UNIVERSITY OF SOUTHAMPTON

Adaptive Resource Management in Large Scale Distributed Systems

by

Vijay K. Dialani

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
Faculty of Engineering and Applied Science
School of Electronics and Computer Science

25th November 2005

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND APPLIED SCIENCE
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Vijay K. Dialani

An emergent trend in large scale distributed systems enables collaboration between large numbers of independent resource providers. Grid computing and peer-to-peer computing are part of this trend. Resource management in such systems is inherently different from that found in traditional distributed systems, the key difference being that the new classes of systems are primarily designed to operate under inconsistent system information and temporally varying operating environments. Although primarily used to enable collaboration of computational resources, these systems have also found application in the field of distributed data management. Although the principles of grid computing and peer-to-peer computing have found many applications, little effort has been made to abstract the common requirements, in order to provide a conceptual resource framework. This thesis investigates the alleviation of such common requirements through investigations in the field of online scheduling, information dissemination in peer-to-peer networks, and query processing in distributed stream processing systems.

A survey of system types is provided to highlight the new trends observed. A top down approach to developing a unifying model seems inapplicable and the range of problems encountered in these system types can only be addressed by identifying common trends and addressing them individually. Consequently, three application domains have been identified in the respective fields of online scheduling, data dissemination and stream query processing. Each of these application class is investigated individually. For each application domain, a review of the state-of-the-art is followed by a precise definition of the problem addressed in the application domain and the solutions developed are substantiated with experimental evaluation. Findings from individual applications have been summarized to generalize the observations towards an overall hypothesis.

The initial discussion of online scheduling requirements in computational grids is used to develop an online guaranteed resource provisioning mechanism. This helps investigate adaptive behavior in systems with centralized control and simple resource descriptions. The investigation also highlighted information management requirements in online scheduling systems. Similar requirements were identified in the field of open hypermedia systems and pervasive computing environments, and a common approach was used to address the problems in these domains. The collective set of requirements is addressed in the generalized context of information dissemination in large scale systems. Adaptive behaviour is investigated in the context of a large number of autonomous resources and the self organizational behaviour of such networks. The self organizational behaviour discussed in the context of information dissemination highlights the fact that adaptive behaviour follows the principal of duality. The findings demonstrate that self-organization can be achieved either by improving the provisioning of resources or by manipulating the workload. It highlighted the fact that QoS definitions will play an important part in future distributed systems. The concept of QoS and adaptive behaviour is investigated in the context of a distributed stream processing system. This investigation has led to the development of a stream query processing architecture with the capability to support multiple query optimizations. A novel query processing language, a query planning technique, an operator scheduling algorithm and an SPJ operator are described in the context of the data stream management system. While maintaining the focus on the overall hypothesis, the thesis provides original contributions in each of the application domains.

Contents

List of Figures

List of Tables

Nomenclature

General	
$G_t < V_t, E_t, F_t, H_t >$	A global dynamic graph
V_t	A time varying set of nodes in a dynamic graph G_t
E_t	A time varying set of edges in a dynamic graph G_t
F_t	A set of vectors to represent the capability of V_t
H_t	A set of vectors to represent the capability of E_t
$g_t < v_t, e_t, f_t, h_t >$	A view of graph global graph G_t
v_t	A set of nodes in the local graph g_t
e_t	A set of edges in the local graph g_t
f_t	A set of local constraints on v_t
h_t	A set of local constraints on e_t
Scheduling	
$J_i < r_i, p_i, d_i >$	Computation Job
p_i	Processing Time
r_i	Release Time
d_i	Deadline
κ	Slack Ratio
$FIFO$	First In First Out ordering in a queue operation
M	Set of Machines
P2P Information Dissemination	
P_i	i^{th} peer in the P2P network
R_i	A set of resources with peer P_i
$C < P_i, P_j, R_i, R_j >$	A coalition between P_i and P_j based on R_i and R_j
H_i	Temporal trace of activities observed by P_i
Q	A query to describe the resources for discovery
Query Optimization	
σ	Select Operator
Π	Project Operator
$\bowtie$	Join Operator

Acknowledgements

Every good thing comes to an end and this is no exception.

My odyssey through different research groups here at Southampton and in the UK in general is finally ending in this thesis. A great learning experience in itself and rewarding in every aspect, this has been a path full of testing times and ample rewards. Lessons learnt in this phase of my life are invaluable and hopefully guiding beacons towards my future destinations. Having reached this destination and getting ready for a new one, I can look back and say "It has been worth it".

I am deeply endebted to Professor Anthony, J. G. HEY and Professor David DE ROURE, for giving me a chance to complete this thesis and for maintaining a firm belief in me. In this fast paced life, it is very rare that a student-advisor association has such a profound impact. I am endebted to my supervisors for not only having devoted time in broadening my horizons of knowledge, but also for taking interest in shaping my outlook towards science and life in general.

I am also grateful for the support of some of the marvellous people at the University who have helped me in numerous ways that cannot be described in words. Eric Cooke has been a pillar of strength and a wonderful mentor. Additionally, I would like to thank friends like Jing Zhou, S D Ramchurn, Partha Dutta, Arouna Wokeu, Sanjay Vivek, Victor Tan, Mark Thompson and all the other colleagues at IAM for providing such a vibrant and intellectually simulating environment.

Last but not least, I would like to thank and express my gratitude to my parents and my wonderful sister 'J', for everything.

To . . .
My Parents, my sister J and Eric Cooke

Part I

Introduction

Chapter 1

Introduction

Non-adaptive computational systems capture only one view of the world, usually the one that was defined at design time. They fail to take into consideration that modern distributed systems by their very nature are non-deterministic and dynamically evolving - a fact also supported by the recent advances in the field of pervasive computing infrastructures. Consequently, non-adaptive systems tend to operate in sub-optimal states. Although, in some cases exhaustive enumeration of all possible states of operation allows application developers to impart adaptive behaviour to their applications, in most cases it remains infeasible to ascertain all the possible operating states for applications at design time. As suggested in the survey, presented in Appendix B, a set of common characteristics exist for most such operating environments. At least two possible approaches could be adopted to explore adaptive behaviour in such environments. The first is to create a generic model and validate it for the given set of applications. The second is to synthesize the hypothesis by addressing the research issues within the application domains. As most of the concepts related to adaptive behaviour in distributed systems have not been exhaustively explored by the research community, the latter approach happens to be more appropriate.

Research in distributed computing systems has been diverse and to the best of our knowledge has almost exclusively addressed the part of the problems highlighted in Appendix B. While most research has focused on isolated issues, it is believed that a better understanding of the issues can be achieved if all of them are considered in the context of an all encompassing pervasive computing infrastructure. Additionally, it is believed that these explorations need to be carried out in the context of the generic application scenarios that are applicable in a wide variety of systems, and therefore this work was carried out in the context of three exemplar applications.

This chapter gives a short introduction to adaptive systems and enumerates the characteristics of the operating environments of interest. Then, the choice of application exemplars is justified, along with the research problems that they address. The contributions and the structure of the thesis conclude this chapter.

1.1 Adaptive systems

In general, adaptive behaviour is the ability of a system to modify its behaviour in response to the prevailing operating environment. However, in the context of a large scale system, it is uncommon for each of the system components to have a complete consistent view of the prevailing operating conditions. In such cases, the adaptive behaviour of the system is closely associated with its perceived operational conditions. A generic non-adaptive system can be considered as one which does not allow any changes to its perceived operating conditions and attempts to achieve a certain objective function, given the prevailing operating conditions.

An adaptive system monitors and models its operating environments and either optimizes it objective function or modifies it in accordance with the prevailing operating conditions. Therefore an adaptive system is characterized by its ability to discover, model and utilize the resources found in its operating environment. A number of complex interactions may exist between the operating environments and the adaptive system components. However, only a subset of these interactions will be applicable in the context of resource management. Adaptive resource management techniques in large scale systems allow the system resources to identify the subset of operating environment characteristics that allow for better utilisation of its resources. For the purpose of resource management in adaptive systems, important amongst these sets of interactions are the ones that determine the characteristics of the operating environment and the influence of the workload on the given resource. While an interaction set determines the ability of the resource to collaborate with other resources, the workload influences the type of objective functions and optimisation strategies adopted by the system.

Consider the example of an online scheduling system that receives a number of computation job requests to be processed on a set of unreliable resources, as shown in Figure 1.1. It is assume that the aim of the scheduling system is to maximize the usage of computational resources. A non-adaptive scheduling system will be designed to maximize a given objective function. However, a predetermined objective function may not be the best choice, given the variations in availability of resources and the properties of the job characteristics. An adaptive scheduling system could observe such variations, adapt its behaviour and choose an appropriate objective function to attain the aforementioned goal of maximizing resource utilisation.

The above simple example highlights the three important characteristics observed in adaptive systems.

(1) Autonomy of actions. In the above example, each of the nodes was able to independently

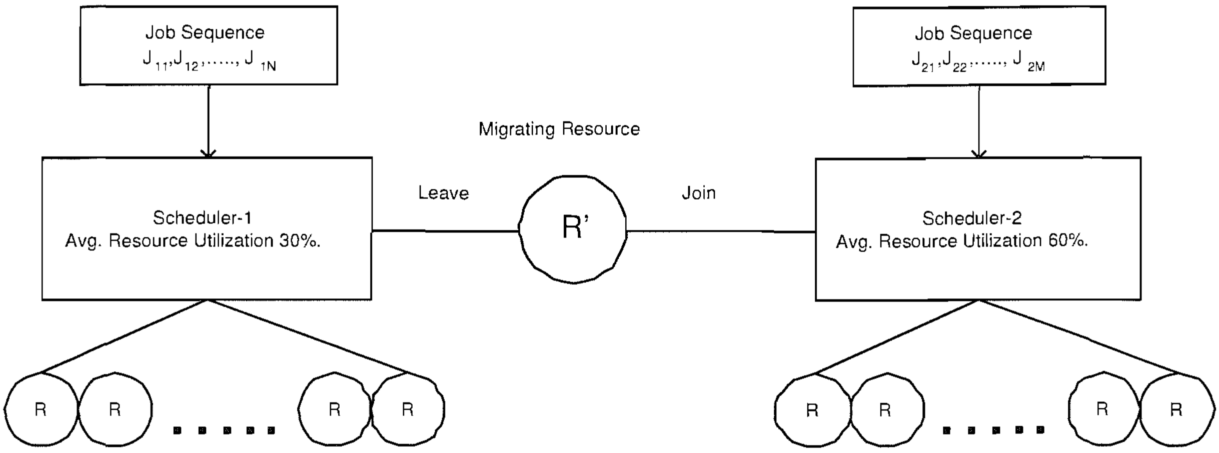


FIGURE 1.1: An online scheduling system with resource migration.

identify whether the resource should be placed under the control of the scheduler. At the same time, the scheduler was autonomously able to determine, when to use the resources. Autonomy of actions is central to this emerging class of systems, it allows them to independently model the operating environments and alter it by cessation of interactions with some parts of the system.

(2) Duality of resource management. The choice of the objective function highlights the duality principle encountered in this class of systems. Abundance of resources translates into a resource provisioning problem, while scarcity of resources leads to a quality of service issue. For example, independent resource providers may cease to participate in the context of the scheduling system if their individual objective functions are not met, while limited resources may force the scheduler to adopt the objective functions which provide a degraded quality of service warranties.

(3) Optimisation over an interval. Adaptive behaviour is instigated when the system components are able to perceive a change in the operating environment. These observations are not always based on observance of a single state change and require the system to observe and model the operational conditions over a period of time. Constant adaptations or too fast an adaptation may lead to instability and the systems need to be able to identify such features.

The list of properties is non-exhaustive and will be strongly dictated by the notion of adaptive systems behaviour. For the set of systems of interest, as described in Appendix B, adaptive resource management is primarily concerned with two types of resources, namely computational resources and information discovery in distributed systems. Additionally, the effects of quality of service on resource management within a single resource are of interest.

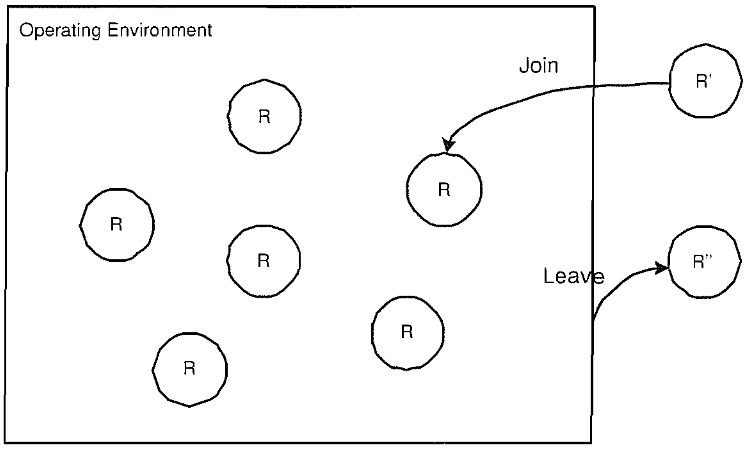


FIGURE 1.2: Adaptive systems with dynamic resources.

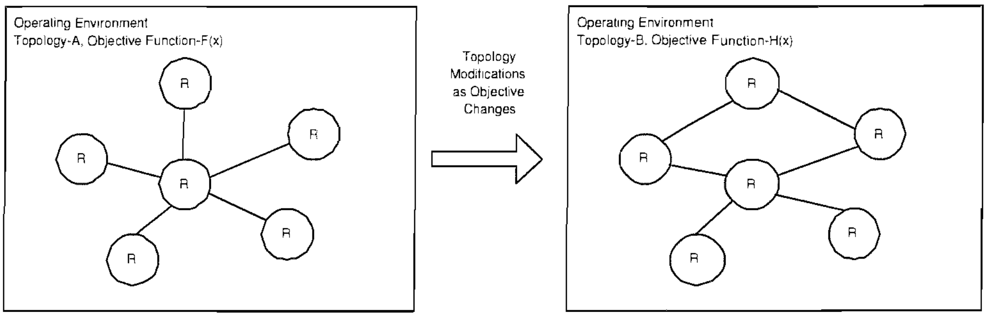

Operating Environment
Topology-A, Objective Function-F(x)
R
R
R
R
R
R
Topology Modifications as Objective Changes
Operating Environment
Topology-B, Objective Function-H(x)
R
R
R
R
R
R

FIGURE 1.3: Adaptive systems with evolving topologies.

1.2 Classification of adaptive systems

Three common types of adaptive resource management scenarios are widely encountered in the set of systems highlighted in Appendix B - namely Resource Adaptation, Topological Adaptation and Quality of Service (QoS) Adaptation. In cooperative operating environments, availability of resources influences task distribution between the available resource providers. Applications capable of maximizing their objective function in response to the availability of resources and ability to degrade gracefully when these resources are no longer available are considered to be resource adaptive 1.2.

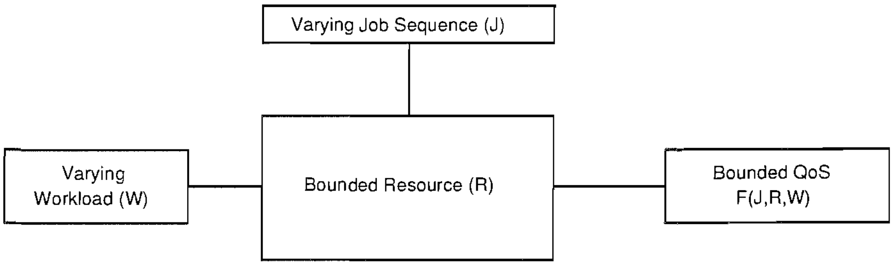


FIGURE 1.4: Adaptive systems with QoS adaptation with load variance, under fixed resources.

Topological adaptation 1.3 refers to the capability of an application to modify its communication behaviour in response to changes in the operating environment. The key difference between resource adaptation and topological adaptation is that the latter approach leads to the formation of a different cooperative structure in response to changes in the environment. In the former type, the variations in resource availability are absorbed within the structure.

Finally, the QoS Adaptive systems 1.4, represent a special case where the operating environment has a finite set of resources. Unlike the previous two cases, where additional resources within the cooperative structure augment the capability of the operating environment, the QoS Adaptive systems try to absorb the variations in the workload or the operating environments and adapt their internal behaviour to adhere to bounds specified by QoS.

1.3 Application domains

Distinct properties are associated with each of the three types of adaptive system. These include resource description and utilisation models, the ability of resources to adapt in any given operating environment and the type of resource optimisation feasible within the given context. In order to select an appropriate application exemplar to investigate the resource management features of systems with the characteristics described in Appendix B.2, common resource definition and adaptation scenarios were considered. To investigate the resource adaptive systems, an online scheduling system capable of providing performance warranties on a set of networks of workstations was considered. Detailed problem definition about the application domain can be found in Chapter 2. Although this exemplar application presents the simple case of computational resource adaptation, most similar systems have not considered the support for objective functions; instead they rely on best effort approximations. The challenge therefore is to devise adequate resource management capability to support objective function, such as warranted completion.

Topological adaptation is widely applicable in the context of sensor networks and application overlay systems. The key feature of these environments is their ability to choose the appropriate members of the overlay in response to the variations of the overlay structure. Although, to date structured and unstructured overlays have been widely employed for this purpose, none of the techniques allows the applications to evolve a topology given the operating constraints. Topological adaptation has been investigated in the context of Peer-to-Peer (P2P) information dissemination systems. However, none of the existing approaches takes into account the resource capabilities of individual peers, or the self-evolution and self organizing aspects of overlay systems. A detailed description of the challenges can be found in section 5.1. Findings from this study are widely applicable in query routing in sensor networks and resource management in P2P system environments.

Investigation into the properties of the QoS based adaptation was carried out in the context of Data Stream Management Systems (DSMS). Processing over the 'append only' data streams specified in terms of continuously executing queries provides an ideal scenario for a static workload, while the variations in the properties of the incoming streams allow simulation of the variations in the operating environment. Finite memory and computational resources provide a resource bound operating environment, one of the key assumptions when investigating QoS. Although, a number of approaches have been advocated in the field of data stream query processing, none of the current approaches investigates the potential resource sharing between concurrent evaluations of queries on data streams. This approach investigates resource sharing and the resultant concurrency control issues in the context of multi-query continual optimisation over data streams. A detailed description of the problem domain and the scope of our work are presented in section 8.3. Findings from this exemplar have helped further state-of-the-art in DSMS query processing, and can be generalized to be applicable to QoS aware systems with bounded resources.

The choice of application exemplars provides a dual opportunity to further the state-of-the-art in application domains, while contributing to generic investigations in the area of adaptive systems. A significant degree of overlap between basic properties like autonomy of actions, incremental access to input data, partial visibility of state information and the temporal nature of the operating environments help to enforce some common behaviour across applications. The following section describes the problems in individual application areas, along with pointers to the seminal work in these fields, which is further extended by the current investigations, and summarizes the contribution of this thesis.

1.4 Contributions

The contributions of this thesis can be divided into two distinct categories: contributions to individual application domains and generic contribution to an overall hypothesis. The hypothesis proposes a model for dynamic graph based representation of large scale systems. The contributions to the application domain are further classified into the following three categories:

Online Scheduling An online scheduling model for Grid environments is proposed as an extension to the original scheduling theory discussed in (Leung 2004). Online scheduling assumes that their is a continually varying demand for resources and jobs. The aim is to determine the feasibility of an optimal planning strategy under the time-varying demand and supply of resources and jobs, and to minimize the cost of optimisation in case a strategy exists. The program allows job migration between multiple resource providers, and there are penalties on commissioning and decommissioning of resources within a resource provider. In order to minimize resource penalties and forecast the usage of any spare resources, the individual resource providers form coalitions to outline the cooperative strategy between various resource providers. The initial motivation of the application domain and the application are described in Chapter 3, the generalized optimisation scenario is described in Chapter 2 and further developed in Chapter 3. The algorithmic contributions are:

1. An online planning algorithm for online scheduling on multiple machines, as described in Chapter 3.
2. The experimental evaluation of the algorithm as provided in chapter 4

Information Dissemination Information dissemination focuses on information management in an ad hoc network environment. The motivating applications are described in Chapter 5 and in the information discovery and management techniques in Peer-to-Peer environments (Stoica, Morris, Liben-Nowell, Karger, Kaashoek, Dabek, and Balakrishnan 2003; Zhao, Kubiatowicz, and Joseph 2001; Rowstron and Druschel 2001). The approach for this thesis uses the characteristics of the information and its demand and supply characteristics to create an overlay. The overlay is optimised for probabilistic routing mechanisms, also known as search mechanisms. An empirical evaluation of the approach is presented for a finite number of system nodes. The algorithmic contributions for this are as follows:

1. A distance based similarity search algorithm is described in Chapter 5
2. A self-organizing overlay mechanism is described in Chapter 5

Query Optimisation The problem of query planning and processing in a data stream management system are described in (Babcock, Babu, Datar, Motwani, and Widom 2002). This scenario is used to examine the complexity of the combinatorial optimisation in dynamic environments and use dynamic graph techniques to address a multiple optimisation problem. A dynamic graph based data structure is used to represent a query planning scenario that needs to produce an optimal operator ordering to reduce the cumulative resource utilisation across multiple queries. The motivating examples, related database literature and query planning and processing algorithms are described in Chapter 8. The algorithmic contributions presented in Chapter 8 are:

1. A dynamic programming based query planning algorithm for queries on data streams
2. An interval search tree-based select, project join operator.
3. An algorithm for concurrency control of multiple query processing for stream data management systems.

4. PSQL, an extension to SQL, for specifying queries over streaming data.

1.5 Structure of the thesis

The literature review for each of the application domains is presented in the chapters that are directly related to the discussion sections and is distributed across chapters to closely associate it with the new contributions. The next chapter describes the general theoretical basis for the work presented in the three application domains. The rest of the thesis is organized as follows.

Chapter 2 : Adaptive Systems This chapter additionally outlines the characteristics of adaptive systems and proposes an ad hoc resource group (AHG) based model for representing large scale distributed systems. The AHG's are modelled as dynamic graphs which capture the characteristics of autonomy and partial information visibility. The properties of the model are then defined as verified by the application exemplars.

Chapter 3 : Online Scheduling Online scheduling techniques are investigated for computational resource sharing over a set of federated and autonomous resources. The algorithms for admission control are presented in online job Grid scheduling systems and analysis is provided for the case of finite time horizons.

Chapter 4 : Evaluation of Online Scheduling This chapter presents the empirical evaluation of the algorithms described in Chapter 3 and provides comparison with the Earliest eXpiry First (EXF) algorithm. It considers the case of uniform and variable jobs and evaluates the algorithm for various values of slack k, for k=*0*, $k \geq 1$ *and* $k \leq 1$, and for variable job arrival rates. It then describes the use of resource advertisement as a means of developing an overlay of online scheduling systems, the information needs for which are further explored in the next chapter.

Chapter 5 : Resource Management in P2P Systems This chapter introduces the adaptive overlay formation and maintenance for resource management in peer-to-peer networks. A brief review and application domain description is followed by a comparison of the existing overlay management techniques. Modifications to these techniques and a generic model for probabilistic overlay creation and maintenance are described.

Chapter 6 : P2P Coalition Formation and Search Algorithm This chapter discusses the application of the generic algorithm described in Chapter 5, to the domain of Peer-to-Peer Open Hyper-media Systems (OHS). It introduces the application scenario and presents a formal description of the search algorithm.

Chapter 7 : Evaluation of the Search Algorithm This chapter provides empirical evaluation of the algorithm developed in chapter 6.

Chapter 8 : Query Optimisation This chapter begins with a description of query processing in data streams and states the relevance of the application domain to adaptive systems. The model used to describe data streams provides an analogy for the infinite sequences of data items that are evaluated for a set of queries. This is followed by a description of a dynamic graph-based query processing algorithm. An IBS-SPJ operator is presented for shared range predicate evaluation. Query re-optimisation techniques and operator scheduling techniques for efficient evaluation conclude the chapter.

Chapter 9: DSMS - Implementation, Evaluation and Analysis This chapter presents the architectural details of the DSMS implementation and the experimental evaluation and analysis of the query optimisation algorithm, the IBS Operator and the Operator Scheduling algorithm.

Chapter 10 : Conclusions This chapter correlates the observations in the various chapters and summarizes the contributions of the thesis. It is followed by a discussion of the potential for future work and directions arising from the above work.

1.6 Suggested order for reading

A legend presented in figure 1.5, is provided to allow easy navigation through this text. It is suggested that all readers familiarize themselves with Part I, which outlines the objectives and scope of the work presented in other parts of the thesis. Part II, Part III and Part IV can be read independently of each other, and are complete pieces of work in their own right. Relevant sections of Part V may be read to see how each of the Parts II, III and IV relate to the initial discussion in Part I. It is hoped that the organisation diagram helps navigation through this complex text.

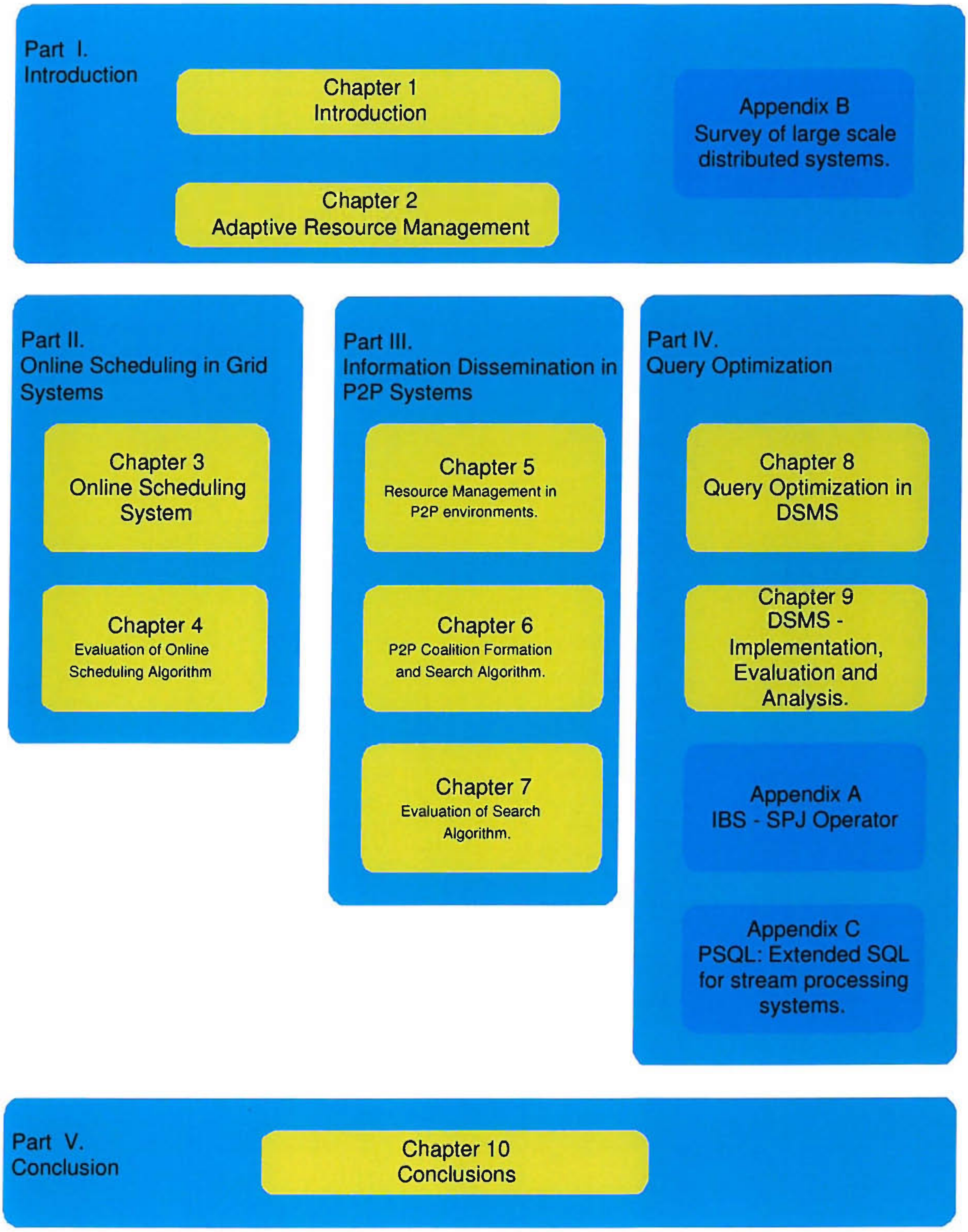

Part I.
Introduction
Chapter 1
Introduction
Chapter 2
Adaptive Resource Management
Appendix B
Survey of large scale distributed systems.
Part II.
Online Scheduling in Grid Systems
Chapter 3
Online Scheduling System
Chapter 4
Evaluation of Online Scheduling Algorithm
Part III.
Information Dissemination in P2P Systems
Chapter 5
Resource Management in P2P environments.
Chapter 6
P2P Coalition Formation and Search Algorithm.
Chapter 7
Evaluation of Search Algorithm.
Part IV.
Query Optimization
Chapter 8
Query Optimization in DSMS
Chapter 9
DSMS - Implementation, Evaluation and Analysis.
Appendix A
IBS - SPJ Operator
Appendix C
PSQL: Extended SQL for stream processing systems.
Part V.
Conclusion
Chapter 10
Conclusions

FIGURE 1.5: Organisation of thesis.

Chapter 2

Adaptive Resource Management in Large Scale Systems

The World Wide Web (WWW), Grid Computing, Peer-to-Peer (P2P) systems and Semantic Web represent emerging technologies employed to develop large scale computing systems. Almost all of them are enabled by cooperative resource sharing between multiple autonomous resource-providers. While the web represents one of the most scalable implementations of distributed system for sharing data/information content, it primarily remains a client-server system with content providers in exclusive control of the resources. Recent advances in the above mentioned technologies such as Grid Computing and P2P, attempt to achieve a higher level of resource integration by incorporating coordinated use of computational and data resources across multiple resource providers/consumers, with decentralized control. However, unlike the simple data content resource sharing on WWW, the coordinated use of computational resources and data presents greater challenges of synchronisation, management and utilisation of resources, as is summarized in Appendix B, which presents a review of these open issues and summarizes the state-of-the art in the above mentioned technologies.

From current trends in these emerging technologies, one could safely infer that future systems may use a common representation based on "virtual organisations (VOs)" to represent a dynamically associated set of resource providers and consumers. A virtual organisation is synonymous to an ad hoc resource group created solely for the purpose of sharing computational and data resources amongst its participant members. To date, VOs have been represented as a loosely connected set of resources and very few investigations have focused on the management of resources within the VOs, and interactions between multiple VOs. This study presents a VO based representation of large scale distributed systems, for this purpose. A VO is created by dynamic association of resource providers and consumers, and the association may evolve over a period of time as the resource providers and consumers join and leave the VO. During its lifetime, a VO needs to provide the necessary mechanisms to facilitate collaborative resource usage and management mechanisms. This chapter investigates the resource management features in these organisations and describes their usage in different application scenarios.

The rest of the chapter is organized as follows: Section 2.1, introduces ad hoc resource groups (AHG's) as generic VO's and enumerates its characteristics. Section 2.2, describes the resource management requirements in AHG's and describe the open issues in managing resources in them. Section 2.3 describes the generic model for description and management of resources in this thesis; it also describes how the work will be developed in the following chapters. Section 2.4 describes the rationale for choosing application scenarios in the investigation. Section 2.5 provides an alternate view of how advances in these application scenarios can be used to develop a distributed stream management system.

2.1 Ad hoc Resource Groups

Increasingly large scale systems are being formed by dynamic online association of computational and data resources. Electronic market places, virtual organisations, peer-to-peer computing and networks of workstations are examples of the emerging type of structured/semi-structured AHG's for managing a practically infinite number of computational resources in online collaborations. Most such ad hoc resource groups are created to provide a service to the members of the institution and are either based on social, economic or utilitarian models. AHG's assume a structure that is better suited for accomplishing their objective function; examples include a market model for environments with scarce resources and a cooperative model for partitioning the large search space. These institutions may impose certain restrictions on the behaviour of the participants - restrictions may assume forms of behavioural guidelines, online checks or dominant strategies - to effectively manage the ad hoc resource group. Enforcement of such restrictions may require monitoring of the communication between participants, monitoring the state of each or some of the participants, and using a reward/penalty mechanism. These rules, restrictions and behavioural patterns of the ad hoc resource group are referred to as the "operating environment".

Operating environments provided by an ad hoc resource group may be either centrally managed or collectively managed by a set of distributed managers, or they may be unmanaged - evolving from the collective behaviour of its participants. Most AHG's facilitate interactions between its participants. However some AHG's, for example, auction sites may prohibit direct interaction between the participants. In cases where AHG's facilitate interaction between the participants, they need to provide additional mechanism to coordinate and monitor such interactions. These constraints imposed by the operating environment restrict the autonomy of the individual participant. Examples of such restrictions include constraints on sharing of state information and management of resources. Constraints enforced by the operating environments determine the "visibility" of the individual participants. In most large scale systems/AHG's, no single participant will have the complete knowledge of the entire state of the system. Each of the participants observes a partial environmental state, which in turn determines the scope of its influence on the operating environment and the behaviour of other participants. Considering that all the participants in an AHG exhibit rational behaviour, the state visibility of the participants will determine the sub-space in which they can affect the behaviour of the system. Participants use this observed state information to autonomously determine their "operational behaviour".

It is presumed that the AHG's are dynamic entities that evolve over a period of time. The temporal nature of the observed state may require the participants to take into account the historical evidence relevant to their future actions. In such dynamic environments, an autonomic participant needs to determine its operational behaviour in terms of an objective function on a partially observed temporal state. The temporal nature of the environment and the variation in the observed state usually results in adopting multiple objective functions for different observed conditions. A set of such objective functions is referred to as a "policy". In certain cases, the participants may use a single policy or may consider re-evaluation of the policy during the lifetime of the AHG. Most AHG's that support use of dynamic policies impose constraints on the time period for the validity of the policy, as the policy adopted by one of the participants may influence the behaviour of the other participants.

Dynamic operating environments present a unique challenge. It is not possible to envisage all the operating states at the design time. Consequently, participants need to adapt to dynamic run time environments. As described above the adaptive behaviour of the participant is constrained by the spatio-temporal data visibility of its operating environment and the information they hold on the state of the other participants.

To summarize, no single participant will have knowledge of the complete state in most large scale systems; consequently they need to be organized into groups of participants under ad hoc resource groups (AHG's). Interactions between the participants will determine their ability to collaborate in order to share their resources to accomplish their desired individual goals. Participants in such collaborations will exhibit autonomic behaviour. The dynamic nature of the collaboration means that the participants will need to adopt policies that allow them to accomplish their individual objectives under the prevailing operating environment. Adaptive polices will need to be adapted in temporally evolving operating environments.

2.1.1 Discussion

Autonomy plays an important part in resource management in most distributed systems. For example, consider the case of WWW and Service Oriented Architecture (SOA). As described in Appendix B, both these system types exhibit autonomous resource control. Although autonomous resource control is crucial from the systems administration perspective, it may be argued that it is also crucial for creating scalable distributed systems. The failure of component based middleware systems is widely attributed to their inability to provide wide area distributed resource management. Two approaches have been proposed to address this issue: firstly, stateless Message Oriented Middleware (MOM), and secondly, systems designed as collaborations of manageable hierarchies of resources. Web Services are an example of the former class, while P2P and Semantic Web fall into the latter class, while Grid Computing systems lie somewhere in the middle. This thesis is concerned with the systems of latter type.

Given the premise that the large scale system is viewed as a collection of autonomous resources, a few questions need to be answered.

Why are aggregations of resources referred to as AHG's? Does this refer to Electronic Institutions as defined in the artificial intelligence research community?

Resources are aggregated to achieve some common objectives of the participant resources. For example, an auction site is an Electronic Institution created to support a particular mechanism for negotiations on goods and commodities and represent an ad hoc resource group corresponding to a physical institution, such as the stock market. Similarly, if one considers the aggregation of computing cycles and data resources, these aggregations assume some form and semantics for collaborative usage of the resources. It may be argued that the semantics of these aggregations can be classified and standardized to reflect the standard usage pattern in systems, where each usage pattern reflects a type of Electronic Institution (EI). For example, consider the case of SETI@HOME, the system allows participants to allocate their computational resources to an EI. The EI restricts the state of visibility of the system participants and prohibits direct interaction between them. However, the central EI has a set objective to utilise the collective resources by allocating computational jobs, in order to maximize the throughput. Thus collections of resources are organized in a star topology, with a centralized server acting as a central EI controller. As collaborative use of resources in such aggregations relies on a well-defined overall objective, organisational structure and communication patterns such aggregations are referred to as AHG's.

The notion of EI's as defined in the field of AI deals with the organisation of roles and responsibilities in an agent's community, along with the necessary restrictions in terms of permitted and forbidden actions. However, as far as is known the notion of EI's has never been applied to the development of large-scale systems, this thesis is the first to propose such an extension. The extensions proposed are in some specific characteristics of the AHG's. The first step is to look into the self-organisational behaviour of the participant's, with a specific example of participants sharing computational cycles, secondly, to look into the resource description and discovery support required by these ad hoc resource groups, in the domain of the peer-to-peer service discovery, and thirdly, to look into the effects of task definitions, and task profiles on the performance of a single resource provider, with an example of a stream database management system.

How are AHG's formed?

An AHG comes into existence when some self managing participants of a large scale system form an aggregation that bounds the behaviour of these participants by means of some norms on their behavioural pattern. These norms may be agreed upon at the inception of the AHG or may evolve from the behavioural pattern of the participants. The monitoring of such norms is enforced by the participants of the AHG by means of a reward and penalties mechanism for AHG's, in economic organisations, or by means of reduced influence on the environment in case of social organisations. However, both these organisational types assume that the AHG's are able to enforce some desired behaviour on the participants by means of local restrictions. It is envisaged that the large scale systems can be built as an aggregation of numerous AHG's. The restrictions on behaviour of the AHG's are imposed by means of enforcing appropriate policies.

What are the types of resources they share?

In general, AHG's remain capable of trading any resources encapsulated by the participants. For example, in an Electronic Market place, participants may transact goods and services of all kinds. Considering the case of AHG for studying characteristics of the large scale distributed systems: first of all the computational cycle sharing for high-throughput computing, and secondly the effects of the organisation on information dissemination within an AHG. Finally,there is a study on the effects of resources and task variations on resource provisioning on a resource provider. These characteristics can be mapped to Parts II, III, and IV respectively.

What are the organisational models for AHG?

The AI definition of the AHG does not impose strict restrictions on the way the resources are organized in an AHG. However, when applied to the case of distributed systems the organisation of AHG may have to take into account the physical attributes of the operating environment. For example, Part III describes the scenario for a sensor network, where the participants need to form an appropriate overlay to capture the constraints of the communication network. Such an organisation of the overlay can be captured by means of an overlay network. As described in part III, there are two ways to form an AHG, either using a predefined structure and restricting the objective functions appropriate to the structure or by using a set of objective functions to determine the overlay structure. However, the second case is observed more frequently and is further investigated in part III of the thesis.

Are there some common data structures and algorithms common to multiple types of AHG's?

In ad hoc resource groups, no entity has complete knowledge of the entire system state. The participants need to monitor the operating environment for changes. These changes assume the form of change in known participants, or relations with those participants, over a period of time. These characteristics point to the need for a data structure capable of handling spatio-temporal data. The use of dynamic graph structure to encapsulate such information is considered. Details about the data structure and its use can be found in Part II of the thesis.

What are the semantics of interaction between these AHG's?

When considering a large scale system composed of a number of ad hoc resource groups, it is imperative to allow the participants to discover other such ad hoc resource groups and interact with them (and vice versa). Cases where the participants can autonomously choose the AHG's that they want to be part off require no specific interaction among them. However, when the AHG's are responsible for the effective use of the participants resources, they need to allow exchange of resources between the collaborating AHG's. In such cases, a hierarchical organisation of AHG's may emerge, with a higher level overlay forming between the AHG's.

The above analogies can be applied to various scenarios in Grid computing and P2P systems, where a number of participants form the "virtual organisations" or "peer groups" respectively. Membership of such groups can either be obtained by the autonomous actions of the independent participants or may emerge from interaction between the groups. From the above discussion, one could envisage modelling a large scale system as a collection of such interacting AHG's. From the distributed systems perspective, one needs to establish the effectiveness of such a model in developing distributed applications. A number of different features have been associated with distributed applications, but few are commonly observed. Such common features include: sharing of computational resources and data resources, and orthogonal to such goals are the issues related to the discovery of resources in those systems.

2.2 Resource management in ad hoc resource groups

Ad hoc resource groups provide mechanisms for sharing resources between the various participants and facilitate resource sharing between resource providers and consumers. Resource providers and consumers in an AHG may deal in virtual resources such as stocks and commodities, but when applied to the computational infrastructure these AHG's are assumed to deal in computational resources, data resources or network resources. The scope of this thesis is restricted to AHG's used to manage resources in distributed computing systems. Most distributed systems consider computational or processor cycles as a default definition of a resource. However, a more generic definition of resources is taken to include computational, data and network resources. The term resource is defined as an entity whose state can be controlled and affected by the operating environment. It is therefore natural to consider computing cycles as one of the primary resources managed by AHG's and has proven measures in terms of processor speed, the cache, memory availability, which are quantifiable resource. Other resources such as data remain more qualitative and difficult to describe and manage. Attributes of data resources include provenance, data visibility and synchronisation requirements.

As described above, an AHG consists of resource providers and consumers, with AHG providing the mechanism for maximizing the gain for both classes of participants. An AHG needs to provide the mechanisms for resource providers to publish their set of resources for subsequent discovery and consumption. On the other hand, the AHG needs to provide a means for allowing consumers to be able to express their request in terms of tasks, for which the publishers can provide resources. This matchmaking process can be facilitated by means of a service discovery mechanism within the AHG. Discovery services have been widely deployed in distributed systems to address problems and related ones in resource discovery. However, distributed AHG's may require multiple such discovery services for publication and discovery of various resources, but supporting such a service for an AHG with a practically infinite number of participants is unfeasible. Thus resource providers and consumers need to be organized into groups of collaborating participants- requiring mutual participation to discover the resources. The interaction graph representing this collaboration can be considered as an overlay network for discovery of resources.

Assuming that the participants in an AHG have the necessary discovery mechanisms to acquire the information about the resources of their choice, the use of such resources may be autonomously controlled by each individual provider, or a number of providers may coordinate the use of their collective resources. Coordinated use of resources may be facilitated by the environment (by building it into environmental constraints) or the participants may create ad hoc means to allow such coordination.

This thesis investigates these and related issues in the context of the resource sharing using online scheduling in a Grid computing environment.

2.2.1 Resource description and resource monitoring

In general, a resource can be represented using a set of attributes that define its properties. Applications use relational, semi-structured or an RDF representation of the resources. A resource description reflects the state information about the resource and enumerates its properties. It may be used as an advertisement for resource matching in distributed systems.

This thesis focuses on the structure of the ad hoc resource groups formed primarily for sharing computational and data resources. Irrespective of the organisation of these electronic institutions, a few common issues need to be managed in each, namely: 1. Discovery of resources. 2. Managing computational resources. 3. Managing data resources.

The above patterns of resource management in large scale systems are studies. First one in which the tasks and their profiles are known, while the number or resources available remains uncertain. The second case, studies the effects of collaboration where the task is known, but cost is reduced by controlling the state visibility of the system. Finally, a system where different task profiles need to be managed, given a definite set of resources is presented.

2.3 Modelling the ad hoc resource groups

The notion of operating environments was introduced in section 2.1. However, to the best of our knowledge their exists no generic model to describe an operating environment. This section, presents a hypothetical model that allows to capture the notion of ad hoc resource groups (AHG's).

A dynamic graph based representation to model the infinite set of resources in an ad hoc resource group is employed. The dynamic graph G_t is used to represent the global view of the entire state of interest. The graph G_t is composed of nodes V_t and edges E_t, the capabilities of the node V_t are described by a vector F_t, while the capabilities of the edge are captured by the vector H_t. The computational resources providers are mapped as the nodes in the global network G_t. The graph G_t represents a completely connected graph if all the nodes are able to communicate with each other. The communication channels are represented by means of the edges of the dynamic graph [1]. A temporal dynamic graph representation captures the temporal behaviour of the system dynamics. In cases, where the graph G_t is a subset of a completely connected graph and the edges

[1] Wired networks like the ethernet allow a n-to-n connection, however physical factors constrain communication in an ad hoc networks, which in turn rely on their neighbouring nodes to route the message

E_t represent the point-to-point communication path between the nodes, the graph G_t represents an overlay network [2].

