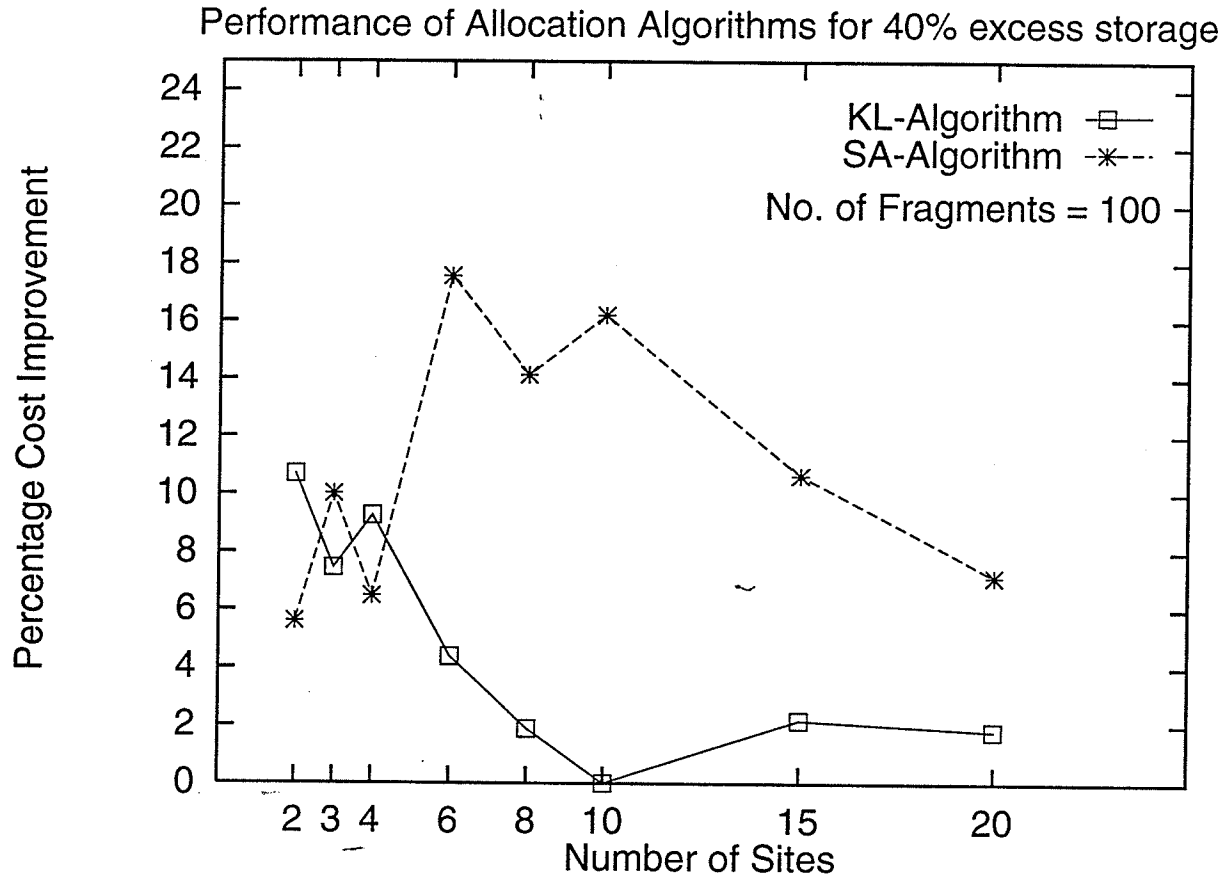

Figure 5.13: Comparative Performance of the Allocation Algorithms for 40% excess storage and 100 fragments