The computational costs associated with the maintenance of such a dynamic graph increase exponentially with graph size, making it unfeasible to maintain a centralized system state. Distributed localized views can be used to maintain the global system state. Synchronized maintenance of the localized views allows the system to maintain a global view, suggesting the use of sparsification techniques (Eppstein, Galil, Italiano, and Nissenzweig 1997) to construct the global graph properties from local graphs. However, it is not always mandatory to maintain synchronised local views. Alternately, each node can maintain its local view, hereafter referred to as the local view g_t, which is composed of nodes v_t and edges e_t. It should be noted that a local view may be maintained by an individual node, or a group of nodes may maintain a shared local view.

It is assumed that each node maintains a local view of the graph in order to maximize its objective function. Considering that the messaging costs are directly proportional to the radius of the graph g_t, the diameter of the graph will be restricted by the associated state maintenance costs. Costs associated with maintenance of the localized view are usually weighed against the benefit acquired by maintenance of such a state. The nature of this association between the local view and the objective function is incumbent on the requirements of the application domain, and cannot be generalised across the model. However, the dynamic graph model allows expression of such constraints in the form of the vector functions associated with the nodes and the edges of the graph.

Constraints on the function vectors of the nodes and edges of the graph raise issues related to the topology of the graph G_t. The topological constraints remain crucial to the systems ability to self-organize itself in the event of change. Using a temporal representation of the graph allows capture of the evolutionary aspect of the AHG's. Some applications may consider a time series representation of the graph parameters. Standard time series inferencing techniques could be applied to monitor the behaviour of the overlay network, using historical data to predict the changes in the AHG's.

The sharing of resources in AHG's is a dynamic process, whereby the system state changes dynamically over a period of time. Such changes may be modelled as changes in the neighbourhood (by means of e_t), changes in the availability of nodes (by means of v_t) and changes to the global properties (by means of f_t and h_t). Considering that the AHG's are primarily constructed to achieve some local or global objective function, these changes will reflect on the applicability of the objective functions. The temporal nature of the graph allows us to capture the notion of optimisation over a period of time. It is envisaged that such temporal optimisations will be the norm in most complex systems, as compared to the static optimisation techniques employed today. Optimisation overtime introduces an important concept of time boundaries. Systems that optimize over a

[2] A review of structured and unstructured overlay networks can be found in Part III of this thesis.

fixed interval in time are referred to as finite-horizon, while the rest are classified to optimize for infinite horizon systems, which can both be represented in this graph model. Analogously, the objective function of the node also exhibits temporal behaviour.The temporal nature of the system also applies to the availability of the information and is orthogonal to online computing paradigms.

The above model is applicable to a class of problems, much beyond the scope of this thesis. The emphasis of this thesis is to verify the above hypothesis in real application domains. Consequently, it is restricted to a distinct class of problems details of which can be found in the following section. If validated, the hypothesis can be used to formally represent the characteristics of AHG's in large scale distributed systems. A formal representation of AHG's will help in devising appropriate mechanisms for developing applications in an AHG based environment.

2.4 Relation to applications

The above model for maintaining the AHG's utilizes a graph based representation $G_t = (V_t, E_t, F_t[], H_t[])$, where each of the variables has a temporal dimension. The model is based on the hypothesis that adaptive large-scale distributed systems can be built as structured/unstructured AHG's, under the constraints of partial state visibility, temporal constraints, disparate resource definitions and resource management requirements. It is envisaged that the study of AHG's will:

1. Help identify common resource management requirements across a class of systems.
2. Provide insights into the temporal behaviour of large-scale distributed systems.
3. Identify the effect of the operating environment on the self-organizing behaviour of AHG's.

Consequently, one has chosen different application domains, namely online scheduling in Grid systems, information dissemination in P2P systems, and multiple query optimisation in stream database management systems. These applications have been chosen to investigate the adaptive resource management aspects of large scale systems with disparate resource definitions, objective functions and application models. It should be noted that, in addition to attempting to prove the above hypothesis, the thesis advances the state-of-the-art in all the application domains. The application level problem definition and contributions to the application domain have been summarized in the parts corresponding to the respective applications.

2.4.1 Motivation

While addressing the online scheduling problem for Grid systems the issues related to the node function V_t, the collective objective function, the effect of partially available information of the job profiles are investigated.

The second application - Information dissemination in P2P systems - investigates the issues related to selecting a subset of edges e_t, in order to satisfy the global objective function of reduced communication costs, where communication costs are expressed in terms of H_t. The application also investigates the effects of changes in graph diameter.

The third and final application studies the effects of variation in task profiles in a stream data management system. In this case the interest is to understand, how changes in task profile affect the choice of the object function of the given application ?

2.4.2 Objectives

Table 2.1 summarizes the application-level objectives for the described application domains.

2.4.3 Focus of the work

Each part of this thesis (namely Part II, III, IV) focuses on its specific application domain. This part, describes the focus of the research in each application domain [3].

Online Scheduling in Grid Systems: Stochastic and online scheduling techniques have been widely studied under a wide variety of scheduling scenarios. Grid Systems exhibit the characteristics of both these system types (for details refer to Part II, Chapter 4, which also describes a Grid Scheduling System). While stochastic scheduling provides mechanisms to formulate a scheduling policy, given a job distribution and resource availability profile, online scheduling allows for scheduling decisions to be performed in an online manner. Similar research issues have also been highlighted by (Leung 2004). The focus of the work conducted in this area has been:

1. To demonstrate the effects of organisation on the behaviour of network of workstations.
2. To design admission control strategy for online scheduling systems in Grid scheduling environments.

[3] At the start and the end of each part, a brief summary is provided to map the application level goals to the high level goals described in this section.

Information Dissemination in P2P systems Structured and unstructured system topologies used in peer-to-peer systems have been widely used to form ad hoc peer groups. Part III of this thesis investigates the correlation between resource distributions and topology structures in P2P systems. The primary objective of the investigation is to ascertain the effects of change in information distribution and workload on the characteristics of the overlay system.

DSMS - Query Optimisation Query processing in data stream management systems represents a special case, in which a fixed set of resources need to be efficiently managed in order to satisfy the demand on resources. Although, the research issues in the field have been highlighted by (Babcock, Babu, Datar, Motwani, and Widom 2002), the issue of multiple query optimisation and time based optimisation has not been addressed in the field of query processing in DSMS. Part IV of the thesis, proposes multiple query planning and re-optimisation techniques, which take into account the temporal behaviour of the input streams.

2.4.4 Application level goals

Most of the applications encountered have used static optimisation techniques and were not designed to function in dynamic environments. Therefore, applications were chosen from the well-understood application domains, and were examined under the setting of adaptive resource availability. Each of the applications was studied under different resource availability criteria and conditions, details of which can be found in Table 2.1. A number of limitations of the applications were identified in the process of adapting these applications. Such an exploration into the applications was put into the perspective of the application level goals.

2.4.5 Discussion

In its base form, resource management represents a constraint satisfaction problem, where applicable solutions maximize the objective function of the system components and the system as a whole. Application to large scale systems with variable availability of the resources poses an important challenge as to how to represent the collection of resources and choose appropriate organisation, so that the topology of the organisation is fit for the particular application. It is assumed that although the exact details of resource management may vary across the system classes, there definitely exists some common usage patterns across these classes. This thesis examines the resource management features outlined in Table 2.1, further details of which can be found in the following parts. A case that discusses the use of all the techniques in the context of a single application scenario is presented in the following section.

2.5 Alternative systems view

The thesis is organized into a number of parts, each of which discusses the techniques in a specific application scenario. While the techniques remain specifically suitable for the application scenarios in question, they can also be applied to develop similar systems. This section describes a Distributed Stream Data Management System, which is based primarily on the techniques discussed in the thesis.

Part IV of this thesis introduces a central Data Stream Management System (DSMS), capable of supporting concurrent and continuous queries over streaming data. A DSMS provides optimized performance over a multi-variable objective function, which consists of time, memory and computational resource usage. The objective is maximized by coordinating the use of computational and memory resources to achieve the desired level of response. The details of this optimisation problem are discussed in chapter 8, which assumes a single processor for execution of the query. The scheduling system of DSMS operates on a sequence of unrelated operator schedules, and requires strict warranties on timeliness of response. The DSMS scheduler requires a subset of the functionality of the scheduling system discussed in Part II of the thesis. The DSMS scheduler can be modified to take the advantage of the techniques described in Part II.

The techniques developed in Part III of the thesis can also be integrated, if one considers the Publish-Subscribe paradigm for access to continuous queries. The Pub/Sub model for access to query processing on streaming data requires publishers of data streams to allow discovery of their data resources. The information dissemination methods discussed in part III, can be used to wrap the schema information as advertisements of resources. The algorithmic principles developed in Part III can then be employed to share the information between a number of DSMS.

Although, the approach adopted by the thesis is in line with the primary focus of exploring the notion of adaptive resource management, the above example demonstrates that the techniques can be combined and will find use in multiple systems.

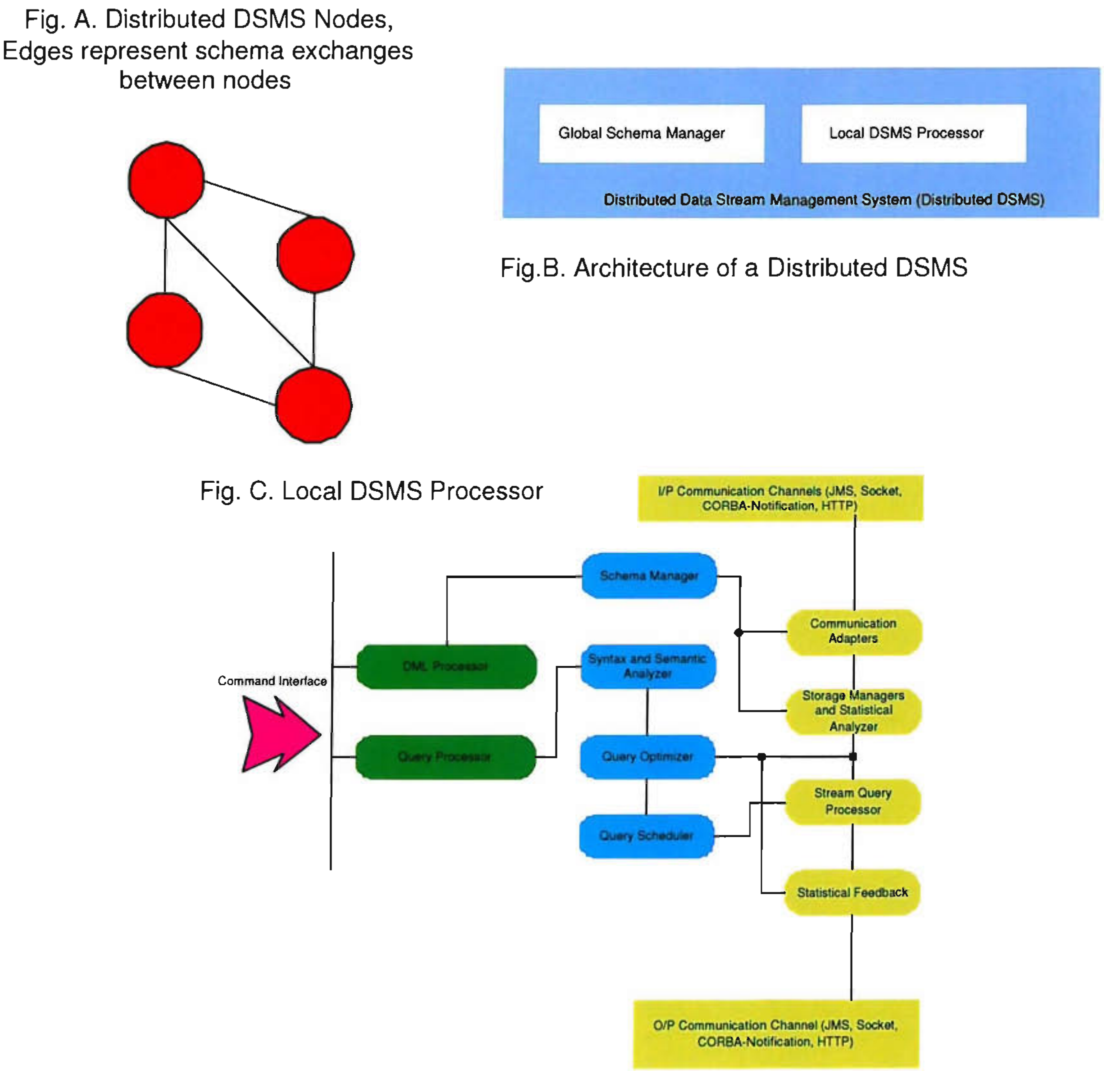


Fig. A. Distributed DSMS Nodes, Edges represent schema exchanges between nodes

Fig.B. Architecture of a Distributed DSMS

Fig. C. Local DSMS Processor

Global Schema Manager
Local DSMS Processor
Distributed Data Stream Management System (Distributed DSMS)
I/P Communication Channels (JMS, Socket, CORBA-Notification, HTTP)
Schema Manager
Communication Adapters
Command Interface
DML Processor
Syntax and Semantic Analyzer
Storage Managers and Statistical Analyzer
Query Processor
Query Optimizer
Stream Query Processor
Query Scheduler
Statistical Feedback
O/P Communication Channel (JMS, Socket, CORBA-Notification, HTTP)

FIGURE 2.1: Distributed data stream management systems.

Properties	Online Scheduling	P2P Information Dissemination	DSMS-Query Optimisation
Information Visibility	• Peers have no knowledge of their subordinate peers, restricted visibility exists between the Master and Slave nodes in Master or Slave configuration. • Job information is revealed at release time; optimisation can only be based on the current information.	• Peers have knowledge of immediate neighbour and may obtain update on information of additional neighbours. • Peers have information on the queries routed and the routing path. Changes in resource information are propagated to interested peers. • Historical records of the query routing and resource information may effect the future decisions of the peers.	• Query processor has limited visibility of stream characteristics and absolutely no information on tuple contents and arrival rates. • Future queries may not have any knowledge of existing queries.
Temporal Characteristics	Task profile and resource profile varies over time.	Coalition profile varies over a period of time.	Resource utilisation of individual query varies in accordance to stream characteristics.
Optimisation Type	Moving finite horizon optimisation.	Instantaneous optimisation based in historical data.	Continual optimisation with finite horizon.
Objective Function Types	Each peer tries to maximize its individual objective function.	Peers coordinate activities to improve group objective function.	A DSMS tries to optimize the objective function of multiple concurrent tasks.

TABLE 2.1: Mapping application objectives to hypothesis

Part II

Online Scheduling in Grid Systems

Chapter 3

Online Scheduling

this chapter introduces online scheduling algorithms for scheduling jobs on multiple machines. This is followed by a thorough review of existing algorithmic techniques for interval scheduling of independent jobs on multiple independent machines, for preemptive and non-preemptive scheduling. The following section describes the existing admission control mechanisms for allocating jobs on a single machine and subsequently introduces the scheduling algorithm, which provides admission control for scheduling on multiple related machines.

3.1 Introduction

Scheduling has been studied extensively in many varieties and from various viewpoints for application to practical computer systems. The basic situation requires processing a sequence of jobs on a set of machines. In the most basic problems, each job is characterized by its running time and has to be scheduled for that time on one of the machines. Other variants introduce additional restrictions and relaxations on the schedules allowed. Most scheduling algorithms are designed to maximize an objective function for a given sequence of jobs and the resultant schedule is considered appropriate if it maximizes some objective function. The notion of an online algorithm is intended to formalize the realistic scenario, where the algorithm does not have complete access to the whole input instance. Instead, the algorithm learns of the input piece by piece and has to react to new requests with only partial knowledge of the input sequence.

Most online scheduling problems are classified on the basis of which part of the problem is given online. Sgall (Sgall 1998) introduces one such classification which has been repeated here. His paper introduces the following classification:

Scheduling jobs one by one. In this paradigm the jobs are ordered in some list and are presented sequentially from this list. The scheduling algorithm assigns these jobs to some machine and time slot(s) before the next job can be seen. The assignment needs to be consistent with other constraints given by the problem. It is assumed that the job characteristics, including running time, are known at the time the job is presented. It is allowed to assign the jobs to arbitrary time slot(s), even if this incurs penalties. However, alteration of the schedule, subject to the visibility of future jobs, is not permitted.

Unknown running time Unlike the previous case, this case assumes that the running time of jobs is not known at the start time and that the total execution time can only be calculated at the time of completion. However, at any time, all the currently available jobs are at the disposal of the algorithm; any one of them can be started, preempted or delayed on any machine(s).

Jobs arrive over time In this paradigm the algorithm has the freedom to start, preempt or delay any of the currently available jobs and, in addition, the running time of each of the job is known at the time of submission. The only online feature is lack of knowledge of the job's arrival time.

Interval scheduling All the previous paradigms assume that a job may be delayed. Contrary to that assumption, interval scheduling assumes that each job has to be executed at a precisely given time interval. A job is rejected if it cannot be executed within the specified interval. In this case, the length of the schedule generated is essentially fixed; hence, tardiness, makespan and/or delay based objective functions are not applicable in this case. Instead, measuring the weight (or the number) of accepted jobs is generally applicable.

Most of the above-mentioned characteristics, such as, online arrival of jobs, interval scheduling and scheduling jobs as they arrive are also observed in recent applications such as Grid Scheduling and scheduling on a Network of Workstations (NOWS). SETI@home (SETI), CONDOR (Litzkow, Livny, and M.W.Mukta 1990); they represent systems that receive jobs in an online fashion. However, the objective is to maximize throughput in these systems . Other objective functions, such as interval scheduling and co-scheduling have been the focus of recent research in the field of Grid Computing (Foster and Kesselmann 1999). This chapter proposes the use of online interval scheduling with admission control for scheduling in a dynamic environment such as Grids. A case for the applicability of online interval scheduling is presented in the section 3.1.1. The following section, 3.2, introduces the relevant definitions and presents a formal description of an online interval scheduling algorithm (Section 3.5). Section 3.5.1 presents proof that the algorithm with admission control performs just as well as the Earliest eXpiry First (EXF) algorithm, and Section 3.6 summarizes the chapter.

3.1.1 Discussion

Most of the online scheduling algorithms have been studied in the context of real-time scheduling systems. However, most of the above-mentioned properties that characterize an online scheduling problem are also applicable to evolving computing paradigms such as Grid computing (Foster and Kesselmann 1999). A characterisation of different types of Grid systems was provided by Fox et.al. (Fran Berman (Editor) 2003), which introduces a taxonomy for various types of Grid systems. Computational Grids represent one such system type that facilitates sharing of computational resources between multiple resource providers and consumers. Section 4.2, describes a scenario inspired by scheduling system in computational Grids that schedules jobs on multiple independent machines. This scenario is inspired by existing Grid applications such as those described by (Abramson, Buyya, and Giddy 2002; Nabrzyski, M., and Jan 2004), where both producers and consumers collaborate to provide a virtual computational resource. In each of these applications the scheduling system learns about the job at its release time and needs to schedule feasible jobs on a set of available resources. Various approaches have been suggested, which include but are not limited to an economics based approach (Abramson, Buyya, and Giddy 2002), a reservation based approach (Graham, E.L.Lawler, J.K.Lenstra, and Kan 1979) and throughput maximisation based approach (Litzkow, Livny, and M.W.Mukta 1990). While some of the Grid scheduling systems accept the jobs to maximize there own objective function, the others derive their objective function from the quality of service guarantees specified in the job description. The case of a former type of scheduler is considered. It tries to maximize a given objective function and accepts or rejects jobs according to its scheduling policy.

No Grid scheduling system has a prior knowledge of all the jobs to be scheduled by the system, and needs to evaluate jobs on arrival. Some scheduling systems accept all the incoming jobs and attempt to schedule them before the expiry of the job. In such cases, the scheduling system does not reject the job unless the job can no longer be scheduled to meet its processing requirements. The drawback of such scheduling systems is that the job processing system does not provide any guarantee on the completion of the job. However, rejection at expiry time may not be advisable in certain time critical systems which require strict guarantees on the completion of the job. Specialized scheduling systems that provide job completion guarantees have been investigated in the field of online scheduling systems, and employ admission control systems to selectively accept or reject jobs at release time. The following sections that precedes section 4.2, introduce the definitions and a review the existing online scheduling algorithms, and suggest online admission control for scheduling jobs on m related machines.

3.2 General definitions and review

3.2.1 Definitions

Following the standard notation introduced by Graham et al. (Graham, E.L.Lawler, J.K.Lenstra, and Kan 1979), it is considered that all jobs are independent with no precedence relationships between the jobs and no communications or synchronisation requirements between the jobs. Jobs are revealed to a scheduler at their respective *release time* r_i. At release time, the scheduler learns about the processing time p_i, and the *deadline* for the job d_i. The job J_i has a slack $s_i = d_i - p_i - r_i$, which represents the amount of time between the arrival of the job and the last possible time at which it could be started to meet its deadline, also known as its *expiry time*. The *minimum slack ratio*, also known as the *patience* of the job scheduling problem, is defined as $\kappa = min_i(s_i/p_i)$, so that every job J_i has a slack of at least $s_i \geq \kappa \cdot p_i$. Δ denotes the ratio of the largest and smallest processing time of the jobs in the schedule σ. The *gain* of the schedule σ on a instance I is defined as $\sum_{J_i \in \sigma} p_i$. The gain of the schedule is maximized, subject to the objective function for the schedule. Candidate object functions that have been investigated include make-span optimisation (Graham, E.L.Lawler, J.K.Lenstra, and Kan 1979; Albers 1997), weighted job optimisation (Lee 2003; DasGupta and Palis 2000). While a non-preemptive job is considered to be successfully processed if it was allocated the resources uninterruptedly for the duration p_i, a preemptive job can be paused and restarted any number of times and should be allocated resources for the cumulative duration p_i, where completion in both cases is subject to deadline d_i. The scheduling system is considered to be clairvoyant if the processing time of the job is known at release time, while cases with unknown running time are referred to as non-clairvoyant systems.

The scheduling algorithm A deals with allocation on single or multiple machines. In either case, the performance of A is measured by comparing its gain with the gain of an optimal *(opt)* off-line scheduling algorithm, which has complete prior knowledge of the jobs when creating the schedule. Online algorithms are classified into deterministic algorithms or randomized algorithm (Borodin and El-Yaniv 1998a). An online A deterministic online algorithm A is *c-competitive* if $gain_{opt}(I) \leq c \cdot gain_A(I)$, for all input instances I. When considering randomized online algorithms, the competitiveness compares the gain of the optimal schedule to the *expected* gain of a randomized algorithm. Competitive ratio c is determined by determining the worst case input for an algorithm. For details on the use of adversary based techniques used to determine the competitive analysis and the adversary-algorithm(A) interaction models, refer to (Borodin and El-Yaniv 1998b). Adversary based techniques (Lee 2004) create the worst case input sequence I and have complete knowledge of algorithm A and utilize it to create the sequence I.

An online scheduling algorithm can schedule any job J_i before its expiry time e_i. In order to maximize its gain a scheduler is not required to complete all the jobs and may reject some of them. The notification of acceptance or rejection of the jobs can be deferred until time e_i. However, admission control mechanisms (Goldwasser 2003; Goldwasser and Kerbikov 2003; Garay, Naor, Yener, and Zhao ; Goel, Meyerson, and Plotkin 2001) have been suggested for cases where the notification for the acceptance or rejection of the job needs to be provided before e_i. An admission control mechanism is not usually required under underload conditions, but assumes prominence under overload conditions.

The above nomenclature and definitions are equally applicable to online scheduling on single or multiple machines. However, following additional definitions introduced below are specifically applicable to online scheduling on multiple machines. In the case of multiple machines it is possible to co-allocate (Schwiegelshohn and Yahyapour 2004) the job to more than one machines. However, the scope of this discussion is restricted to the case where a job can be assigned only to a single machine at any given time.

At any given time, each job can only be assigned to one machine, in the set of available machines M_k. M_k is considered to be a set of uniform machines if $\forall M_i \in M_k | M_i = M_j$ for any i and j. At any given time t, a number of jobs may be available for scheduling. A scheduling system may maintain all the jobs in a single queue or may allocate them to separate queues on arrival. While a scheduling system that does not pre-allocate the jobs usually maintains a single queue for all the jobs, multiple queues are maintained in cases where the job allocation precedes the actual allocation of the job. A scheduler is considered online if it processes the job in an exact FIFO order. However, in certain cases, sorting and selection on the job queue is permitted. For example, the algorithm presented by (Lee 2003) is considered online. It describes a scheduler that sorts the queue of available jobs and select the job with maximum length. In online scheduling a job allocation cannot be subjected to any future changes. However, if the jobs once allocated to the queue are reassigned, then the scheduler is considered to be semi-off-line in nature.

In cases where a scheduler maintains a single queue for all the incoming jobs, each individual queue is aware only of the currently executing jobs. In the case of non-preemptive scheduling, the set consists of a single currently executing job. A job is feasible on a machine if its allocation allows all the jobs to attain their deadlines. In some cases the scheduling decision may be invoked on arrival of the job while in other cases the decision may be invoked when a machine becomes available. If the scheduling decision is made on the arrival of the job, it is likely that the job may be equally feasible on more than one machine, also referred to as a tie. The performance of an online algorithms relies crucially on the way it breaks ties; however it should be noted that some algorithms break ties arbitrarily.

3.2.2 Review

It is not possible within the scope of this section to provide a detailed review of online scheduling algorithms. A detailed review of online scheduling algorithms can be found in the (Borodin and El-Yaniv 1998a; Leung 2004). The scope of this section is limited firstly of all to highlight the state of the art the online scheduling algorithms for multiple machines and secondly, to summarizing the current admission control mechanism for multiple machines. The review is not restricted to algorithms by the choice of a particular objective function, but gives emphasis to interval scheduling.

Graham et.al. (Graham, E.L.Lawler, J.K.Lenstra, and Kan 1979) introduced the concept of online scheduling and provide formal analysis of their algorithm, also known as the List Scheduling Algorithm. It allocates an incoming job to the least loaded machine and has a proven lower bound of (2+1/m) for makespan minimisation, where m being the number of machines. Later, Albers (Albers 1997), provided an improved bound at a competitive ratio of *1.923* for $m \geq 2$, this algorithm maintains two groups of lightly loaded and heavily loaded machines; a job is allocated to lightly loaded machines only if it cannot be scheduled to a heavily loaded machine. These online scheduling algorithms represent the first studies in the case of online scheduling for makespan minimisation on multiple machines. There have been a number of studies for scheduling on multiple machines for different objective functions.

The general model of online scheduling was further refined by Lipton and Tomkins (Lipton and Tomkins 1994), who also introduced the concept of online interval scheduling. In interval scheduling all jobs request immediate use of resources and need to be completed before a fixed interval. Their model (for a single resource) implicitly assumes that a scheduler has no prior knowledge of the value of Δ. They prove the fact that, with jobs of equal length, greedy interval scheduling is guaranteed to find an optimal schedule. For jobs with two distinct lengths, the authors provide a randomized 2-competitive algorithm, and, for jobs with arbitrary length the algorithm is $O((log\Delta)^{1+\epsilon})$ - competitive. This result was improved upon by Goldwasser (Goldwasser 2003), to prove that the competitive ratio for multiple machines is bounded by $(2 + 1/\kappa)$. This result for a single resource was also proven to be valid for multiple resources (Kim and Chwa 2001). However, all the results for interval scheduling on multiple resources rejected the job at its expiry, and, as far as is known (Goldwasser and Kerbikov 2003) have written the only paper to consider admission control for a single resource in the case of interval scheduling. The work by Goldwasser et.al. (Goldwasser and Kerbikov 2003) remains the only work to study the effect of resource admission control on the performance of an online interval scheduling system. It makes three important contributions. First, it provides a 4-competitive randomized algorithm capable of providing immediate notifications, secondly, a 3 - competitive randomized algorithm with no notifications and thirdly, it proves that no randomized algorithm which provides immediate notification can be better than *7/3*-competitive. A few other alternative approaches for online interval scheduling on multiple machines are Lee (Lee 2003) and Das (DasGupta and Palis 2000). For online mechanism design, Lee (Lee 2003) introduces a static classification based randomized algorithm, for a specific case of $k < 1$ and is an important result in the field. The work by Das (DasGupta and Palis 2000), highlights the effects of online scheduling with rejection and restart. The most recent results from mechanism design deal with scheduling on a single machine under unreliable job information, and prove that adequate mechanisms can be designed to provide performance guarantees on the schedule generated by such collections of resources.

3.3 Problem definition

The problem of online allocation of clairvoyant jobs on a variable set of resources/nodes is considered with the aim of developing an admission control system. Each node is represented by a queue, and is autonomous in the sense that a node may join or leave the collection of nodes. However, as a part of the system each node accepts jobs over a finite horizon, each queue is considered as a possible candidate of the incoming job, if it can process the job while retaining its prior commitments. The decision to allocate a job to the queue is irrevocable, and jobs are reallocated only in the case of a queue failure.

At any given time the queue holds a list of the current job and allocated jobs. The state of each of the queues is completely visible to the scheduling system, which must decide the allocation of the next job. Job acceptance by the scheduler is constrained by the finite horizon imposed by each job in the queue. This finite horizon is always at a constant distance from the current time instance. This continual time representation, when analyzed at any instance in time, reduces to a bin packing problem, with constraints on allocation strategy.

The basic scheduling system in this setting has to:

1. Maximize the competitive ratio of the scheduling of jobs on resources with equal queue lengths.
2. Maximize the competitive ratio in case of unequal queue lengths.
3. Minimize the penalties in case of resource failures.

Associating resource costs and weighted jobs represent extensions to the existing basic scheduler.

3.4 The semantics of job allocation

Resources/nodes are represented as queues, where each resource in the scheduling system exhibits autonomous behaviour. There are at least two possible ways of implementing such a decentralized scheduling system. The first alternative is to assume a centralized job queue, with the scheduler acting as a broker (also known as matchmaking), queues competing with each other to provide the best possible allocation strategy for the maximisation of the objective function. The second alternative involves a master-slave configuration, where the queue relinquishes the control to a centralized scheduler, which adopts appropriate job allocation strategy to maximize the objective function. As demonstrated in experiments (refer to next chapter), the first strategy of matchmaking results in disproportionate allocation of jobs, consequently resulting in a decrease in the overall competitive ratio. The master-slave online scheduler and the admission control system performed better and were used as the preferred architecture.

In a master-slave configuration, the central scheduler has complete knowledge of the state of each of the queues. As the sequence of jobs is known at release time, the centralized scheduler has to choose jobs that can be appropriately allocated to each of the queues. When presented with a job the scheduler needs to ascertain if it should accept or reject the job. If accepted, it needs to allocate the job to either of the queues. As described in the previous section, once allocated to a queue the job cannot be reassigned. Therefore, a scheduler may defer the actual allocation of the job, but accept it on the basis of the feasibility criterion. By deferring the actual allocation of the job, the scheduler retains the flexibility to reassign the jobs to an appropriate node and achieve higher competitive ratio. However, the deferral process incurs additional processing costs of the complexity $O(n^3)$, detailed discussion of which can be found in (Brucker 2001).

In order to maximize the competitive ratio of the online allocation, the scheduler should ensure that no resource/node remains unallocated during any interval. Consequently, a greedy strategy is adopted for allocation of resources. A greedy strategy allocates the job to any idle queues. However, if each of the queues has a currently executing job, the scheduler needs to assign the job to the most appropriate queue. Two prominent strategies can be adopted. A scheduler may back fill the current queue before starting to allocate the jobs to the next available queue; alternately it could try to ascertain the best resource it can allocate the job to. The proof in section 3.5 demonstrates that the best fit strategy performs better then the strategy of allocating jobs to the first available queue, a fact also validated by experimental analysis presented in the next chapter.

3.4.1 State transition representation of job status

An online scheduling system processing a continuous stream of jobs classifies the jobs in accordance to their state. Consider a scheduling system in which jobs can be maintained

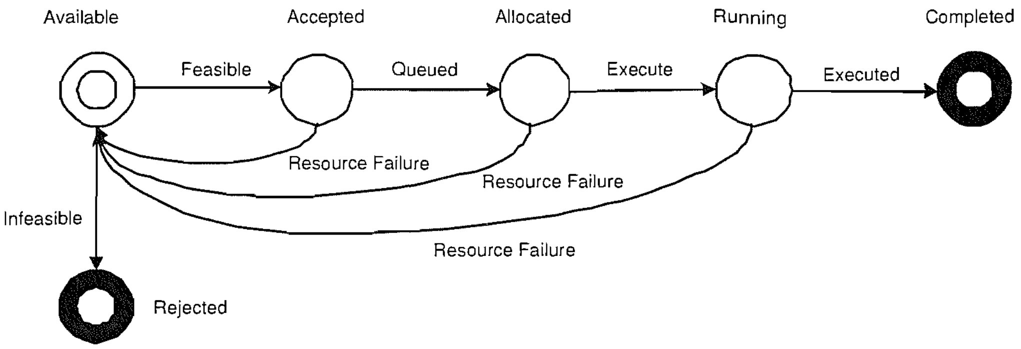


FIGURE 3.1: State transition of a job in a Grid scheduling system.

in either of the following states:

Available A job submitted for allocation is queued in this state.

Accepted If the job is feasible in any of the job queues it is classified as accepted.

Allocated A job assigned to a job queue is referred to as an allocated job.

Rejected A job is rejected either for being infeasible or having passed its expiry time due to resource failure.

Running Jobs being executed at any of the queues are classified as running.

Completed Jobs successfully executed are marked completed. Only completed jobs contribute towards the gain of the online algorithm.

The possible lists of transitions have been represented in the following figure 3.1:

3.5 Algorithm - Best Fit Interval Scheduling (BFIS)

A cluster of resources is considered, where each resource is represented as a queue, as introduced earlier. Theoretically, each of the resources can have infinite capacity. However, as the cluster of resources has been formed by the dynamic association of the resources, practical systems requirements introduce the bounds on warranties and only limited queue sizes are considered. The length of the queue determines the finite horizon - the point in time beyond which the queue ceases to accept allocations. Such a bound can be expressed in terms of Δ and the maximum permissible slack κ_{max}. The size of the queue represents the maximal permissible interval allocation permissible and can be considered synonymous to fixed bin size in the bin-packing problem.

In most practical scheduling systems of this type, κ_{max} is determined by the average lifetime of the resource's participation in the cluster. While the maximum acceptable length of the job may be decided by the penalties introduced on the failure to process the job.

At any given instance time t, the bin has a length, and the item can be shifted.

Best Fit is a greedy algorithm and is based on the following heuristics:

1. Allocate a job to empty queue.
2. For all allocations within a queue, use the EDF semantics for executing jobs.
3. On arrival of a job, consider its feasibility on the current set of machines. If feasible on multiple machines, use the best fit criterion for breaking the tie for allocation between multiple machines.

The following example illustrates the use of the above scheduling strategy for a sequence of jobs $J = \{J_1, J_2, J_3, J_4, J_5\}$ that are scheduled on a set of machines $M = \{M_1, M_2\}$, where each of the jobs is represented as:

$$J_1 = <0,\ 1,\ 3>$$

$$J_2 = <0,\ 1,\ 2>$$

$$J_3 = <0,\ 1,\ 2>$$

$$J_4 = <0,\ 1,\ 2>$$

$$J_5 = <0,\ 1,\ 2>$$

On arrival of job J_1, both machines have an empty queue and are equally best fit, and hence the tie is broken arbitrarily. For the sake of this example, consider that job J_1 is allocated to machine M_1. On arrival of J_2, the machine M_2 is empty and it receives the assignment of job J_2. The job J_3 remains feasible on either of the machine and happens to be best fit for M_2, while J_4 and J_5 remain feasible on M_1 alone.

Consider an optimal online scheduling strategy OPT. Let J represent the sequence of jobs $J_1, J_2, J_3, ..., J_n$ that are to be scheduled over a set of machines M. Let A represent the best fit scheduling strategy used for allocating the jobs in a queue. Each job $J_i =< r_i, p_i, d_i >$, to be scheduled on the set of machines, is made visible at the release time r_i. Depending on their release time, the jobs in the sequence J can be classified into two distinct categories: those released during the *busy* interval and those released during the *free* interval. An interval is considered free for the algorithm A if at least one of the machine queues is empty at the time. As A happens to be greedy, all jobs released during the *free* interval are scheduled by both A and also by OPT.

Let J' represent the set of jobs released during the free interval. As mentioned above, both OPT and A process the set J'. Let J'' be the set of jobs released during the busy interval and let J''_A and J''_{OPT} represent the subsets of J'' processed by A and OPT respectively.

$$CompetitiveRatio(c) = \frac{J' + J''_A}{J' + J''_{OPT}} \tag{3.1}$$

$$\frac{J' + J''_A}{J' + J''_{OPT}} \leq \frac{J''_A}{J''_{OPT}} \tag{3.2}$$

The above equations prove that the effects of admission control are evident only in case of *busy* intervals and hold true in both the cases of preemptive and non-preemptive job scheduling. Although, the use of preemptive techniques is a common occurrence, non-preemptive scheduling has been considered in Grid scheduling. A non-preemptive job allocation provides exclusive control of the resource - an advantage considering the security and provenance requirements in a Grid environment.

An analysis of algorithm A for a non-preemptive scheduling strategy is presented in the following section.

3.5.1 Analysis

Under the following conditions, algorithm A will reject the job while OPT will accept it.

Already executing a job As A uses a non-preemptive scheduling technique, it cannot admit a job while executing a current job. However, OPT, with complete knowledge of the input sequence will not start a new job (provided it is not tight) if it expects a job to be released during the execution time of the already accepted job. Consider Δ to be the ratio of the longest job to the shortest job and κ_{min} as the minimum slack ratio required for the admission of the job. Considering that the longest job J_h has been released at time t=0, the queue will start scheduling the job on that machine. While executing J_h, algorithm A will not accept any jobs that expire within the interval $< 0, d_h >$, where d_h is the deadline for job J_h. However, OPT, with complete prior knowledge of the input job sequence J, will be able to accommodate jobs within the slack of job J_h. From the above discussion, under worst case circumstances, the total gain of A is p_h, while the possible gain of the algorithm OPT is $2 \times p_h - \kappa_{min} \times p_l$, where $\Delta = \frac{p_h}{p_l}$. This derives from the fact that, if any job is released during the interval $(p_h - \kappa_{min} \times p_l - \epsilon,\ p_h]$, algorithm A will be able to accept the job. Hence the competitive ratio for the online non-preemptive case is given as:

$$c = \frac{p_h}{2 \times p_h - \kappa_{min} \times p_l} \tag{3.3}$$

$$c = \frac{\Delta}{2 \times \Delta - \kappa_{min}} \tag{3.4}$$

Arbitrary allocation in case of a tie A processes the sequence of jobs in their order of arrival. The incoming jobs are either immediately assigned to a queue or rejected. At any given instance these jobs need to be bin packed into the available queues. The order of packing determines the maximum moment available to the jobs in each of the queues. Consequently, the scheduler may either assign the jobs arbitrarily to a queue amongst the set of queues on which the job is feasible or it can break the tie by use of an allocation strategy. First-Fit, End-Fit and Best-Fit are the three most common strategies employed in the domain of bin-packing. Amongst the three possible strategies, the Best-Fit Strategy retains maximum moment between the allocation of the jobs, and therefore achieves the highest packing density amongst the three, and was chosen to implement online scheduling.

Inadmissable Job Slack Lipton and Tomkins (Lipton and Tomkins 1994) introduced the concept of interval scheduling under minimum slack requirements. However, their (and the subsequent results in the field) impose no restrictions on maximum admissible slack. This is primarily attributed to the fact that most of these algorithms assume that resources will be available throughout the life-time of the algorithm. However, practical operating scenarios, as described in the next chapter, require the imposition of maximum slack. Hence, in cases where J consists of the jobs with a slack ratio greater than κ_{max}, A is bound to reject the jobs and hence perform poorly. As the two classes of algorithms differ significantly, in such cases, the competitive ratio is indeterminable.

3.6 Summary

The motivation for computational resource sharing is described in Appendix B. This chapter has proposed a queue based model to capture interval based resource scheduling in Grids and similar environments. A best-fit online interval scheduling algorithm for non-preemptive jobs was described and analyzed. The performance evaluation of the algorithm can be found in the following chapter.

```
EXFMNotify()
input  : Queue of incoming jobs A, Set of Machines $M_k$
output: An online schedule S

event job arrival invokes routine ProcessJob($job$)
ProcessJob (job)
begin
if $A = \emptyset$ then
    Accept job and schedule on any random machine
end
else
    $M_p \leftarrow$isFeasible($job, M_k$);
    if $M_p = \emptyset$ then Reject job ;
    else BestFit($M_p$,$job$)
end
end

event Continuous processing at each machine $M_i$ is the routine ExecJob($Q_i$)
ExecJob($Q_i$)
begin
Let $J_k \in Q_i$ be the job with earliest deadline.
$currentjob \leftarrow J_k$;
$nextidle \leftarrow currenttime + p_k$;
$Q_i \leftarrow Q_i - \{J_k\}$;
Allocate Resources to $J_k$;
if $currenttime = nextidle$ then
    if $Q_i \neq \emptyset$ then  ExecJob($Q_i$);
end
end

isFeasible($J_h, M_k$)
begin
for $i \leftarrow 1$ to $|M_k|$ do
    Order all jobs of $Q_k$ by non-decreasing deadlines $J_{1k}, J_{2k}, J_{3k}, J_{4k}, ...J_{qk}$;
    Calculate $index \leftarrow$ possible location of the job.;
    if $e_h >$ PreviousCommitments(index) then
        continue
    end
    Calculate $availableSlot \leftarrow$
    PreviousCommitments($index$)-FutureCommitments($index$);
    if $p_h \leq availableSlot$ then
        $M_p \leftarrow M_p + M_k$
    end
end
end
```

Figure 3.2: Online resource allocation with admission control to provide notifications at release time

Chapter 4

Evaluation of the Online Scheduling Algorithm

The first section of this chapter presents the experimental evaluation of the interval scheduling with admission control. It uses the standard simulation techniques used for evaluating scheduling systems (Kiran 1998) and compares the performance of the algorithm for variable slack ratio, job length, arrival rate and resource failure rates. The performance of the algorithm is compared against the EXF (Earliest eXpiry First) algorithm - an online scheduling algorithm with the best known competitive ratio. Section 4.2 describes the application of the above work in the context of an online Grid scheduling system, and the subsequent section summarizes the work on online scheduling systems.

4.1 Experimental settings

Experiments were conducted using a single source of job sequence that generates jobs with a specified probability distribution for job length, rate of job arrival and resource failure rates. Table 4.1 below summarizes the parameter values used for the experimental evaluation.

4.1.1 Job generator

A job generator had been devised for generating a sequence of jobs used to analyze the performance of the scheduling algorithm. The job generator creates instances of jobs with the desired characteristics of execution time and slack. Each job instance has three parameters: first, *release time*, r_i - is determined in accordance to the stochastic distribution of job arrival; second, the *processing time*, p_i - is determined in accordance

Parameter Description	Parameter Value
Simulation interval	10000 jobs
Number of nodes used for simulation	100
Maximum Job length	1000
Job distribution type	Poisson distribution, zip distribution
Job arrival rate	Poisson distribution
Slack constant	Constant for some experiments, varied from 1.1 to 12.0
Failure rate	Gaussian distribution
Mean time to failure	0.75 of maximum job length

TABLE 4.1: Simulation settings for evaluation of online scheduling algorithm.

to the job length distribution. Candidate distributions include unit job lengths, uniform job length, poisson distribution and zipf distribution. Thirdly, the *deadline* of the job is calculated in accordance to the slack, where deadline of a job $d_i = r_i + (\kappa + 1) \times p_i$. A finite sequence of jobs with the desired job and job arrival characteristics is generated and is evaluated using an off-line scheduling algorithm, Earliest eXpiry First (EXF) and the online-interval scheduling.