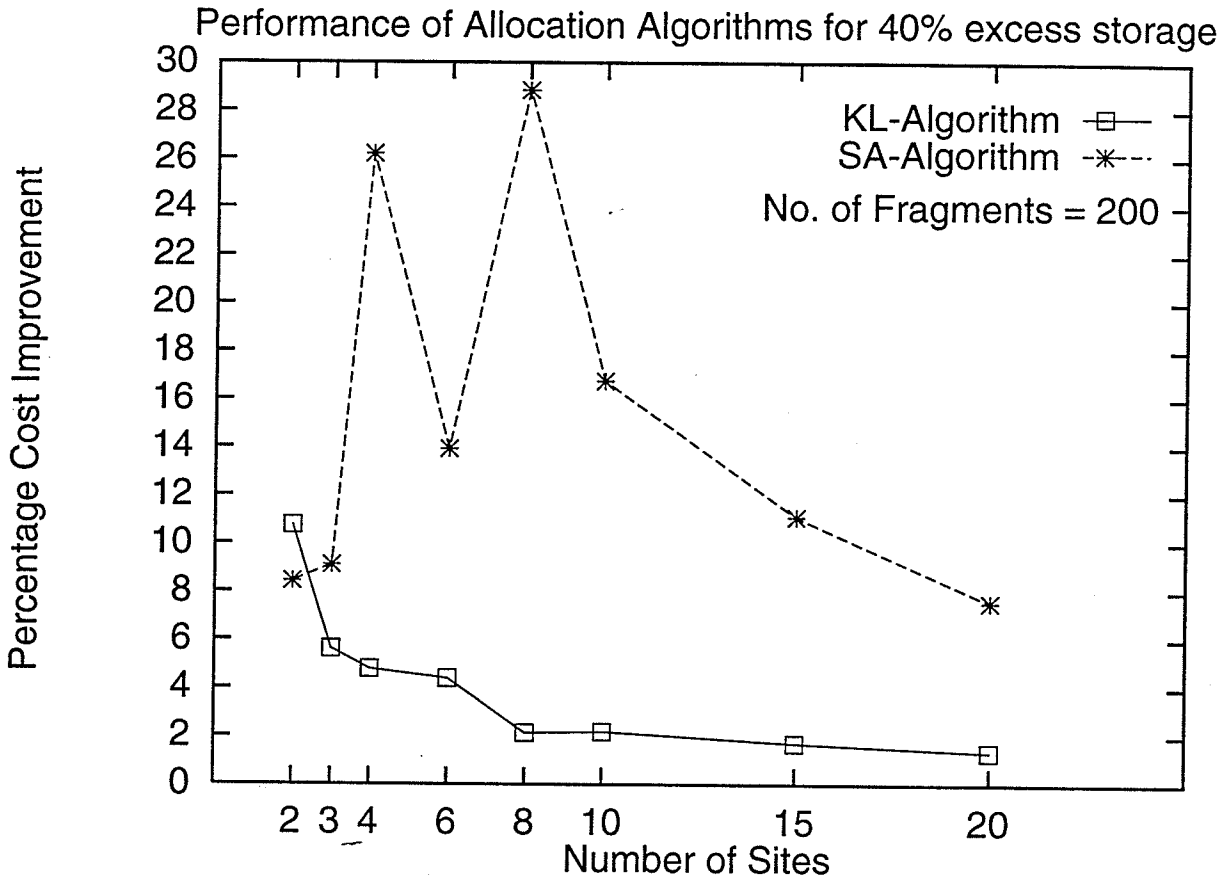

Figure 5.14: Comparative Performance of the Allocation Algorithms for 40% excess storage and 200 fragments

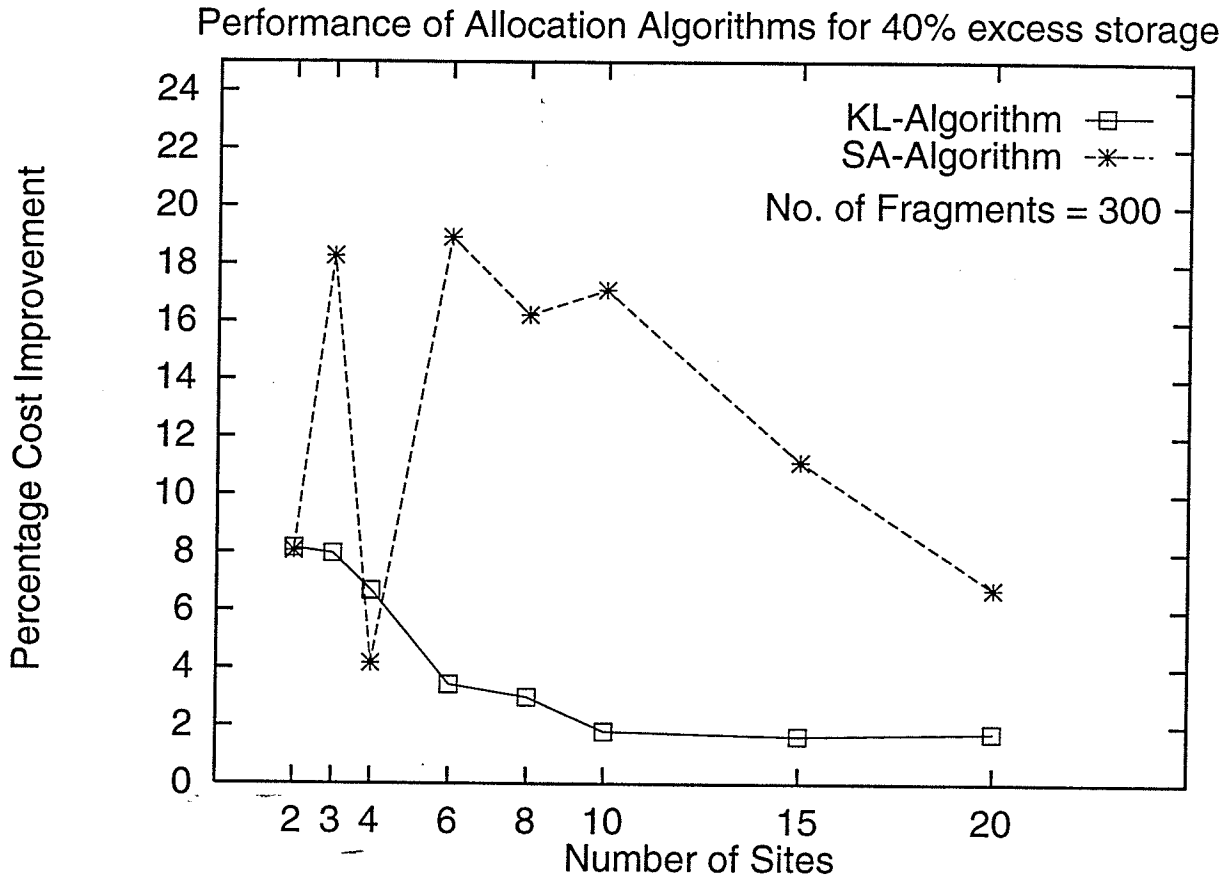

Figure 5.15: Comparative Performance of the Allocation Algorithms for 40% excess storage and 300 fragments

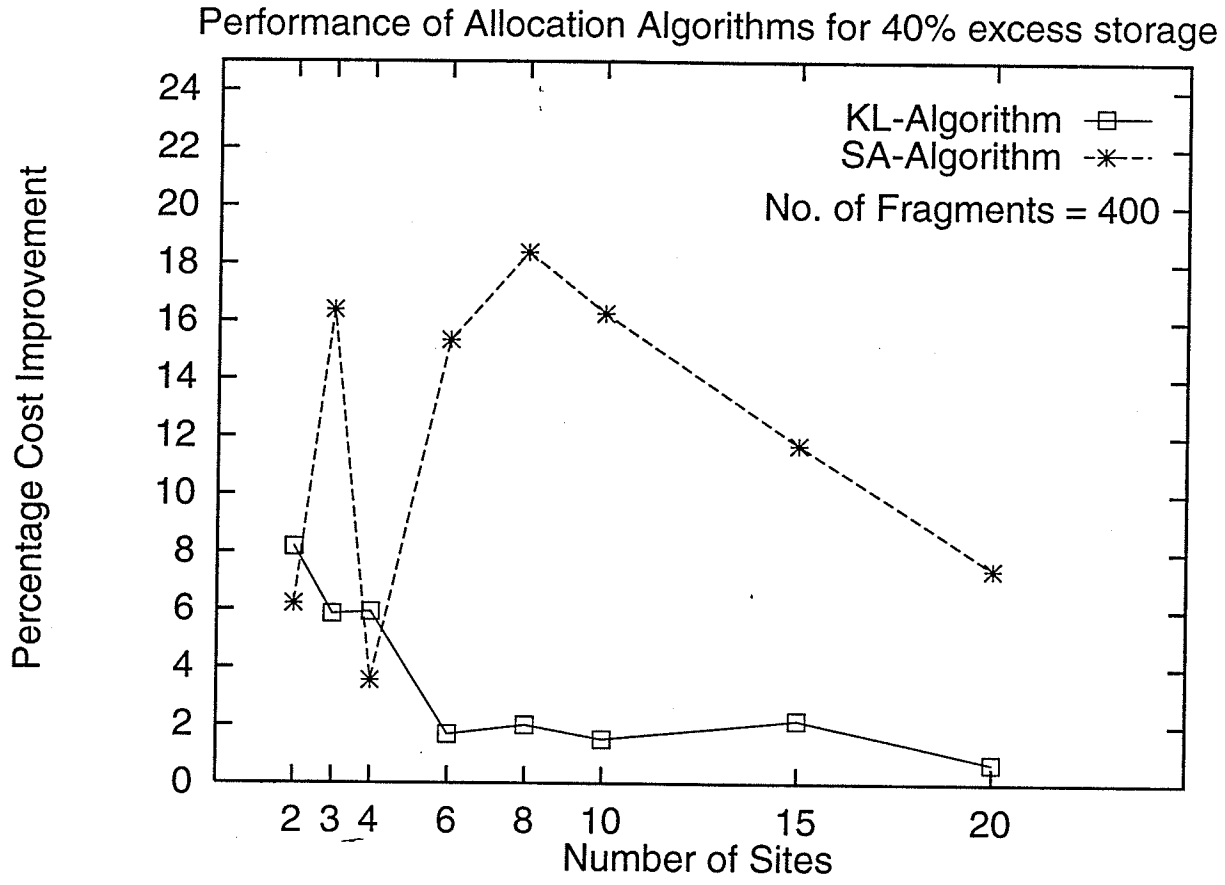

Figure 5.16: Comparative Performance of the Allocation Algorithms for 40% excess storage and 400 fragments