Only integer job lengths were considered for simulation. Integer job lengths were considered for the relative ease of scheduling the jobs against a virtual time clock, as it uses integer incremental time steps. Unit and uniform job lengths represent a very specific case of workload observed in certain web server workloads, while the zipF distribution represents the job lengths observed in super computing center workloads. The Poisson distribution represents a generic distribution of random workload observed in batch jobs. For job arrival distribution,Poisson distribution was considered.

4.1.2 Scope of the evaluation

The purpose of the evaluation is:

- To determine the effectiveness of the scheduling algorithm vis-a-vis the performance of an off line scheduling algorithm, the EXF algorithm and the interval scheduling algorithm with no notification.
- To evaluate the performance of the algorithm under varying load conditions.
- To compare the overhead associated with the admission control mechanism.

4.1.3 Analysis

The competitive ratio for the various algorithms is presented in figure 4.2. The figure represents the relative performances of BFIS, EXF and the offline algorithm for a set

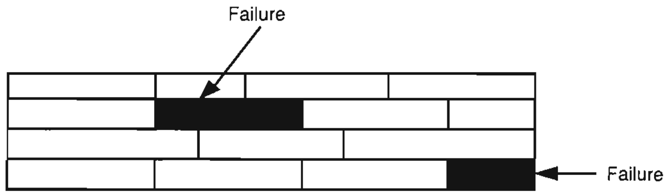
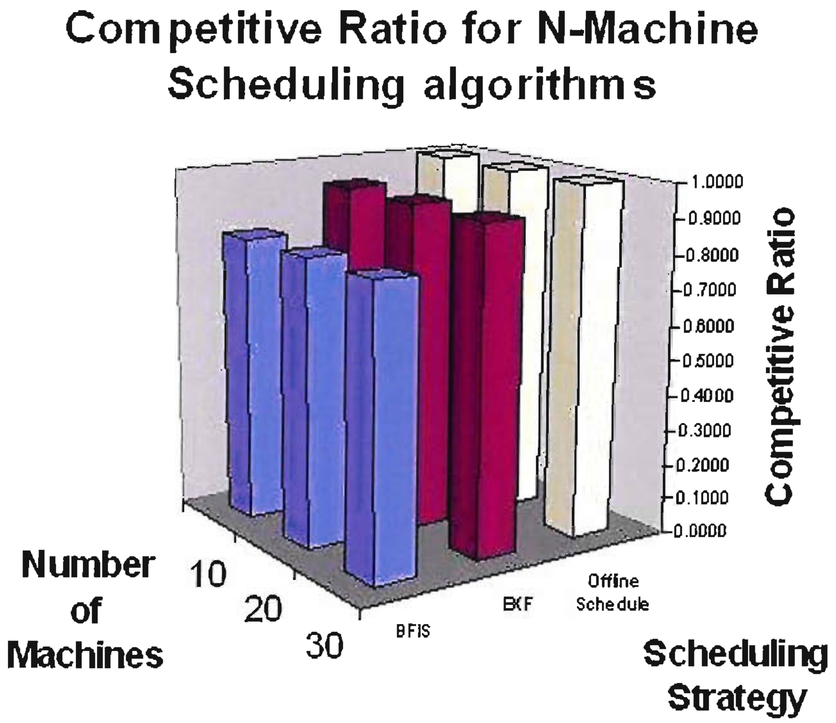


FIGURE 4.1: Simulating resource scheduling with failures.



	10	20	30
🟦 BFIS	0.8177	0.8201	0.8166
🟥 EXF	0.9362	0.9336	0.9308
□ Offline Schedule	1	1	1

FIGURE 4.2: Comparison of BFIS, EXF and off-line scheduling strategy.

of 10, 20 and 30 machines. The figure represents the average competitive ratio for K > 1, randomly varying between the values of 1<k<10; the maximum bin size for each machine was κ_{max} *maximum job length*, for the case of BFIS. For EXF, the job queue was sorted and the EXF was allowed to choose the longest job. The input job sequence had at least one feasible schedule that would bin pack all the queues. It is assumed that any off-line algorithm is able to detect the existence of one such ideal schedule. The competitive ratios of BFIS and EXF demonstrate that the performance of both of these algorithms is unaffected by the number of machines and is relatively resilient to variations in slack factor. With a competitive ratio of approximately 0.8, the additional overhead in BFIS (approximately 10 percent) is introduced by admission control.

4.1.4 Discussion

As NOWS assumes prominence, both the resource providers and consumers will require mechanisms for managing the agreements for some a fore-mentioned finite horizon. The BFIS algorithm presented in this section presents one such mechanism, which allows the resource providers and consumers to dynamically reserve resources for an incoming online sequence of jobs. Unlike other online scheduling algorithms, BFIS uses the concept of finite horizon, and, unlike in random breaking of tie the best fit ensures that the queues are filled to provide even distribution of jobs for the horizon dictated by each of the incoming jobs. The approach is similar to online bin packing for a dynamic horizon. The load distribution allows the scheduler to optimize the use of selected resources and minimize the resource usage of others. Such segregation of resources based on job sequence characteristics allows the scheduler to dynamically determine the optimal number of resources required for the particular job distribution. The next section describes an online scheduling system for Grid systems as a probable use case for BFIS.

4.2 Description of Information exchange between Grid schedulers

Traditional resource management is *commonly* used to describe all aspects of the process of locating various types of capabilities, arranging their use, utilizing them, monitoring their state and providing traceable evidence/audit of their usage. As described in Appendix B, Grid Systems (Foster and Kesselmann 1999) represent an emerging class of systems that assume dynamic operating environments, which facilitate coordinated use of distributed resources. Appendix B introduces computational Grids - a class of Grid systems that allows coordinated use of computational resources. It is possible to conceive a computational Grids existent under a single administrative domain and with a centralized resource management system. However, in most cases, the a Grid based resource management system will operate over a set of unreliable resources spread across multiple administrative domains.

Grid Schedulers, as described in (Nabrzyski, M., and Jan 2004), represent types of Grid resource managers that, by their very definition, are involved with managing resources across multiple administrative domains. Grid Scheduling can be applied to many types of resources: a machine, disk space, a QoS network and so forth, although the rather generic definition usually applies to the management of computational resources. In (Nabrzyski, M., and Jan 2004) Nabrzyski et.al. describe the structure of a generic Grid scheduling system and introduce the concept of hierarchical organisation of local and higher level schedulers. Hierarchies of such schedulers use different job allocation techniques to coordinate the allocation of resources. The scheduling systems try to optimize the usage of resources in accordance with their respective objective functions. (Nabrzyski, M., and Jan 2004) introduced guaranteed completion time of allocations as one of the objective functions used in Grid scheduling systems. The interval scheduling algorithm described in the previous chapter was conceived to operate under such Grid scheduling systems. The queue based model of the algorithm allows it to map resource allocations across multiple resource providers and can also be adapted (by changing the feasibility test described in the algorithm) for use with autonomous queues.

The interval scheduling algorithm can be used to manage resources within the context of a single resource manager. As a hierarchical Grid scheduler is reliant on the participation of local schedulers, which in turn autonomously derive their objective functions, it is imperative that local schedulers dynamically collaborate or cease to collaborate with the schedulers at a higher level. In most cases, these changes are in response to changes in resource and load characteristics. Information on such changes needs to be exchanged between the instances of the Grid scheduling systems. Scheduling systems may use job brokerage or resource brokerage as a means of resource management across multiple scheduling systems. ClassAds (Litzkow, Livny, and M.W.Mukta 1990) used in CONDOR represent one such system of job brokerage. Information exchange between scheduling systems may happen along the established hierarchical topological order as used in Globus Information Services. However, if one considers the relaxed model of peer-to-peer systems, information exchange may influence the choice of topological ordering. The next part describes the techniques for the creation of such overlay networks.

4.3 Summary

This part of the work (Part II) introduced and evaluated an interval based scheduling system for dynamic environments. It was proven that online interval scheduling can be used to provide guaranteed resource availability for computational resource allocations. The queue based model takes into account the intermittent availability of resources and creates an adaptive schedule with a bounded competitive ratio. Subsequently this chapter discussed a way of employing the above algorithm in the context of the Grid scheduling system. The autonomic organisation capability can be sustained by permitting information exchange between Grid scheduler instances. Topological organisation of the nodes is discussed in Part III, which introduces a mechanism to design a semi-structured overlay network between resource providers.

Part III

Information Dissemination in peer-to-peer systems

Chapter 5

Resource Management in P2P environments

The previous chapter, presented a use case of an online scheduling system that involved potential collaboration between multiple resource providers. The multiple resource providers used resource advertisements as a means of communicating resource information between providers and consumers. The scenario presented in section 4.2 served as motivation to investigate the issues of resource discovery in systems composed of autonomous resource providers (referred to as peers when set in a P2P architecture). The scenario 4.2 considered one variant of resource discovery under a peer-to-peer (P2P) (Clark 2001) system environment.

This part presents a framework that facilitates resource discovery in P2P systems. The discussion spans three chapters. This chapter reviews discovery techniques used in a peer-to-peer environment, elaborates the application scenarios and presents a generic model for overlay construction and management in peer-to-peer(P2P) systems. This is followed by the detailed description of the algorithm in Chapter 6 and an experimental evaluation described in Chapter 7. The primary contributions of these chapters towards the thesis are that they:

1. Outline the evolution of resource discovery in peer-to-peer systems and describe their limitations when applied to the application domains of mobile services and Open Hypermedia Systems;
2. Provide an algorithm for creation and maintenance of an adaptive overlay;
3. Present an experimental evaluation of the same.

Section 5.1 presents a general overview of P2P systems. Section 5.2 describes existing search techniques and highlights their limitations. Motivating applications are the subject of section 5.5, while resource definitions and search techniques are described in section 5.6. This is followed by a description of a generic framework for the creation and maintenance of an overlay network, in section 5.8. Section 5.7 discusses the characteristics of the proposed overlay.

5.1 Peer-to-Peer computing

There is no definitive description of peer-to-peer systems (Oram 2001), but the system characteristics are often used to describe this class of systems. P2P systems are typically characterized by decentralisation of control, where each node plays the parts of the client as well as server, often leading to the creation of ad hoc communities of collaborating peers. This is exemplified by applications such as Napster and has also been widely adopted by file sharing applications such as Gnutella (GNUTELLA), Freenet(Clarke, Sandberg, Wiley, and Hong 2001), and OceanStore (Kubiatowicz, Bindel, Chen, Eaton, Geels, Gummadi, Rhea, Weatherspoon, Weimer, Wells, and Zhao 2000). Most P2P file sharing systems are classified as unstructured P2P systems; their topology evolves as peers join in or leave the network. A large body of work has focused on developing structured P2P computing networks. Examples include Tapestry(Zhao, Kubiatowicz, and Joseph 2001), Chord (Stoica, Morris, Liben-Nowell, Karger, Kaashoek, Dabek, and Balakrishnan 2003) and CAN (Ratnasamy, Francis, Handley, Karp, and Schenker 2001).

The P2P approach of creating ad hoc networks of collaborative peers has been applied to various application domains, including, large scale distributed computing as Grids (Buyya, Abramson, and Giddy 2001; SETI), file sharing (Clarke, Sandberg, Wiley, and Hong 2001; GNUTELLA ; Kubiatowicz, Bindel, Chen, Eaton, Geels, Gummadi, Rhea, Weatherspoon, Weimer, Wells, and Zhao 2000), and service oriented computing platforms such as JXTA (Qu and Nejdl 2001). The following section provides a brief overview of both structured and unstructured P2P systems.

5.2 P2P systems

Resource discovery techniques are central to both structured and unstructured P2P systems. While the unstructured P2P systems utilize some sort of heuristics to guide the search, the structured P2P systems use the properties of the overlay to selectively propagate the search query to locate appropriate resources. The following section reviews the search techniques for systems belonging to each of these two P2P system classes.

5.2.1 Unstructured P2P systems

Freenet Freenet is a P2P file storage system in which peers share their available disk space to create an internet-scale virtual file system. Each of the participating peers is required to provide some storage space; in return Freenet provides the user with a secure means of storing their files on the virtual file system. To add a new file the user provides the file and a location-independent, globally unique identifier (GUID) (also known as keys). A file addition results in the file being replicated and stored at a number of locations.

In the Freenet system, every node maintains a routing table that lists the addresses of the other nodes and the list of the keys it thinks that they hold. On receipt of a query, the node finds its own store; and returns the file if it is found in the local store, otherwise it forwards the request to the node with the numerically most proximal key to the one requested. To prevent flooding of the network, Freenet mandates that each query be associated with a Time To Live (TTL). In addition, Freenet maintains the search paths for previous queries to train the routing table sets. These trained sets are used to cluster the files with similar keys on the same data store. The simulation studies on Freenet show that the path length grows approximately logarithmically to network size.

Gnutella Gnutella is a file sharing application and relies on participant peers to form an unstructured overlay network. A peer can join the Gnutella network by contacting one of the participant peers, which leads to subsequent overlay formation amongst the other participant peers. Once attached to the network, each of the Gnutella nodes processes the incoming query requests. Early Gnutella algorithm relied on broadcasts to propagate queries between neighbouring nodes. The range of the query broadcast is restricted by the TTL associated with each of the query requests. Subsequent changes to the algorithm have been proposed by (Lv, Cao, Cohen, Li, and Shenker 2002), which substitute flooding by a set of random walkers. The simulation based study described in (Lv, Cao, Cohen, Li, and Shenker 2002) demonstrates marked improvement in performance. However, subsequent studies (Ritter 2001)have demonstrated that the Gnutella approach remains non-scalable.

5.2.2 Structured P2P systems

Structured P2P techniques impose a set of topological constraints on the construction of the overlay network. Most of the structural constraints were designed to organize the overlay and facilitate efficient search algorithms. For example, CAN and Chord partition the search space into a distributed hash structure, while Tapestry utilizes specific naming mechanisms to create an overlay. The following is a description of three structured P2P systems.

Content Area Network (CAN): Content Area Network (CAN) (Ratnasamy, Francis, Handley, Karp, and Schenker 2001) uses an internet scale hash table maintenance technique to uniquely map a "key" onto "values". Central to the algorithm is the creation and maintenance of a d-dimensional co-ordinate space that allows hash table equivalent functionalities such as insert, deletion and look-up of key value pairs. The *d-dimensional* co-ordinate space is just a logical representation, and nodes can be dynamically repartitioned amongst all the nodes at any time.

A typical CAN network consists of many nodes, each storing a part of the hash-table, known as a "zone", in addition to information about some adjacent zones. Each node belongs to a unique zone and is neighbour to nodes that overlap with it in at least *d-1* dimensions. Each node maintains the state of its neighbours, with the maximum number of neighbors limited to *2d*. A typical query specifies the destination co-ordinates. The query routing mechanism of each node uses the neighbor state to route the query along the d-dimensional space, to a node that is the closest in the d-dimensional space. For a d-dimensional space with n-equal partitions the average path length is $(d/4)(n^{1/d})$, where the path length grows by $O(n^{1/d})$ by addition of the node.

CAN provides reliable mechanisms to recover from failed nodes. When a node leaves CAN, it explicitly hands over the zone to one of its neighbors, which thereafter maintains both zones. However, failure to communicate zone and neighbor information with immediate neighbors is considered as a failure. A neighbor detecting a failure starts the takeover mechanism. The first neighbor to successfully complete the takeover mechanism informs neighbours about the completion of takeover to all the neighbours of failed nodes.

Chord: Chord is a P2P protocol that results in the formation of a structured overlay. The chord protocol is specifically designed to uniquely map the key to nodes. It uses consistent hashing to allocate keys to nodes and arranges the space of an *m-bit* identifier into a circle of modulo 2^m identifiers. In steady state each of the nodes in chord maintains state about O(log(N)) nodes in a N-node system. The chord maintains information about neighboring nodes as they join and leave the system, with a very high probability that the reorganisation results in no more than $O(log^2(N))$ messages. Central to chord is the concept of creating a circular identifier space, where each node stores the information about the identifiers between itself and the next node in a clockwise direction.

Each of the chord nodes maintains information about approximately $O(log(N))$ neighbors and uses collaborative replication to improve the resilience of the P2P network. Each of the participant nodes is identified by the key obtained by hashing its IP address. Each node stores information about the keys located between its identifier and that of its immediate neighbor. Queries in Chord are processed by passing them around to successive nodes with identifiers lower than the identifier being queried.

Tapestry: Tapestry is an application level P2P protocol that extends the unique naming scheme introduced by Plaxton trees (Plaxton, Rajaraman, Andr, and Richa 1997). The overlay organizes the 2^n nodes into l levels, and nodes in each level maintain their respective neighbor maps. In an l level tree, each node is identified by an $w=n/l$ bit identifier. A node with a label, say xyz, where x,y and z are the bit digits, will have a routing table with

1. 2^w entries of [*,X,X]
2. 2^w entries of [x,*,X]
3. 2^w entries of [x,y,*]

where * denotes every digit in 0,1,..., $2^w - 1$, and X denotes any digit in 0,1,..., $2^w - 1$.

Using the above routing state, a packet is forwarded towards the destination label node by incrementally resolving the destination label from left to right. Each node forwards the packet to a neighbor whose label matches the destination label in one more digit than its own label. The average path length for a network of n nodes is O(log(n)), and requires the system to maintain a state of O(log(n)) neighbors.

5.3 Related algorithms and systems

Similar resource discovery problems have also been identified in related research areas. These involve maintenance of state tables as summarized below:

5.3.1 Distance Vector and Link State based algorithms as applied to ad hoc computing

Distance Vector based and Link State based algorithms have been widely used for IP routing in ad hoc computing environments. Examples include DSDV (Perkins and Bhagwat 1994), AODV (Perkins and Royer 1999), ZRP (Haas and Pearlman 1998) amongst others. These algorithms maintains partial knowledge of system topology in order to route messages between mobile nodes in an ad hoc communication network. The algorithms have a striking resemblance to structured P2P networks: both system types maintain partial routing tables in order to propagate queries to their final destination.

5.3.2 Domain Name System (DNS)

A domain name system maintains a table that maps a unique name to its IP address, and, in a sense provides a functional capability similar to that of Distributed Hash

Techniques. For any group of computers partaking in the DNS naming scheme there is likely to be a single definitive list of DNS names and associated IP addresses. The group of computers included in this list is called a zone. A zone could be a top level national domain or a university department. Within a zone, a DNS service for subsidiary zones may be delegated along with a subsidiary domain. The computer that maintains the master list for a zone is said to have authority for that zone and will be the primary name server for it; there will also be secondaries for that zone. The DNS server may be able to resolve the request for name from its own local database/cache. If a DNS server is unable to resolve the key, it solicits help from another DNS server higher up in the hierarchy or one in the destination zone.

5.3.3 Coalition formation in Agent-based systems

Coalition formation is an area of active research in the field of agent-based systems (Wooldridge and Jennings 1995) and deals with the design of mechanisms where a number of independent agents come together to act as a collective entity. Coalition formation has been studied in the context of multi-agent systems and has been applied to various fields, such as e-commerce (Tsvetovat and Sycara 2000) and Grid computing (Foster and Kesselmann 1999). Coalition formation is based on the notion that the collaborating agents are better off acting collectively rather than individually in a multi-agent system. The coalition formation in multi-agent systems can be viewed as being composed of three main activities (Sandholm, Larson, Andersson, Shehory, and Tohm 1999), as follows:

Coalition structure generation Coalition structure describes the sets of collaborating agents and determines the scope of interactions among coalitions. The process usually involves partitioning the group of agents into smaller groups of collaborating agents, such that the partitioning results in an exhaustive and disjointed set of coalitions [1]. For a set n agents $\{p_1, p_2, p_3, ...p_n\}$, there exist $2^n - 1$ possible coalitions and $2 \cdot n - 1$ coalition structures. For example, for a set of three agents $\{p_1, p_2, p_3\}$, there exist $\{p_1\}$,$\{p_2\}$,$\{p_3\}$,$\{p_1, p_2\}$,$\{p_1, p_3\}$,$\{p_2, p_3\}$,$\{p_1, p_2, p_3\}$ coalitions and $\{\{p_1\}, \{p_2, p_3\}\}$,$\{\{p_1, p_2\}, \{p_3\}\}$,$\{\{p_1, p_3\}, \{p_2\}\}$, $\{\{p_1, p_2, p_3\}\}$ and $\{\{p_1\}, \{p_2\}, \{p_3\}\}$ disjointed coalition structures.

Optimizing the value of individual coalitions The coalition structure can be optimized to maximize a certain objective function, also known as the coalition value. Each agent in the coalition pools its resources and associated tasks to maximize the coalition value. An overall coalition structure is designed to maximize the cumulative coalition values of the entire structure.

[1] Some research also considers the case of non-disjointed coalitions, where agents can simultaneously belong to more than one coalition

Pay-off distribution All agents in the coalition pool their resources to maximize the coalition value function in expectation of a certain payoff. The payoff can be equally or proportionally divided amongst the members of the coalition. Payoff distributions are common in agent-based e-commerce applications, where each autonomous agent enters a coalition to maximize utility function in the process of maximizing coalition value.

In this study coalition formation algorithms are classified into two distinct categories, payoff maximizing coalitions and coalitions 'value maximisation'. The classification derives from the two prevalent approaches used in agent based systems, namely competing agents and collaborating agents. In the former case of pay-off maximisation, each agent enters a coalition with a sole purpose of maximizing its utility, and the coalition structure formation assumes a lesser priority. However, in coalition value maximisation, the collective utility of all the agents in a multi-agent system supersedes the pay-off distribution objective. In such cases the coalition structure is used to measure the good of the coalition formation process.

This research is interested in systems in the coalition structure formation process and its application to the formation of overlay topologies in peer-to-peer networks. Previous research in the field has focused on the formation of super-additive coalitions (Kahan and Rapoport 1994), in which any two coalitions are better off by merging together. However, super-additive coalitions are not appropriate for the coalition structure generation, as a grand coalition comprising all the agents will be the most appropriate coalition structure. Hence, the current focus is on exploring coalition structure formation for non-super-additive environments. Coalition formation for non-super additive environments has been considered by (Sandholm, Larson, Andersson, Shehory, and Tohm 1999; Dang and Jennings 2004) who suggest algorithms and provide the worst case bounds for the creation of coalition structures for multi-agent systems. The systems consider multi-agent environments with a static set of agents and a fixed coalition value. Recently, attention has been paid to more dynamic environments, where the coalition values are not fixed and agents constantly join or leave a coalition (Klusch and Gerber 2002). However, no performance bounds have been provided to represent the complexity of coalition formation.

5.4 Discussion

The simplistic approach adopted by unstructured P2P computing systems leads to higher average query path lengths and often results in unnecessary broadcasts of messages and utilisation of network resources. Use of structured P2P systems provides bounds on message path lengths and the amount of state held by the node. It should be noted that the creation of a structured P2P overlay does not take into account physical network characteristics and results in longer than actual query paths. Certain structured P2P overlays like Chord are also susceptible to uniform workloads resulting in over-utilisation of certain peers.

Neither of the system types takes into account the characteristics or the specific requirements of the application domain and they represent the extreme ends of a spectrum. While the unstructured by its very nature does not impose an overlay structure, the structured overlay seems to be too inflexible to be generically applicable. The next chapter introduces the mechanism for the creation of an adaptive overlay that attempts to overcome the limitations of the above two approaches. The following section introduces certain application domains which were used as exemplars to verify the approach and are presented here to highlight the specific requirements of the specific application domains.

5.5 Additional application scenarios

In addition to the application scenario described in section 4.2, the work is also applied to the domain of P2P based Open Hypermedia Systems and is equally applicable in collaborative service discovery in mobile environments. The initial work was carried out in the context of Open Hypermedia Systems and was published in (Zhou, Dialani, De Roure, and Hall 2003). This section provides a brief overview of the two application domains and introduces resource definitions and the search criteria used in the field.

5.5.1 Peer-to-Peer Open Hyper Media Systems

Open Hypermedia (Wiil 1997) is a model that has been adopted by the hypertext community for many years. It is principally characterized as having hypermedia link information stored separately from the documents that it describes. The links are stored in linkbases. This approach allows links to be managed and maintained separately from the documents, and different sets of links can be applied to a set of documents, as appropriate.

The development of the first Open Hypermedia System ("Microcosm") (Fountain, Hall, Heath, and Davis 1990)) predates the Web. The first implementation of the Microcosm philosophy on the Web was the Distributed Link Service (DLS) (Carr, De Roure, Hall, and Hill 1995), (De Roure, Walker, and Carr 2000). This was extended so that link resolution was also distributed around the Web (De Roure, Carr, Hall, and Hill 1996), and the service paradigm now extends to recent developments, such as ontology services (Carr, Hall, Bechhofer, and Goble 2001). COHSE (Carr, Hall, Bechhofer, and Goble 2001) provides tools for the Semantic Web that builds upon the concept of the DLS and ontologies.

The Semantic Web (Berners-Lee, Hendler, and Lassila 2001) augments current Web technologies by associating machine understandable annotations (also known as metadata) with contents. Metadata provides an abstract representation of information and is primarily produced to facilitate inference techniques to co-relate information from different providers. Current search techniques used in Semantic Web technologies focus on annotating static information, but fail to take into consideration dynamic and asynchronous variation in content. Some may consider services based architectures like DAML-S (Ankolekar 2001), which use Semantic Web technologies, to be a form of dynamic content system. This research differs from (Sycara, Lu, Klusch, and Widoff 1999) and consider it to be an application of the Semantic Web to active entities rather than dynamic entities. In the proposed approach, the Semantic Web is considered to be dynamic, if it is created spontaneously by a set of collaborating nodes, where each node can dynamically update its published contents. While Semantic Web technologies are generic in their application, this scenario restricts their application to collaborative environments, which facilitate resource sharing between dynamic collections of participants. As the participant can act both as a resource provider and a resource consumer, a peer network is constituted by collaborating entities.

These collaborative P2P-OHS publish and consume resource descriptions usually expressed in RDF (Miller 2004) format. Summarized metadata information in a link base known as "topics vector" is advertised by each link base, and a list of similar topics is used to create an overlay that binds the participant peers in the peer-network. Each of the participating peer caches the "topic vectors" of its immediate neighbors and uses the informational inferences from these "topic vectors" to route the query amongst its neighbors. The search is expressed by means of an RDF query and is accomplished by propagating the query among a number of participating peers. A typical search expression is represented in section 5.6. Peers collaborate to maximize the number of link bases searched with minimal query routing and processing overheads.

The following subsection presents the last of the application scenarios and then summarizes the specific requirements of the application domain, and contrasting them with the current capability of the peer-to-peer networks discussed above.

5.5.2 Collaborative service discovery in Services Oriented Architecture

Service discovery mechanisms are crucial to service architectures such as web services and mobile services. Whether in a wired network environment or a wireless network environment, the service providers need to publicize service descriptions for subsequent discovery and utilisation by client applications. Discovery services such as UDDI (UDDI 2004), JINI (Kumaran and Kumaran 2001) and UPnP (Michael and Weast 2003) are widely used to support the discovery of services in wired as well as mobile network environments[2]. While wired network environments can utilize a centrally located discovery service such as UDDI, this is not the case for mobile environments. As no node has the resources to maintain the complete state of the ad hoc system, individual nodes should collaborate or form coalitions to discover resources in an ad hoc system.

(Chakraborty 2004), (Ratsimor, Chakraborty, Tolia, Khushraj, Kunjithapatham, Joshi, Finin, and Yesha 2002), and (Chakraborty, Joshi, Finin, and Yesha 2004) present a set of techniques and a framework for discovery and composition of services in ad hoc computing environments. The limited storage capacity of the mobile nodes limits, coupled with their mobility requirements introduces unique constraints on service discovery in such mobile environments. As described in (Chakraborty 2004), service descriptions and service compositions can be described in DAML-S. Each of the participating peers caches the service descriptions and facilitates the search on these cached advertisements. A service discovery request is expressed as a DAML-S search syntax and involves a complex search criteria. Examples of these are presented below: [3].

Our research groups initial work on service discovery (Miles, Papay, Dialani, Luck, Decker, Payne, and Moreau 2003a; Miles, Papay, Dialani, Luck, Decker, Payne, and Moreau 2003b) also highlights similar complexities in service discovery. A peer network consisting of such service providers needs to find an optimal way to disperse the service advertisements and adequate query routing mechanisms to locate mobile services.

5.6 Search requirements

As demonstrated by both the above systems, the search criteria tend to be much more complex, as compared to the standard identifier based search dictated by almost all the current P2P systems. The above examples use an RDF representation to specify the resource, and, correspondingly, the search criteria need to locate resources with similar advertisements. In P2P systems, each search translates into the location of a single unique identifier. However, most practical systems may be composed of resources that cannot be guaranteed to be unique and one can not rule out the existence of multiple resources with similar characteristics. Hence, this study proposes the use of a coalition based search mechanism to locate the resources of interest.

5.6.1 RDF representation of a query in P2P OHS

The search criteria for P2P-OHS were presented in a paper by (Zhou, Dialani, De Roure, and Hall 2003) and are repeated here as an example case (see figures 5.1 and 5.6.1). A typical linkbase contains a list of topics. In this example the link base is capable

[2] The definition of wired and mobile networks, is described in the MANET documents

[3] Includes the DAML-S service description and the DAML-S search criterion

```
<?xml version=‘‘1.0’’ encoding=‘‘UTF-8’’ ?> <rdf:RDF
xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
    xmlns:rdfs=http://www.w3.org/2000/01/rdf-schema#
    xmlns:lb=‘‘http://www.semanticweb.com/rdf/linkbase-ns#’’>
    <rdf:Description about=‘‘http://www.semanticweb.com/linkbase
          /research/linkbase.xml’’>
        <rdf:type resource=‘‘http:// www.semanticweb.com/rdf
          /linkbase-ns#Linkbase’’ />
        <lb:topic>theory</lb:topic>
    </rdf:Description>
</rdf:RDF>
```

Figure 5.1: A linkbase expressed in RDF Syntax, taken from our publication

```
<rdfq:rdfquery>
  <rdfq:From eachResource=‘‘http://www.semanticweb.com/
         collabrative_environment_x/peer_linkbase’’>
    <rdfq:Select>
      <rdfq:Condition>
        <rdfq:and>
          <rdfsq:sequal>
            <rdfq:Property name=‘‘lb:topic’’/>
            <rdf:String>Theory</rdf:String>
          </rdfsq:sequal>
          <rdfsq:sequal>
            <rdfq:Property name=‘‘lb:topic’’/>
            <rdf:String>Practice</rdf:String>
          </rdfsq:sequal>
        </rdfq:and>
        <rdfq:or>
          <rdfsq:sequal>
            <rdfq:Property name=‘‘lb:topic’’/>
            <rdf:String>Thoughts</rdf:String>
          </rdfsq:sequal>
        </rdfq:or>
      </rdfq:Condition>
    </rdfq:Select>
  </rdfq:From>
</rdfq:rdfquery>
```

Figure 5.2: A typical query specification, taken from our research group's publication

of providing information about a topic "theory" located within it. A typical query expression ss expressed in attempts at locating the link bases that provide information about such topics. For example the above query expresses interest in locating resources that provide the information either about the topics "Theory and Practice" or about "Theory and Thoughts".

The above search expression involves conjunctive and disjunctive operations and such complex query expressions cannot be translated into a simple key location expression, as mandated by structured P2P location techniques.

5.7 Discussion

From the above scenarios, it may be observed that:

1. It is not always possible to transform a search request into a unique identity location problem.
2. It is not always possible to cache the entire state of the neighbors and hence resilience cannot be guaranteed.
3. It is not always beneficial to form a collaboration without ascertaining the associated communication and processing costs.

The above reasons highlight the inadequacy of the existing techniques used to create P2P overlay systems. In addition, structured P2Ps mandate the number of neighbors and the amount of state information held by a peer. While structured overlay systems introduce highly efficient mechanisms for locating identities, they overlook the crucial application characteristics. Due to strong structural requirements, the structured overlay approach constrains the autonomy of the participating peers.

This research considers an alternative approach, which relaxes the stringent structural requirements of the structured P2P systems and allows collaborating peers to evolve an appropriate overlay topology that caters for the specific needs of the application domain. The approach described below considers each peer to be an autonomous entity independently determining the number of neighbours, and visibility of state information and suitability of its neighbors. However, to maintain certain structural properties, some constraints on the autonomy of the peers are introduced. The prescribed approach advocates the adaptive overlay formation described in the following sub-sections.

5.8 Adaptive overlay formation

An overlay structure captures topological information about interacting peers. The topological structure affects:

interactions between immediate neighbors An overlay structure determines the type of associations that a peer has with its neighbors. For example, in a P2P-OHS the probable neighbors should be peers that either have similar resource profiles or their shared resources are mutually beneficial to their neighbors. In this case, the structure of the overlay changes if either of the neighbors changes its resource specification. Thus, the criterion used to define the overlay influences the type of interactions between the participant peers.

constraints on communication mechanism between the immediate neighbors An overlay may or may not depict the characteristics of the underlying communications network.

the visibility of the state information between the systems An overlay is resilient to the arrival and departure of peers. In order to achieve this resilience the participating peers need to maintain adequate state information about their neighbors. The nature of this state information varies between application domains.

the co-ordination mechanisms The overlay creation and maintenance mechanisms determine the type of interactions required to allow the participation of new peers, to overcome the failure of peers and to handle the semantics of any updates about state information between the neighbors.

Most of the above requirements can be expressed as constraints on the behaviour of an individual peer in the peer network. As described earlier, each peer in the network has limited visibility of the entire system state and it uses this limited state visibility to autonomously decide its set of neighbors. The criterion used to prioritize amongst a number of neighbors is referred to as *relative utility*. The relative utility function determines whether two peers will form part of a coalition. A coalition can exist only if both peers derive mutual benefit from its existence. For example, in the case of P2P-OHS, a successful coalition can be formed with peers with similar resources. Both peers mutually benefit from successfully routing the query and the savings in communication costs justify the continual costs of maintaining the coalition.

It is considered that each peer autonomously evaluates the benefits derived from participating in a coalition. It also assumes that each of the participating peers have a complete knowledge of:

1. Its own resources.
2. Its existing coalition(s).
3. The resources offered by the potential coalition partner.

It should be noted that the scope of information visibility is restricted to the resources offered by the coalition partner and do not consider the cases where a coalition partner provides the knowledge of its existing coalitions, which is also referred as "derived coalition". Derived coalition formation requires managing and mapping dependencies between different coalitions and happens to be much more complex to model. Extending the current approach to include derived coalitions forms an important part of future investigations.

It should be noted that each individual peer can simultaneously participate in a number of multiple coalitions, where each of coalition results in acquiring a number of neighbors. The resultant union graph of these multiple coalitions results in the formation of the adaptive overlay network. The above mentioned approach was tested for the domain of P2P-OHS, details of which can be found in chapter 6.

The peer-to-peer overlay formation algorithm is an event driven ongoing algorithm based on the principle of the local constrained optimisation. As peers acquire greater visibility of the state information, they periodically optimize their set of neighbors, subject to local constraints on peers. Changes to the resource definitions of the peers are communicated to immediate neighbors and may lead to re-evaluation of the coalition between two peers. Variations in communication costs may also lead to re-evaluation of the coalition and result in the reorganisation of neighbours. A peer iterates through this communicate-optimize cycle indefinitely, after a random initialisation into the peer network. It is assumed that, before a re-optimisation, a peer is aware of the changes to the constraints and coalition variables. The overlay achieves its adaptivity from continual local re-optimisations. If the set of variables that underpin the coalitions stabilizes, it leads to the formation of a stable overlay. However, an unstable system results if the variables change more frequently then the re-optimisation capability of the peer network.

5.8.1 Formal description

Let G_t be the graph representing the overlay at any given time instance t. The graph $G_t = (P_t, E_t)$, where P_t is the set of peers in the peer network and E_t represents the sets of edges connecting the nodes in the overlay. The set $E_t = \{E_{12}, E_{13}, ...E_{ij}\}$ represents a set of edges at any given instance t. An edge E_{ij} exists between the two nodes P_i and P_j, if and only if, the nodes P_i and P_j are involved in a coalition. It should also be noted that graph G_t can only be constructed if a node has complete knowledge of all the other nodes in the network. However, in the present case, each peer P_i has partial knowledge of the graph represented as G_i. Refer to G_i as the visible state of the node P_i.

Each peer P_i has a set of resources R_i and is a member of coalitions C_i. Each coalition, $C(P_i, P_j, R_{ci}, R_{cj})$, such that $C \in C_i$, strictly consists of two participating peers, in this case P_i and P_j, where each participating peer contributes resources R_{ci} and R_{cj} respectively, such that $R_{ci} \in R_i$ and $R_{cj} \in R_j$. Each coalition partner is referred to as a neighbor, and each neighbor can only be a part of a single coalition.

As per the semantics of the coalition, peer P_i and peer P_j notify each other of any changes to the states of the resources R_{ci} and R_{cj} respectively. Each peer derives some utility from being a part of the coalition and constantly re-evaluates its participation in light of the other available coalition options. It should be noted that the maximum number of coalitions for peer P_i is restricted by the visibility G_i. However, the visibility of peer P_i may increase with the number of coalitions it is involved. For example, in the case of P2P-OHS, the visibility of the peer increases in the process of answering queries originated by non-neighboring peers.

Let $[t_i,t_j]$ be the last interval of observation of coalition between the participating peers. During this interval, each of the peers observes the state of the resources provided by the neighboring peer and re-evaluates the coalition to choose a set of partners from the set G_i. The coalition re-evaluation algorithm is presented in 5.8.1.

5.9 Summary

This chapter has reviewed the state-of-the-art of P2P systems. It was argued that the current techniques for developing P2P systems fail to fulfil the requirements of complex applications, such as service discovery, P2P-OHS and resource discovery in a network of schedulers. For these specific application scenarios, an adaptive overlay formation approach was proposed. The approach highlights the fact that each peer is completely autonomous and should be allowed to describe its policies for:

1. choice of neighbours
2. number of neighbouring nodes
3. visibility of the application state, and
4. its co-ordination mechanism.

The approach advocates the formation of coalitions amongst peers and allows for a peer to be a part of multiple coalitions. As each peer belongs to multiple coalitions, the cumulative effect leads to the creation of an adaptive overlay. The structure of this overlay changes, subject to the arrival and departure of peers and to temporal and spatial changes to the distribution of resource distributions.

The next chapter presents one such prototypical system used to validate the approach advocated in this chapter.

```
input : Resources advertised by the peer - $R_i$
        A set of existing coalitions - $C$
        A visibility graph to choose the probable neighbors from $G_i$
        Number of coalitions allowed $k$
        Utility of Coalitions - H, valid for time $[t_i, t_j)$
        A utility function to compare two probable coalitions
$CompareCoalition(H, C_c, R_i, P_j)$ , returns a comparable value for selection of a
best probable neighbor, given existing coalitions $C_c$ and the availability of
resources $R_i$, provided by this peer.
output: Modified list of Coalitions, $C$
        New Visibility Graph $G_i$
$ListofPeers \leftarrow$ ListofNodes($G_i$);
$ListofPeers \leftarrow ListofPeers +$ GetCoalitionPartners($C$) /*ListofPeers
contains non duplicate entries of known peers */;
$no\_of\_peers \leftarrow |Listofpeers|$;
if $no\_of\_peers < k$ then
    $ListofPeers \leftarrow ListofPeers +$ Broadcast
end
$C \leftarrow null$;
while $|C| \leq k$ do
    $temp \leftarrow null$;
    $next \leftarrow null$;
    for $j \leftarrow 1$ to $no\_of\_peers - |C|$ do
        $P_j \leftarrow ListofPeers[j]$;
        if $temp \leq CompareCoalition(H, C, R_i, P_j)$ then
            $temp \leftarrow$ CompareCoalition($H, C, R_i, P_j$);
            /* Localized Hill Climbing with successive maximisation based on
            local view,
            refer to discussion for constraints on CompareCoalition function */;
            $next \leftarrow P_j$;
        end
    end
    $C \leftarrow C + next$;
    $ListofPeers \leftarrow ListofPeers - next$;
end
$G_i \leftarrow$ CreateVisbilityGraph($C$);
return $C, G_i$
```

Figure 5.3: Continual reorganisation through coalition re-evaluation

Chapter 6

P2P Coalition Formation and Search Algorithm

This chapter extends the P2P overlay creation mechanism described in the previous chapter and describes its application in the context of a P2P Open Hypermedia System. Though the work is described in the context of a P2P-OHS it is equally applicable to all the previously described scenarios. This chapter is organized as follows: Section 6.1 describes the overlay creation process and introduces the software architecture of a peer node. Section 6.2 provides the formal description of the relevant data structures and query semantics. Section 6.3 describes the operations on the data structures introduced in section 6.2. The search algorithm is described in section 6.4 and a formal description of overlay reorganisation is given in section 6.5, which considers reorganisation under failure of nodes and change in resources provided by peers. A few of observations are made and some negative results are discussed in Section 6.6.

6.1 Introduction

This section describes the application of the generic algorithm described in section 5.8 to the domain of P2P-OHS, of which our motivating application was described in section 5.5. As described in section 5.5, a P2P-OHS consists of a number of peers that host link bases, which have been annotated in accordance to an application specific domain ontology [1]. Annotated resource descriptions, also known as "topics", represent the resources provided by each peer. In terms of the notation described in section 5.8, where each peer P_i hosts a set of resources R_i, the list of topics is referred to as R_i. Each peer enters a number of coalitions to maximize the discovery of its resources. The

[1] Domain ontology refers to the application domain ontology, for example ontology in the biomedical domain

TABLE 6.1: Choosing neighbours for coalitions formation

$R_i = \{T_1, T_5, T_7\}$			
Resource Type	Overlap	Degree of Overlap	Additional resources
$R_1 = \{T_1, T_2, T_3, T_4\}$	$\{T_1\}$	1	$\{T_2, T_3, T_4\}$
$R_2 = \{T_3, T_4, T_5\}$	$\{T_5\}$	1	$\{T_3, T_4\}$
$R_3 = \{T_1, T_3, T_5, T_7\}$	$\{T_1, T_5, T_7\}$	3	$\{T_3\}$

relative utility of the probable coalition partners are selected to reduce the cumulative communication costs incurred in routing the discovery queries across the peer network.

The cumulative routing costs are minimized using a simple cluster heuristic to create coalition among peers with similar resources. Peers form coalitions with peers with high degrees of overlap. For example, consider a peer P_i, with a possibility to enter a coalition with peers P_j, P_k. The tie will be broken in favour of P_j, if $|R_{ci} \bigcap R_{cj}| > |R_{ci} \bigcap R_{ck}|$; otherwise the tie is broken in favour of peer P_k. The query routing mechanism uses this overlap information to route the queries to the resources with the similar resources, please refer to section 6.2.1 for further details on query routing. The query throughput is maximized and the communication costs for query routing are restricted to communication costs with similar peers. However, in certain cases, peer P_i may not be able to decide on suitable candidates to whom to route the query, and in such cases resort to local broadcasts to neighbours.

For example, figure 6.1(A) represents a probable outcome of the clustering process. In this figure, the red dots and edges represent the peers that are part of a coalition on a particular resource. Once a query is routed to any node in these subgraphs the query is routed via the edges of the graph. However, to communicate between the sub-graphs, the nodes may resort to local broadcasts. The query costs can be reduced to a theoretical minimum if the all peers with a particular resource are clustered to form a connected graph, a case depicted in figure 6.1(B).

At the time of entering future collaborations, each peer has a complete knowledge of its existing collaborations, as described in section 5.8. Figure 6.2, represents one such case, in which a peer needs to choose a maximum of two coalitions from a set of existing coalitions. If the resources held by peer P_i is $R_i = \{T_1, T_5, T_7\}$, the overlap calculations are as in table 6.1. It should be noted that if the peer chose candidates without prior knowledge of existing coalitions, it would end up choosing both the resources of type R_3. However, after having chosen the first resource of type R_3, the peer attempts to maximize the resource types and chooses a resource of type R_1 over a resource of type R_2, maximizing the total resources known to peer P_i. It should be stressed that peer P_i does not know about the existing coalitions of other peers.

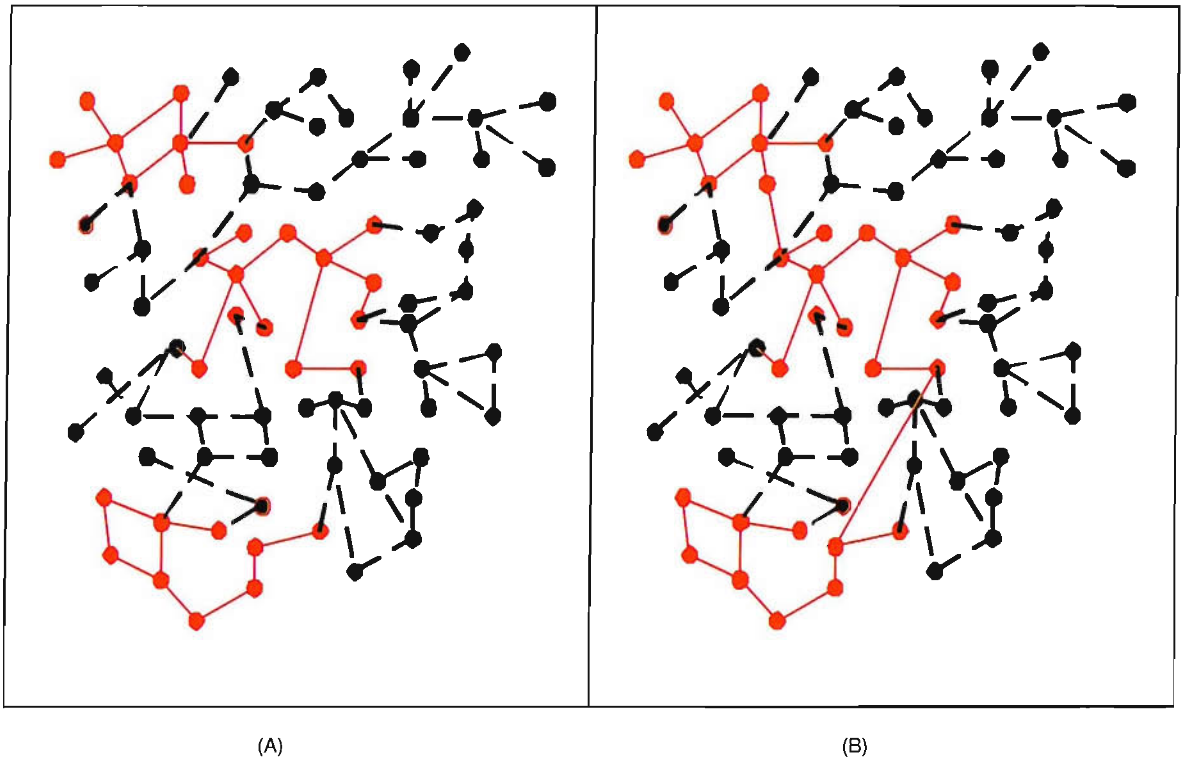


FIGURE 6.1: Peer Network, (A) An overlay showing disconnected sub-graphs clustered over a single attribute, (B) An ideal overlay with a connected graph clustered over a single attribute.

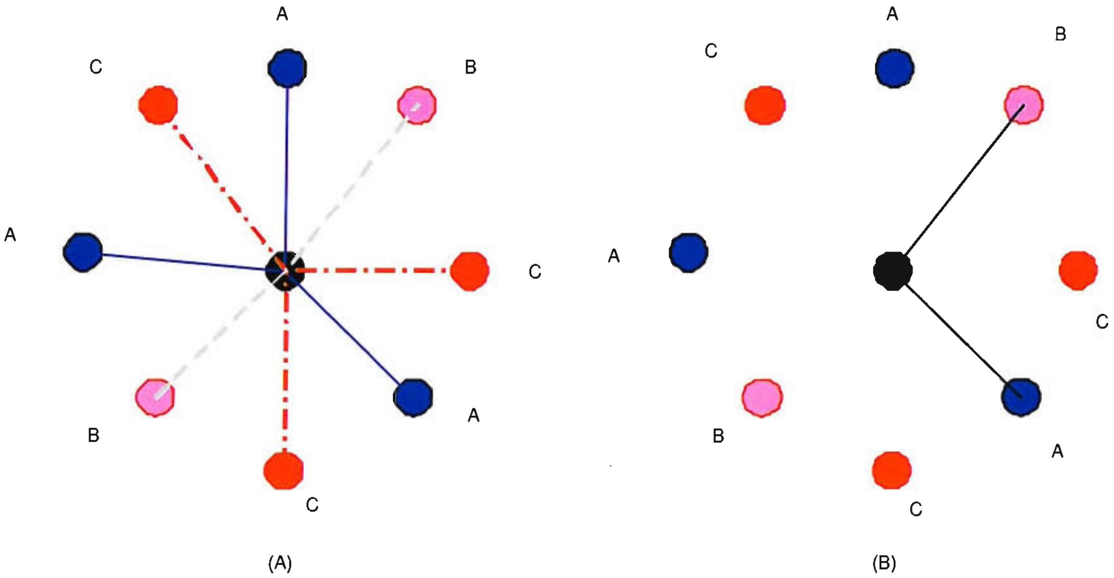


FIGURE 6.2: Overlay Selection - (A) A peer that selects the neighbours based on maximum resource overlap (B) A peer that selects neighbours to maximize the resources based on overlap and query routing history.

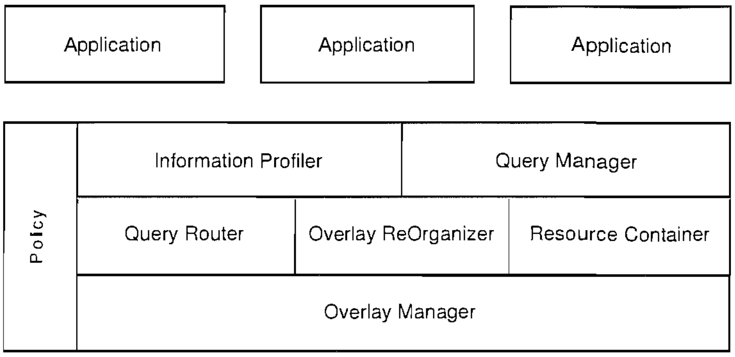


FIGURE 6.3: Peer architecture

6.1.1 Peer architecture

The overlay creation and formation process is based on a coalition formation and maintenance protocol, the details of which are described in the following sections. The overlay network does not necessitate adherence to any architecture, but is only limited to the semantics and causality of the messages. However, to provide an insight into the state maintained at each node and the implementation of the system, the peer architecture is described in this section, the block diagram is presented in the figure 6.3. Each peer node consists of the following building blocks:

Overlay Manager The overlay manager maintains the list of active coalitions and monitors the changes to any parameters in any of the coalitions. It also maintains state visibility information and manages the visibility graph G_i for a peer P_i.

Resource Container The resource container manages the list of resources made available by P_i. It uses the overlay manager interface to notify the members of the affected coalitions of resource changes.

Overlay Reorganizer The overlay reorganizer determines the long term participation of P_i in a coalition C_{ij}. It maintains statistics on communication costs and query profiles and encapsulates the inferencing techniques for overlay reorganisation.

Query Router The query router implements the query routing heuristics and tries to minimize the communication costs by choosing appropriate coalitions for routing the query contents. It also implements logic to uniquely identify the query, process queries in FIFO order and implement the query routing semantics.

Query Manager The query manager is responsible for creating and managing queries generated by P_i and provides an interface to the application logic.

Information Profiler The information profiler uses the graph G_i and the inferences from the overlay reorganizer to create a model used to select future coalitions.

Policy Manager The policy manager dictates how the local resources are utilized. The candidate policies affect the size of the routing table and memory utilisation for maintaining statistics on past queries.

6.2 Notation

Table 6.2, introduces the notation used to describe the algorithm. The general graph G_t, is composed of a number of peers. Each peer P_i provides a list of available resources LT_i and maintains a list of coalitions LP_i. While the resources are described by means of their advertisements (Id_{topic})and possible state transitions (S_{topic}), the coalitions are maintained in a separate table, also used for routing the query. A number of coalitions are maintained in the table structure, which is used to maintain the coalitions, and is also used to route the queries; hence it is also known as the routing table. Each row in the table captures information about the neighbour (Id_{peer}), the list of resources that it brings to the coalition ($LT_{resources}$), the similarity with the resources of this peer (LT_{common}) and topological information about the peer. As coalitions are treated independently of each other and as each coalition involves only two peers, the number of rows in the table equals the number of neighbours associated with the peer. The size of the table is restricted by peer parameter k_i. Each of the peers may maintain the information of a peer that is more than a single hop away from the peer P_i, the maximum number of hops being limited by the *radius*. Thus the diameter of the visibility graph is limited to twice that of the *radius*. Each peer incurs communication overheads for maintaining the state of the neighbours, and the peer can adjust the communication costs by varying the radius of the graph G_i.

6.2.1 Query structure and routing semantics

The overlay maintenance mandates that each peer maintain relevant information to route the query to the appropriate resources. This requirement is similar to that imposed in structured P2P systems where each peer maintains appropriate resource information, in this case an identifier map, to successfully locate the resource. In our case, each peer maintains the resource advertisement information for each of its neighbours and constantly updates the information to reflect the current state of the resource. The resource discovery request is expressed as a query on these advertisements. A query is identified uniquely by its globally unique identifier (GUID). The node that creates the resource discovery request formulates the query expression and associates its identifier with the query. The query manager at the node determines the TTL/max hops that determine the range of query propagation. The originating peer than evaluates the query and forwards it to chosen list of neighbouring peers (please refer to section 6.2.1 for further details on query evaluation). Each peer appends its path to the query description

TABLE 6.2: Notation used for describing the algorithm.

G_t	G(V_t,E_t)	Overlay network topology at time instance t
	V_t	Set of nodes at instance t
	E_t	Set of edges at instance t
$P_i \in V_t$	Id_{peer}	Peer identifier
	LP_i	list of coalitions
	LT_i	list of topics
	G_i	sub graph known to the peer
	$radius$	Determines the radial distance of the peers known to P_i
	k_i	maximum number of permitted neighbours for peer P_i
LT_i (List of topics published by the individual peer)	Id_{topic}	Resource Advertisement
	S_{topic}	List of possible resource states
LP_i (List of neighbouring peers)	Id_{peer}	Identifier of neighbouring peer for this coalition
	$\mathrm{LT}_{resources}$	list of resources committed by the neighbouring peer
	LT_{common}	common resources in the coalition
	α_{dist}	degree of overlap
	$\beta_{direction}$	direction of edge (only if directed graph is permitted)
	depth	The radial distance of this peer from the neighbour P_i
H_i	$time/size$	Temporal records maintained for the statistical inferencing, the list is either time or size limited
	timestamp	timestamp when the item was added to History
	$Q_{timestamp}$	The description of the query processed at the specified timestamp
Q	q_{id}	universally unique query identifier
	q_{desc}	query statement (e.g. XQuery Expression, RDF query expression)
	TTL	time to live
	hops	maximum number of hops
	root	the originating node for the query Q
	path	list of nodes visited by Q, in the order visited starting with root

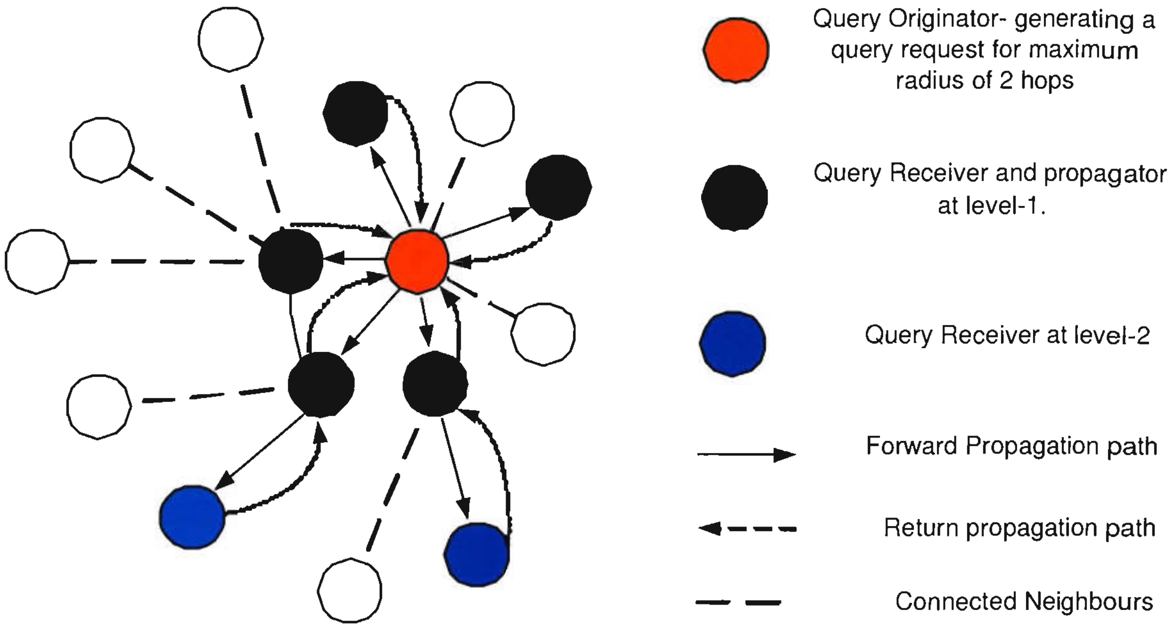

Query Originator- generating a query request for maximum radius of 2 hops
Query Receiver and propagator at level-1.
Query Receiver at level-2
Forward Propagation path
Return propagation path
Connected Neighbours

FIGURE 6.4: Schematic representation of query routing

and reduces the number of remaining hops before propagating the query to the next peer. On receipt of the query each peer evaluates the resource discovery request against its local list of resources, and, if an appropriate match is found, the query results are returned to the route peer and are propagated along the path of the query propagation. If the query has not expired and/or has not propagated to the desired diameter, the query is recursively routed to the next probable list of neighbours. The schematic representation is presented in the figure 6.4

6.3 Algorithm and message types

The above section introduced the notation and associated data structures. This section defines the continuously re-optimizing event driven algorithm used to facilitate search in P2P overlay networks. It should be noted that part of the algorithm executes in response to the occurrence of certain events. These events may originate from changes to the environment (i.e. changes in the state of the peer or the state of the neighbouring peers) or may be internally generated by the peer in response to the changes in the internal state. The following sub-sections presents the list of messages handled by the peer and its subsequent processing.

There is no particular order associated with the occurrence of most of the events. However, certain events assume precedence over other events, for example an initial request to join the overlay.

6.3.1 Request to join the overlay

Each newly joining peer needs to be associated as a neighbour to at least one of the peers that happens to be a member of the overlay. The newly joining peer P_i publishes a list of topics LT_i and generates queries for the peers with similar resources. The query requests and responses are routed through the initially contacted peer and the visibility of peer P_i increases as the peer obtains further results. The peer P_i tries to form a coalition with the newly acquired list of related peers.

Variables: $\mathrm{LT}_i := 0$, $\mathrm{LP}_i := 0$, when P_{new} joins graph G,

- Online := true
- Allow Queries := false
- Randomly generate a list of queries Q_{random} from the resource space and randomly select a set of peers from the responses, denoted as P_{random}

6.3.2 Processing the query responses for (Q_{random}) from Each Peer in P_{random}

- For each received $\mathrm{LT}_{response}$ from the randomly chosen peers,
  - Calculate α^{dist} := number of topics in $\mathrm{LT}_{response} \bigcap \mathrm{LT}_{new}$
  - Add to LP_{new} the list of peers, distance, and the intersection set with $\beta^{direction}$:= true
  - If received, $\mathrm{P}_{response}$ already exists in LP_{new}, select another set of peers
  - If $\mathrm{LT}_{response} \bigcap \mathrm{LT}_{new} = \{\}$, store the information as a uni-directional set, where LP_{new} contains the list of peers at α^{dist} = null and the intersection set with $\beta^{direction}$ = false

6.3.3 Request for processing query

Any of the peer's neighbours could invoke a request for query processing. As there may exist a number of possible paths for query routing, duplicate processing is prevented by uniquely identifying the query by its associated query identifier. Once the uniqueness of the query has been established, the query processor selects the list of probable routing targets to propagate the query. If it is unable to find any suitable coalitions that can be delegated to the query, it resorts to a local broadcast, whereby it transmits the query to all the neighbouring peers amongst all its coalitions.

6.3.3.1 Query processing at Peer P_i

Let LT_{query} represent the list of topics in the query. Let n represent the number of topics in LT_{query}.

- If not already processed query.Id_{query}
- begin
- For each α^{dist} in $\mathrm{LP}_i \geq n$,
    - If $\mathrm{LT}_{query} \bigcap \mathrm{LP}_i \rightarrow \mathrm{LT} = \mathrm{LT}_{query}$, propagate the query
    - If $\mathrm{LT}_{query} \bigcap \mathrm{LP}_i \rightarrow \mathrm{LT} = \{\}$, forward the query to all the neighbours in LP_i
- end

6.3.4 Request for resource description

Each peer maintains an advertisement of the resources that form a part of its coalitions. Depending upon the visibility information and the communication costs, a peer also caches the information of the peers at a radial distance *radius*. In order to obtain the information a peer raises a request for a resource description message. The message enumerates the list of resources that the peer is interested in and the radial distance of the intended recipients of the message. A peer uses this information to estimate the resource availability in its immediate neighbourhood. It should be noted that the peer does not enter into coalition with the peers at a hop distance greater than 1, but uses the information solely for the purpose of query routing.

6.3.5 Notification of change in resources

Each coalition monitors a set of resources and the state of the resources. A push monitoring mechanism is used, where each resource provider issues notifications for any changes to the resource state. Each coalition partner subscribes to the notifications requests of the resources contributed by its subordinate peer. Peers may monitor the communication costs for the notifications messages and consider the associated overheads while re-evaluating the coalition.

6.4 Search mechanism

Each search request to locate any particular combination of resources is formulated as a resource discovery request. The peer that initiates the search matches determines the upper bound on the total query propagation costs and specifies the time to live for the query of a particular type. Each such search request is formulated as a query routing request and incurs a cumulative communication cost across the peer network. This communication cost is affected by:

1. The topological distribution of the resource providers and consumers.
2. The relative availability of similar resources.
3. The coalition preferences of the cumulative network.

The TTL determines the yield achieved by the query routing mechanism. Each originating peer uses the TTL to specify the requests for a particular resource type as inferred from its information model. The peer constructs this resource model by observing three states of the system:

1. The state of the coalitions.
2. The state from the historically routed queries.
3. The state from observing the state models of other peers.

Once a query request has been formulated it is routed by using the state table at each of the routing peers. The query statistics capture the number of broadcasts and the relative yield obtained from routing the query. Each peer maintains these statistics over a period of time, also known as query history. The query history infers the probable distribution of resources across the peer network and also estimates the probable availability of the resources in such a network. This data can either be maintained for a fixed number of query records or it may be maintained in accordance to temporal constraints.

6.5 Overlay reorganisation

The overlay is an abstract representation of a list of coalitions between a number of participant peers. Each of the partners in the coalition autonomously decides on which coalitions to retain or which coalitions to forfeit. Under static resource and query distributions, the overlay would stabilize, and, under stable conditions, all the peers would have settled for the best possible static topology. However, in most practical cases the peers would not have complete knowledge of the system; hence, even temporal changes will result in certain coalitions being regarded as inefficient, while certain others will assume greater importance. In addition, certain unforseen conditions like the failure of certain nodes will result in the recreation of the state and the reorganisation of the overlay network.

The decision to prefer one coalition over another can either be made on the basis of a short temporal observation or can be considered over a longer duration. The duration over which the peer gauges the utility of the coalition depends upon the nature of coalition. Peers that maximize the short term gain and are interested in re-evaluating their strategy on the basis of short observations are preferred for the following reasons:

1. The frequency of changes to the system state may be too infrequent to take advantage of any previous system state for long durations.
2. Considering a large scale network the costs involved in maintaining the resource information at each of the peers may be more than the derived benefit.

Considering that a peer uses state information observed over a finite interval, a peer will be able to create the profile of the information held on the peer and the information routed through the queries. The information is used to model the resource type and distribution profile, known as the local model. A number of peers may share their model of the environment and create a model based on the coalition. The shared model can be used to create the approximate resource distribution, which can be used to choose probable partners for future coalitions.

6.6 Observations

The coalition formation algorithm described above is based on a simple heuristic used to form clusters of related resources. This intuitive approach reduces the query routing costs, as the resources in a cluster share a common objective of routing queries for their "similar" resources. However, participating in a coalition only reduces the routing costs for the resources that form the basis of the coalition. As described earlier, a coalition $C < P_i, P_j, R_i, R_j >$, is based on the sharing of resources R_i, R_j between peers P_i and P_j. Generally, a peer P_i, will have a set of resources R, such that $R_i \in R$, and is likely to enter multiple coalitions with an upper bound on the number of coalitions a peer could enter. As part of the cluster, each of the peers is obliged to cooperate in routing the queries (or requests) sent by neighbours. A peer may either satisfy a request and/or may propagate it to a neighbour that may satisfy such a request, resorting to local broadcast when both the previous options are infeasible. Routing query requests serves two purposes: First, the query routing details increase the visibility of the peer P_i by informing it of the existence of other peers. Secondly, query content allows the peer to map the resource availability in its surroundings. Based on query routing and overlay information, a peer maintains the model of its peers and classifies them into three distinct categories, firstly, peers with complete overlap of resource information, secondly, a peer set with partial overlaps, and finally, a set of peers with no resource overlap. On the basis of the above information, a peer needs to determine the most suitable candidates for coalition formation. During experimental evaluation, it was observed that if the overlay is constructed solely on the basis of the resource overlap information, the resultant cluster has a very high density of packing. High density clustering reduces the query routing costs of the highly available resources, while pronouncing the costs of the sparingly available resources. However, if the overlay formation is based on both the resource information and the query routing information the resultant clusters consists of resource neighbours from all three categories, namely with full, partial, and no overlap.

It should be noted that the clustering algorithm does not mandate adherence to any particular topology. As the topology of the overlay evolves from the coalition formation process, it is possible that the average communication costs for locating the resources and throughput of the query will vary. The variation is attributed to two factors: First, if a query originates outside the cluster of resources, it incurs local broadcast costs, until it encounters at least one of the resources within the cluster, which subsequently directs the query. However, if the TTL of the query exceeds the cluster size, the query is diffused into the surrounding cluster for propagation through local broadcasts. Secondly, as resources become scarce, the resultant clusters are few and far between, and additional communication costs are incurred in locating the similar clusters.

6.6.1 Limitations of the approach

The following is a list of limitations of this approach:

Network Partitioning: As each peer in the overlay is individually responsible for determining its immediate neighbourhood, it is impossible to rule out network partitioning effects. Although a local broadcast used by the query routing mechanism aids the reformation of the connected overlay, the chance of network partitioning could not be eliminated.

Unstable Overlay: As each of the participant peers autonomically determines its participation in individual coalitions, it is imperative that these reorganisations will happen asynchronously. As described in the previous chapter, the reorganisation mechanism requires a certain amount of time to attain an optimum solution. However, frequent changes to the query profile or resources held at the peers result in an unstable overlay. Such instability is also observed in the structure overlay and is also attributed to frequent modifications to the peers.

Warranties on forming optimal topologies: The possibility of the coalition formation resulting in a non-optimal solution was proven by Sandholm et.al (Sandholm, Larson, Andersson, Shehory, and Tohm 1999). As the above work derives from a similar approach, albeit in a more relaxed and distributed environment, the observations by Sandholm et.al. remain equally applicable, and the process may result in the formation of non-optimal topologies.

Local Optimum: The mechanism described in the previous chapter relies on the use of the latest observations of the dynamically evolving network and attempts to optimize the topology in what can be seen as a naive hill climbing technique. However, such a technique may result in identification of the local optimal and with no means to identify the global optimal, the algorithm may result in suboptimal solutions.

6.7 Summary

Application of the coalition based overlay network as applied for content management in P2P-OHS has been discussed in this chapter. The exemplar was used to route the resource discovery requests using the overlay characteristics. Reorganisation of the overlay to reflect the changes in the query distribution and resource distribution was discussed as an extension to the original approach. Observations from the actual implementation of the above have highlighted some limitations, as discussed in the previous section. The experimental evaluation of the above approach is discussed in the next chapter.

Chapter 7

Evaluation of the Search Algorithm

This chapter presents the results of the experimental evaluation of the algorithm described in the previous chapter. Section 7.1 provides the general overview of the simulation environment and elaborates on experimental settings and experimental evaluations. Experimental evaluations, results and analysis are presented in section 7.2, and the findings are summarized in section 7.3

7.1 Introduction

The simulation requirements overlap with those of peer-to-peer systems and ad hoc systems. However, as far as is known, there is no standard peer-to-peer simulation environment. Though such environments exist in the field of ad hoc networking, most of the ad hoc simulators are restricted to simulation of lower-level communication protocols. For example, the use of Network Simulator 2 (NS2) and mobisim was evaluated, [1]but considered inappropriate for simulation of an information based dynamic overlay formation. Consequently, a new simulation environment was developed, referred to as InfoSim.

InfoSim is a generic infrastructure to simulate query routing in a P2P environment. Queries are represented, communicated and routed as messages. The messages are declarative in nature and possess a context, including sender, receiver, send time, expiry time and message path. The simulator uses a logical clock and provides capability for scheduling a sequence of events. The events are cached at the recipient, for further processing. However, on expiry, the messages are discarded from the recipient queues.

[1] References could not be provided as there exists no published article that describes the simulator's internals. The web reference for NS2 is $http : //www.isi.edu/nsnam/ns/$, November 2005.

7.2 Experimental evaluation

The results presented in this section can be classified into three main categories of comparisons pertaining to:

1. Query Routing strategies;
2. Effects of variation in the link state table on routing costs;
3. Effects of topology on objective function.

Query Routing strategies: Three candidate query routing strategies are compared to determine the effectiveness of the approach. The performance of the similarity based query routing strategy is compared against the performance of random walk and the broadcast mechanism [2]. The set of experiments described in the latter sections were conducted under exactly similar routing table and resource states across the P2P system. All the routing strategies employed TTL based query expiry criteria and were compared on the basis of routing efficiency, where routing efficiency is determined as the ratio of resources discovered for the set of messages exchanged between the nodes. Further details of the experiments can be found in section 7.2.4

Effects of variation in link state table on routing costs: Maintaining the link state table incurs the messaging costs. This set of experiments investigate the effect of variations in the size of the link table on the overall query routing performance of each of the node in the network. Two situations are considered: the first set of experiments presents the effect of changes in state table size on the query routing performance of the system, and the second set of experiments presents the effects of changes in the spatial radii of the state table. The results of this set of experiments are summarized in section 7.2.4.2.

The effects of topology on objective function: Each node in the P2P system has a finite set of neighbours, and the combined set of neighbourhood information describes the topology of the P2P system. The main aim of our clustering heuristics is to create a topology to minimize the query routing costs. This set of experimental analysis observes the behaviour of the P2P overlay topology for a finite set of query distributions. Snapshots of the topology are obtained at regular intervals and are compared against the theoretically optimal topology. As there exists no comparative operators to determine equivalence or quantify the topologies, the cumulative gain in objective function is used as the measure of comparison. Details on how to obtain the theoretical optimal and results can be found in section 7.2.4.2.

[2] Broadcast and guided random walk techniques have been widely used for comparing the query routing strategies in unstructured P2P systems

Parameter	**Configuration - 1**	**Configuration - 2**	**Configuration - 3**
Simulation time steps	10000	10000	10000
Number of nodes	20 to 200	20 to 200	200 to 1000
Maximum number of coalition	3 to 50	3 to 50	3 to 50
Radii of topological visibility	1 to 3	1 to 5	1 to 10
Frequency of queries	3/time step	variable	variable
Number of unique resource definitions	200	200	1000
Maximum Resources per node	100	variable 10 to 100	variable 10 to 100
Resource Distribution type	zipf	zipf	zipf

TABLE 7.1: Configurations used in P2P simulation

7.2.1 General setup

An initial P2P network topology is assumed to represent the sets of neighbors for each of the participant peers. A global resource distribution is used to populate the list of resources hosted at each individual peer. An instigating peer is chosen at random to initiate a query and determine the appropriate radii of query propagation. The query contents are derived from a known resource distribution. Each peer autonomously routes the query and also helps in propagating responses to the queries. The radii of query propagation and frequency of query propagation are uniformly distributed over a range of values. The experiments were carried out with the configurations shown in table 7.1.

7.2.1.1 Resource distribution

Each peer hosts a set of resource descriptions also known as content descriptions. Our experiments consider two such resource distribution parameters, known as the global resource type distribution and the local resource distribution. While the global resource type distribution indicates the relative availability of a resource across the system, local distribution indicates the availability of resources at a particular node. It is observed that peers that host largest variety of contents derive higher values of resource discovery due to their ability to satisfy a relatively larger number of queries then that compared to peers with lower number of content definitions. Hence, local distribution has an effect on the choice of objective function by the peer. While a peer with higher content types can maximize its utility by satisfying queries, the other types of peer can maximize their utility by choosing an appropriate set of neighbors to route their contents to.

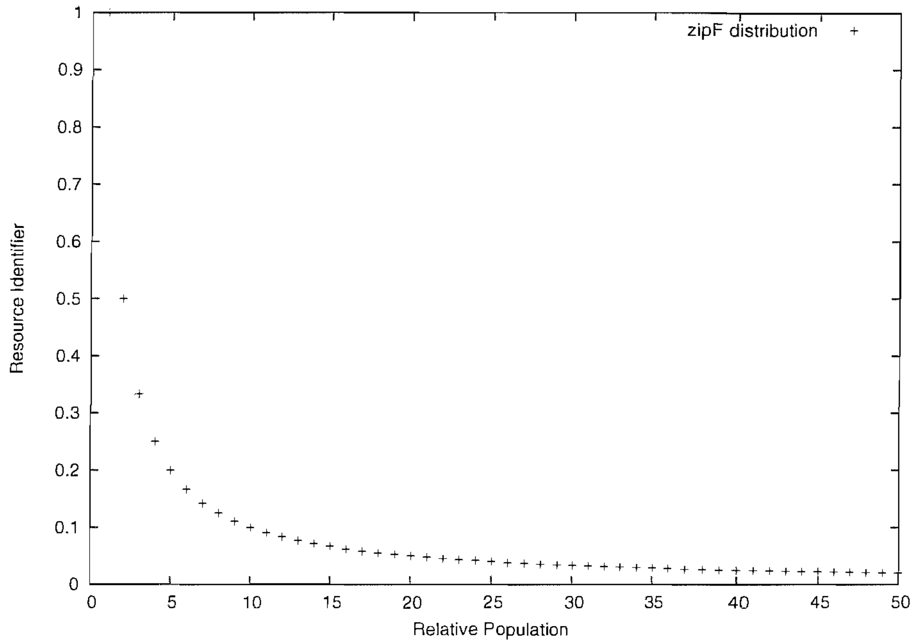


FIGURE 7.1: A zipF resource distribution.

7.2.2 Input data sets and data distribution

Both real and synthetic input data sets were used to validate the hypothesis, as the use of them was predominant. They were selected to reflect the application domain characteristics. For example, the application domains described in the previous chapter exhibit a *zipF* distribution of resources. A *zipF* distribution was used to represent the cumulative resource distribution across all the participant nodes, although individual resource distributions differed across peers. A uniform distribution was used to allocate variable sets of resources to each of the participating peers, while adhering to the overall resource distribution. In addition, uniform and poisson distributions were used to generate queries over a given set of resources. Independent resource and query distribution were used to reflect a real-life scenario, where resource availability and demand for resources are usually independent of each other. Finally, a uniform probability was used to randomly select the node that instigates the query.

7.2.3 Comparison with respect to an optimal topology

Given the resource distribution and the query profile distribution, an optimal topology can be ascertained on the basis of the mean path length required for query propagation. An optimal topology minimizes the mean path for discovering the maximum number of resources that satisfy the query. Consider the above example, the nodes are labelled 1 to 6 and their resource distribution is given as:

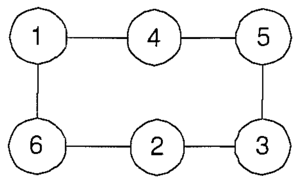


Table 7.2: A simple topology.

$$AdjacencyMatrix(A) = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Table 7.3: Adjacency matrix and its graphical topology for a small network, connectivity k = 2

Node 1: { A,B,C,D}
Node 2: { E,F,J,K}
Node 3: { E,H,M,I}
Node 4: { G,F,C,D}
Node 5: { F,H,K,I}
Node 6: { A,B,I,J}

From the above resource distribution, it can be easily inferred that *Node 1* can benefit the most by forming a coalition with *Node 4* and *Node 6*. Under uniform distribution of queried resources, both these nodes provide resources that minimize the broadcast costs incurred by *Node 1*. The same is also applicable to the topology formed by other resources. An optimal topology for a simple network of six nodes and its adjacency matrix for a maximum number of neighbor connectivity of $k = 2$ is represented in the figure.

In the above example, each of the nodes is restricted to entering a maximum of two coalitions, where, as previously described, a coalition exists between a set of peers. The value of the coalition is determined by taking into account the query profile distribution. A uniform query profile means that maximizing the information of the resource types leads to maximisation of the utility function.

Possible coalitions for *Node 1* and their respective utility values for uniformly queried resources is given as follows:

Coalition(1,2): Utility = 0
Coalition(1,3): Utility = 0
Coalition(1,4): Utility = 2
Coalition(1,5): Utility = 0
Coalition(1,6): Utility = 2

One could similarly calculate the coalition values of the remaining pairs of nodes, as the coalition is commutative: *Coalition(i,j)=Coalition(j,i)*; one can select coalitions that result in cumulative maximum utility. The cumulative coalitions are considered subject to the constraints on maximum number of neighbors k and the connectivity constraints on the graph.

The theoretically maximal coalition value determines the probable candidates for optimal topologies. However, it is not guaranteed that the heuristics discussed in the previous chapter will always converge to the formation of the exact optimal topology. In such cases, the ratio of the cumulative gain values, subject to connectivity constraints, are used to compare the effectiveness of the approach.

7.2.4 Experiments

As described in section 7.2, the experimental evaluation is divided into the above mentioned three categories. The following sub-sections summarize the results in each of these categories:

7.2.4.1 Query routing strategies

Three candidate query routing strategies are considered, firstly, the probabilistic query routing (described in the previous chapter), secondly the broadcast strategy and finally the random walk strategy. The performance of the query routing strategy is compared on the basis of the net throughput of the query and the average transmission costs for locating the resources. By its very definition, a broadcast mechanism should provide the highest throughput and should be able to discover all the resources within the TTL radius of the instigating node. The above fact was validated with the results described in figure 7.3, which plots the query throughput for the increased resource availability. As described in section 7.2.2, the cumulative resource distribution follows a zipF pattern; consequently the resource types with higher availability are more likely to occur then resource types with the least probability. As shown in figure 7.3, the resources with higher availability result in higher throughput and the throughput decreases with relative availability of the resources. The inverse zipF nature of the output verifies the fact that the overlay organisation results in a structure of $radii < TTL$ for the query and the majority of the resources are discovered by the broadcast mechanism. In terms of throughput, the probabilistic query routing mechanism provides throughput comparable to the broadcast routing strategy, as demonstrated by the plot in figure 7.2. However, the random walk strategy resulted in the worst possible query throughput, as shown in the figure 7.4. The comparative graph for the query throughput for each of the three strategies can be found in figure 7.5. It should be noted that the probabilistic routing strategy and broadcast strategy provide nearly the same throughput. The cost analysis of the three strategies can be found in figures 7.6, 7.7, 7.8 and 7.9.

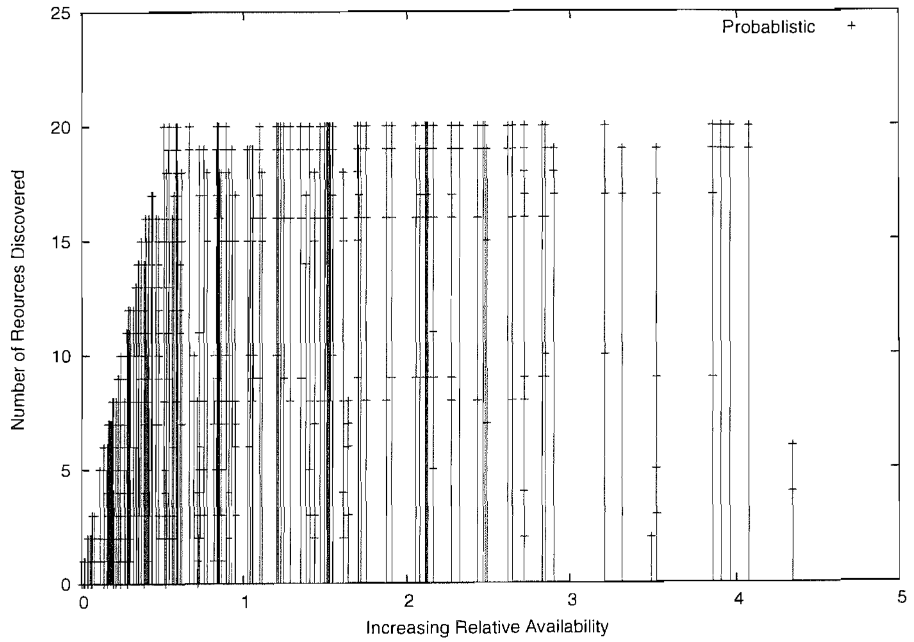


FIGURE 7.2: Query throughput using the probabilistic routing algorithm for zipF resource distribution and variable radius r, $1 \leq r \leq 3$, Number of peers = 20.

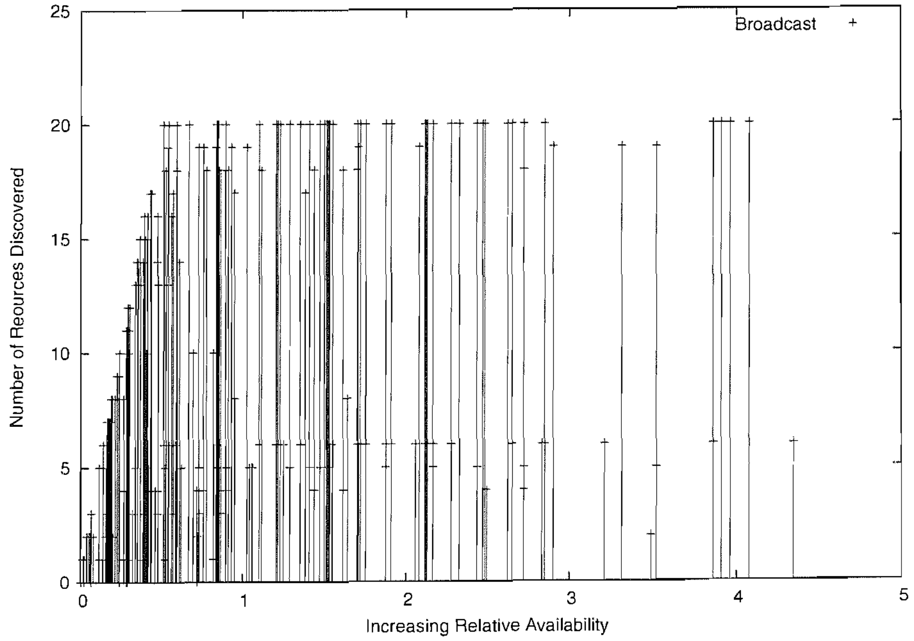


FIGURE 7.3: Query throughput using the broadcast routing algorithm for zipF resource distribution and variable radius r, $1 \leq r \leq 3$, Number of peers = 20.

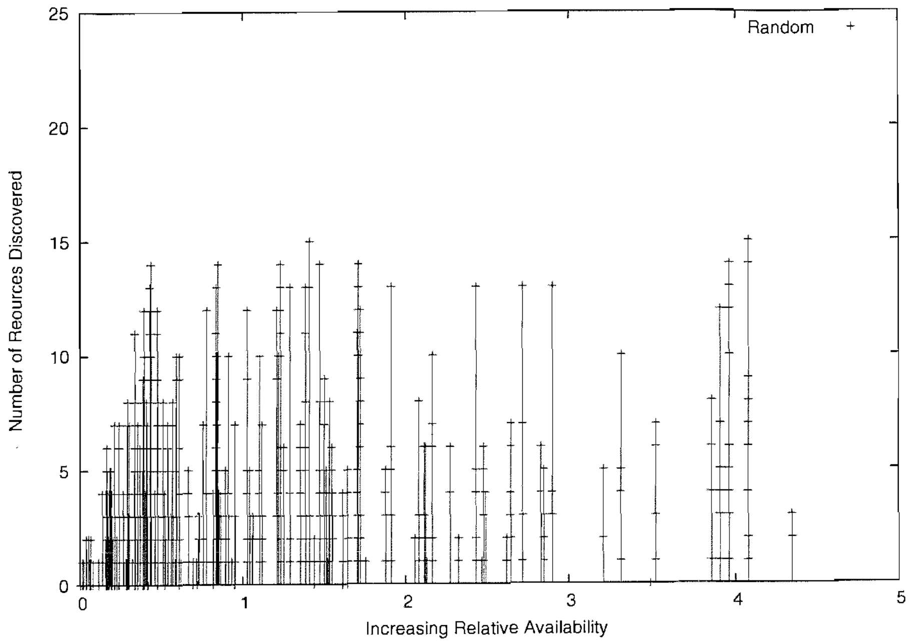


FIGURE 7.4: Query throughput using the random walk routing algorithm for zipF resource distribution and variable radius r, $1 \leq r \leq 3$, Number of peers = 20.

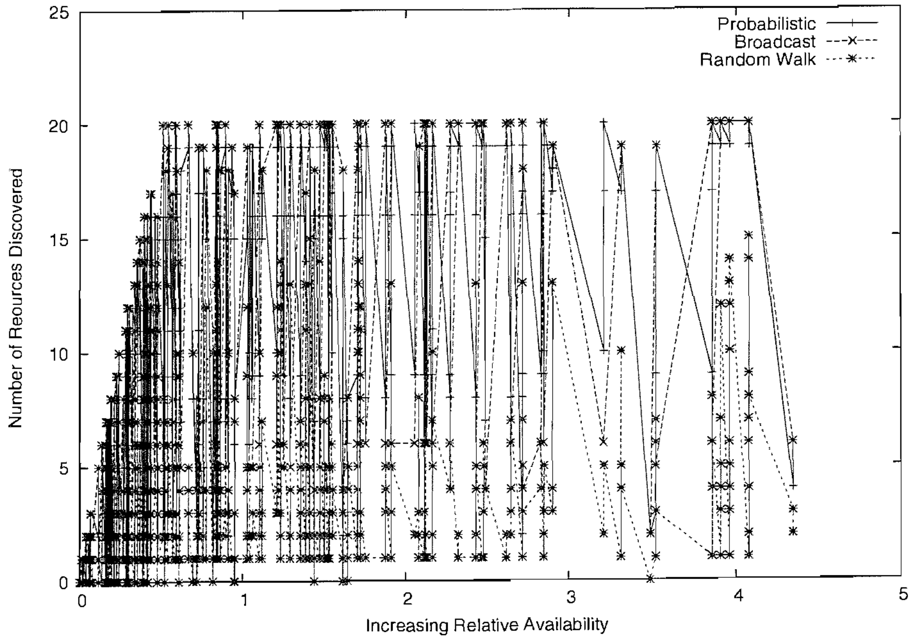


FIGURE 7.5: Comparative query throughput between probabilistic, broadcast and random walk routing algorithms for zipF resource distribution and variable radius r, $1 \leq r \leq 3$, Number of peers = 20.