1. It is observed that for networks of fewer sites (number of sites less than 4), the graphical approach is more capable of producing near optimal solution. Simulated Annealing produces better result for DOBS allocation where the number of network sites is between 6 and 15.

2. The performance of either of the optimization algorithms depend significantly on the starting allocation obtained by the initial allocation algorithm but the effect of the optimization strategies is consistent.

3. For values of design parameters the initial allocation produced by any algorithm may not be "nice" enough to be handled by either of the two heuristic optimization algorithms presented in this dissertation. Therefore, unpredictable and "not so nice" results are not surprising. But that is the inherent nature of any heuristic scheme.

4. The analysis indicates that the conventional benchmark algorithms (eg. the K-L Algorithm) for allocation problems are not efficient and reliable for the DOBS allocation. On the other hand, the simulated annealing algorithm can be applied effectively for allocation design in DOBSs where the number of network sites is between 4 and 15. The majority of the real life DOBS will fall into this category and so for all practical purposes, the simulated annealing approach may be a preferred approach compared to conventional bench mark methods (eg. the graphical approach) for DOBS allocation design.

5. Another important observation is the relative magnitude of the mean interfragment affinity, measured by interfragment references, and the mean site affinity plays an important role in making placement decisions for the fragments. If the site affinity is high compared to interfragment affinity, it plays the dominating role in deciding the placement location and vice versa. The test data sets employed for analysis in this dissertation, have the interfragment references higher than the site affinities.

Therefore, the placement decisions are primarily dominated by the interfragment reference costs.

5.6 Runtime and Complexity Analysis

Since we are addressing a static allocation problem in this dissertation time is not a critical issue. However, any allocation scheme is beneficial only if the improvement in performance achieved by the optimal allocation outweigh the overhead involved in executing the allocation algorithms on the objectbase. Therefore, a brief analysis on the time complexity of the proposed allocation algorithms is necessary for the sake of completeness. The worst-case time complexities are evaluated for the allocation algorithms to demonstrate their efficiency and applicability in real life DOBS allocation.

Assume N_f represents the number of class fragments to be allocated and N_s the number of sites in the network of diameter D. Further, let the number of application executing at different sites of the network be N_a.

The initial allocation Algorithm 3.1 involves the following steps:

1. Sorting of the N_f class fragments (line 1) is a $O(N_f log N_f)$ operation.
2. Sorting of the list of N_s sites (line 2) is a $O(N_s log N_s)$ operation.
3. Considering the worst case, each fragment is tried at every site which is a $O(N_f N_s)$ operation.

Therefore, the overall time complexity of the algorithm can be written as $O(N_f(log N_f + N_s) + N_s log N_s)$. Since $N_f \gg N_s$, we can approximate the time complexity as $O(N_f(log N_f + N_s))$. This analysis does not consider the steps involving constant number of operations as they would add a constant term to the expression for complexity.

For the heuristic optimization method based on graphical approach presented as Algorithms 3.2, 3.3, 3.4, and 3.5, the worst-case time complexity can be estimated as follows:

In Algorithm 3.2 computation of site affinities for all sites (Figure 3.2 lines 10-11) is a $O(N_s N_a)$ procedure since entries for all applications are to be checked for each site in the *ASA* matrix. Computation of the D values of all the fragments in a pair of sites (line 14) is a $O(\frac{{N_f}^2}{{N_s}^2})$ procedure, assuming there are $\frac{N_f}{N_s}$ number of fragments, on an average, at every site after initial allocation. This is because for every fragment in one site, all other fragments in the other site of the pair of sites must be considered.

Step(3) (Figure 3.2 lines 18 - 24) is repeated at most $\frac{N_f}{N_s}$ times and *D values* are evaluated every time Step(3) is executed. Hence, execution of Step(3) involves $O(\frac{{N_f}^3}{{N_s}^3})$ operations. Further, Step(2) (Figure 3.2 lines 13-29) includes Step(3) and Step(4). Assuming the convergence is reached in finite number of iterations, the total number of operations performed by Step(2) is $O(\frac{k{N_f}^3}{{N_s}^3})$, where k is the constant denoting number of iterations required to reach *local optimality.*

Therefore, the total number of steps performed by the Algorithm 3.2 that checks for pairwise optimality is $O(N_s N_a + \frac{k{N_f}^3}{{N_s}^3})$, ignoring steps involving constant number of operations. On simplification, the complexity of the algorithm 3.2 can be expressed as $O(N_s N_a + \frac{{N_f}^3}{{N_s}^3})$, ignoring the constant multiplier.

The Algorithm 3.5 executes the Algorithm 3.2 for each pair of sites (Figure 3.5 lines 8-9) until *global optimality* is reached. This means the Algorithm 3.2 is invoked $O(p{N_s}^2)$ times, where p is a constant denoting the number of times the loop for *global optimality* is executed[3]. Therefore, the overall complexity of the Algorithm 3.5 can be approximated as $O({N_s}^2 * (N_s N_a + \frac{{N_f}^3}{{N_s}^3}))$, which on simplification, gives $O({N_s}^3 N_a + \frac{{N_f}^3}{N_s})$.

Since, $N_f \gg N_s$, the second term is the dominating term in the above expression and therefore, the complexity of the heuristic graphical algorithm can be expressed as

[3]It is assumed that convergence is reached in finite number of iterations.

$O(\frac{N_f{}^3}{N_s})$.

Since the algorithm 4.2 is based on stochastic principles, any closed form expression for its time complexity cannot be derived. However, an application of the algorithm on the set of test cases ensures its convergence in a reasonable time even for large data sets [GS86], [HASG94].

Chapter 6

Conclusions

6.1 Summary and Contributions

In this dissertation, a comprehensive study of issues in the distribution design of a DOBS is undertaken. The objective of this dissertation is focused on the design and analysis of allocation techniques in distributed objectbase systems. The problem of allocation has not been satisfactorily solved for distributed file systems and for distributed relational database systems. Due to additional complexities of the object model, the reported results and analysis of the allocation schemes for DFS and DDBS are not applicable to the DOBS allocation problem. This dissertation contributes by making the pioneering attempt to design a solution to the allocation problem in DOBSs. The significant contributions made by this research work are summarized below.

First, this dissertation makes an important contributions to the database technology in general by presenting an analysis of allocation models designed for DFS and DDBS and classifying them based on an allocation taxonomy. The allocation taxonomy captures the principal characteristics of a general allocation model and therefore, can be used to analyze any other allocation models developed.

Secondly, the allocation taxonomy provides the basis for analyzing an allocation

model for DOBS and identifies the primary characteristics of a DOBS allocation model with respect to its assumptions and constraints that distinguishes it from other DOBS allocation models.

An important contribution of this thesis is the formulation of a nonredundant cost model for DOBS allocation by parameterizing the input information required by the allocation scheme and deriving formal mathematical relationships between them. The model also captures the object oriented features of encapsulation and inheritance to formally express the impact of the input parameters on the allocation decisions. Based on the model, an algorithm for an initial allocation, and two optimization algorithms are designed for DOBS.

The graphical optimization technique, based on the work on graph partitioning by Kernighan and Lin [KL70], treats the class fragments and their cross references as the nodes and edges between them respectively. The algorithm attempts to arrive at a "near optimal" distribution of fragments by exchanging and/or moving fragments between every pair of sites.

The simulated annealing technique is used for the second optimization approach. The cost function that represents the total cost of all references to every fragment by other fragments and applications is used as the objective function for the annealing process.

The algorithms are implemented and are tested with carefully generated test data to obtain a comparative analysis of the performances of both of them. The design, implementation and analysis of the allocation algorithms form the most significant contribution of this thesis. A number of significant insights are derived from the analysis of the results obtained from the two algorithms that can be usefully applied to any real life DOBS allocation design. The applicability of the allocation algorithms for the DOBS model of interest may be verified by a cost benefit analysis based on their efficiency. The optimization of the initial allocation is meaningful only if the improvement obtained by optimization justifies the additional overhead. If significant improvement is not ob-

tainable by optimization then an efficient initial allocation scheme alone may serve the purpose.

6.2 Future Research Directions

This Section outlines the future research directions on the allocation problem in distributed objectbase system (DOBS) as an extension of the research work presented in this dissertation. DOBS design is a complex task that involves a large number of parameters and constraints. In this thesis, a simplified model is used to keep the complexity of the problem manageable. The simplicity of the model helped in understanding the nature and characteristics of the problem and made the analysis possible. The working allocation model described in Chapter 3 needs to be generalized so that the solutions obtained are applicable to real-life DOBS. Therefore, additional constraints and conditions need to be considered and the constraints and conditions captured in the model need to be reviewed to extend the scope and applicability of the research results. However, an allocation algorithm for the distribution of class fragments that considers all issues and constraints and provides a complete solution is beyond the scope of this thesis and may not even be possible [KNM92]. In the following sections the issues of allocation problem in DOBS need to be addressed are briefly presented.