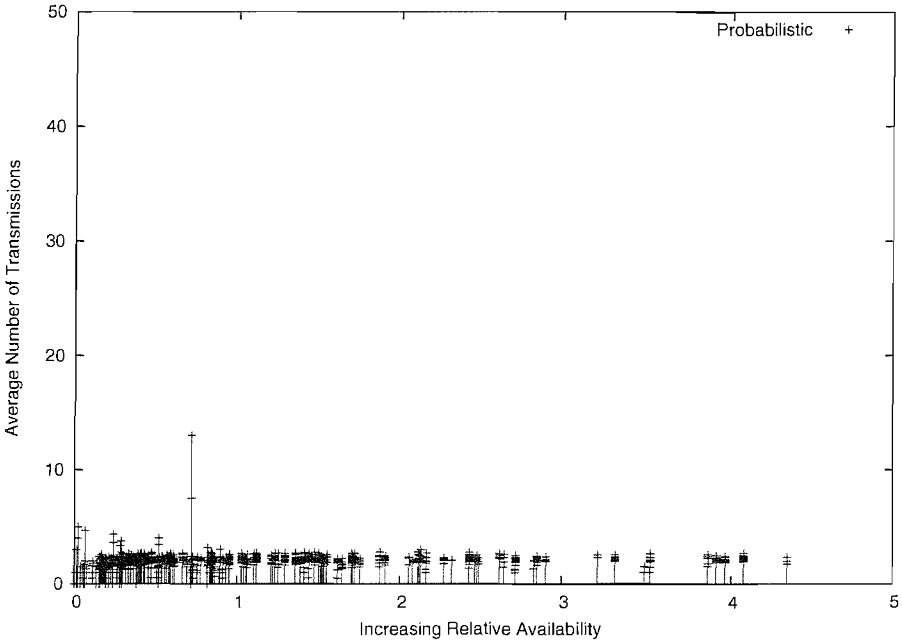


FIGURE 7.6: Transmission costs for the probabilistic routing algorithm for zipF resource distribution and variable radius r, $1 \leq r \leq 3$, Number of peers = 20.

As demonstrated by figure 7.7, the relative transmission costs of the resource discovery increases as the resources become scarcely available. The same trend is observed for the random walk strategy, as shown in figure 7.8. However, the routing costs for probabilistic routing are nearly constant and show little variation in the face of changing resource availability. As shown in figure 7.6, the query costs for the highly available resources and scarcely available resources is nearly the same. This demonstrates that probabilistic routing is effective in detecting neighbours that have a higher possibility of providing resource information. With an average transmission cost of 2.5 and a tightly-bounded theoretical transmission cost of 1.0, the probabilistic routing algorithm performs better than the broadcast and the random walk mechanism.

It should be noted that both the transmission costs and the throughput plots reflect the fact that each point in the graph is measured for a randomly generated query initiated from a randomly chosen point in the network. The variation in the throughput is because the clusters of resources are at a greater distance from the origin of the query, therefore also attracting higher broadcast costs.

7.2.4.2 Effects of variation in link state table on routing costs

It should be noted that one objective is to develop a technique that is equally applicable for both wired and wireless networks. Unlike wired networks, wireless networks, such as sensor networks, consist of nodes with limited resources. The next set of results shown in figure 7.10 demonstrate the effect of changes in radii on the overall performance of

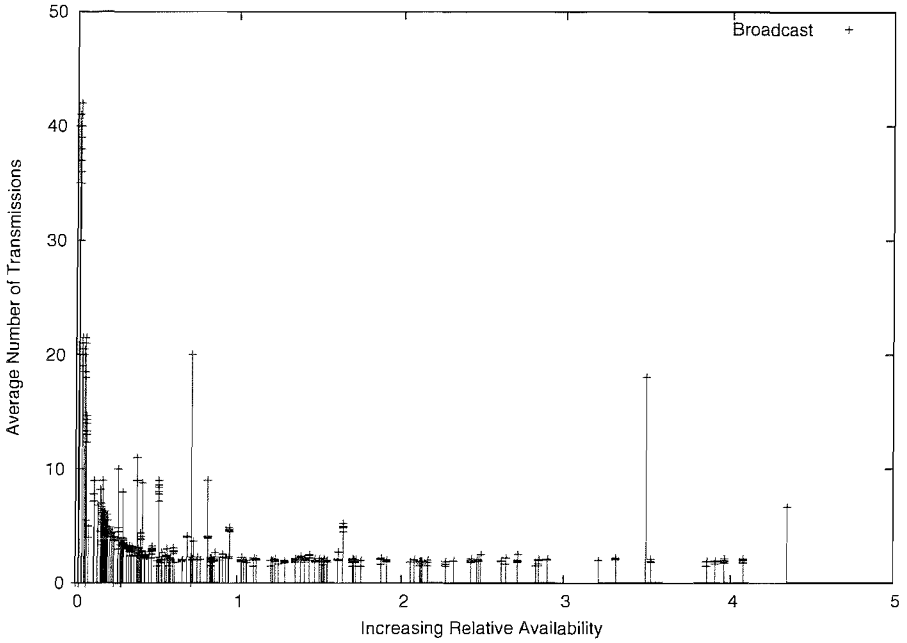


FIGURE 7.7: Transmission costs for the broadcast routing algorithm for zipF resource distribution and variable radius r, $1 \leq r \leq 3$, Number of peers = 20.

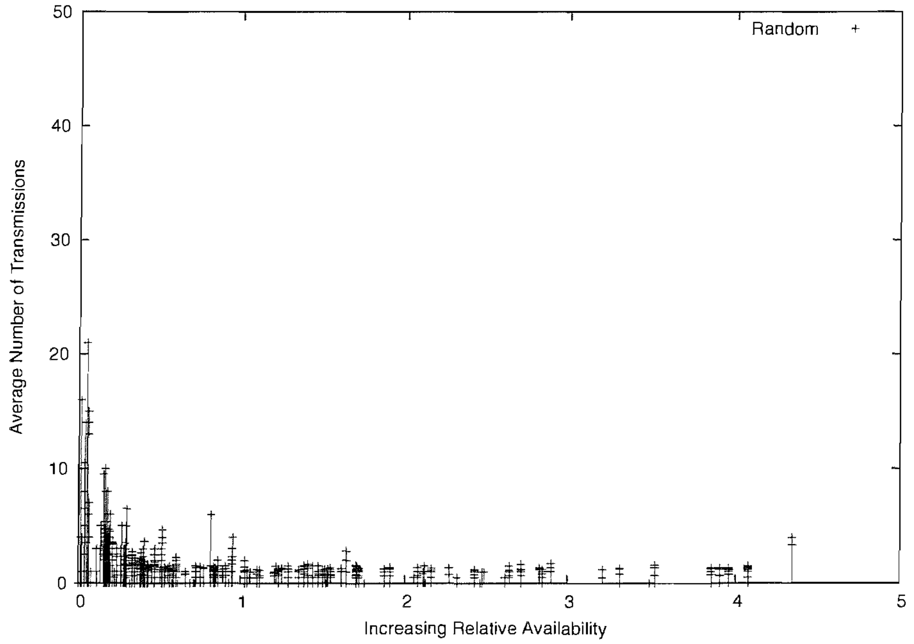


FIGURE 7.8: Transmission costs for the random walk routing algorithm for zipF resource distribution and variable radius r, $1 \leq r \leq 3$, Number of peers = 20.

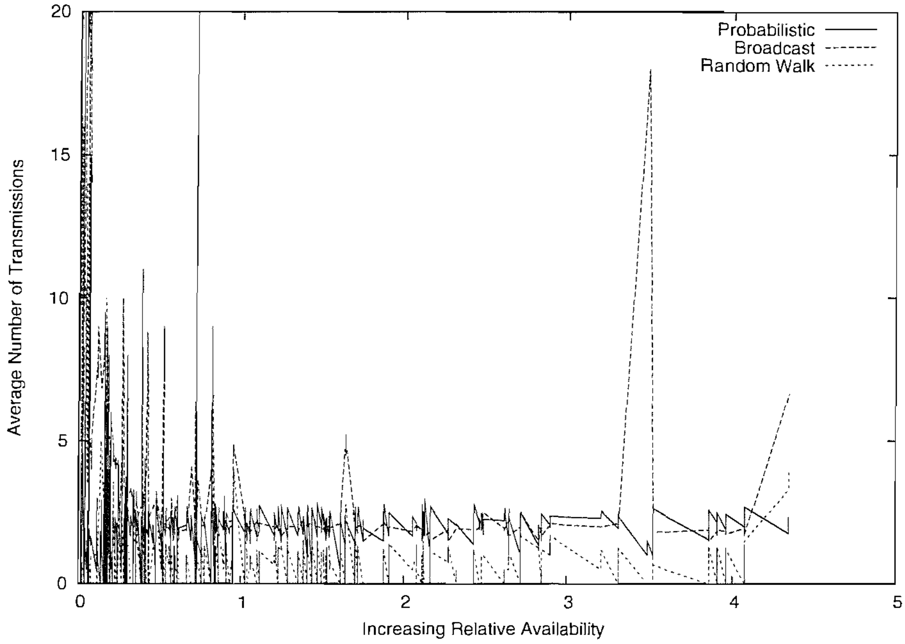


FIGURE 7.9: Comparative transmission costs for the probabilistic, broadcast and random walk routing algorithm for zipF resource distribution and variable radius r, $1 \leq r \leq 3$, Number of peers = 20.

the probabilistic routing strategy. As shown in figure 7.10, the query throughput shows a remarkable increase with increase in radius. Initial increments obtained by increase in radius outperform the further increases obtained. The above property can be exploited by the wireless networks to determine the appropriate radius for dissemination of their resource information.

Some performance observations about the routing algorithm are made in figure 7.10: Firstly, the proportional gain in throughput is higher for the initial increase in radius. Secondly, the variations in query performance reduce significantly as the radius of the graph is increased. As all the above experiments were conducted under similar operating environments except for the state information used for routing the query, it is inferred that the algorithm is able to provide better performance for an increased state information. However, a comparison of the increase in performance with the proportionate cost increments is needed, which have been highlighted in the figures 7.12, 7.13, and 7.14

The transmission cost indicators for the three radius indicate the following trends. Firstly, the clustering strategy initially allows lower transmission costs for the scarcely available resources, while the cost for highly available resources is fairly constant. Secondly, as the radius is increased the cost for query processing of the scarcely available resources increases. In figure 7.13, it is approximately constant for all query types, while it increases beyond the cost of highly available resources in the case of figure 7.14. This indicates that the cluster size has a direct impact on transmission costs, and this effect

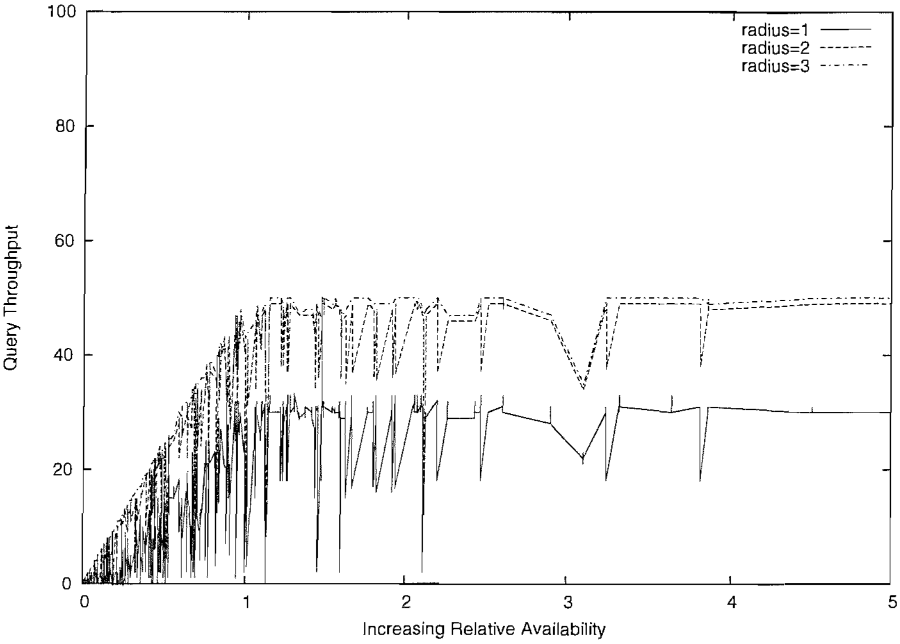


FIGURE 7.10: Effect of variation in radius on effectiveness of the probabilistic routing algorithm for zipF resource distribution and variable radius r, Number of peers = 200.

is more pronounced for scarcely available resources.

Similar studies were carried out for the broadcast strategy, and it verified the theoretical case that a change in the radius of the information has no impact on the throughput and the transmission costs. The experimental verification of the results can be found in sections 7.11 and 7.15 respectively.

7.3 Conclusion

The experimental evaluation presented in this chapter presented the generic trends observed in the construction of a coalition based overlay network. It was observed that probabilistic routing provides a comparable throughput to the broadcast mechanism but at a comparatively reduced cost. Additionally, the transmission costs of probabilistic routing remains the lowest and is constant across the resource distribution. The results also demonstrate that the increase in radius improves the performance of the probabilistic routing. However, such improvements need to be weighed against the associated costs for state maintenance. The above investigation focused on the case for which the transmission costs are linearly proportional to the amount of state being maintained. The overlay characteristics were validated with the individual peers autonomically choosing the radius of their state maintenance and coalition formation. The above results validate the assumption that coalition formation is robust in developing overlay networks for content distribution and information dissemination.

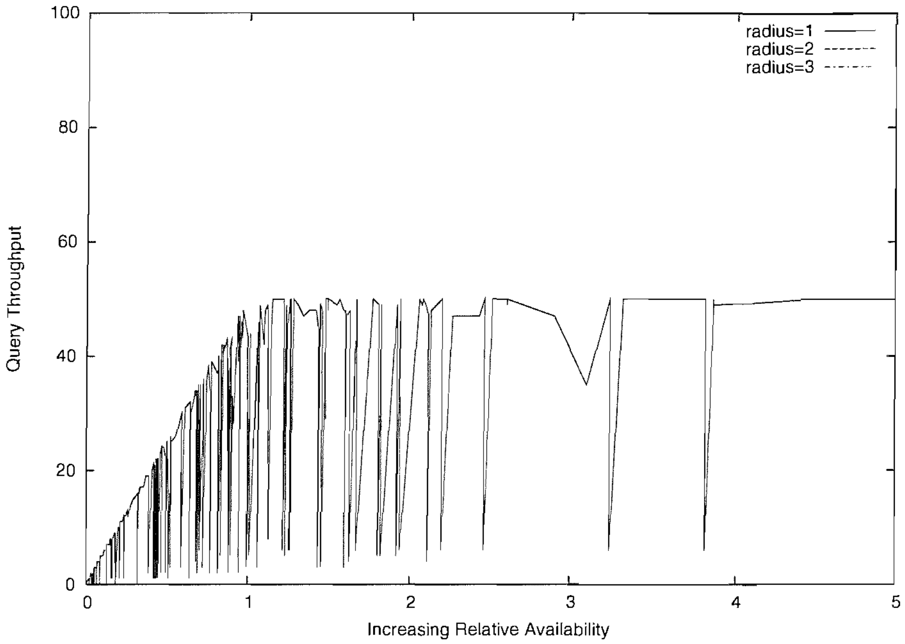


FIGURE 7.11: Effect of variation in radii on effectiveness of the broadcast routing algorithm for zipF resource distribution and variable radius r, Number of peers = 200. Note: Points in the graph overlap, the three curves are similar.

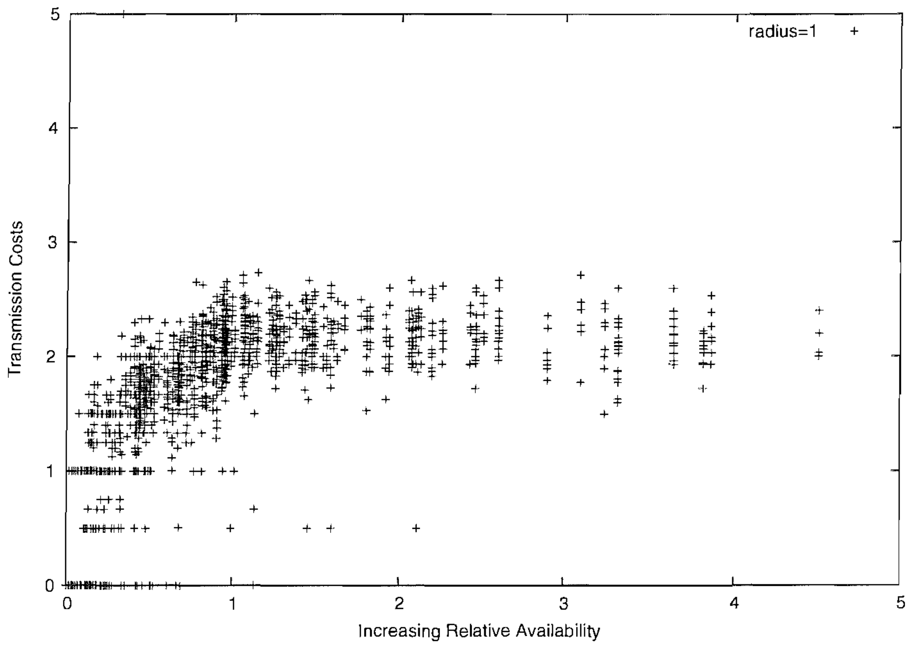


FIGURE 7.12: Cost for r=1 using the probabilistic routing algorithm for zipF resource distribution, Number of peers = 200.

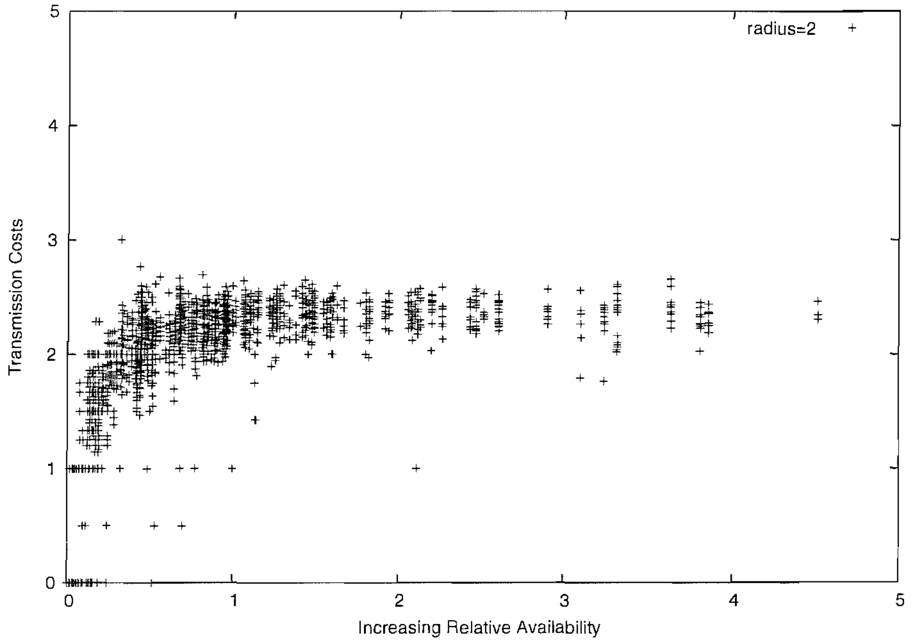


FIGURE 7.13: Cost for r=2 using the probabilistic routing algorithm for zipF resource distribution, Number of peers = 200.

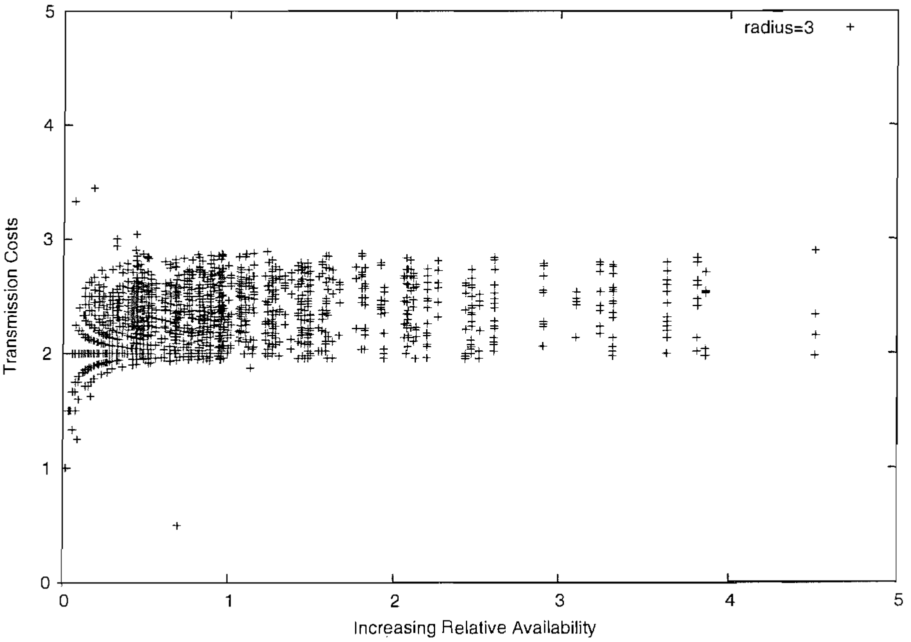


FIGURE 7.14: Cost for r=3 using the probabilistic routing algorithm for zipF resource distribution, Number of peers = 200.

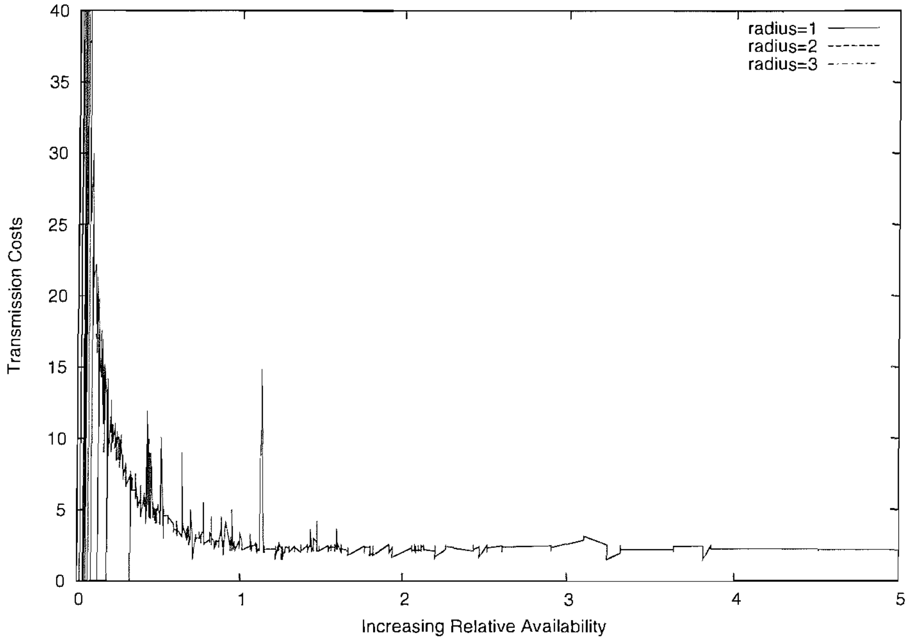


FIGURE 7.15: Cost for r=1, r=2, r=3 using the broadcast routing algorithm for zipF resource distribution, Number of peers is 200. Note: Curves in the graph overlap

Part IV

Query Processing in data stream management systems

Chapter 8

Query Optimisation

Chapter 2 introduced adaptive query processing systems for streaming data and the related motivating applications. This was followed by a detailed description of a model for adaptive information management in chapter 5. This chapter continues the thread of discussion on adaptive information management, albeit with an exclusive focus on the particular application domain - query processing over streaming data. It should be noted that the choice of this particular application domain was influenced by some of its following characteristics, as follows:

Time varying behaviour Query processing over streaming data provides an ideal case for optimisation over time. In a streaming database, data arrival rates happen to be infrequent. Time varying behaviour, coupled with constraints on memory utilisation, computational resources, and the online nature of processing, facilitate the evaluation of our assumptions of optimisation over time in a real application domain.

Distributed and partially visible information The online processing requirements of stream data management restrict the number of parses of incoming information. The system needs to adapt on the basis of a small number of actual observations, and most often these observations are not shared across multiple streams.

A reduction from a fully connected graph to a partially connected graph One of the important operator employed for query processing of multiple streams is a n-way join operator. Traditionally, a query plan for such an operator is represented by a tree, derived from an n-way fully connected graph. This allows us to explore the possibility of representing the scenario using a dynamic graph formulation. Section 8.4 describes a dynamic graph based approach to multiple query optimisation.

A combinatorial cost representation Query processing in streams is constrained, and usually represented by a combinatorial cost formulation described in terms of memory utilisation, computational costs timeliness of response. Usually a cost expression is usually associated with an individual query, but typically a system processes multiple queries over multiple streams. Therefore the application domain is well suited to exploring the relation between graph theoretical representation and combinatorial optimisation.

At the time of writing this thesis, query processing over streams is an emerging area of research in the database community. The theoretical contributions in this chapter are two-fold. Firstly, the hypothesis presented in chapter 2 should be ratified in terms of solving a combinatorial optimisation using graph theoretical techniques in a domain varying with respect to time. Secondly, a new approach is proposed for query planning on streaming data. This part of the thesis consists of two chapters. Chapter 8 describes the problem domain and the proposed solution and chapter 9 describes the implementation and evaluation of the proposed approach. This chapter begins with a brief introduction to query processing for relational data model, as described in section 8.1. However, it should be noted that with such a large number of approaches and techniques developed for query processing, means that it is impossible to provide a comprehensive summary of the whole area of the research. Instead, section 8.1 introduces important concepts and definitions to help understanding of the problem domain. Section 8.2, highlights the differences between traditional query processing and query processing for streams, using a relational data model. Section 8.3 describes the scope of the approach and its relevance to the actual application domain. A set of theoretical solutions to the problems described in section 8.3 are described in section 8.4. The next chapter, provides an architectural overview of the implementation used to verify the hypothesis described in section 8.4. The experimental results and their interpretation are provided in section 9.2. Section 9.3 provides the usual two part summary relating the findings to the application domain and the overall hypothesis.

8.1 Background

Relational data representation (Codd 1970) is a widely used one that allows flexible manipulation of encapsulated data. A part of its success is attributed to the well defined relational algebra (Date 1995), is used to represent the data definition and manipulations. Usually a declarative language - in most cases SQL (9075 1992) - is used to syntactically represent the actual list of intended operations. Most commercial database systems allow concurrent access to the underlying resources by allowing execution of simultaneous queries. These systems convert the syntactical representation into a query plan for execution by the underlying system. Query planning remains central to resource utilisation in relational data management systems. To discuss the specifics of resource management, the rest of the section discusses the query planning techniques in relational database systems.

8.1.1 Query processing in relational database systems

Query processing in relational database systems is usually a multi-staged process. Most query processing systems use a parser to create a query plan from the syntactical representation of the query statement. Initial plans are further optimized by the query optimizer. A query optimizer applies a number of transformations to generate a list of alternative feasible plans (also known as search space). A search mechanism is used to select a most suitable query plan from the list of plans - usually one with minimal costs for a given cost model. A query plan with lowest cost implications is executed by the task management system of the database. Most traditional databases generate a query plan at the start of query processing, while some perform re-optimisations of the query plan during execution. The planning process is usually based on the statistical information gathered from currently available tuples. A query plan with re-optimisation is well suited for query processing in dynamic environments. Such query plans are usually employed in multi-database systems where the unpredictable processing environment necessitates the use of dynamic planning. Re-optimisation of the query plan is not usually employed in single database systems.

Amongst the important aspects of relational query processing are cost estimation and operator scheduling mechanisms, which tend to have a huge impact on optimisation. A seminal paper by Selinger et. al. (Selinger, Astrahan, Chamberlin, Lorie, and Price 1979a) introduced System-R and a widely used formulation for generating query plans. Most database systems extend this model to create a specialized query processing system. System-R accepts a SQL statement and generates an access plan for single selection or join relationship. The cost estimates in System-R are taken into account, the access cost being based on the index information and the join operator algorithm. The generic framework proposed by System-R provides a means to calculate query costs in terms of access and operator costs. Similar cost metrics have been widely adopted in numerous RDBMSs.

The following few subsections discuss some of the important components of a query processing system:

8.1.1.1 Query planning

Query planning is a process by which a declarative data manipulation statement is translated into an execution plan for evaluation by the database system. A typical query planning system translates the declarative statement into a series of operations and sub-operations to express the task as a series of atomic tasks that can be quantified to ascertain the exact resource usage for the task. A query plan is usually expressed as a directed edge graph, which associates data with the operators. A query tree is the most usual representation of the query plan. A query tree is usually composed of a collection of data nodes and operator nodes. It can be expressed as a collection of sub-plans where the output of the internal plan is merged as the input of the higher level plan. The data nodes represent the table space or the tuple being accessed and a number of access operators and join operators form a part of the higher level plan and are used to execute the underlying sub-plans. In some cases, the sub-plans may not be expanded at the compile time and the actual planning process may be accomplished at the run time. Multiple database optimizer represent one such class of query optimizer that evaluate the query plan at the run-time or delegate the task of evaluating the sub-expression to an independent autonomous optimizer.

8.1.1.2 Cost metrics and estimation techniques

The cost of query processing is usually expressed in terms of three important parameters - memory usage (also known as memory utilisation), computational resource uage (represents the computational cost) and the time to response. As a standard case, each of the algorithms used for scan or join operations has a known complexity. For example, a hash join (Date 1995; Luo, Ellmann, Haas, and Naughton 2002), an index scan (Date 1995) and a clustered index scan (Date 1995) each has a different complexity. As most of the algorithms used for query processing exhibit deterministic behaviour, it is possible to calculate the approximate resource requirements for the evaluation of the individual query. The cost of query processing is usually a proportionate mix of the three parameter cost metrics. The exact inter-relations between the cost function depend upon the implementation of database system. As a representative example of cost calculation in database systems, the cost calculation formulae of System-R are repeated. The representation is purely to aid the comparison with the cost metrics for streaming databases, introduced in section 8.4

8.1.1.3 Query planning techniques

An optimizer needs to produce a sufficiently large number of alternate plans so that it can locate a plan with the minimal costs for processing. An optimizer needs to tradeoff the cost of optimisation of a query plan and the time to execution of the actual query. Consequently, the algorithms can be classified into three primary categories (Swami 1989) namely, the *exhaustive search, randomized search and heuristic guided search algorithms.* A brief overview of each of these types is given. A more complete and detailed discussion can be found in Steinbrunn et. al.(Steinburnn, Moerkette, and Kemper 1997)

Exhaustive Search The number of possible query plans for an n-way join increases exponentially with the number of tuples. Consequently algorithms that iterate through these exponentially increasing search spaces exhibit similar complexity in time and space. Examples of this type of algorithms appear in (Selinger, Astrahan, Chamberlin, Lorie, and Price 1979b; Lohman and ONO 1990; Kemper, Moerkotte, and Peithner 1993).

Randomized Search Randomized algorithms are well suited for optimisation in space. However, this has a negative impact on optimisation in time. By their very definition, randomized algorithms are indeterministic in nature, and are likely to have higher time overheads. These algorithms perform better than the exhaustive or heuristic search algorithms for simple queries, but tend to be more appropriate for larger queries, due to lower planning costs and fixed complexity in space. Various variants of the randomized search algorithm can be found in (Ioanndis and Kang 1990; Swami and A.Gupta 1988; Steinburnn, Moerkette, and Kemper 1997)

Heuristic Search This class of algorithms tends to exhibit polynomial complexity in time and space and generally produces query plans that are orders of magnitude more expensive than those produced by the exhaustive search mechanisms. Common examples of this type of algorithms include the "minimum selectivity" algorithm and greedy algorithms (Swami 1989; Steinburnn, Moerkette, and Kemper 1997).

As far as is known, most query optimisation algorithms fall into either of the three above mentioned categories. The algorithm presented in section 8.4 uses dynamic programming techniques that can be classified as a modified exhaustive search algorithm. The actual differences are highlighted in section 8.4.

8.1.1.4 Query re-optimisation

Query re-optimisation is employed to iteratively optimize the query at run-time. A few of the many reasons for run-time optimisation of the query plan include:

Lower confidence in cost estimates. For very large queries, sampling techniques are employed to reduce the scan costs for generation of the initial cost estimates. Additionally, some estimators may only sample a part of the entire dataset to provide a cost estimate.The accuracy of any such estimate depends on the choice of the sampling technique. A number of runtime optimizers (Ng, Wang, Muntz, and Nittel 1999; Ozcan, Nural, Koksal, Evrendilek, and Dogac 1997) adopt a strategy to evaluate a part of the result to verify the estimates and re-optimize in the required cases.

Frequent changes in operational conditions In a number of conditions, the projected estimates may be invalidated by the operational characteristics of the systems. For example, consider an equi-join operation between two tuples, where one tuple is resident in memory and the other tuple is being retrieved from the disk. At runtime, increased disk activity may result in delays in the processing of the join operators. A re-optimisation of query plan in general and operator ordering in specific may be required to reduce the overall space requirements of the query. Another example, consider a multiple autonomous database query involving a multiple database join operation. A central query processor decomposes the query into sub-queries on the participating databases. Each of the autonomous database systems independently feed the join operation. A re-optimisation of the query in light of such changes in the operating conditions is advantageous. An example approach of query scrambling can be found in (Getta 2000).

Parallel query processing Concurrent execution of the operators can either be determined at design time or can be imbibed at runtime. Adaptive query processing on parallel machines necessitates replanning to obtain better optimisation.

Multiple Query Optimisation A multiple query optimisation reduces operational costs by utilizing the resource sharing between queries. The possibility of resource sharing between the queries can only be ascertained at runtime and may require re-optimisation of the query plan.

8.1.2 Summary

The above section provided a brief introduction to query processing in database systems and highlighted some of the important characteristics of query processing systems. It is acknowledged that the above description is by no means an exhaustive one or representative of the enormous complexity of the rich field of query processing in database systems. However, as stated earlier, introduction of some of the important terms will help the discussion in the latter sections. Query planning, the costing model, query optimisation techniques, and query re-optimisation in traditional databases have been presented to contrast them with those used in query processing over streaming data.

8.2 Query processing for streams

8.2.1 Motivation

The online processing requirements of most applications in pervasive environments - for example, network traffic monitoring, fraud detection in telephone networks, sensor networks, data feeds from stock exchanges, online instruments in a Grid environment, and publish/subscribe notification models for Grid environments - necessitate query processing on data streams. For all practical purpose a stream represents an infinite source of data. The large volume of data and the online nature of the applications make it imperative for the applications to process the information in an online fashion. These systems view data as a dynamic commodity, which needs to be made available at the desired locality, at appropriate times and with the desired characteristics of delivery. The traditional notion of the centralized processing of managed data is no longer applicable in such environments. Traditional data processing involves processing on relatively static data on immobile processing nodes. In pervasive environments, data needs to be processed while in a transitory state and the data processing system needs to adapt to variations in the availability of the computing and network resources.

A typical feature of the query processing in data stream systems is the association of multiple long running continual queries with a stream of data. Manipulations on multiple data streams are specified using relational algebra, although the operator semantics and characteristics for data streams are significantly different from their relational counterparts. Query processing in these systems can be expressed as a pipeline join operation between multiple streams interconnected through database operators. As the streams are bound to experience variations in data arrival rates, the resource allocations at each of the operators are bound to fluctuate. As a result, query plans are frequently executed in the conditions that are significantly different from those for which the query plan was generated. Continual re-optimisation of the query plan is also necessitated by continuous fluctuations in data arrival rates (Amsaleg, Franklin, and Tomasic 1998; Urhan, Franklin, and Amsaleg 1998) and changes in the characteristics of the data itself (Avnur and Hellerstein 2000; Madden, Shah, Hellerstein, and Raman 2002). Our aim is to develop a query processing system that adapts to the variant nature of the data streams. Near online processing requirements necessitate that any adaptation of the query plan for the prevailing execution environment should be efficient in time and posses minimal reorganisation overheads. To expedite query reorganisation, an approach is adopted that maintains statistical information for a list of viable query plans and minimizes the query processing cost for a three-variable cost metrics - based on data flow requirements, processing requirements and delay characteristics.

Distributed processing on multiple data streams is similar to multi-database query processing. Query processing in relational database systems exploits the similarity between query sub-expressions to optimize the processing cost over multiple queries. Here is proposed a multi-query optimisation in a data stream system that forms an overlay to reduce resource consumption across multiple queries.

8.2.2 Stream data management

A stream represents a infinite source of append only data. However, practical constraints on memory space imply that it is not possible to archive the stream in its entirety, and the scope of any query evaluation needs to be limited to a subset of the streaming tuples.

One of the most frequently used methods is to use a window definition to restrict the scope of evaluation to selected subsets of tuples. Most of the windows use temporal constraints to identify the subset of contiguous tuples in a stream.

A stream S contains a set of tuples specified by an associated relational schema R, where R is an schema with attributes $X_1, X_2, X_3, ..., X_k$. The attribute space of R is defined by the function *att(R)* and the function *O: N* $\rightarrow$ *att(R)* specifies an ordering of R. A sequence $S(R_s, O_s)$ represents a stream with ordered tuples of schema R. A stream maintains a number of tuples N, where $0 <= N < \infty$ is determined by the window specifications of the set queries Q associated with the stream. Details of window specifications on memory requirements and the number of tuples retained by the query are provided later in this section. However, in some cases, the memory utilisation may increase with an increase in window size. Two alternative approaches have been adopted; first, to maintain a synopsis of the previous data, and second, to provide approximate answers to the queries. In this case it is assumed that all the tuples arriving at the stream are processed. An exact description of query semantics can be found in section 8.2.6.

A stream data processing system defines a set of operators to define and implement a query processing system. These operators can either by unary, binary or n-way operators. A generic unary stream operator O_{sp} accepts a single input S_{in} and produces an output S_{out}. While a multiple stream operator O_{mp} accepts multiple inputs $S_{in} = \{ S_i, i \in I \}$. The memory utilisation of the operator depends upon the actual semantics of its implementations. However, it should be noted that each stream operator produces an output stream and the characteristics of output stream are influenced by both the operator as well input stream characteristics.

A query is specified as a relational operation on a set of streams. However, unlike in a relational model, the additional requirements to represent the characteristics of the streams necessitate extensions to the relational algebra. The relational algebra is extended to capture the additional semantics. The approach is very common and is adopted by almost all the current stream processing systems that use relational algebra to capture the database's equivalent operations. In this approach, the query is primarily composed of two separate parts, first the data manipulation part, which specifies a SQL equivalent of query, while the second part consists of specific extensions. The additional semantics reflect how the data are used to specify the mechanism for extracting data from the stream and its buffers, and to define the lifetime of the query. (Guha, Koudas, and Shim 2001) introduce the concept of windows for specifying the buffer. The window refers to the stream tuples that can form a part of current join evaluation. In (Chandrasekaran, Cooper, Deshpande, Franklin, Hellerstein, Hong, Krishnamurthy, Madden, Reiss, and Shah 2003) the authors extend this definition and consequently four types of queries have been identified, a snapshot query, a landmark query, a sliding query and a temporal band join. The semantics of windows heavily impact on the persistence mechanism and the execution of the query. The same query semantics is used in this study as proposed by (Chandrasekaran, Cooper, Deshpande, Franklin, Hellerstein, Hong, Krishnamurthy, Madden, Reiss, and Shah 2003), and a sample query for a single stream is represented as:

```
Select  temperature,timestamp
from furnace_Monitor
where  furnace_charge <= 20 for(t=ST; t<ST+30; t+ =1){
 Windowls( Temperature, t-5, t)
}
```

The above expression translates to select the temperature and the time-stamp for a furnace where the furnace was loaded with some charge. Select the values for the past 6 days and keep the query alive for 30 days. The actual number of tuples that end up populating the window over the period of last 6 days depends upon the data arrival rate. A number of such expressions can be queued on a stream and need to be evaluated simultaneously. The evaluation of a query can occur at the time of the arrival of data or at predefined intervals or at the time of queuing the query or at the time when the data is being invalidated or overwritten due to some window overflow criterion. All such attributes should ideally form part of the scheduling operation. However, almost all the query-processing applications currently available fail to provide any extensions for the scheduling operations. The definition of the above specification is therefore extended to include scheduling options as part of query specification. The actual event that triggers the evaluation of a query is referred to as the evaluation event.

The query specification has the following logical structure.

```
Select [ Tuple specification]
FROM [Tuples]
WHERE [JOIN — RANGE CRITERION]
FOR [Query period]
WINDOW SPECIFICATION
ON
Scheduling criterion such as DATA ARRIVAL | PERIODIC | SNAPSHOT |
    OVERFLOW | QUERY ARRIVAL
```

The scheduling options are important in determining the overall quality of service for evaluation of query results, while the window specifications are important in determining the semantics of the result set generated from the query evaluation. The query specifications and the stream characteristics identified in this section are described in the next sections on window semantics and join operation semantics.

8.2.3 Window semantics and specifications

As far as is known, operational semantics for join operators for stream data processing are not as well defined as in the case of the relational database system - this is primarily attributed to different window definitions and the different result set semantics prevalent in the field. A number of different window types have been defined in individual research projects. The most widely accepted types of query semantics were described by Chandrashekar et.al. (Chandrasekaran, Cooper, Deshpande, Franklin, Hellerstein, Hong, Krishnamurthy, Madden, Reiss, and Shah 2003). Chandrashekar et.al describe four query types namely - Snapshot query, Landmark Query, Sliding Query and Temporal band Join. Each of the four types imposes significantly different requirements on the design of the query processing system. For example, the query evaluation event for a snapshot query is evaluated exactly once per window. A join operation in the case of a snap shot query between streams evaluates in response to an event on the input window of one of the related streams. Similarly, in the case of a landmark query, sliding window and temporal band join, either a temporal event or arrival of data at a stream could result in the evaluation of the join. In addition, the query results may be generated out of order and guarantees may be provided on the eventual correctness rather than immediate correctness at each evaluation step. In this case, the proposed approach considers that the results are correct at each evaluation step. The out of order arrival of tuples from data sources is not permitted.

Consider the stream $S(R_s, O_s)$ and an associated query Q with a window specification W. Let i represent the number of tuples in the sequence represented by the stream, and let N be the number of tuples currently held in the stream at a given time $t_{instance}$. Let t_i denote the time at which the tuple i was added to the sequence, i.e. arrival time of S_i is t_i.

A snapshot query is evaluated against the contents of the stream at a particular instance in time. It should be noted that a snapshot query does not maintain an active window on the data stream; instead it utilizes the current contents of the stream S. If *Proj(S, W)* is defined as the scan of window W predicated on stream S, where if W_s represents the list of windows associated with the stream, then a snapshot of stream S is given by

$$Snapshot(S) = Proj(S, W_s) = Unique(\sum_{i=1}^{n} Proj(S, W_i)), whereW_i \in W_s$$

A landmark query evaluates the query for all rows of the selected streams beyond a particular landmark. Hence, W_L represents a valid landmark window if and only if the landmark l satisfies the condition $S_0.t_0 < l < S_N.t_N$ all the records where $l < S_i$ forms a part of the window. Also, any records added to the S at time $t > t_{instance}$ are also added to the scope of the landmark window specification. A landmark window can also specify

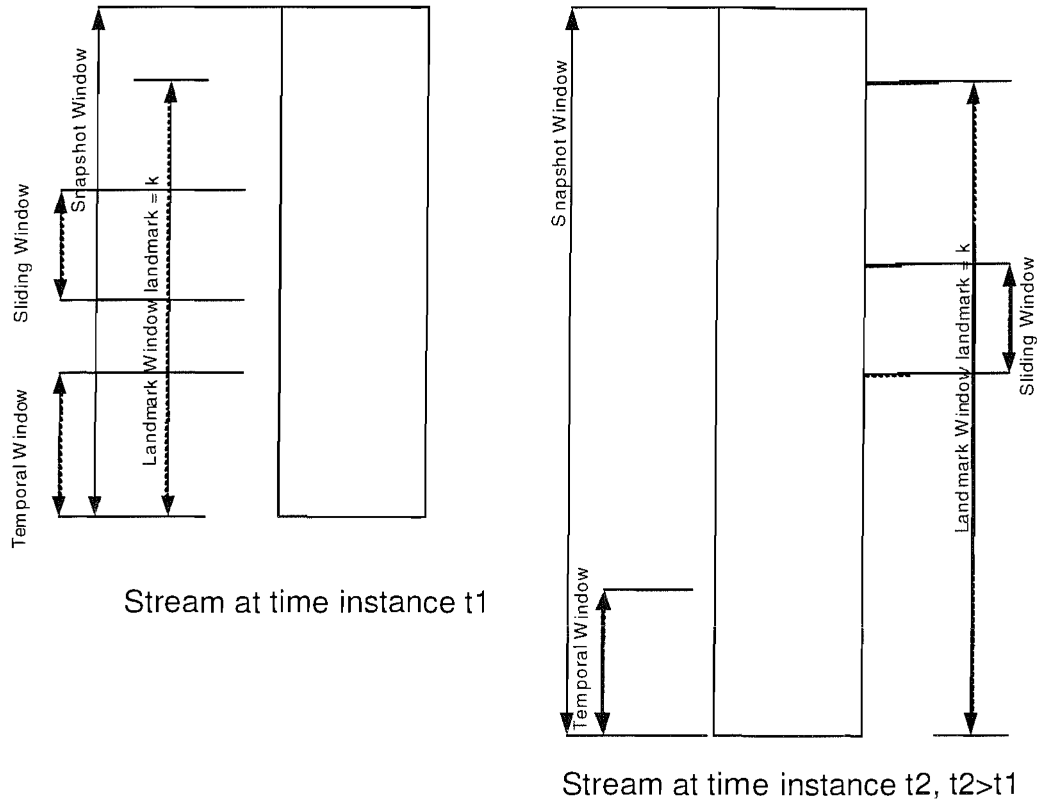

Sliding Window
Temporal Window
Snapshot Window
Landmark Window landmark = k
Stream at time instance t1
Snapshot Window
Temporal Window
Landmark Window landmark = k
Sliding Window
Stream at time instance t2, t2>t1

FIGURE 8.1: Window types

the rules for automatic traversal of the landmark on each evaluation. Theoretically, landmarks may either be specified using temporal properties or derived from the data content properties of the tuples, as in the following restricted case.

$$Landmark(S, W) = scan(S, S.t > W.t)$$

A sliding window over streams is represented by a set of landmarks - an upper bound and a lower bound. A scan on stream S for a query evaluation with a sliding window specification returns the tuples between these two landmarks. A scan for sliding window W_{sl} with a set of associated landmarks $W.L_{upper}$ and $W.L_{lower}$ is represented by:

$$Sliding(S, W_{sl}) = scan(S, W.L_{lower} > S.t > W.L_{upper})$$

As defined earlier, landmark window scan evaluates to records from a given landmark; thus one end of the window is clamped, while the other slides as new data items are appended to the stream. A temporal band join is a reverse phenomenon, with a fixed endpoint and a variable start point. A temporal band window retains a fixed number of items and slides as the data items are appended to the stream.

$$Temporal(S, W_t) = Scan(S, |S_i| - t_k << S_i << |S_i|), \tag{8.1}$$

where t_k is the window size.

It is assumed that streams are 'append only' containers; therefore each of the operators can process the data on arrival. However, in certain cases, streams may be allowed to modify the data within a window. In such cases the operator requires an explicit invocation of the query evaluation event. These two different semantics have varying effects on the way in which the queries are processed by the system. However, the following discussion considers 'append only' data streams.

8.2.4 Continuously Adaptive Continuous Queries (CACQ)

Query processing over streaming data can either follow snapshot semantics, thereby evaluating the query expression over the current states of the stream, or otherwise allow a continuous evaluation as new data items are appended to the stream (also known as continuous queries). A continuous query expression is scheduled in accordance to the window specifications of the query. Continual query processing requires maintenance of some intermediate state between subsequent evaluation of the query expression. Unlike their traditional counterparts, data operators for stream data management need to maintain an intermediate state to minimize the cost of regenerating the state at each query evaluation. The continual query paradigm necessitates the use of pipelined operators.

Traditional query plan evaluation generates an intermediate state during the query evaluation and discards the state on completion of the query. A continual query creates, maintains and modifies the state with respect to the changes in the environment. Candidate examples for such data operators include a flexible hash join (Shah, Hellerstein, Chandrasekaran, and Franklin 2003), a ripple join (Haas and Hellerstein 1999) or an eddy operator (Avnur and Hellerstein 2000). Modified versions of these operators have been used to provide CACQ capabilities in stream management systems.

Flexible data operators were designed to adapt to the variations in availability of resources used for query evaluation. Consider a query Q = $A_1 \bowtie A_2 \bowtie A_3 \bowtie A_4 \bowtie A_5 ... \bowtie A_n$, an n-way join between the tuples $A_1, A_2, ..., A_n$. A sequential n-way join operator will block until the data is available for each of the tuples. A flexible operator performs the incremental joins between the data tuples as and when they become available, it also optimizes join performance by reordering the join ordering of the tuples. While sampling the relationship in an n-way join, the join operator scans a single relationship for changes and evaluates the incremental results.

Assume that a join has been evaluated for the initial scans on the relationship, such that:

$$Q = S_1 \bowtie S_2 \bowtie S_3 \bowtie S_4 \bowtie S_5 ... \bowtie S_n \tag{8.2}$$

If the sampling of the relationship indicates that the new tuples ${S_1}'$ have arrived for the schemas S_1, the incremental result

$$Q1 = (S_1 \cup {S_1}') \bowtie S_2 \bowtie S_3 \bowtie S_4 \bowtie S_5 ... \bowtie S_n \tag{8.3}$$

$Q1 = (S_1 \bowtie S_2 \bowtie S_3 \bowtie S_4 \bowtie S_5 ... \bowtie S_n) \cup ({S_1}' \bowtie S_2 \bowtie S_3 \bowtie S_4 \bowtie S_5 ... \bowtie S_n)$
$\because (A \cup B) \bowtie C = (A \bowtie C) \cup (B \bowtie C)$

$$Q1 = Q \cup ({S_1}' \bowtie S_2 \bowtie S_3 \bowtie S_4 \bowtie S_5 ... \bowtie S_n) \tag{8.4}$$

$Q1 = QU\delta$, where $\delta = ({S_1}' \bowtie S_2 \bowtie S_3 \bowtie S_4 \bowtie S_5 ... \bowtie S_n)$ represents the incremental processing for ${S_1}'$. A scan of the tuples in the relationships results in an incremental result set generated at a minimal cost while retaining the state of the previous evaluation. CACQ operators are most suitable for query over stream data as the incremental tuples arriving at each of the stream window can be continually evaluated resulting in a consistent result set, where the increments can be managed by manipulating the increments in the input tuples. The join semantics of CACQ provide an incremental result set for the join operation and are well suited for 'append only' join processing, i.e. landmark windows. The effects of CACQ on sliding window queries are discussed in the section 8.2.6, which is preceded by a brief discussion on adaptive join operators.

8.2.5 Adaptive join operators

In the case of query processing over streams an n-way join operator provides improved space and computational usage compared to a series of binary joins. A typical n-way join operator does not retain any intermediate results, while an n-way join implemented as a series of binary joins needs to maintain and may need to re-index the intermediate results to improve the join efficiency of the intermediate joins. A number of adaptive n-way join operators have been proposed in the literature , for example, (Avnur and Hellerstein 2000), (Haas and Hellerstein 1999). An n-way join operator centralizes the operator state for the n-way join facilitating easy optimisation. It reduces the co-ordination costs for adaptive query planning. For example, an eddy (Avnur and Hellerstein 2000) operator creates individual stems (Raman, Deshpande, and Hellerstein 2003) for each of the participating tuples. The tuples are sampled for new data items and the n-way join is accomplished by routing the tuples through appropriate stems. An eddy minimizes the intermediate state, thereby allowing flexibility to adopt an individual routing policy for a given set of tuples. The StreamDB illustrates a list of alternate query routing policies. In their paper, Madden et al. (Madden, Shah, Hellerstein, and Raman 2002) discuss a ticket based routing mechanism to circumvent the problem of selectivity estimation. An alternative approach is suggested by the same (Madden, Shah, Hellerstein, and Raman

2002) that uses the random routing policy to route the tuples between the operators. Similar approaches have also been discussed by (FengTian and Witt 2003). For SPJ, the authors provide a nested loop implementation of an adaptive operator that adopts to changes in the data arrival rates and operator selectivity.

An adaptive join operator for streaming data needs to optimize the usage of three different types of resources, memory space, computational resources and response time. Alternative approaches to formulating the memory constraints are highlighted by two very distinct approaches. The first, is PSoup (Chandrasekaran and Franklin 2003)in which the authors describe a mechanism to reduce response time by an early and lazy materialisation of the result set, while (Arasu and Widom 2004) Arasu et al. explore a different problem of identifying the set of queries that can be executed under memory resource constraints. The approach highlighted in section 8.4 in a way extends Arasu and Widom's approach for defining the resource constraints in an adaptive data operator. As a typical query plan consists of a number of such operators, they need to be scheduled on scarce computational resources. A number of scheduling options for optimizing computational costs and improving response time have been suggested in the operator scheduling strategies of (Babcock, Babu, Motwani, and Datar 2003; Hammad, Franklin, Aref, and Elmagarmid 2003).

In general, n-way adaptive join operators can be compared on the basis of their adaptivity in solving a three-parameter cost metrics of memory usage, computational resource usage and responsiveness. Section 8.3 provides description of the combinatorial problem and the solution in section 8.4.

8.2.6 Join semantics

Relational algebra provides a de facto definition of join operations. It introduces inner join, outer join and equi-join semantics. However, relational algebra was defined for static data items, and is equally applicable to window joins over streams. However, as defined in section 8.2.4, different window types generate different types of scan objects for query evaluation. If the windows hop from one set of intervals to another, then the overall semantics of the join over the period of time are maintained and are similar to the results of the non windowed join. However, in the case of a sliding window join, subsequent window joins may share a set of tuples. A union of all the result sets generated during each of the evaluations may contain duplicate data items. The cumulative result is that the actual semantics of the join are not maintained.

8.3 Problem Definition

There are at least three different types of optimisation scenario a figure prominently in query optimisation over streams, first, a very basic type of optimisation that aims at reducing the computational cost of an individual query, secondly, optimising utilisation of system resources over multiple queries, given that a number of simultaneously executing queries over a set of streams provide a potential for resource sharing, and finally, scheduling the queries so that effective utilisation is minimized for both the individual and the group of queries.

Consider a set of streams S = $\{S_1, S_2, S_3, ..., S_n\}$, where each stream is expressed as $S(R_s, O_s)$, as introduced in section 8.2.3. A list of queries Q =$\{Q_1, Q_2, Q_3, ..., Q_m\}$ are queued for execution over the set of streams. Each query Q_i is represented as a set of operations on a set of streams S_{Q_i}, such that $S_{Q_i} \in S$. Suppose that the query operation Q_i represents a multiple join operation, it can be expressed as, $Q_i = S_{Q_1} \bowtie S_{Q_2} \bowtie S_{Q_3} ... \bowtie S_{Q_k}$, where k = $|S_{Q_i}|$. Each of the streams in S_{Q_i} has an associated rate of data arrival d_{Q_i}, where $d_{Q_i} = \sigma R_{S_i}$, where R_{S_i} represents the *average* rate of data arrival at stream S_i, and σ represents the selectivity of Query Q_i over stream S_i for attribute $R_j \in R_s$.

Each of the data streams S_{Q_i} receives a set of tuples through constant evaluation of the query Q_i over S_i. Though the collective selectivity of the query is constant, processing for the overall join operation is minimized by using appropriate join ordering. On the other hand, the join operator scheduling aims at reducing the memory utilisation of the query and to minimize join operator costs. The query processing is characterized by the following:

1. Query routing allows the join to be either individually evaluated for each of the tuples or it can be evaluated for a group of tuples.

2. As discussed in section 8.2.4, at any given instance, a ripple join process can only progress in one dimension. While the query is evaluating tuples from a single stream, the memory utilisation at other streams increases as the new tuples are queued at those streams. The utilisation cost during the processing is directly proportional to the data arrival rates at the waiting streams. Hence, the choice of a stream to be processed needs to be optimized against the costs accrued by memory utilisation at the blocking streams.

3. Each subsequent operator ordering should result in a reduced result set. This implies that operator reordering needs to follow selectivity estimates. Selectivity at any given join execution is directly related to the current contents of the stream windows. If a statistical tool is used for calculating join estimates, it needs to reduce computational costs by avoiding the re-estimation of costs for each window hop.

4. The process of continuous optimisation has its own control costs (such as recalculating the hash tables and recreating the intermediate results), which need to be minimized.

5. The response time of the query is directly proportional to the delays introduced by the individual join operators. The response time delays can be minimized by maximizing the parallelisation of the query evaluation. Note, parallelisation necessitates the creation of a more bushy query plans, and may have higher synchronisation overheads.

6. The ripple join calculates the results in terms of the previous query results. A query can reduce the cost of re-optimisation if it can identify a subset of the previous results, which can retain between subsequent re-evaluations of the query. It should be noted that the retention of intermediate results is constrained by the size of the cache.

The above mentioned constraints are applicable to a single query optimisation scenario. In this case, techniques to achieve the local minima for memory, processing and delay are considered effective.

Multiple queries continuously executing on a common set of streams provide possibilities of resource sharing between multiple query evaluations. The simplest form of resource sharing can be applied at the level of select and project operators, whereby a single filter is used to scan the data for each of the selection and projection operations. The design of one such operator is discussed later in this section.

An alternative form of resource sharing relies on sharing of intermediate results. Although, resource sharing between queries has traditionally relied on identification of common sub-expressions between a set of queries, it is equally applicable to query processing on streaming data. Query processing over streams provides an added advantage of routing the tuples in such a way that multiple evaluations can be simultaneously carried out by the multi-stream operators. In addition to the identification of the sub-expression, the queries have to be evaluated for window semantics, as the two queries with a shared set of tuples in their window definitions are bound to have reduced cost if they share the costs of processing. The shared expression can be used to select the appropriate order for routing of the tuples, such that the combined processing costs of routing the tuples are minimized.

The third and final type of optimisation relates to resource consumption by the individual queries. Data items queued at each of the streams occupy memory resources. In certain cases, for example the sliding window, memory resource utilisation can be optimized by appropriate prioritisation of the order in which the streams are processed. The optimisation problem can be summarized as generating the appropriate schedules and identifying the appropriate sets of queries that can share this atomic set of operations.

To address all the above mentioned concerns three algorithms are devised in the following sections. The following sub-sections, provide further refinements on the above-mentioned set of problems.

8.3.1 Selection and projection filter

A selection and projection operation over a data stream differs from the selection and projection over a relational database table, in that the scope of selection and projection in a streaming data processing system is limited to the scope of the current window. A number of queries may involve selection and projection over the stream data, thereby providing ample opportunities for sharing selection and projection costs, by maximizing the co-evaluation of the selection and projection operation.

A select and project operator results in the creation of intermediate result sets for each of the queries. The size of the result sets can be reduced by maintaining the references to the rows of data and not replicating the actual tuples. These references are used to create a scan for further evaluation of the query. The queries may share the actual scan or may proceed independently. The processing costs of shared evaluation can only be sustained if the resulting costs accrued for maintaining the result sets are less then an independent evaluation. However, the select project filter should be capable of simultaneously evaluating multiple queries and should be able to tradeoff the costs of maintaining the results against the cost of evaluating multiple evaluations.

In order to decide such a trade-off the operator should be capable of estimating the sizes of the resultant datasets and also the estimated costs of maintaining the result set. The size estimates can be obtained by using the histograms for range queries. However, the dynamic contents of the stream data render the regular techniques for histogram evaluation ineffective.

8.3.2 Complex queries

A set of database operators defined for relational algebra can be found in (Date 1995). The scope of this discussion is limited to a set of select, project, join, logical and cross product operators. The list is restricted to keep the discussion focused. Most of the observations detailed for these operators are equally applicable to other database operators with minor or no modifications. Combinations of the select, project and join operators comprise a query expression. Evaluating the strategy against a number of query types provides an opportunity to determine the efficacy of the approach under extreme query types.

Some of the examples derived from the previous work in the field are described below. The paper by Gouda and Dayal (Gouda and Dayal 1981) introduces three distinct query

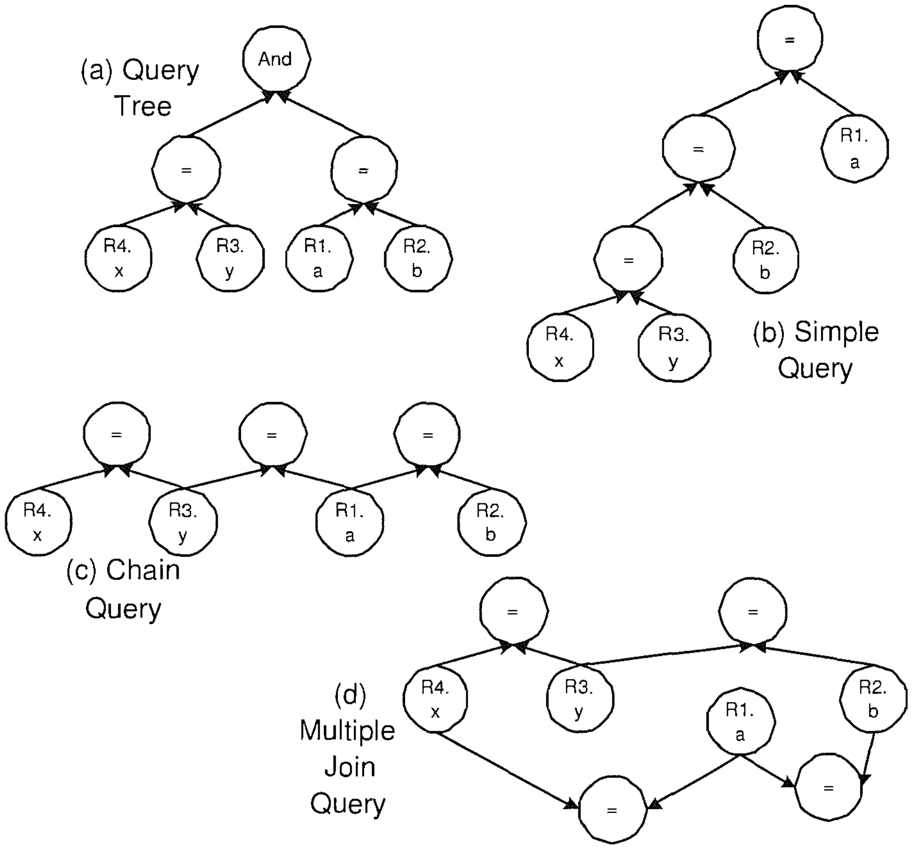


FIGURE 8.2: Different Query Types

types, namely a tree query, a simple query and a chain query. A tree query represents a multi-way join between a number of tuples. The join criterion assumes that any tuple in a tree query participates in only one join operation. A simple query is a special case of a tree query, such that a common joining attribute in one of the tuples across the joins operations. Similarly, a chain query is a special case of a tree query in which each tuple has at most two join attributes and participates in two join operations. Figure 8.2 provides graph representation of the tree join, simple query and chain query. In addition to these three query types, a fourth query type, a multi-join, was introduced in a paper by Lee et al. (Lee, Shih, and Chen 2001). This is a generalized case of multiple join operations in a query, and with multiple join ordering combinations.

8.3.2.1 Pipeline query

A pipeline query is composed of a hierarchically arranged series of join operators. A typical pipeline query operation can be expressed in the form $R_a \bowtie_b S_b \bowtie_c T_c \bowtie_d U$. A query plan for the evaluation of the pipeline join operation can be represented as a left depth tree, and optimisation is achieved by reordering the join operators. An n-way pipeline join can be evaluated by using a single n-way join operator or by a series of binary operators. A series of binary operators may require multiple scans on the intermediate results, where the indexing costs increase the total cost of query evaluation. It is also important in the case of query evaluations, where producing initial results early is important. It also provides an ideal case for evaluation of non-blocking algorithms, which do not stage the data either on the disk or in the memory.

A CACQ expressed as a pipeline join over the streams is the most simplistic query type and has been studied in other projects (Chen, DeWitt, Tian, and Wang 2000).

8.3.2.2 Star queries

A star query is represented as a join with a single attribute in one of the participating tuples. A typical star query can be expressed as $(R_a \bowtie_b S) \bigcap (R_a \bowtie_c T) \bigcap (R_a \bowtie_d U)$. A star query representation is well suited for single hash join operators. A single shared attribute across the joins results in reduced indexing costs across the join operators. A star query also reduces the scan and indexing costs between subsequent join evaluations sharing a tuple. The star query representation is of particular importance to the study of n-way stream operators, as it presents unique challenges for an operator scheduling system.

8.3.2.3 Cyclic queries

Cyclic queries represent a special class of queries, where the query graph between the join operators assumes a cyclic order. Figure 8.2 (d) represents one such example of a cyclic query operation. The join ordering in a cyclic query evaluation needs to ascertain the ordering of the joins and the concurrency of the join evaluation.

8.3.2.4 Cross products

In addition to the join operators, query plans are also expressed in terms of logical operations between the tuples. Considering that all the possible logical operations can be expressed as a combination of conjunctive, disjunctive and negation logic (AND-OR-NOT logic), the discussion is restricted to these three primary types. A logical operator differs significantly from a join operator in terms of computation and memory utilisation, and hence is included in the present investigations.

The above sections introduced the breadth of different query types that will be used to evaluate the effectiveness of the approach introduced in this study. The different query types presented in this section are representative of the wide variety of query types that are frequently encountered in relational query processing. However, it should be pointed out that most of the investigations in the area of stream data processing have been limited to the pipeline query evaluation. Thus, the comparative studies presented in the following chapter primarily focuses on this particular query type.

8.3.3 Bursts of data arrival

A stream represents an infinite sequence of data items that are evaluated as and when they arrive. The actual rate of arrival of the data items determines the contents of the query windows activated on the stream. The cumulative memory utilisation of the streams is directly proportional to the difference between the arrival and departure rates of the data items. If the cost of a query evaluation is directly proportional to memory utilisation, prioritized processing of the streams with higher data rates will result in reduced query processing costs. In addition, the cumulative delay in response to data arrival at a stream can be reduced if the tuples are served in order of arrival and at a rate comparative to the arrival rate of the data items. The query processing between subsequent evaluations can also benefit from the partial caching of the intermediate results. The above three desired characteristics can be achieved by processing the streams in increasing order of data arrival rates.

A way to ascertain the bursts in the arrival of data is to use self similarity techniques to predict arrival rate similarities and attempt to reduce the costs by using historically effective techniques.

8.3.4 Cost metrics

The effectiveness of query optimisation is adjudged according to its capability to reduce certain costs against particular cost metrics. Traditionally, the performance of a query processing system is compared on the basis of memory utilisation, computational resource utilisation and response time. However, in most cases, the query evaluation lasts for a finite amount of time, and, as variations in the underlying processing environment are limited, it allows the approximate costs of the query plan to be estimated in advance. In case of query processing for streams, the total resource utilisation is accumulated over a period of time and it is necessary to calculate the costs of replanning and migration to an alternative query plan.

One can use the traditional definition of memory utilisation and represent it as the measure of the storage space required by the query evaluator. However, streams differ significantly. In this case, the memory utilisation increases on arrival of additional data items. As a result, a modified measure is used that represents memory utilisation as a cross product of memory utilisation x time. The modified definition implies that memory utilisation increases linearly on the arrival of data items.

Consider a query Q, that represents an n-way join between streams in the set $S = \{S_1, S_2, S_3, ..., S_n\}$. Let each of the streams S_i contain a number of data items $|S_i|$, where the delay of each processing item is δ_j. The cumulative memory utilisation of the query is given by the following equation:

$$memoryutilisation(Q) = \sum_{i=0}^{n} \sum_{j=0}^{|S_i|} S_j \delta_j$$

Note, in the case of multiple query optimisation, the memory utilisation is reduced by proportionally sharing the cost of retaining a tuple across multiple queries.

While the computational resource usage of the query evaluator is calculated as the utilisation of the individual query operators, the average response time is calculated as the average delay between the input and output of the data items from the stream operator. Computational resource usage is directly related to the choice of the join algorithm and the concurrency of the various join operators. The details of computational resource usage are discussed in section 8.4, here represented as *computational cost.* The objective function of the query optimizer is thus represented as:

$$Optimize(Q) = Min(memoryutilisation, Computationalcost, Cumulativedelay)$$

As the overall optimisation is expressed as an explicit function of a time varying parameter, namely memory utilisation, the optimisation equation represents a time varying entity, and needs to be optimized over an interval. As discussed in Chapter 2, the above equation for optimisation is solved for the finite and infinite horizon; the trade-offs are explained in section 8.4.

The above formulation summarizes the constraint resolution problem for the case of an individual query optimisation and can be used to compare the effectiveness of the approach against a theoretical optimal. However, most practical systems are designed to perform simultaneous evaluations for a number of queries. These systems operate under certain resource constraints. The optimisation in this case needs to maximize the cumulative processing of a number of queries. In the previous case, the effectiveness of the approach can easily be determined for the cases where the theoretical optimal is known. However, in most practical cases, calculating the theoretical optimal will have additional cost implications. An alternate way to measure the effectiveness of various approaches is to consider resource utilisation on a fixed set of resources. Hence, the above constraint resolution problem can be expressed as an optimization problem, that given a fixed amount of memory and computational resources maximize the number of concurrent query evaluations.

Let $Q_s = \{Q_1, Q_2, Q_3, ..., Q_p\}$ represent the set of queries that are allocated to a scheduler, with a fixed amount of computational resources. In this case consider the design of a scheduler that can ascertain the cost of additional queries and the effect on the cumulative quality of service[1].

A set of refined requirements on individual system components were described in the above section. The following sections, start with a description of the query optimisation algorithm and proceed to describe the developed SPJ algorithm and scheduling and monitoring infrastructure.

8.4 Query optimisation

The query processor for the data streams, as designed for the system, consists of following stages of processing: query plan logical generation, a physical plan generation, the query scheduling and query monitoring. These stages describe the overall processing cycle for processing the query over multiple streams. A planning approach is used for optimizing query evaluation for streaming data. The queries submitted in the format expressed in section 8.2.4, are translated to a query graph, composed of data nodes, as the leaf nodes and non-data nodes, representing the operators. Initially there is no optimal solution, but the query evaluation performance is gradually improved over a period of time. The query plan starts with an initial state and an optimal solution is formulated as a transition from the present state to the new desired state. A dynamic programming technique utilizing the statistical information from the query evaluations is used to determine transition to the alternate state of the query plan.

The logical plan generator accepts the query string and creates an access plan that happens to be the dynamic graph. The graph representation of the query plan has been used in various previous formulations. There are two predominant forms of graphical representation of the query plan. The first is a representation proposed by (Avnur and Hellerstein 2000) in which the data tuples and the operators act as the nodes. All the data tuples are leaf nodes of the query tree, while the relational operator forms the higher-level nodes of the query tree. A second representation proposed by (Lee, Shih, and Chen 2001) represents the joins as the nodes in a query tree and the inter-operator communications are labels along the edges. The former representation of the query plan is used to help create a dynamic graph. For the purpose of clarity, cost metrics are not immediately introduced, nor the statistical information available for the individual streams. Instead the focus is on the data structure used by the logical plan generator to communicate a list of feasible plans to the physical plan generator. In the case of relational systems, the query plan generator usually selects a single most likely plan, estimated to give the best performance. However, in the case of the streaming data, the weights associated with the nodes and edges of the graph change so infrequently that it

[1] Need to provide an appropriate mathematical representation of the problem

is not feasible to recalculate the entire optimisation space for each cycle of optimisation. In this case, the entire set of solutions that are technically feasible and yield the desired output are retained irrespective of the current cost of calculating and implementing the execution plan. A dynamic graph representation is made where some of the edges and/or nodes may be conditionally connected to each other. The dynamic graph allows a perfect representation to capture the time varying nature of the associated edge and node weights. An example of how such a graph is created from a query string and how it is maintained is explained in section 8.4. The discussion of graph generation is preceded with a short description of the query types that our algorithm seeks to address.

8.4.1 Plan generation and re-optimisation

Rather than starting with a description of a fully-fledged graph representation of the query graph, a phased approach is used to introduce the process of creating a query graph. It starts with the description of a query tree and introduces the relevant set of notations, operations and constraints, gradually developing the notation to include the representation of the optimisation space.

A query tree is composed of two types of nodes, data nodes and operator nodes. The data nodes represent the actual tuples or, to be specific, in our case a number a streams of data. Each data node has at least one parent node. A data node has no child node, which implies that all the data nodes represent leaf nodes of a query graph. Each operator node has at least one child node and may or may not have a parent node. Although, an operator node can have multiple child nodes, an edge connecting a node to its child node is referred to as the input and the edge connecting a node to its parent is referred as the output of the node. An operator node can have multiple inputs, but it can only have one output. All the edges are directional. For example, for any given node, an input edge is an incoming edge and an output edge is an outgoing edge. A directional edge graph for one such query is represented in figure 8.3.

A query tree represents one of the many logical query plans. Alternate query plans can be obtained by modifying the query plan to either make it less or more bushy. A list of valid query tree transformations have also been discussed by Getta (Getta 2000). Similar operations are equally applicable to the query tree, though the specific technique of reduction of data tuples may not be applicable to streams which encounter bursts of data.

Nodes and edges in the query tree are labelled with statistical, relational and performance observations at each of the nodes. For the purpose of brevity, the discussion of the labels is deferred to section 8.5.5. By the definitions of certain operators, a few transformations are uniquely applicable to the streaming data. For example, the structure of the probes for a streaming database allows close association between the data operator and the

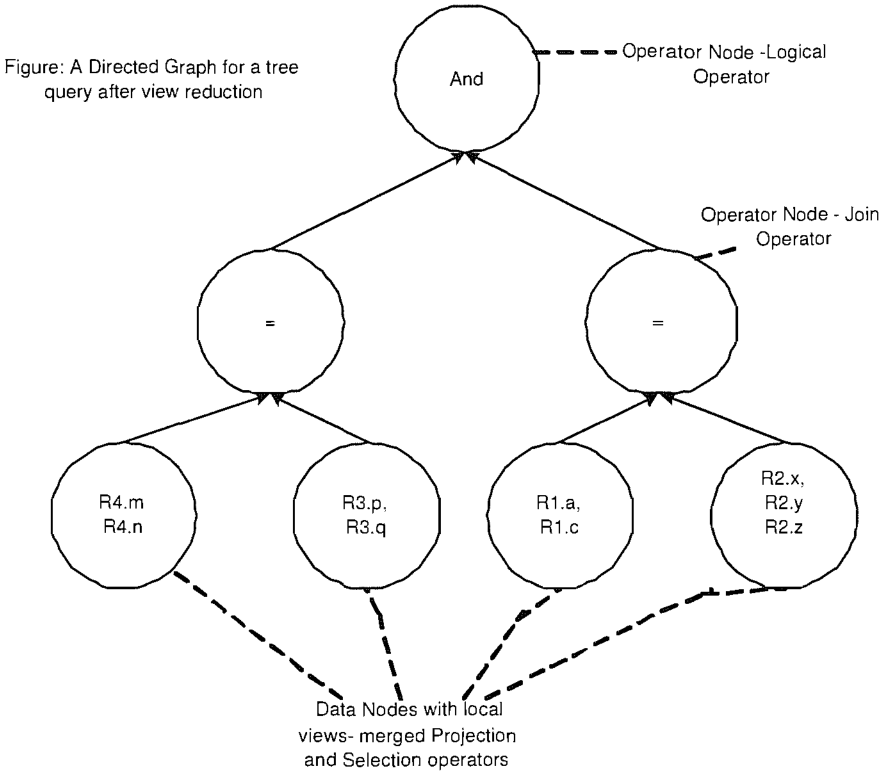


Figure: A Directed Graph for a tree query after view reduction
And
Operator Node -Logical Operator
Operator Node - Join Operator
=
=
R4.m R4.n
R3.p, R3.q
R1.a, R1.c
R2.x, R2.y R2.z
Data Nodes with local views- merged Projection and Selection operators

FIGURE 8.3: A typical query tree

selection operation. In this case, rather than describing selection and projection as a separate operator, merger of the two nodes is allowed and the resultant node has modified labels to reflect the merger of the two nodes. The resultant node is still considered to be a data node, though its tuple output is superseded by the local view, constructed on the basis of selection or projection.

Considering that the scope of stream operators is limited to the select, project, join, and logical operations. A query tree for this restricted set of operations has leaf nodes represented by the local views that are described as by transformation of the stream and the select project operators, while the non-leaf nodes represent the join and the logical operators. A transformed query tree is shown in figure 8.4.

A logical query plan forms the basis for generation of the physical query execution plan. At this point, it may be pointed out that the optimisation of a query plan is a continual process throughout the lifetime of the query. As an example, consider a simple graph that involves a join operation between two streams. The output of the join operation acts as the input to its parent operator. Figure 8.5 represents the logical plan and the possible physical plans. Edges of similar colour are dependent, and can only exist in pairs. The graph represents three possible locations of the join operator. The operator can be collocated with either of the tuples or at the parent node of the join operator. The figure represents three feasible plans that ensure that the output tuple of the logical

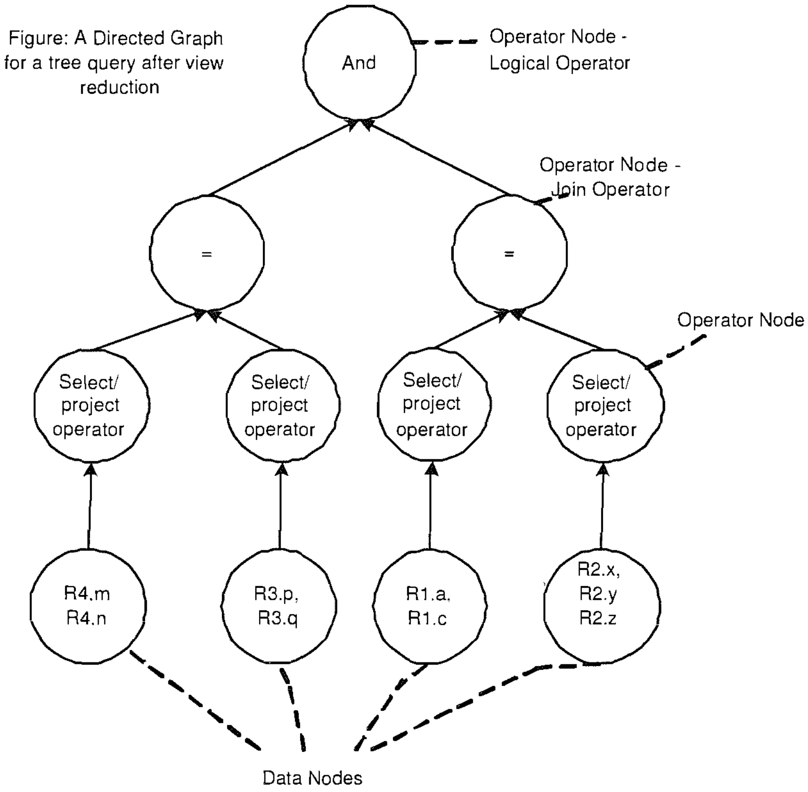


FIGURE 8.4: A typical query plan

plan remains unaffected, due to the selection of either of the physical plans. In addition to the operator semantics, the choice of either of the plans will in turn depend on the objective function of the query evaluation, the rate of data arrival at the tuples and the rate of data that need to be transferred between the tuples.

8.4.2 Query re-optimisation

The above subsection restricted the discussion to a most primitive query tree. This subsection uses induction to prove that the same result can be applied to the entire query graph. This subsection steps through the algorithm as per our discussions in previous section and begins with the tree query example. Figure 8.4 shows a logical query plan generated by the query parser. It use a physical transformation to generate a list of access plans for the query tree. The hints in the query specification are used as a policy to improve the responsiveness of the query executor.

The algorithm uses a depth first search technique to identify the operator at the highest level. The local optimisation problem is solved in accordance with the process described in the previous subsection. A virtual node replaces the sub graph in the parent graph and the attributes and subsequent labels are calculated for the virtual nodes. Each virtual node in turn acts as a virtual data node to the higher levels, as the tree is gradually reduced by successive application of sparsification (Eppstein, Galil, Italiano, and Nissenzweig 1997). During sparsification, at each stage, the tree is further optimized for performance characteristics. The type of transformations applicable after each reduction are discussed in subsection 8.4.2 and 8.4.2.2. The technique of sparsification produces a number of nested certificates that can then be monitored individually to determine the extent of the re-organisation required in response to change in the dynamic behaviour of the operators. The aim of introducing a tree of sparse certificate is to reduce the amount of reorganisation that may be required in response to bursts of data streams. When the data rate at an operator changes, the operator invokes a certificate recalculation. In response, the local optimal is recalculated and the request is propagated higher in the sparse tree. The propagation of the sparse certificate is terminated if the change in the lower level certificates does not result in significant changes at the higher level of the query tree.

Though the above reduction was discussed in the context of operators capable of processing two streams, the approach is equally applicable to multiple join operators. M-Join and Eddy are two such operators for join operations on multiple streams. There are two ways to incorporate the join operator, either in the logical query plan or during the reduction of the physical query plan. Introduction of a multiple join operator in the query plan increases the combination of the query plans applicable in the reduction. A plan diagram for one such case of 3 way join is represented in Figure 8.5 below. As in the case of the two way join operator, reduction of the multiple join operator results in a single virtual node in the reduced graph.

The selection of a multiple join operator in a physical query will be equivalent to selecting query plan 3, as represented in Figure 8.5A. A physical plan based reduction makes use of statistical information to estimate whether there is an effective advantage of using the multiple join operator. Query plan 3 is selected when the join selectivity is greater that unity. However, in certain cases, where a multiple join operator may exist, an optimum choice is to select query plan 3 irrespective of the join selectivity. If operator selectivity is lower than unity, the choice of either Plan 1 or Plan 2 reduces the overall flow. However, when the certificate is calculated, the overall memory requirements and the delay are bound to increase. While an increase in memory requirements is attributed to the fact that a queue is maintained at two different operator locations, the introduction of an additional processing element increases the overall delay of processing the query. The trade-off with the cost is the determining factor for selection of a multiple join operator or a two-way join operator. Subsequent to the selection of a operator, the sub-tree is subjected to flow optimisation within the sub-tree.

Recursive application of the sparsification based algorithm leads to the formation of a physical access plan. At this point, some properties of the logical and physical plan are highlighted. In both the logical and the physical plan, the edges are directional. In a logical plan, the edges are from a child point to a parent node, and there are no

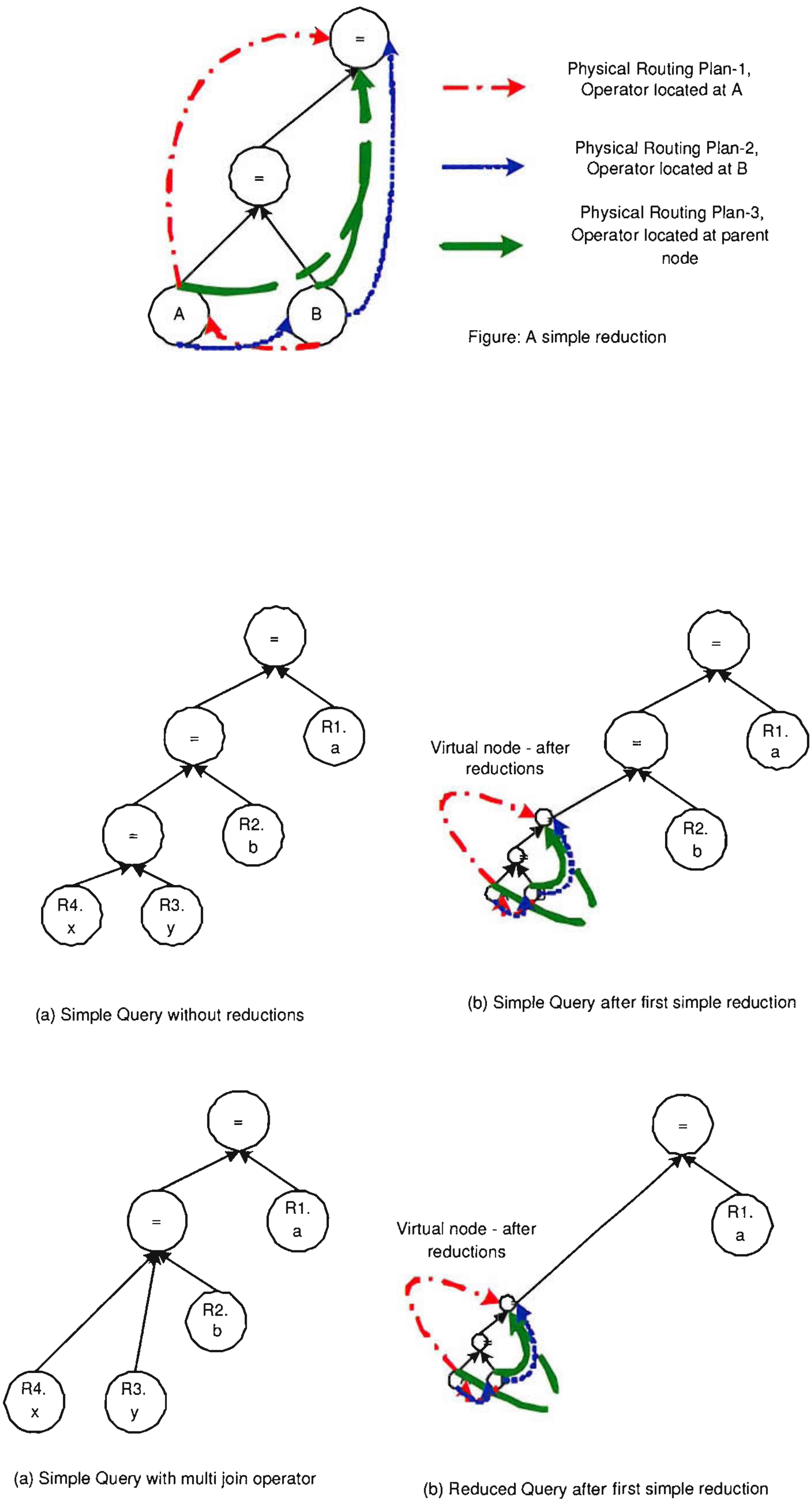

=
=
A
B
Physical Routing Plan-1, Operator located at A
Physical Routing Plan-2, Operator located at B
Physical Routing Plan-3, Operator located at parent node
=
=
=
R1. a
R2. b
R3. y
R4. x
Virtual node - after reductions
R1. a
R2. b
Virtual node - after reductions

Figure: A simple reduction

(a) Simple Query without reductions

(b) Simple Query after first simple reduction

(a) Simple Query with multi join operator

(b) Reduced Query after first simple reduction

FIGURE 8.5: Reductions for different query types.