6.2.1 Extension of Static Cost Model

In this section some of the possible directions for extending the static cost model for DOBS and the algorithms designed for it are enumerated.

The central issue of the DOBS cost model presented in Section 3.1 is the cost function which is used as the objective measure for minimization. Only communication costs due to nonlocal references to the fragments are taken into consideration in forming the cost function for the proposed model. This is done because communication cost

dominates other cost factors due to concurrency control, I/O *etc.* These cost factors must also be taken into account in order to make the model complete and accurate. Therefore, the cost function need to be modified to a more generalized cost function that includes other cost factors as well.

6.2.2 Static Performance Model and Static Load Balancing

The primary objective of DOBS design is to achieve enhanced performance. Therefore, performance issues must be captured in the objective function of allocation model. Keeping the cost associated with an allocation in mind, a performance model for DOBS and algorithms to support it must be developed.

An application is a set of method invocations on class objects and the overall performance of DOBS will depend on how efficiently each of the methods are executed. The efficiency is measured against some performance metric. Usually overall throughput and/or response time is chosen as the metric. A common factor that influence both throughput and response time is delay. For every method execution, the factors that contribute to the overall delay include network delay, processing delay and I/O delay.

Due to the complex execution hierarchy of the methods, performance analysis in DOBS is a nontrivial task. Moreover, applications may have different performance requirements. To meet different requirements of the applications, different sets of constraints for each type of applications must be defined (thus the application may be required to be classified based on their processing requirements and performance criteria).

One of the underlying assumptions of the model presented in this dissertation is that the sites have sufficient processing power to handle processing loads of arbitrary size. Unfortunately, this is not realistic. Processing capacities available at individual site are limited and to guarantee acceptable performance we must ensure that the distribution of objects do not overload any particular processor while some other processor is sitting idle. A *load balancing strategy* for DOBS design that distributes the processing demand uni-

formly across the sites of the distributed network to reduce queuing delay which thereby improves performance is needed. The distribution scheme must ensure that the total processing power required for a particular site considering all active objects at that site should not exceed the available processing power of all the processors located at that site.

6.2.3 Allocation Model for Partially Replicated Objectbase

The DOBS allocation model presented in this dissertation assumes that there is only one copy of each class fragment in the objectbase. Replication of some of the fragments may be necessary to increase reliability, data availability and system throughput, and to ensure acceptable response time. Obviously, any practical objectbase system design allows a certain degree of redundancy to meet the performance requirement. While additional copies of fragments improves data availability and reliability additional storage cost and cost of updates, concurrency control and maintaining integrity must be considered. Thus a trade-off is necessary in determining the optimal number of fragment copies and their locations. This becomes an additional decision problem in the DOBS allocation model that supports partial replication.

6.2.4 Dynamic Environment and Dynamic Allocation

The allocation scheme for DOBS presented in this dissertation is designed for a static environment where objects as well as applications are stationary. In a dynamic environment, this initial static allocation may not continue to be the best possible distribution due to changing object access pattern. Objects will migrate from one site to another, replicas of existing objects will be created, new objects will be created, and some old ones will be deleted or updated. Therefore, as time progresses the objectbase is expected to undergo continuous change. There may be change in underlying network too. Further, there may be changes in location and access pattern of applications over time. The allocation scheme must adapt to these changes to ensure the best possible distribution

of the object fragments at every configuration of the distributed system. With multiple copies of the object fragments moving around the network, an application may refer to different copies of it in multiple accesses to that fragment. The allocation scheme should dynamically decide the placement locations for the object fragments. By extending the static model to capture these dynamic characteristics in the dynamic allocation model for DOBS a more realistic allocation scheme will emerge.

Further, due to constant change in distribution of class fragments and class objects, the processing load on any particular site of the DOBS will undergo constant change. A dynamic load balancing scheme is needed to adjust the load distribution as the distribution of class fragments and class objects changes.

The allocation taxonomy presented in Chapter 1 needs to be reviewed with respect to the dynamic environment. Although the issues that characterize an allocation model are the same as in static model, the way they impact the design of an optimal allocation scheme in a dynamic environment will be significantly different from that in the static model.

Bibliography

[AG73] S. R. Arora and A. Gallo. Optimization of Static Loading and Sizing of Multilevel Memory Systems. *Journal of ACM*, 20(2):307–319, 1973.