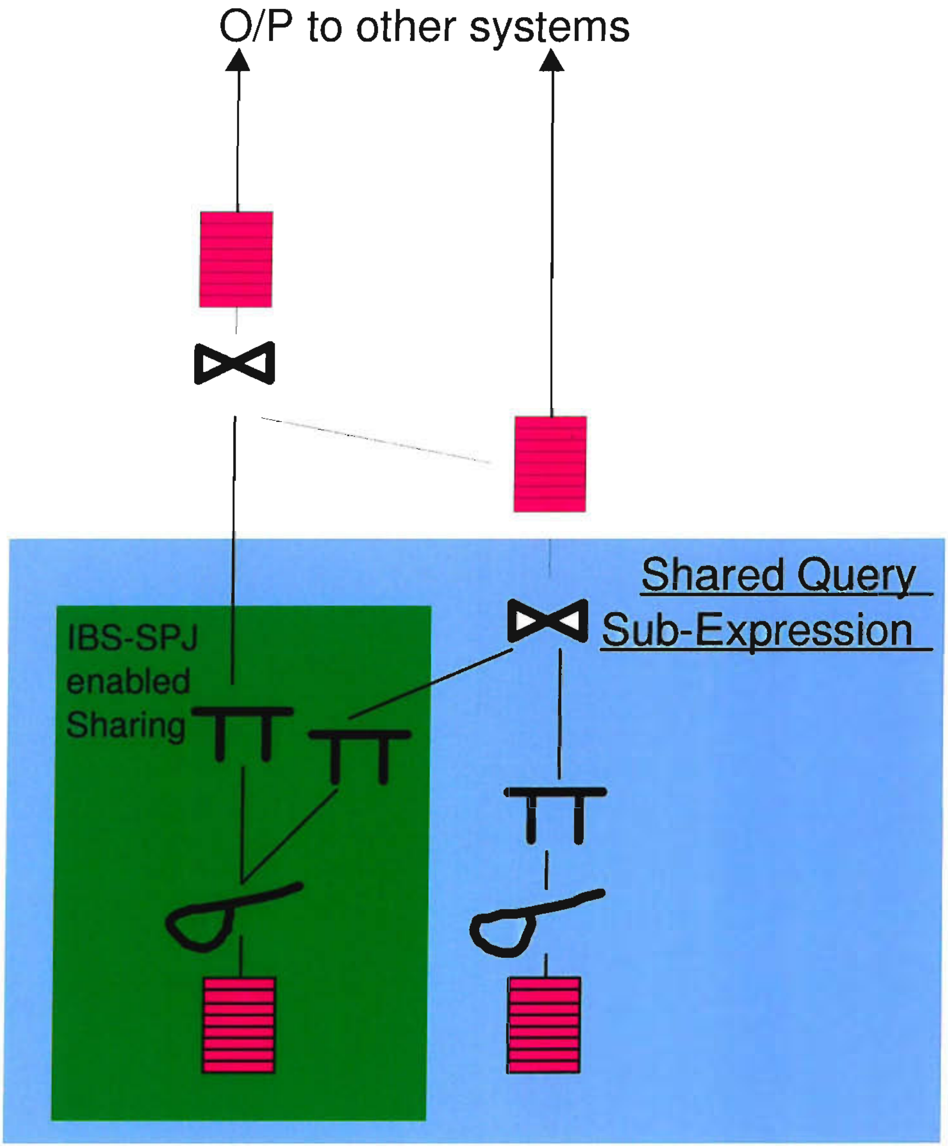


FIGURE 8.6: A typical operator flow generated by the planning algorithm.

edges from the parent node to the child node. However, in a physical plan, the edge direction is not restricted. A directed edge can connect a node to its parent node or to its sibling at the same level. While in a logical plan all the edges are static, the edges in the physical plan are dynamic and are interdependent. Unlike the pure dynamic graph, where all the edges can coexist, the edges in the physical query plan are related by the principle of mutual exclusion. In the above figure, only one of the three sets of plans can be selected. This implies that, if a dynamic graph structure is maintained to represent the above scenario, the graph algorithms should be able to support insert, update and delete operations for multiple edges. The dependency between the edges necessitates modifications to a dynamic graph MST maintenance algorithm to allow conditional selection of the nodes as the edges are considered.

8.4.2.1 A special case of n-way join

The previous section described the algorithm that generates an initial plan for enacting the query. Execution of a physical plan involves translation into sub-queries on individual streams and appropriate allocation of the operators. This sub-section assumes the existence of a physical plan for a simple logical query and discusses the process of re-optimisation of query. Consider the following figure, which represents the join between three streams. Figure 8.5 shows the initial location of the operators, their selectivity estimates and the rate of arrival of data. It is assumed that the operators are located at nodes A and C. It is also assumed that the plan was generated under the conditions that $[rateB < rateA]$ and that $[rate(AB) < rateC]$. Consider that at some time instance $[rate(AB) > rateC]$ is consistently true. Hence the flow diagram for the certificate at level 1 is represented in Figure 8.5. As a result of the change in the characteristics of stream C, plan 2 is considered to be a better option at level-1, which means that the flow of data into the virtual node is considered optimal. The level 1 is thus optimized to select plan 2, with a request to recalculate the stream at level 0. The re-organisation means that level 1 requests the operator to be placed at one of the stream locations at level 0. If node A supports a multi-join operator, the node merges the join operation, or else the flow from the two streams B and C is directed to stream A. At some time step after the reorganisation, assume that the $[rateA < rateB]$ condition materializes. The change will result in reorganisation at level 0 and may require shifting of both the level 0 operator and the virtual operator to node B.

8.4.2.2 Reductions for chain query

In all the above examples being considered a simple query tree that involved a join on a single join attribute and in which the query tree structure allowed easy selection of the sub-tree for sparsification. In the special case of chain query, also known as the pipelined query selection of a sparse is not as straightforward. In a chain query, each of the data streams can be simultaneously part of two sparse certificates. As each stream is the subject of selection in calculation of either of the sparse certificates, the reduction technique is modified to represent this specific case. Figure 8.7 is a representation of the reduction for such a specific case. The algorithm starts the reduction from one of the end nodes of the chain query. In general the following reduction can be applied to any case where a data node has more than one parent node and where the tuple join criteria for the two edges of the node are dissimilar. As one of the input tuple forms a part of more than one data operator the reduction process involves a combinatorial operation involving a mutually exclusive set of nodes. As shown in figure 8.7, the operator edges are distinguished into two edge groups, with only one of them being able to form a part of the final physical plan. The initial certificates were created using a similar process as described above. The first step in reduction involves the calculation of the probable

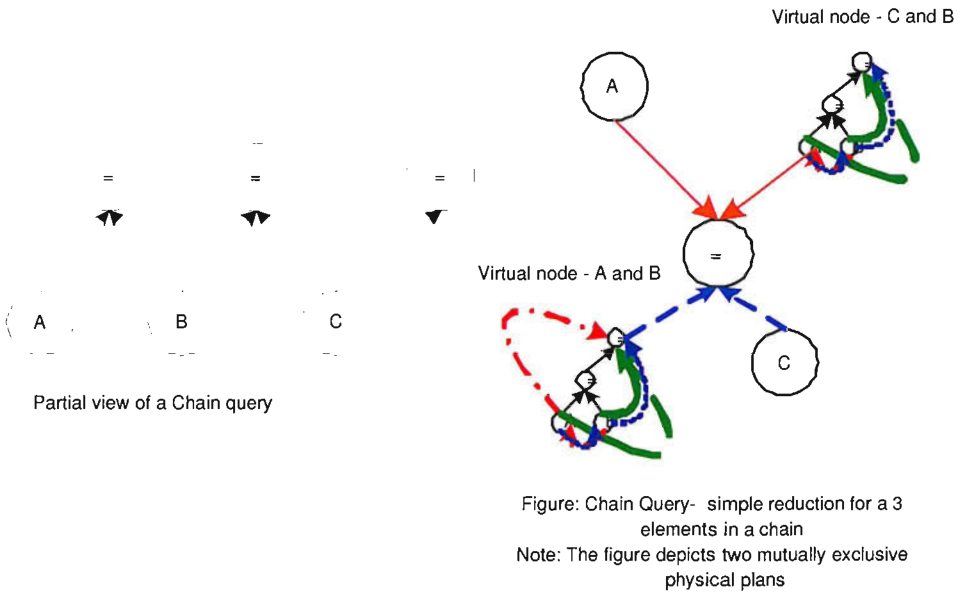

Virtual node - C and B
A
=
=
=
=
Virtual node - A and B
A
B
C
C
Partial view of a Chain query
Figure: Chain Query- simple reduction for a 3 elements in a chain
Note: The figure depicts two mutually exclusive physical plans

FIGURE 8.7: Reductions for chain query.

certificates with the two mutually exclusive sub-graphs. The reduction aids in comparing the relative merit of either of the two combinations and leads to the selection of one of the two candidate solutions. On selection, a second step of reduction, one similar to the adaptive simple query reduction as described above, generates a physical plan for directed flow between the streams. The process can then be recursively applied to the chain of queries.

8.5 Application to distributed DSMS

The above mentioned query planning techniques can be applied in the context of both centralized and distributed data management systems, for the following reasons:

Using sparse certificates allows partitioning of the query plan Each operator sees either a data node or a virtual node. A certificate represents the characteristics of the virtual node and shields the operator from the underlying complexity. These certificates can also be used as partitioning points, thereby allowing the certificates to be hosted on different machines in a distributed setting. The partitions allow independent modifications to the subtree of the query plan owned by each of the certificates.

Incorporation of the monitoring parameters without modifications The monitoring mechanism does not assume any processing architecture for any of the data processing nodes. Operator and architecture independence allows the use of the algorithm in a heterogeneous data management system.

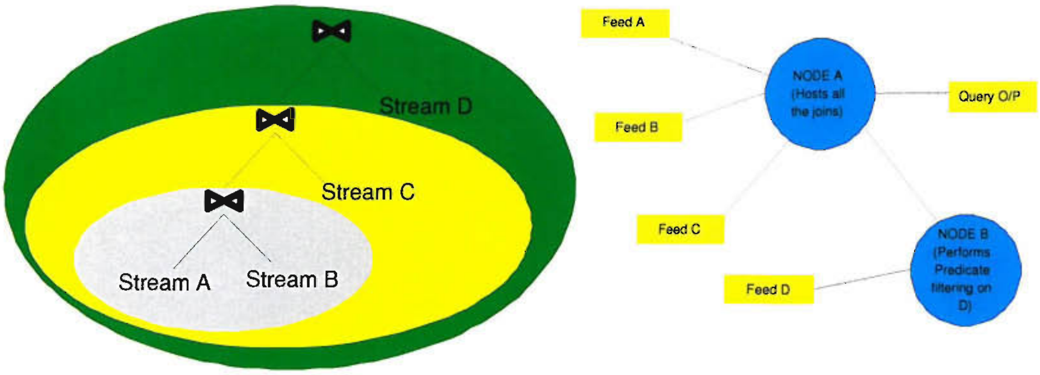

⋈
Stream D
⋈
Stream C
⋈
Stream A
Stream B
Feed A
NODE A (Hosts all the joins)
Query O/P
Feed B
Feed C
NODE B (Performs Predicate filtering on D)
Feed D

FIGURE 8.8: Sparsification based query planning applied to distributed query planning system.

Stream properties are exploited to create overlays The common sub-expressions represented by the certificates can be advertised for consumption by other nodes in the system. Use of certificates enables potential sharing of sub-expression between numerous data management systems in a distributed DSMS, as depicted in the figure 8.8.

8.5.1 Selection operator

The above specification allows the system to prioritize the order of query processing. For the purpose of clarity, it is assumed that a join over multiple streams can be scheduled with the same scheduling policy across all relevant databases.

On arrival, each query is added to the stack of queries valid for the current scheme. All the queries on the stream can be reduced to a selection, projection or self-join operation. In case of queries across streams, the query is broken down into one of the above sub-query types for ease of evaluation. The range of queries are stored in a predicate tree. On the occurrence of an evaluation event, the value is compared against that held in the predicate tree to calculate the range of the queries affected. The affected queries are notified of the data arrival and the resultant dataset after the query evaluation has been maintained by a bit-array. A termination of the query results in removal of the row, while invalidation of the data results in column elimination. The results are shipped in accordance with the operator scheduling and are maintained in a separate result structure. In certain cases the evaluation of the query may not be able to be able to cope with the amount of data that arrives in a particular stream, thereby necessitating that certain data items go unprocessed. A number of such techniques for selective probing of the incoming data have been suggested. The data is sampled in accordance with the sampling criteria, dominated by the distribution of the variable being sampled. However, in this study the sampling is not enforced and the tuples are not eliminated unless the query processor explicitly enforces the policy.

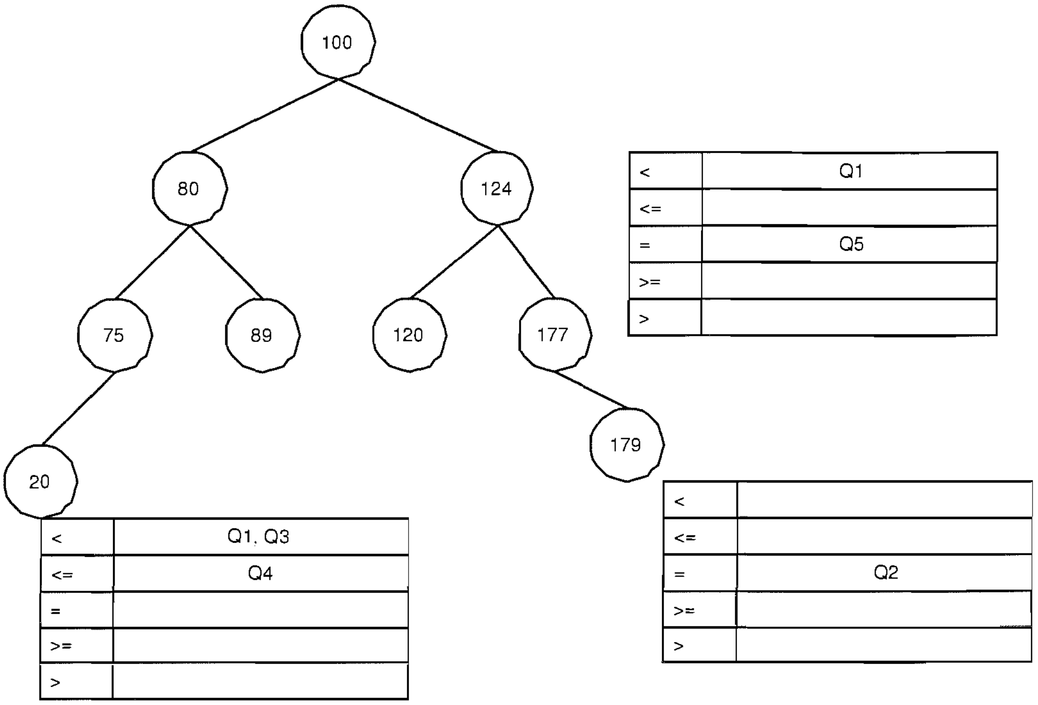
8.5.1.1 Data structure associated with the operator

In a regular database each of the query is processed individually and the selection of multiple attributes of the same tuple can be performed simultaneously. It is definitely possible to process the queries on a stream in sequential fashion. However, that would lead to increased cost, and may lead to significant memory overheads if the tuple arrival rate is too frequent as compared to the processing rate of the operator. A partial reduction in processing time can be achieved by maintaining predicate trees to manage the 'range queries'. The approach adopted provides considerable improvements in performance of queries when a single variable is considered. However, the predicate trees cannot be used to manage the 'range queries' for multiple predicates and a separate tree is maintained for each of the query attributes for a query involving a selection for multiple predicates. The probe selection can thus perform the search on each of the predicate trees to ascertain the queries that are satisfied by the appended tuples.

In most cases, the total number of attributes in a tuple normally exceeds those used as selection attributes across queries. Maintaining a separate predicate tree for each of the tuple attributes is not a viable option. The predicate tree for a particular attribute is created if and only if there exists at least one query that specifies the tuple in its selection criterion. To facilitate the mapping of queries with multiple attributes and to minimize the number of accesses required to predicate trees, a data structure is used that maintains the relationship between the queries and also prioritizes the order of attribute based selection.

The manipulations on the above data structure are described for the arrival of data and the queuing/removal of queries. When a query is queued with the probe, it is analyzed to represent the list of AND OR and NOT operators. For each logical AND expression in the query a row is inserted in the data structure and the process is repeated for the blocks connected by a logical OR operation. The rows in the data structure are sorted by the query ID. The attributes that need to be accessed to evaluate the query are set in the bit array, and the predicate tree of the attributes is appended to capture the relationship between the query and the attribute value. The attribute count (the k-value) is incremented for the attributes affected by the query. On removal of the query, all the rows corresponding to the query identifier are removed from the data structure.

On arrival of the data, the incoming tuple is loaded into the select operator, with the attribute k-value. The predicate tree of the attribute is accessed to ascertain the list of successful queries. For each query that is not satisfied the attribute condition is removed from the query list by reducing the data structure with respect to the list of queries returned by the index predicate search. The reduction only effects the queries that had the bit mask set for the query attribute being evaluated. The reduction does not eliminate the queries that are not dependent on the attribute. Successive reductions are carried out using a similar method, until no more queries remain or no more attributes



<	Q1
<=	
=	Q5
>=	
>	

<	Q1, Q3
<=	Q4
=	
>=	
>	

<	
<=	
=	Q2
>=	
>	

R.a (ka=6)	R.b (kb=0)	R.c (kc =4)	R.d (kd=6)	R.e (ke=3)	
1	0	1	1	1	Q1
0	0	1	0	0	Q1
1	0	0	1	1	Q2
1	0	1	1	0	Q5
0	0	0	0	0	Q6
1	0	0	1	0	Q4
0	0	0	0	0	Q3
1	0	0	1	0	Q7
1	0	1	1	1	Q8

FIGURE 8.9: Selection with modified IBS Tree

with predicate trees are detected. The final reduction set refers to the queries that are satisfied by the incoming data tuple. Creating this type of probe structure allows the system to express any query and join as a composition of the probes on multiple tuples. The reduction technique uses the attribute with the highest k-value to achieve maximum reduction in the early stages of processing.

8.5.2 Algorithms for select and project operators

The probe sets the bit flags for the tuples that are selected by a query. On occurrence of the evaluation event, the data items are selected from the stream and the data is sent to the selected into the output array. The cost of selection and projection is bounded by the following cost equations. Let R be the tuple with attributes R_i, in this particular example R<a,b,c,d,e,f>. Associated with each tuple is a list of queries that are stored in Q_L. Each query has a range constraint on a number of attributes of R. The query may have a number of combinatorial constraints on the same tuple, which are converted to AND, OR and NOT logical operators.

The following algorithm is used to process the queries on the

On Query queuing

This method is invoked when the query is added to a stream.

1. Add the query to the data structure.
2. Modify the k-values of the attributes affected by the query.
3. Update the query predicate trees for each of the affected attribute.
4. Modify the result data structure to add the query to the bit array.

On Query remove

This method is invoked on removal of the query from the stream.

1. Modify the result bit array structure to remove the column representing the query.
2. Modify the query data structure and remove all the rows for the query.
3. Modify the k-values of the attributes in the query data structure and destroy the predicate tree, if the count is zero.
4. Modify the predicate trees to remove the query entries and prune the tree.

On Data process

This method is invoked to process every incoming data tuple, with the result that the data structure is modified to reflect selection of the tuple.

1. Select the attribute with highest k-value.

2. While the query map is not null, perform reductions

   Use the attribute value to obtain the list of affected queries.

   Perform reduction of the query list.

   If query list is exhausted, then terminate the processing, the row will not appear in the result bit array.

   Select the next attribute with the highest k-value.

3. Append the result bit array for all the queries existent in the query list.

8.5.2.1 Cost of processing the query with this probe

Only the computational and memory costs need to be considered, as all the data structures and stream are considered to reside locally. Assuming that all the data is maintained in the main memory, the I/O costs will be of negligible significance, and the entire complexity of processing will be due to computational complexity.

Consider that the queries are processed sequentially without any attribute based reduction. The cost of processing a single tuple for all the queries can be given by:

Summation (*log(n)* * no. of attributes accessed) for all the queries. = *k*q*log(n)*

Considering the reduction based technique the cost of processing the query can be given as

Summation(*log(n)* * no. of queries remaining) for number of iterations < *k*q*log(n).*

The overall complexity in both cases is of the form *nlogn*, but in the latter case the reduction leads to lower complexity; the worst-case complexity may be equal to the complexity of the first case.

8.5.3 Multi-way join operator

Multi-way join operators provide increased memory utilisation by eliminating the need to maintain intermediate results, and may also reduce the computational costs in the case of non-blocking data stream processing. A multi-way join operator processes the incoming data items on individual streams. As illustrated in section 8.2.4, a stream operator can schedule the order of routing the tuples through the individual joins, in order to reduce memory and computational costs. The cumulative memory utilisation of the query being processed by the multi-way join operator is given by the equation:

$$memoryutilisation(Q) = \sum_{i=0}^{n} \sum_{j=0}^{|S_i|} S_j \delta_j \tag{8.5}$$

where Q is the query represented by the join operator, and the streams are represented by the set S. In addition to the memory utilisation of the incoming streams the Multi-way join operator may also exhibit some additional memory utilisation, due to the caching of the intermediate state and results. The previous sub-section illustrated a sparsification and iterative dynamic programming based approach to determine the next feasible minimal state for query evaluation, which is extended here.

Consider a query Q that represents a multi-way join between streams S. Assuming that there are no intermediate states, then each tuple arriving at any of the streams needs to be routed through $|S| - 1$ other streams. Let us represent the route of each of the tuples arriving at stream S_i with a directed graph G_i. As G_i is the ordered graph representing tuple routing and as there exists no parallel execution, the graph G_i is acyclic, and each node in the graph has at most one incoming and at most one outgoing edge. The nodes in the graph are ordered to minimize the flow of number of tuples at each hop in G_i. The ordering of the nodes in G_i, each ordered graph G_i is selected on the basis of selectivity estimates. S_i is by default the first node of graph G_i. The second node is chosen from a list of the nodes $(|S| - 1)$, the third is chosen from the list of $(|S| - 2)$, and so on. The objective function used to iteratively select these nodes in graph G_i is given by the formula:

$$\sigma(N) = min_{nodes(|S|-j)atstage(j+1)}\{\sigma_{(|S|-j)}\} \tag{8.6}$$

Here N represents the next node to be added to graph G_i, and the process is repeated for $(|S| - 1)$ times.

Let G represent the union of all graphs G_i, such that G $= \{G_1, G_2, G_3, ..., G_i\}$ and $|S| = |G|$. Graph G is a directed graph with $|G|$ directed spanning trees. The limitation of the previous join operator was that only one tuple could be allowed to advance its scan in order to maintain the consistency of the ripple join operation. Using this graph formulation could introduce concurrent execution of various scans by selectively locking the routing paths.

Consider that query Q selects a scan on item S_i. As the scan is on stream $|S_i|$ locks the tree rooted at stream S_i. The multi-way join operator selects a list of alternate streams that need to be scanned at the completion of the current routing path. All such candidate streams are queued for execution. For the join operations in progress, the streams are routed through the path represented by the spanning tree. As the items are routed from node N_i to node N_j, such that $e_{i,j}$ exists in G_i, the lock on the node N_i is released. Thus the candidate node N_i is available for the scan of its tuples and advances in its direction are permitted. Selection of this second node allows it to lock the other nodes in accordance with its query graph G_i.

8.5.4 Operator scheduling

Query processing for streams is represented as a set of graph flows, which represents the order in which the data from the streams is processed. The last sub-section introduced a mechanism for concurrent execution of scans from multiple streams. However, the data items should not be considered as individual tuples processed on arrival; instead they are considered as blocks of tuples processed as blocks of memory units. The block size and ordering of the stream scan are selected in order to reduce cumulative memory utilisation. Here it may be pointed out that the memory utilisation of the stream is not dependent on the execution of the simple query, but is intertwined with the operations of the other query. Operator scheduling has two distinct objectives - first, to reduce the cost of individual query evaluation and secondly, to dynamically arrange the ordering so that the global costs introduced on the data items are also minimized.

A number of strategies can be adopted in ordering the scans on the streams of an individual query. The approach of cost based reduction is investigated. The memory utilisation of a particular query Q was described in equation 12.5. Assume that the query processor can evaluate the tuples that occupy size M. The reduction achieved in the complexity of the query is as follows:

$$memoryutilisation(Q) = \sum_{i=0}^{n-1}\sum_{j=0}^{|S_i|} S_j\delta_j + \sum_{j=0}^{|S_n|} S_j\delta_j \tag{8.7}$$

The scan on S_n should be maximized in order to reduce the overall memory utilisation of the query. The objective function in that case is given by

$$F = Max\{\sum_{j=0}^{|S_n|} S_j\delta_j\} \tag{8.8}$$

In cases where the query manages to process all the data items pending in the stream the resultant memory utilisation is given by

$$memoryutilisation(Q) = \sum_{i=0}^{n-1}\sum_{j=0}^{|S_i|} S_j\delta_j \tag{8.9}$$

while in cases where partial processing has occurred, the resultant is given by

$$memoryutilisation(Q) = \sum_{i=0}^{n-1}\sum_{j=0}^{|S_i|} S_j\delta_j + \sum_{j=0}^{|S_{n1}|} S_j\delta_j \tag{8.10}$$

where $(n1 < n)$. The formulation presented in equation 8.8 represents the objective function of the operator scheduler. Adoption of the cost based scheduling strategy usually is liable to the starvation effect. However in this study, the objective function has a temporal component, due to which the resultant costs increase with the passage of time, there by necessitating the evaluation of all the streams and eliminating any chances of starvation.

Here it should be pointed out that temporal costs of query evaluation are liable to occur in the streams that experience arrival of new data items and in the windows that roll-over the data. The temporal costs are inapplicable to data items once they have been processed. Thus, streams with lower data rates as well as static tables do not incur additional costs for retaining the data items collected as a result of the previous scans.

Multiple queries share the cost of maintaining the data in the stream. This cost of data items is shared between multiple data queries. As the scope of the queries is limited to the scope of the window, a window moment may invalidate the requirement of maintaining certain data items within the stream. A resultant window moment results in spreading the cost of the data item between the remaining queries. Such a resource management feature is required to allow variable cost mechanisms for various window mechanism. The actual cost function can be adjusted in accordance with the memory management policy. The policy will affect the way in which queries are evaluated. Thus, if considered, an equi-cost representation for evaluation of the memory costs, it can be proven that the landmark window functions are penalized as compared to the sliding window or the clamped window functions. The aim in that case is to reduce the cost of the query function; thereby, certain queries will receive higher prominence in the order of processing. The effects of operator scheduling and related effects on cumulative query processing are studied in section 9.2.

8.5.5 Statistical information collected

To determine the appropriate ordering of data operators, and prioritize the queries, a DSMS maintains statistical information about the intermediate data items. To assure the extensibility of the statistical monitoring environment, it is required that the monitoring of the data operators is independent of operator characteristics, an objective achieved by the use of a black box model to denote operators.

The model collects fine grained information on the flow and size of tuples, as they happen to govern the amount of memory required at each operator. It inferences meta-level delay information based on tuple and flow characteristics. The flow information can also be used to infer selectivity information on the historic data flows observed by the operator. In addition, the instrumentation monitors resource sharing between multiple queries, being enabled by the operator.

The following tables provide details about the statistical information collected.

TABLE 8.1: Stream Query Processing - Tuple and Stream Instrumentation Details

Parameter Name	Symbol	Cardinality	Description
Arrival time	T_a	Data tuple	A timestamp to capture time on arrival
Selection time	T_s	Data tuple	A timestamp for the tuple after it has been processed.
Shipping time	T_{sh}	Data tuple	A timestamp to indicate when a tuple was shipped to the next operator.
Arrival rate	S_a	Stream	Number of tuples arriving per second.
Maximum cache size	M_a	Stream	Average memory requirements for storing the data in the stream.
Local views	N_L	Stream	Total number of select/project queries being supported by the probe on the stream.
Scheduling	S_c	Stream	Type of operator scheduling at the stream.
Window type	W_t	Stream	Used to specify the semantics of the window.
Tuple structure	T_s	Stream	Used to specify the schema of the stream.
Tuple size	T_{si}	Stream	Size of a row in the stream.

TABLE 8.2: Stream query processing - Operator instrumentation details

Parameter Name	Symbol	Cardinality	Description
Input tuples	Q_{ti}	Per input @ node	The number of tuples that have arrived since the last landmark.
Output tuples	Q_{to}	Per output @ node	The number of tuples that have been transferred from the last landmark.
Input tuple size	I_{TS}	Operator	The combined size of all the input tuples.
Output tuple size	O_{TS}	Operator	The size of the output tuple.
Input tuple rate	I_{TR}	Per input @ node	The rate at which the tuples have arrived in at each of the inputs.
Output tuple rate	O_{TR}	Per output @ node	The rate at which the output tuples have been collected from the operator.
Cache size	Q_a	Operator	The cache available at the operator location.
Selectivity	S_a	Operator	The selectivity of the query operator,calculated as the (no. of output tuples) / (Cartesian product of input tuples) .

8.6 Example

A schematic representation in the following section uses an example to describe various optimisation scenarios encountered during the query processing in DSMS. The previous section introduced an IBS-SPJ operator, which provides range predicate sharing between multiple queries. This section describes the multiple query optimisation using pipelined join operators and IBS-SPJ operators. A modified join operator is represented in figure 8.10. The non-blocking join operator is augmented with statistical capability, ability to register its intermediate results as temporary streams, tuple dropping and sparse

TABLE 8.3: Parameter description and symbols

Parameter name	Symbol	Cardinality	description
Tuple structure	T_s	Edge	The structure of the tuple that is being transferred over the link.
Tuple size	T_{si}	Edge	The size of a single tuple that is being transported through the edge.
Flow rate	F_l	Edge	The rate at which the tuples are being transported.
Edge group	Id	Edge	The group of edges that need to be simultaneously inserted or deleted from the solution graph.

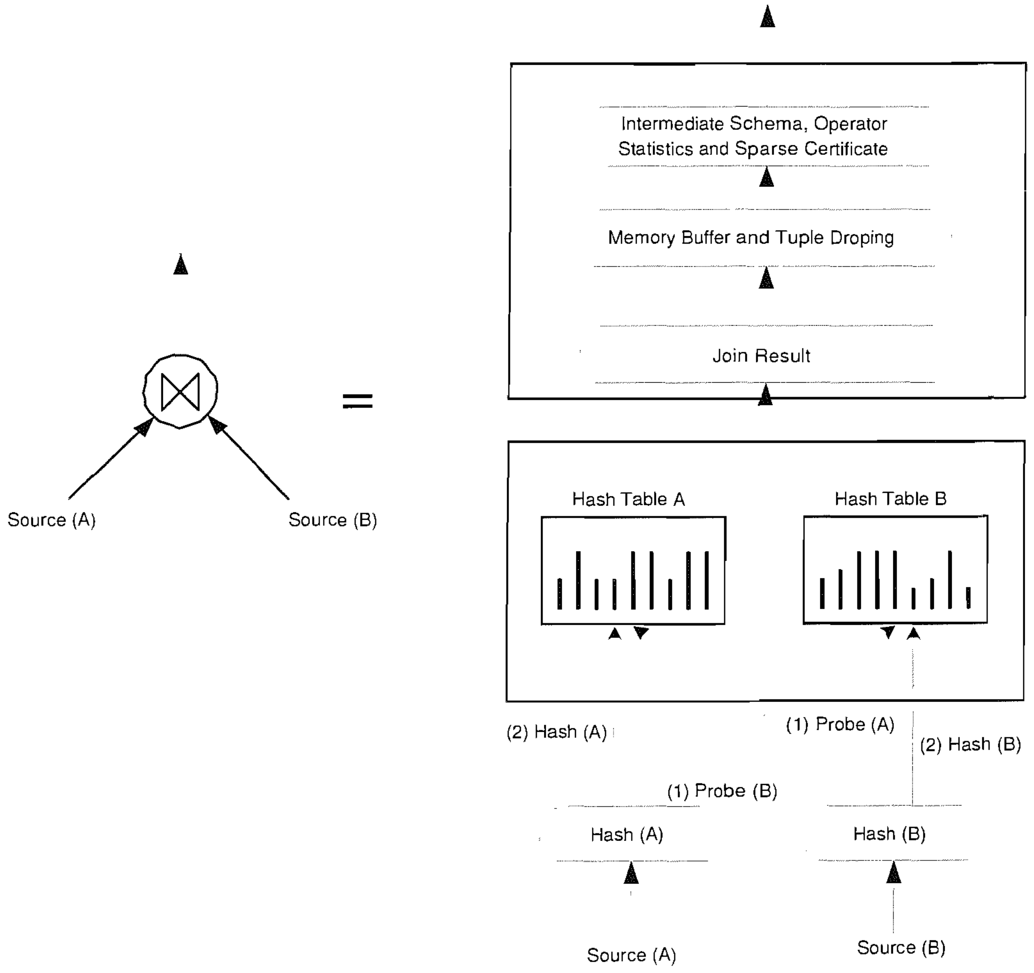


FIGURE 8.10: A pipelined symmetric hash join with monitoring information

certificate capability. The certificates specify the goals for each of the operators and are usually allocated by the parent operators to their child operators. The result of the join operators on multiple input streams is a single output stream, registered as an intermediary table for continual optimisation in conjunction with logical plans.

For this example, consider the case of the following three queries:

TABLE 8.4: Stream query processing - example queries.

Query 1 (Q1)	Select * from A, B, C, D where A.a = B.b AND B.b = C.c AND C.c = D.d AND A.a = 10 AND B.b =20 [Window Specifications];
Query 2 (Q2)	Select * from A, B, C, E where A.a = B.b AND B.b = C.c AND C.c = E.e AND A.a = 10 AND B.b = 20 AND E.e = 90 [Window Specifications];
Query 3 (Q3)	Select * from A, B, G, H where A.a = B.b AND B.b = G.g AND G.g = H.h AND A.a = 50 AND B.b = 60 AND G.g = 60 AND H.h = 7 [Window Specifications];

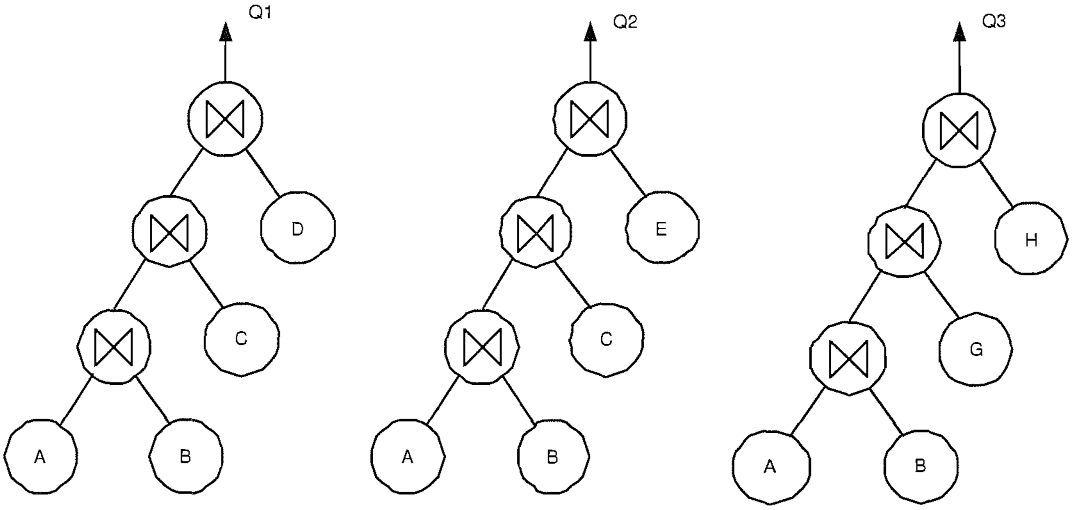


FIGURE 8.11: Example Queries

The individual query plans for each of the queries is represented in figure 8.11. As query Q1, Q2, Q3 are added to the system the logical plan is modified as depicted in figures 8.12, 8.13 and 8.14. The self-referencing edges represent the range predicate selections and are translated into expressions on IBS-SPJ operators, while the edges between the nodes are translated into join operators. An optimizer collocates a common join operator for streams that share expressions and have similar consumption rates and window specifications. The execution plans that share operators are represented in figures 8.15 and 8.16, while figure 8.17 represents the case where the execution engine partitions the resources so that a different quality of services can be met.

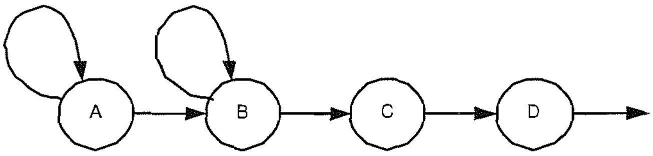


FIGURE 8.12: Logical plan for a single query.

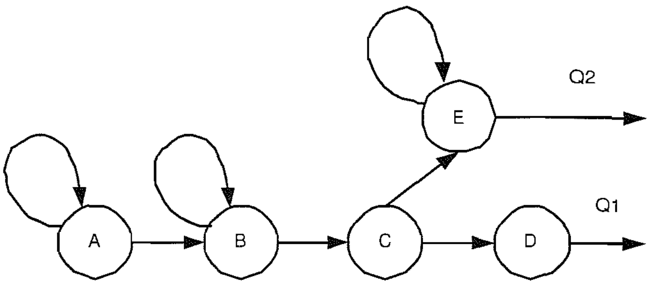


FIGURE 8.13: Logical plan for a two queries.

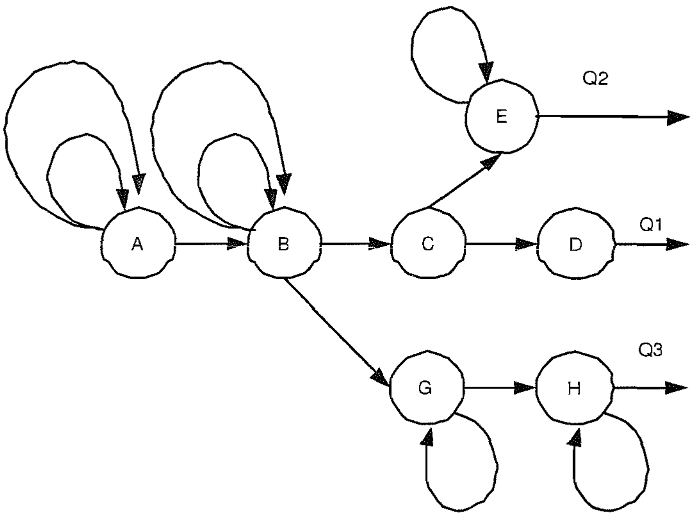


FIGURE 8.14: Shared logical plan for three queries.

8.7 Summary

Query optimizing in data streaming systems has been the focus of many recent projects, for example, StreamDB (Arasu, Babcock, Babu, Datar, Ito, Motwani, Nishizawa, Srivastava, Thomas, Varma, and Widom 2003), TelegraphCQ (Chandrasekaran, Cooper, Deshpande, Franklin, Hellerstein, Hong, Krishnamurthy, Madden, Reiss, and Shah 2003) and NaigaraCQ (Chen, DeWitt, Tian, and Wang 2000). Three distinct query optimisation techniques have been proposed: first, a tuple routing approach used by Eddy (Avnur and Hellerstein 2000), secondly, a rate-based query optimizing technique (Viglas and Naughton 2002). Thirdly, an operator ordering based query optimisation (Babcock, Babu, Motwani, and Datar 2003). The above techniques adapt to stream characteristics and optimize memory utilisation by reducing the state information maintained at each stage of query processing. However, the above techniques have limited applicability due to the following reasons:

Limited context - single query All of the above techniques attempt to minimize resource utilisation in the context of a single query. Optimizing the queries individually does not guarantee the optimal strategy for the DSMS, which typically

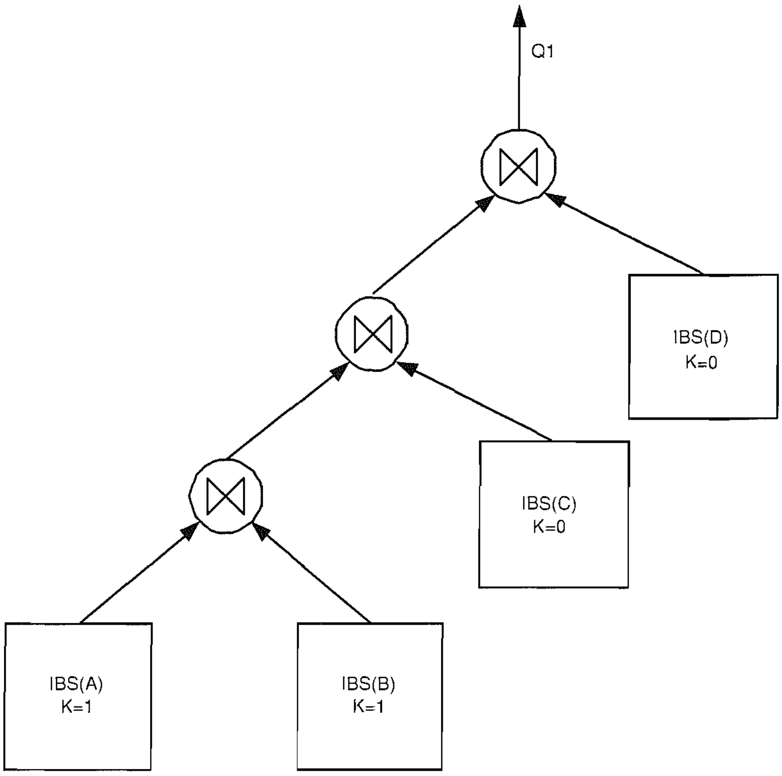


FIGURE 8.15: Execution plan for a single query.

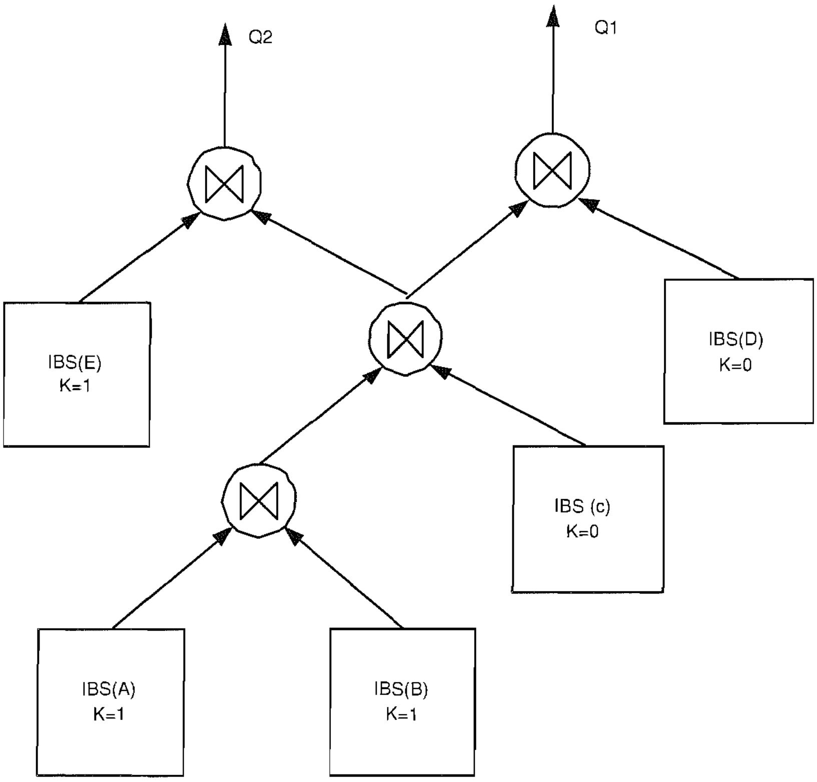


FIGURE 8.16: Execution plan for a two queries.

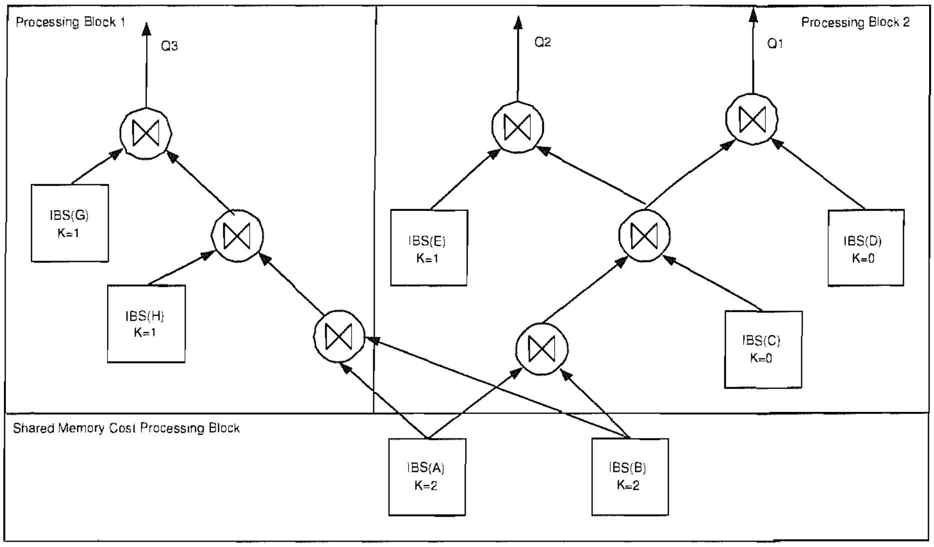


FIGURE 8.17: Execution plan for three queries, with parallelisation.

executes multiple queries at any give time. Unlike the queries in traditional databases the queries in stream databases happen to be continuous, and need to be executed over a period of time.