[Ape88] Peter M. G. Apers. Data Allocation in Distributed Database System. *ACM Transactions on Database Systems*, 13(4):263–304, September 1988.

[BG74] J. P. Buzen and P. S. Goldberg. Guidelines for the use of infinite source queuing models in the analysis of computer system performance. In *Proceedings of AFIPS National Computer Conference*, volume 43, pages 371–374, 1974.

[Buz71] J. P. Buzen. *Queuing Network Models of Multiprogramming.* PhD thesis, Harvard University, Cambridge, Mass., 1971.

[CA80] Peter Pin-Shan Chen and Jakob Akoka. Optimal Design of Distributed Information Systems. *IEEE Transactions on Computers*, C-29(12):1068–1080, December 1980.

[Cas72] R. G. Casey. Allocation of Copies of a File in an Information Network. In *Proceedings of AFIPS Spring Joint Computer Conference*, volume 40, pages 617–625, 1972.

[CDY90] B. Ciciani, D. M. Dias, and P. S. Yu. Analysis of replication in distributed database systems. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):247–261, June 1990.

[CH76] K. M. Chandy and J. E. Hewes. File allocation in distributed systems. In *International Symposium on Computer Performance Modeling, Measurement and Evaluation*, pages 10–13, March 1976.

[Che73] P. P.-S. Chen. Optimal File Allocation in Multilevel Storage Systems. In *Proceedings of AFIPS National Computer Conference*, volume 42, pages 277–282, 1973.

[Chu69] W. W. Chu. Optimal File Allocation in Multiple Computer System. *IEEE Transactions on Computers*, C.18(10):885–889, October 1969.

[CNW83] S. Ceri, S. Navathe, and S. Wiederhold. Distribution Design of Logical Database Schemas. *IEEE Transactions on Software Engineering*, SE-9, July 1983.

[CP84] S. Ceri and G. Pelagatti. *Distributed Databases : Principles and Systems.* McGraw Hill Inc., 1984.

[CR90] Ge-Ming Chiu and C. S. Raghavendra. A model for optimal database allocation in distributed computer systems. *IEEE COMPUTER*, pages 827–833, May 1990.

[CY89] D. W. Cornell and P. S. Yu. On Optimal Site Assignment for Relations in the Distributed Database Environment. *IEEE Transactions on Computers*, 15(8):1004–1009, August 1989.

[DF82] L. W. Dowdy and D. V. Foster. Comparative Models of the File Assignment Problem. *ACM Computing Surveys*, 14:287–313, June 1982.

[EB93] C. I. Ezeife and Ken Barker. Horizontal Class Fragmentation in Distributed Object Based Systems. Technical Report 03, University of Manitoba, October 1993.

[EB94] C. I. Ezeife and Ken Barker. Vertical Class Fragmentation in Distributed Object Based System. Technical Report 04, University of Manitoba, April 1994.

[Esw74] K. P. Eswaran. Placement of Records in a File and File Allocation in a Computer Network. In *Proceedings of IFIP Congress.* North-Holland, 1974.

[FB76] D. V. Foster and J. C. Browne. File assignment in memory hierarchies. In *Proceedings of Modeling and Performance Evaluation of Computer Systems*, pages 119–127, October 1976.

[FDA81] D. V. Foster, L. W. Dowdy, and J. E. Ames. File assignment in a computer network. *Computer Networks*, 5:341–349, September 1981.

[FH80] M. L. Fisher and D. S. Hochbaum. Database location in computer networks. *Journal of the ACM*, 27(4):718–735, October 1980.

[Flo89] R. A. Floyd. Transparency in distributed file systems. Technical Report 272, University of Rochester, January 1989.

[FO81] M. Fridrich and W. Older. The felix file server. *Operating Systems Review*, 15(5):37–44, December 1981.

[FO84] M. Fridrich and W. Older. Helix: The architecture of a distributed file system. In *Proceedings of the 4th International Conference on Distributed Computing Systems*, May 1984.

[Gav87] Bezalel Gavish. Optimizing Models for Configuring Distributed Computer Systems. *IEEE Transactions on Computers*, C-36(7):773–793, July 1987.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A guide to the theory of NP Completeness.* W. H. Freeman and Company, 1979.

[GMM92] D. Ghosh, I. Murthy, and A. Moffett. File allocation problem: Comparison of models with worst case and average communication delays. *Operations Research*, 40(6):1074–1085, November-December 1992.

[GP86] Bezalel Gavish and Hasan Pirkul. Computer and Database Location in Distributed Computer Systems. *IEEE Transactions on Computers*, C-35(7):583–590, July 1986.

[GS86] Bruce L. Golden and Christopher C. Skiscim. Using simulated annealing to solve routing and location problems. *Naval Research Logistic Quarterly*, 33:261–279, 1986.

[GS90] Bezalel Gavish and Olivia R. Liu Sheng. Dynamic File Migration in Distributed Computer Systems. *Communications of the ACM*, 33(2):773–793, February 1990.

[GW94] Shahram Ghandeharizadeh and David Wilhite. Placement of objects in parallel object-based systems. In *Proceedings of 10th IEEE International Conference on Data Engineering*, February 1994.

[GWLZ93] Shahram Ghandeharizadeh, David Wilhite, Kaiming Lin, and Xiaoming Zhao. Object placement in parallel object-oriented database systems. Technical Report USC-CS-93-558, University of Southern California, Los Angeles, California, 1993.

[Had63] G. Hadley. *Linear Programming.* Addison-Wesley, 1963.

[HASG94] D. T. Hiquebran, A. S. Alfa, J. A. Shapiro, and D. H. Gittoes. A revised simulated annealing and cluster-first route-second algorithm applied to the vehicle routing problem. *Engineering Optimization*, 22:77–107, 1994.

[HM73] P. H. Hughes and G. Moe. A structural approach to computer performance analysis. In *Proceedings of AFIPS Spring Joint Computer Conference*, pages 109–120, 1973.

[HRSV86] M. D. Huang, F. Romeo, and A. Sangiovanni-Vincentelli. An efficient general cooling schedule for simulated annealing. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 381–384, November 1986.

[IK82] Keki B. Irani and Nicholas G. Khabbaz. A Methodology for the Design of Communication Networks and the Distribution of Data in Distributed Supercomputer Systems. *IEEE Transactions on Computers*, C-31(5):419–434, May 1982.

[JFK78] L. G. Jones, D. V. Foster, and P. D. Krolak. A goal programming formulation used in the file assignment problem. In *Proceedings of R. J. Duffin Conference*, July 1978.

[KGV83] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simualted annealing. *Science*, 220(4598):671–680, 1983.

[KL70] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, pages 291–307, February 1970.

[Kle76] Leonard Kleinrock. *Queuing Systems:Computer Applications*, volume 2. Wiley, 1976.

[KNM92] K. Karlapalem, S. B. Navathe, and M. Morsi. Issues in distributed design of object-oriented databases. In *International Workshop on Distributed Object Management*, pages 47–65, August 1992.

[LL83] Laurence J. Laning and Michael S. Leonard. File Allocation in a Distributed Computer Communication Network. *IEEE Transactions on Computers*, C-32(3), March 1983.

[LM77] K. D. Levin and H. L. Morgan. Optimal Program and Data Locations in Computer Networks. *Communications of the ACM*, 20(5):315–321, May 1977.

[Mac89] M. H. MacDougall. *Simulating Computer Systems: Techniques and Tools.* The MIT Press, 1989.

[MD82] J. Mitchell and J. Dion. A comparison of two network-based file servers. *Communications of the ACM*, 25(4):233–245, April 1982.

[MR76] Samy Mahmoud and J. S. Riordon. Optimal Allocation of Resources in Distributed Information Networks. *ACM Transactions on Database Systems*, 1(1), March 1976.

[ÖV91] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, 1991.

[Pie75] W. F. Piepmeier. Optimal Balancing of I/O Requests to Disks. *Communications of ACM*, 18(9):524–527, Sept 1975.

[RC70] C. V. Ramamoorthy and K. M. Chandy. Optimization of Memory Hierarchies in Multiprogrammed Systems. *Journal of ACM*, 17(3):426–445, 1970.

[SW83] D. Sacca and G. Wiederhold. Database partitioning in a cluster of processors. In *Proceedings of 9th International Conference on Very Large Databases*, pages 242–247, October 1983.

[Wah84] B. J. Wah. File Placement on Distributed Computer Systems. *IEEE Computer*, pages 23–32, January 1984.

[WDIY88] Joel L. Wolf, Daniel M. Dias, Balakrishna R. Iyer, and Yu Philip S. Yu. A Hybrid Data Sharing-Data Partitioning Architecture for Transaction Processing. In *4th International Conference on Data Engineering,Los Angeles, CA*, pages 520–527, February 1988.

[WM94] P. A. Walsh and D. M. Miller. Goal-Directed Simulated Annealing and Simulated Sintering. *Microelectronics Journal*, (25):363–382, 1994.

[YSLC83] C. T. Yu, M. K. Siu, K. Lam, and C. H. Chen. File allocation in distributed databases with interaction between files. In *Proceedings of 9th International Conference on Very Large Databases*, pages 248–259, October 1983.