One dimensional optimisation Query optimisations for DSMSs have focused exclusively on either minimizing memory utilisation or improving the throughput of query evaluations. However, query optimisation needs to be based on a complex Quality of Service (QoS), based on memory utilisation, throughput and the computational resources required for evaluation. Guaranteeing a complex QoS requires multivariate optimisation an issue that has not been addressed so far.

The approach described in this chapter is the first approach to multiple query optimisation for DSMSs that overcomes both of the above-mentioned limitations. The sparsification based technique described in this chapter is the first such query processing technique that considers resource sharing between multiple queries. Resource sharing between multiple queries is detected from the query semantics specified in PSQL [2]. Resource sharing has been extended to the query planning and query execution stages. It remains the only known approach that tries to exploit the correlation between the memory utilisation, computational resource utilisation and throughput. In addition, it is the only known approach that considers re-optimisation of query plans as the integral part of query processing in DSMS.

While this chapter concentrated on the theoretical aspects of query optimisation, the more detailed implementation and practical aspects are discussed in the next chapter, which also provides a detailed evaluation of the techniques described in this chapter.

[2] PSQL was developed specifically for identifying resource sharing between query definitions in DSMS. (For details refer to Appendix C)

Chapter 9

DSMS – Implementation, Evaluation and Analysis

This chapter describes the DSMS implementation and experiments carried out to evaluate its performance. Section 9.1 describes the DSMS implementation which supports PSQL (described in Appendix C) and implements the algorithms described in Chapter 8. Section 9.2 describes the experiments conducted to evaluate performance. A summary of the findings can be found in section 9.3, which also summarizes the contribution to the application domain and correlates the application level findings and overall hypothesis from the Chapter 2

9.1 Implementation details

A DSMS was implemented for the sole purpose of evaluating the performance of the algorithms described Chapter 8. A DSMS implementation could have been developed as an extension to the open source relational database. For example TelegraphCQ (Chandrasekaran, Cooper, Deshpande, Franklin, Hellerstein, Hong, Krishnamurthy, Madden, Reiss, and Shah 2003) extends the PostGres database system. An alternative was to develop a dedicated implementation of a DSMS. The following considerations influenced the choice of the latter:

Support for streams Most relational database systems support standard containers such as tables and views. These systems provide very little support for in-memory representation of the containers such as streams. Temporary in-memory relational tables present the most suitable data structure to represent streams. However, most such implementations use secondary storage (or disk space) to swap table space, which, as discussed in the previous chapter, is not recommended for the manipulation of streams. Also, when temporary tables were used to capture the sequence-like semantics of streams, additional processing costs were incurred in removing the items from temporary storage. Automatic removal of data tuples is the significant difference between tables and streams. Data tuples in a stream expire, if not processed within a bounded interval. Such temporal characteristics are also observed by relational operators, where the resultant tuples are assigned a time-stamp derived from the time-stamps of the input tuples at an operator. The significant difference in container properties entails modifications to access mechanisms and memory management to support streams in a relational database system.

Support for PSQL PSQL extends the SQL standards supported by most relational database implementations. Incorporation of PSQL support will require modification to the syntax and the semantic parsers of the relational database system. Incorporation of PSQL is crucial in validating our tuple based resource sharing approach from an end-to-end systems perspective. Also, PSQL introduces an important notion of considering both table-spaces and streams as datasets, window operations and scheduling characteristics. Verification of the capability of the PSQL in identifying the similarity between the queries on streaming data was confirmed by a system implementation.

Unique monitoring requirements Uncertain data arrival patterns and variations in data characteristics have a significant impact on the estimation capability of the query processing system. Cost based query processing in stream data management systems requires a capability to analyze and predict the behaviour of a time varying dataset. Standard relational database systems do not provide such a monitoring system. Any attempt to incorporate such a capability in a standard database requires modifications to the query processing system.

Continual Query Optimisation A DSMS provides support for a continual query execution, while most database systems provide support only for evaluating a query instance. Optimal continual query evaluation requires support replanning and changes to the physical query plan. Such changes require additional functional capabilities to be incorporated into the query planning system.

As described above, implementing a DSMS using an existing relational data management system requires significant modification to a wide range of system components. To overcome this limitation, a new DSMS implementation was developed. The architecture of the implementation is described in the following section.

9.1.1 Architecture

Figure 9.1 depicts the architecture of the DSMS implementation. The input to the DSMS is composed of two different blocks the data definition block and the query processing block. The DDL (Data Definition Language) block deals with the creation, deletion and modification of the data schemas and data containers. The Data Manipulation Language (DML), which is also a part of the SQL specification, provides constructs for manipulation of data in standard containers such as tables and adapters are associated with the stream data containers. The implementation provisions data adapters for consumption of data from Java Messaging Service (JMS), socket based communication and synthetic data generators, with the possibility for supporting additional adapters for communication services such as CORBA Notification and HTTP communication protocol. Only those data items that conform to the specified schema are accepted from these incoming communication channels. Temporary storage is provided for staging the data items, which are subsequently evaluated for a list of continuous queries. The query processing block processes the query statements and appropriately determines the physical and the logical query plans. The query plans determine the resource sharing between multiple queries and are used to derive the schema and memory requirements of the intermediate results. The query plans are used to determine the operator scheduling for subsequent evaluation of the queries. Online statistics are maintained for the data items encountered by the queries. Our implementation provides the possibility to collect the semantics both before and after the actual query processing. The semantics allows refinements of the estimates and the query plans derived from these estimates. The output generated from the data streams also happens to be streams, and the resultant data streams are propagated using the various communication channels specified by the query semantics.

9.1.2 Illustration memory management and scheduling

As described in the previous sections a query statement is translated into a logical query plan, which in turn is translated into a physical query plan. Figure illustrates one such query graph that is formed by addition of multiple queries to the query graph. At the bottom of the graph are the selection operators that provide tuple level resource sharing between the query plans, while the output of the queries is obtained at the top of the query plan. The intermediate graph is determined on the basis of the shared query sub expressions. The cost of the shared subexpressions is shared between the queries that share the expression. In simple terms, the cost of the subtree is shared by the queries that receive the fan out from the subtree. The shared cost concept allows the operator scheduling mechanism to determine the net effect of scheduling the chains of operations within the graph.

Associated with the each node of the graph is the queue that retains the operator state. Assuming the familiar case of the sliding windows, the state maintained at the base of the graph is relatively static. However, efficiency of query evaluation depends on the ability of the scheduler to allocate an appropriate amount of memory at each of the

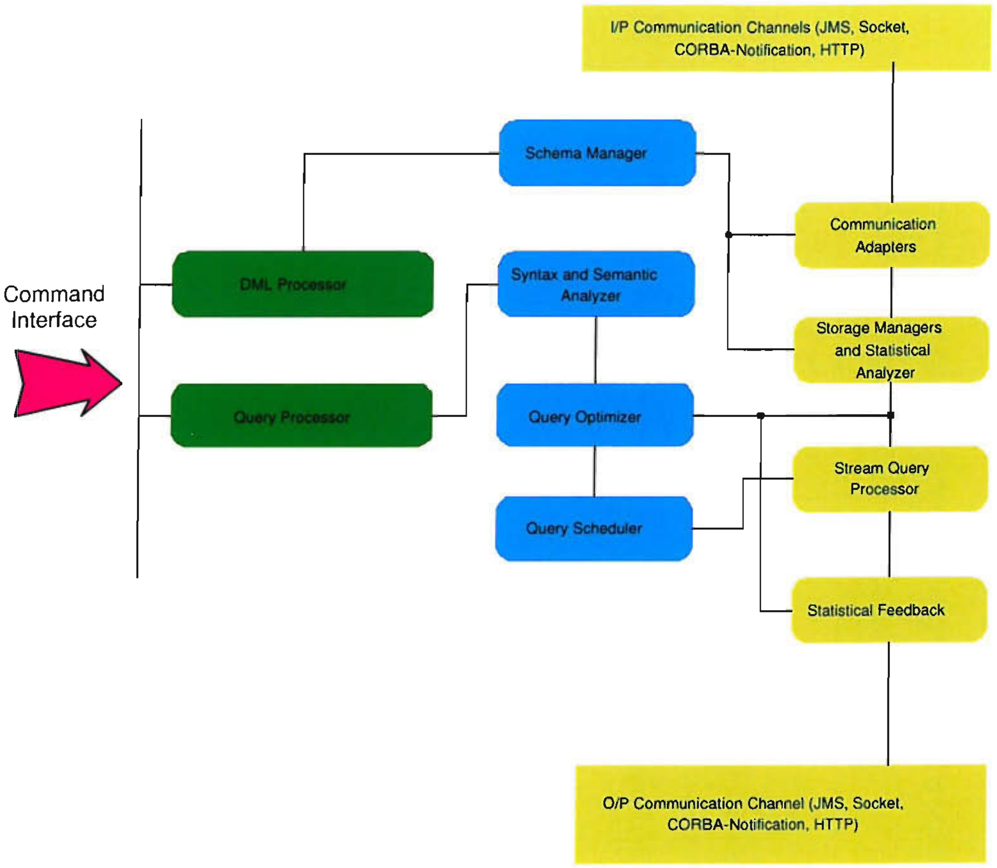

I/P Communication Channels (JMS, Socket, CORBA-Notification, HTTP)
Schema Manager
Communication Adapters
Command Interface
DML Processor
Syntax and Semantic Analyzer
Storage Managers and Statistical Analyzer
Query Processor
Query Optimizer
Stream Query Processor
Query Scheduler
Statistical Feedback
O/P Communication Channel (JMS, Socket, CORBA-Notification, HTTP)

FIGURE 9.1: Block diagram: Query Processing Engine (QPE)

subexpression queues. The mechanism was described in the previous chapter and the evaluation of the same can be found in the following section.

9.2 Experiments

The experimental evaluation of DSMS presented in this section covers important individual components and proceeds to provide the evaluation of the complete DSMS. The evaluation focuses on the IBS-SPJ operator, the memory evaluation, and the operator scheduling. These experimental evaluations were conducted using a synthetic data workload, details of which (schemas and their distribution) can be found in Appendix C.

9.2.1 Select project operator analysis

Select project operators belong to the group of stateless operators which do not maintain any state information to evaluate the query expression. Hence, unlike join operators, these operators remain unaffected by the use of different window types. The schema used for evaluating the data operator is presented in the following Table 9.1. It evaluates

Schema of the Stream: Create Stream IT (ID INT, Name CHAR(200), PanNo INT, Income REAL, Ideal INT);								
	Depth of IBS filter tree							
Number of Tuples	**k = 1**	**k = 3**	**k = 5**	**k = 7**	**k = 8**	**k = 9**	**k = 11**	**k = 13**
10000	65.5	27.4	21.866	16.8	13.76	12.566	9.3571	9.187
20000	112.6	67.1	43.233	35.175	28.76	25.733	22.771	19.525
30000	170.1	93.05	69.233	53.9	40.02	37.5	34.142	30.262
40000	223.4	133.6	89.1	72.3	56.22	48.466	45.328	39.85
50000	293.8	157	108.833	85.925	67.18	63.016	55.157	49.8125
60000	334.5	186.75	134.4	107.375	81.88	73.15	66.7	58.975
70000	389	222.65	152.6	123.1	94.66	87.516	81.271	71.662
80000	448.5	253.1	183.866	142.175	110.96	98.166	87.057	79.1125
90000	517.3	279.7	200.5	155.075	123.12	110.65	102.242	90.2375

TABLE 9.1: Performance data of an IBS based SPJ operator

the performance of the operator as a number of queries are added to the operator. The queries are either conjunctive or disjunctive or predicate range expressions. The expressions are evaluated for an increasing number of tuples. The evaluation time for the queries was averaged for 10 runs of the data with varying selectivity. Figure 9.2, represents the query processing time for varying number of queries being simultaneously evaluated by the operator. The query processing time increases linearly (see figure 9.3) with the increase in the number of tuples being processed. However, the time per query decreases as the number of queries increases. Hence, for a large number of queries the cumulative gains obtained by the use of this operator increases linearly with an increase of query expressions. Figure 9.3, which presents an alternative view, highlights the fact that the cumulative average processing time reduces due to the tuple sharing enabled by the operator.

9.2.2 Query planning under variable data rates

Adaptive query processing allows the query evaluators to be able to adapt their resource usage in response to changes in the data rates. The aim of these experiments is to demonstrate the capability of the algorithm to detect, restructure and schedule the query operators in response to variations in the data rates. The experiments compare the algorithm against the cost-based non-adaptive query planning algorithm, to show that it remains capable of adapting to variations in resource availability.

To evaluate the performance, a candidate query consisting of three streams and two join operators is used to compare the performance of the query planning algorithms. The selection and projections being stateless operators, they were not considered for performance comparison. Performance is compared on the basis of cumulative memory and processing requirements. Historical statistics maintained on the container were used

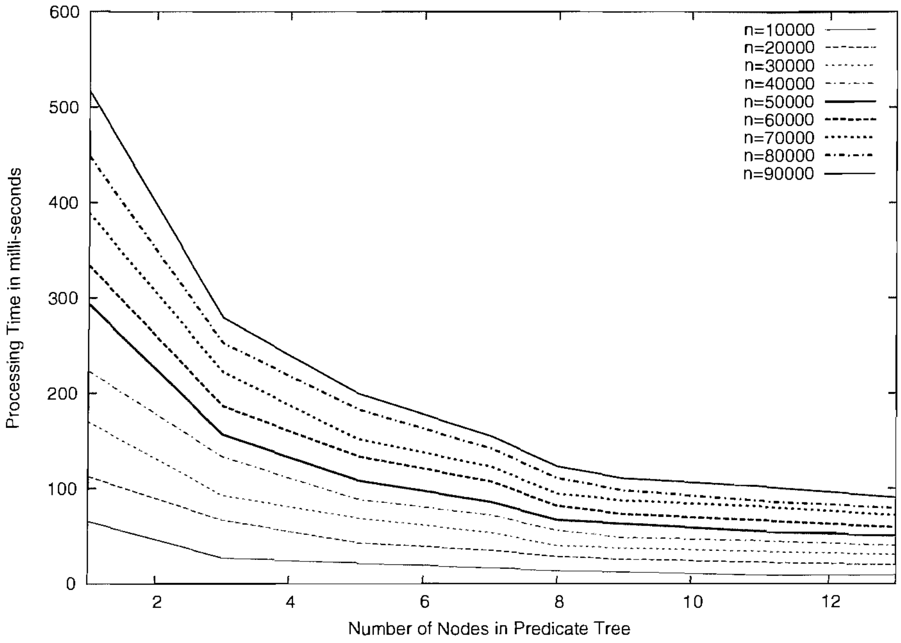


FIGURE 9.2: Effect of the IBS on time complexity.

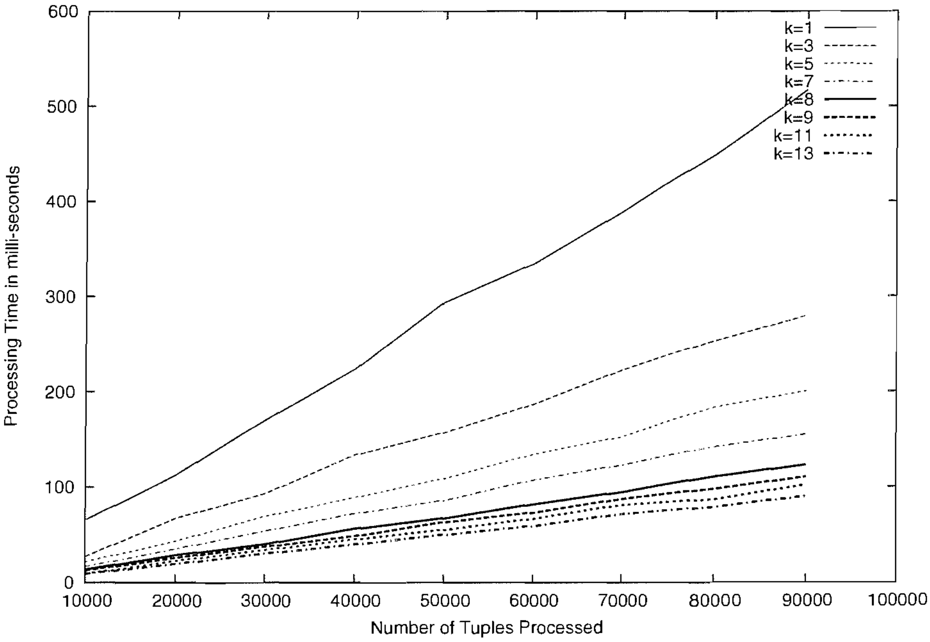


FIGURE 9.3: SPJ individual query performance.

to construct the static query plan, which was used to evaluate the continuous query to its completion. The performance was compared against the dynamic query planning algorithm. To provide a fair comparison between the planning algorithms a round robin operator scheduling policy was used to execute the query. Figure 9.5 illustrates the use of the algorithm for two categories of change in the stream data processing: variations in the data rate and variations in the selectivity of the join operators. Synthetic data sets and streams were used to simulate the behaviour of the algorithm.

Figures ?? illustrate the cumulative memory and computational resource usage in case of variations in data rate. The data arrival rates at streams A,B,C are given in the following figure 9.4. Stream A is a blocking/unblocking stream, with data arriving at constant speed for a fixed interval, while the data arrival rates at streams B and C follow triangular and gaussian distributions respectively. In the absence of any known benchmark for comparing the performance of the various stream processing systems, these three patterns were arbitrarily selected from the stream characteristics observed in real systems. A bounded number of tuples were used to perform the evaluation. Considerable care was taken to evaluate the system exclusively in the main memory and idle resources were provisioned so that the algorithm's performances are not effected by the resource constraints imposed by the system.

Figure 9.5 shows the variation in the total queue sizes over time for the two algorithms. It was observed that the static query planning algorithm incurs considerable penalties, because of its inability to adapt to variations in the resource requirements of the data streams, while the adaptive algorithm minimized the memory requirements of the query processor.

Figure 9.6, shows the variation in throughput in the face of variations in the join selectivity algorithm.

9.2.3 Operator scheduling analysis

This sub-section provides evaluation of the operator scheduling algorithm described in the previous chapter. Evaluation of the algorithm is carried out in two scenarios: operator scheduling for memory minimisation for a single query and Operator scheduling for multiple queries. In case of the single query optimisation, a comparison is made against FIFO, Greedy and Chain Strategy. Chain is excluded from comparison in the case of multiple query processing, as chain scheduling for multiple queries remains undefined.

As can be observed from the following figure the chain strategy and adaptive scheduling algorithms outperform FIFO and greedy scheduling algorithms. As both chain and adaptive scheduling rely on an information push mechanism, the performance comparison highlights the adaptive scheduling algorithm's ability to allow concurrent evaluation of

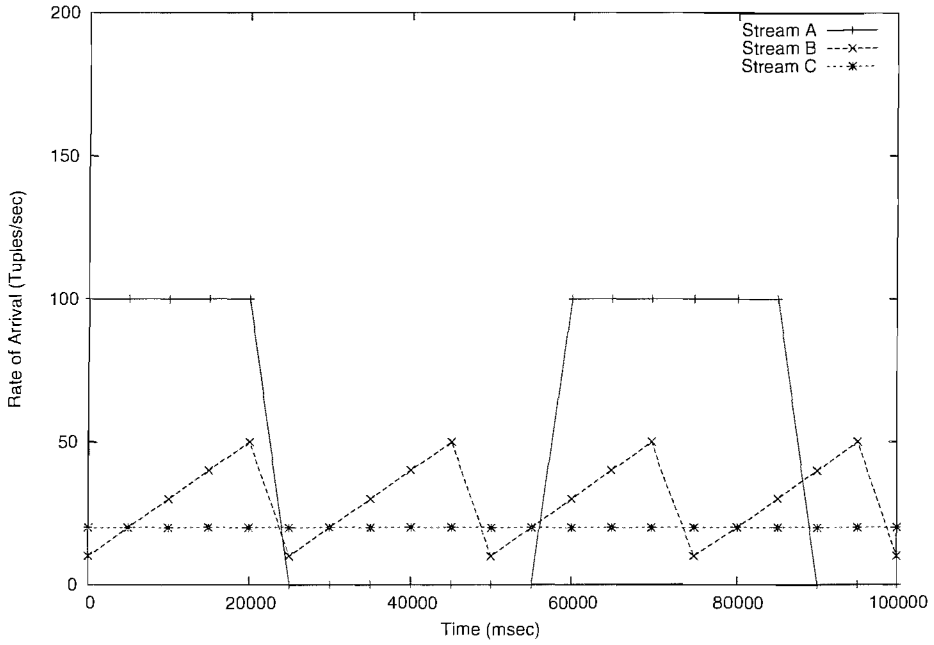


FIGURE 9.4: Data arrival rates of various streams and sub streams.

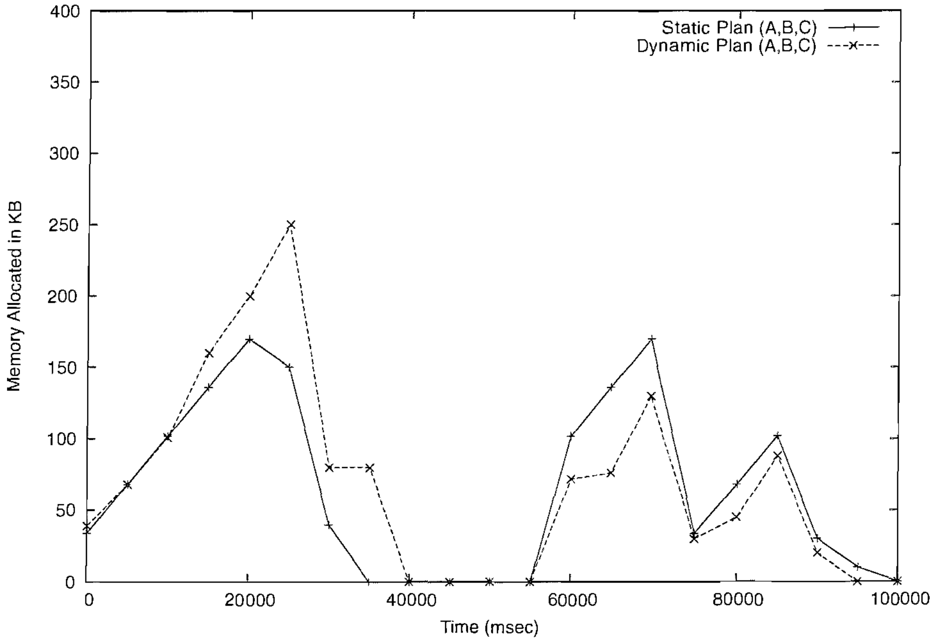


FIGURE 9.5: Memory requirements of various streams and sub streams.

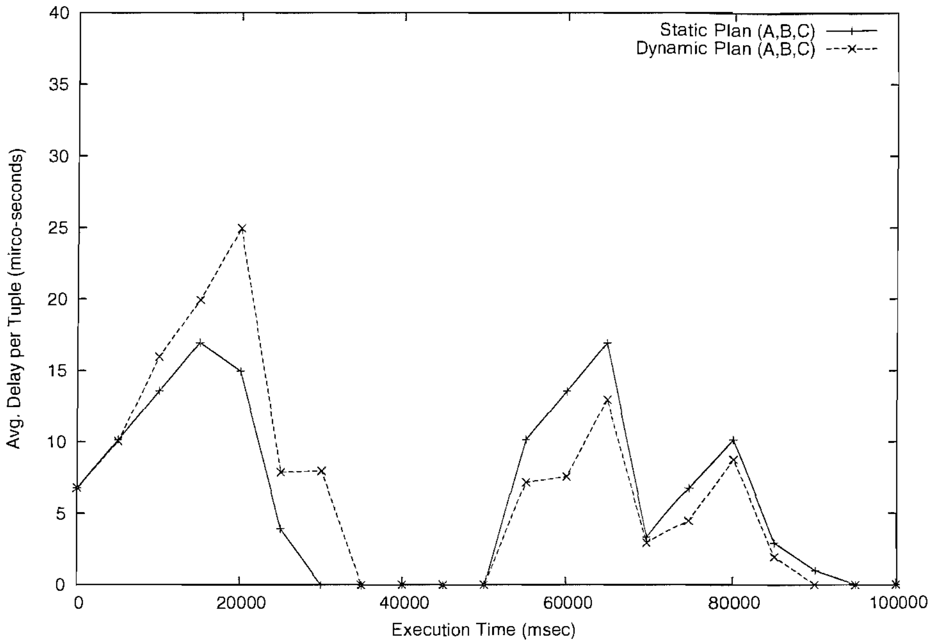


FIGURE 9.6: Delay Characteristics of various streams and sub streams.

parallel paths, unlike Chain, which allows data to be propagated through a direct chain from leaf to root node of the query plan. Memory usage and computational resource usage are depicted in the following graphs.

In the case of multiple queries, the FIFO and Greedy algorithms fail to optimize the amount of state held in operators shared across multiple queries. This results in disproportionate states being held at the operators. However, the adaptive scheduling algorithm adopts a back filling strategy, which allows it to reduce memory utilisation by associating dynamic bounds on computational resource usage. Figure 9.7 illustrates the memory and computational performance of adaptive vis-a-vis FIFO and Greedy scheduling strategies.

9.3 Summary

A DSMS systems consists of four distinct phases of processing: the logical query planning phase, the physical query planning phase, the operator selection phase and the scheduling phase, strictly in that order. An adaptive DSMS system can be constructed by incorporating adaptive behaviour at any stage of processing. In such a scheme, the decision made at the higher stages affects the decision made in subsequent phases. To date, two distinct approaches have emerged in the field: a block-based query processing approach advocated by Aurora (Carney, etintemel, Cherniack, Convey, Lee, Seidman, Stonebraker, Tatbul, and Zdonik 2002) an a individual query optimisation approach advocated by TelegraphCQ (Chandrasekaran, Cooper, Deshpande, Franklin, Hellerstein, Hong, Krishnamurthy,

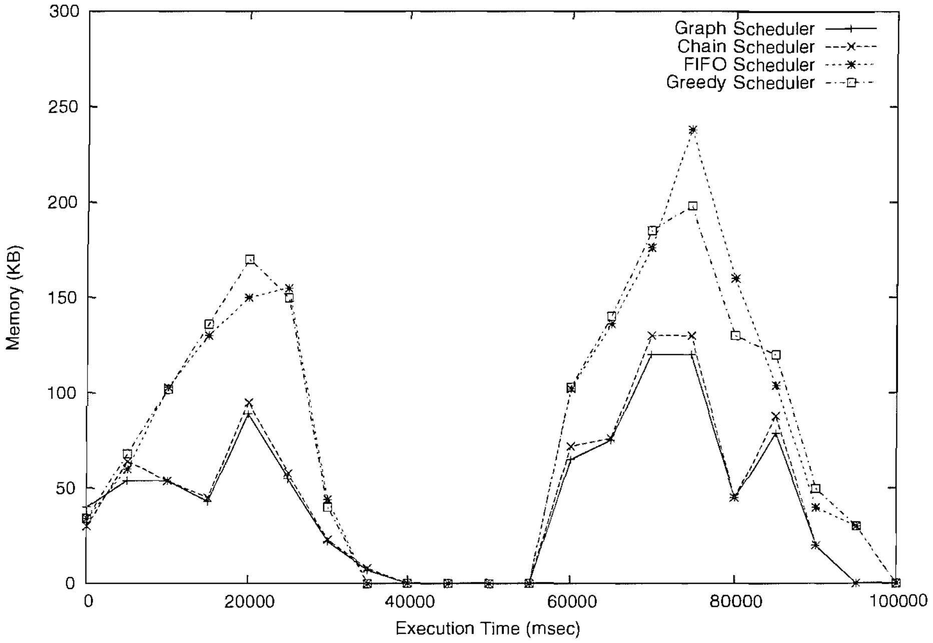


FIGURE 9.7: Comparative memory performance of scheduling strategies.

Madden, Reiss, and Shah 2003) and StreamDB (Arasu, Babcock, Babu, Datar, Ito, Motwani, Nishizawa, Srivastava, Thomas, Varma, and Widom 2003). Aurora adopts the multiple query processing approach of producing all the necessary tuple combinations required by the queries and applies late filtering to produce the required tuples, while the rest of the systems ignore the case of multiple query processing. While Aurora does address the issue of multiple query processing, it does not incorporate the notion of adapting the logical/physical plan in accordance with runtime query statistics. On the other hand, systems like TelegraphCQ adopt the tuple routing strategies, which introduces adaptive behaviour at planning and the operator level. For example, the Eddy operator used in TelegraphCQ makes localized optimisation decisions. To date our approach remains the only approach that tries to integrate the different phases of adaptive behaviour in query processing systems. It is the only approach that allows adaptive behaviour at the query level and can also impose global restrictions to optimize query performance across the system.

The performance evaluation in this section showed that the use of a combined planning and scheduling strategy results in a better performance than the approaches that adopt independent planning and execution architecture.

9.3.1 Contributions

A number of novel techniques for developing DSMS were introduced in the previous chapter, and their evaluation was presented above. The specific contribution of the above approach are:

1. The incorporation of multiple query optimisation approach for DSMS, which allows differential QOS to be supported for individual queries.
2. The incorporation of a novel query planning and operator scheduling algorithm.
3. The Proposal for PSQL to be used as a query language for stream systems.

Part V

Conclusions and Appendices

Chapter 10

Conclusions

10.1 Concluding remarks

Dynamic aggregation of resources has become a common trend in an emerging class of distributed systems. This thesis presented investigations related to adaptive resource management for three exemplar applications. Based on the limited experiences of these investigations, it is concluded that there exists at least two sub-categories of adaptive systems in large scale distributed systems, provisioning systems and quality management systems. A provisioning system retains a common policy throughout execution; it uses resource augmentation to adapt to variations in operating conditions. Scheduling systems and P2P systems (presented in the previous parts) fall under the provisioning systems category. The stream query processing system is classified as a quality management system. An important characteristic of quality management systems is their ability to adapt their operational characteristics in response to the operating environment. Not every aspect of these two classes of adaptive systems was discussed, but a brief discussion of observations that apply to adaptive systems in general is provided below.

It was observed that, unlike most distributed system properties, adaptive system behaviour cannot be expressed without explicit incorporation of the temporal dimension. However, the way the notion of time is captured may vary with respect to the system characteristics. For example, our example of online scheduling uses a shifting finite horizon, while the information dissemination example addresses the issue by using a continual representation of time, and finally, the tuple structure in the DSMS example explicitly incorporates the notion of time. The notion of time provides a convenient way to represent that the system has moved from one optimal state to another. In retrospect, this was also observed in our applications with a moving finite horizon in online scheduling, with optimisation cycles in a dissemination scenario and with changes to query plan in DSMS indicating such state transitions. From the observations made in developing these systems it is concluded that adaptive behaviour manifests itself in various different ways. However, in order to achieve adaptive behaviour it remains crucial that systems are able to determine the existence of a more optimal state and identify some means to transition to that state. For example, the addition of time slots to a queue in online scheduling indicates that additional resources are present, that current packing may soon become non-optimal, and the system then attempts to optimise its objective function. Similar parallels can be observed in the other two exemplars.

When a system becomes capable of identifying the transition to an optimal state with respect to its current state, it can be augmented with a capability to explore the transitions to these states of interest. While online scheduling provides a limited means to explore this space, ample scenarios have been provided in the remaining exemplars. In the cases studied, the state transition scenarios were translated into an optimisation problem. For example, in information dissemination, "Which neighbor to choose?" represents the optimisation problem that was solved to provide adaptive behaviour in a peer network. However, it was found that in some specific cases, adaptive behaviour can be accomplished by exploring the search space, while in some other cases it can be achieved by refining the search space. For example, a query processor could exhibit adaptive behaviour by modifying either the logical plan, physical plan, operator behaviour or schedule, exclusively by exploring the search space in each of these layers, or it could refine the search space by accomplishing adaptive behaviour by optimisation across the layers.

Our investigations into adaptive systems also highlighted their limitation in autonomic management of resources. All our system prototypes required a set of policies that defined their objective function and restrictions on their optimisation space. Whether considering utilisation maximisation in online scheduling, search cost minimisation in information dissemination or query optimisation in DSMS, these systems had a predefined set of objective functions. Our future work on adaptive systems would be to design systems that determine their objective function on the basis of operational characteristics and policy statements. The interaction of policies with operational environment and the verification of policies will be an important research aspect of such systems. It will be interesting to see if these systems incorporate the notion of stability. The next few sections present conclusions for each of the application types.

10.1.1 Online scheduling

The investigations into computational resource aggregation systems are convincing in that, unlike the resource reservation schemes, online scheduling schemes are bound to dominate scheduling in the Grid environment. Low scheduling overheads, ability to maintain autonomous control of resources and provide probabilistic guarantees on resource utilisation are definite advantages of this approach. On the negative side, it is not clear how communication models between jobs will affect the scheduling of jobs. It is bound to be the case that the incorporation of communication patterns will increase the system's sensitivity to failures of resources and necessary redundancies will have to be incorporated to increase the robustness of the system.

10.1.2 Information dissemination

The investigations into these system types reinforce the assumption that overlay networks will be commonplace for describing the context for resources involved in adaptive resource management. Adaptive overlays can be effectively used to incorporate the notion of partial visibility of state information and to enforce localized policies. However, although a graph theoretical approach is proposed to capture the dynamic behaviour of the individual resources, it is the system dynamics and the choice of the objective function that predominantly determine the suitability of the overlay structure. It was observed that a mismatch between the structure of the overlay network and the objective function leads to frequent reorganisation, leading inturn to an unstable system.

10.1.3 DSMS - Query processing

Data Stream Management Systems represent a unique class of application systems, which highlight the case that, under limited resources, adaptive systems can be designed to provide variable quality of service guarantees. In addition, the approach in this thesis provides a query planning mechanism that has the unique capability of adapting to a distributed system setting with no additional modifications. To date, this remains the only approach to provide such a sophisticated level of control on resource utilisation for such problems. However, with exclusive focus on adaptive query processing, the approach fails to take into account the additional complexity introduced by schema modifications in streams and differential QoS. It will be interesting to see how such features can be incorporated into future editions of the system.

10.2 Future Work

The end of each part of the thesis indicated how had been extended the state of the art in each of the application areas. A viable approach will be to continue to investigate the issues in each of the application domains, while continuing to conceptualize a model to capture adaptive behaviour in large scale systems. Although the above approach was used during the investigations of this thesis, it has some limitations. At times, there was insufficient overlap between the application areas and the general theoretical approach. As part of future work two separate areas of investigations are identified, which will separate the theoretical aspects and the application aspects.

The future plan is to investigate two research directions to address the above problem, first, the theoretical modelling of adaptive behaviour in distributed systems. Although a generic hypothesis has been presented in the thesis, formal specification to capture the concept of adaptivity needs to be further refined. The current hypothesis remains the proof of the concept of the initial investigations, but this will have to be further validated to include models for policy driven adaptive behaviour. At the same time, it will be required to equip the model with the notion of comparison between adaptive behaviours. Such a feature will aid the classification and comparison of adaptive systems. As pointed out in previous sections, the temporal dimension will necessarily be part of any such model. At some point during the investigations, possible directions in the field of time series representation or in semi Markov decision processes were considered, as possible means to capture the temporal aspect in large scale distributed systems. This seems to be an approach that requires further investigation.

Second, by considering the development of a distributed DSMS, as mentioned in the chapter 2 of this thesis, the emergence of sensor networks and content based routing systems has highlighted the need for DSMS combined with semantic overlay. Gryphon (Strom et al. 1998) and INFO-Dissemination (Dialani, Gawlick, Madsen, Malaika, and Mishra 2005) represent the emerging class of systems that perform in network processing on structured and semi structured data. It will be interesting to see how the various algorithms developed for this thesis can be combined to provide a distributed DSMS system capable of processing the data streams flowing between various nodes in the overlay network. It is presumed that the semantic overlay technique developed in part III of the thesis can be used to provide schema location and matching services. While the query planning techniques described in part IV will be employed for distributed query processing, the scheduling model can be employed to prioritize operator scheduling in cases where multiple computing resources are available.

Appendix A

Appendix: Continuous Query Semantics

Consider a query Q such that the join criterion is defined as Q = $T_1 \bowtie T_2 \bowtie T_3 \bowtie ... \bowtie T_k$, representing a pipeline join between a set of tuples T = { T_1, T_2, T_3,...,T_k }.

Let Q^p represent all possible query plans for evaluation of the join operation, such that:

$$|Q^p| = (2 * k - 2)!/(k - 2)! \tag{A.1}$$

Only an exhaustive search based optimisation technique will iterate through the entire set of possible query plans; most other optimisation mechanisms will reduce the search space to minimize the optimisation costs. An optimal plan, namely:

$$Q_e = Q_e \in Q^p$$

minimizes costs for given characteristics of T. However, any change in the characteristics of T may result in Q_e being a non-optimal solution. Selection of an alternative optimal plan necessities a new search through the space.

The following is a list of probable reasons that may result in a given optimal query plan being rendered non-optimal:

1. Each stream in a query plan represents a list of rows in its active window. The addition of new rows, passage of time and other external events are likely to result in change in the number of rows involved in a join operation. A change in the number of rows participating in a join operation is reflected in the access plan cost and operator costs.
2. The correlation between the join parameters may change with respect to time, rendering previous selectivity estimates to be inaccurate.

3. A change in the arrival rate of the tuples at individual streams increases the indexing and maintenance costs of the individual tuples.

Two possible alternatives for re-optimising the query plan include either optimising a sub-plan from the original query plan or recreating the optimisation search space and recalculating the solution. Chapter 8 proposed an approach for recalculating the new optimal query plan from an existing query plan.

It is assumed that continuous queries are used to implement the DSMS and incremental changes to the resultant result set are calculated on the basis of the following equation:

$\delta\mathrm{Q} = \delta\mathrm{T}_1 \bowtie \mathrm{T}_2 \bowtie \mathrm{T}_3 \bowtie ... \bowtie \mathrm{T}_k$

Here, δQ represents the incremental tuples generated in response to the addition of tuples $\delta\mathrm{T}_1$, to the stream T_1. The query semantics assume that the state maintained by window operations on each of the streams is adequate to produce a semantically correct answer, and that the window moments are synchronized across the streams.

Ideally, an output tuple should be generated when a tuple is appended to either of the streams. SQL99 describes similar semantics for windowing functions. However, the same semantics has not found acceptance in DSMS systems. Overload conditions and unordered tuple arrivals in DSMS systems led to the adoption of additional query semantics.

Tuple Dropping Strategy Under overload conditions, the DSMS may not be able to process the high volume of incoming tuples with acceptable delays. Such systems adopt a tuple dropping strategy and hence the results during overload conditions may not be semantically consistent.

Compensating Tuples Out of order arrival of tuples may lead to generation of an inconsistent result set. As the result set cannot be corrected through re-evaluation, some DSMS provide compensating tuples to rectify the resultant result set.

The above described semantics affect the ways in which the results are generated and the optimisation strategies adopted by the stream management systems. The query semantics were summarized to highlight the differences in DSMS implementations.

Appendix B

Appendix: Survey of Large Scale Distributed Systems

This appendix presents a survey of the emerging class of large scale distributed systems. It outlines the common trends observed in applications of these large scale distributed computing infrastructures and explores the adaptive behaviour exhibited in them.

B.1 Examples of Large Scale Systems

The hypothesis is that the current trend in large scale computing systems is being driven by two complementary advances, firstly, the notion of providing computing as a utility, and secondly the notion of the pervasive nature of such an infrastructure. The notion of providing computational and data services in the form of utilities has been partly inspired by the success of the World Wide Web (WWW). WWW has provided the impetus for exploring the sharing of various types of computational and data resources across institutional boundaries (Foster and Kesselmann 1999; Foster, Kesselman, Nick, and Tuecke 2002; Tsvetovat and Sycara 2000). It is a commonly held belief that the advances in the field of web technologies, which have allowed asynchronous content delivery, can be extended to provide integrated access to data resources and computational resources across institutional boundaries. At the same time, advances in hardware (specifically sensor technologies) have provided the impetus to the pervasive aspect of the computing infrastructure. A number of research ideas are being pursued to create infrastructure where data and computational resources are seamlessly integrated to form a pervasive computational infrastructure. The set of systems described in this survey were selected for their support of pervasive environments and is applicable to a large set of applications. The following subsections review a subset of the relevant technological/research approaches followed in developing large scale systems. The review is not exhaustive and is primarily focused on highlighting the commonalities, rather than differences between the surveyed approaches.

B.1.1 Services Oriented Architecture

Services Oriented Architecture (Graham, Davis, Simeonov, Boubez, Neyama, and Nakamura 2001) has been proposed as an important paradigm to support the development of distributed applications in a heterogeneous computing environment. Most service oriented architectures use a platform neutral messaging specification to describe the basic communication protocols. For example, the use of SOAP, XML are used to describe the messages for web services communication. Service Oriented Architecture allow system components to be accessed through a set of methods, with the aide of message types defined for the system components. Service advertisements contain the descriptions of the messages and the various port types. These advertisements are published in discovery services such as UDDI, which can aid the discovery of services across the system. At times, the service advertisement is augmented with semantic information associated with the service definitions to allow complex patterns in the discovery process, work (Miles, Papay, Dialani, Luck, Decker, Payne, and Moreau 2003a; Miles, Papay, Dialani, Luck, Decker, Payne, and Moreau 2003b), which the author of this thesis has been involved in present one such extension to UDDI and enables metadata assisted service discovery. A number of services may be composed with the aide of workflow technologies. The workflow may use fault-tolerant web service implementations (Dialani, Miles, Moreau, De Roure, and Luck 2002), or may use dynamic rebinding to compose reliable services.

Service Oriented Architecture (SOA) based software implementations such as .NET, J2EE, OGSA(Foster, Kesselman, Nick, and Tuecke 2002) provide software platforms for developing web service implementations and are designed to operate in a heterogeneous resource environment, while maintaining compatibility at the messaging level. Dynamic compositional capability coupled with the ability to operate in heterogeneous environments have enabled the use of Web Services to develop applications that support dynamic integration of data and computational resources.

SOA enables adaptive behaviour by allowing applications to dynamically rebind to services that support identical interfaces, where identical services are identified according to their syntax and semantic properties. To support this adaptive behaviour, the applications need to be able to share the service advertisements in a scalable fashion. These advertisements may be shared between the applications using a centralized service such as UDDI or can be cached using an adaptive overlay network. An adaptive overlay is particularly useful in cases where centralized registries could not be supported, for example, mobile services environments.

B.1.2 Grid computing

The Grid computing paradigm has evolved around the notion of the virtual organisation. Virtual organisations are described as dynamically created associations between users and resources across various administrative domains and institutional boundaries. A number of alternative approaches have been suggested for the management of such virtual organisations. Examples include economics-based resource allocation (Abramson, Buyya, and Giddy 2002), structural organisation (Litzkow, Livny, and M.W.Mukta 1990; Foster and Kesselmann 1999), unstructured organisation of resources (SETI) and Services based systems (Foster, Kesselman, Nick, and Tuecke 2002). The theoretical aspects of the virtual organisations have been discussed in (Dang and Jennings 2004; Norman, Preece, Chalmers, Jennings, Luck, Dang, Nguyen, Deora, Shao, Gray, and Fiddian 2003). An important aspect of Grid computing is its dual focus on the co-allocation of data and computational resources, in an dynamic computational environment composed of unreliable resources.

Grids have been used for a wide variety of applications, which include but are not limited to data integration (Atkinson, Chervenak, Kunst, Narang, Paton, Pearson, Shoshani, and Watson 2004), high throughput computing (Frey, Tannenbaum, Livny, Foster, and Tuecke 2001), biomedical applications (Goble, Pettifer, and Stevens 2004) and sensor networks (Hill, Szewczyk, Woo, Hollar, Culler, and Pister 2000). Though the Grid applications vary in their software architectures, they exhibit some common characteristics, which are: first, resource discovery, second, by dynamic/run-time composition of software services, and thirdly, orchestration of distributed data and computational resources. These applications provide virtual organisations either for direct access to data and computational resources or through application service encapsulating these resources.

A classification of various types of Grid Systems has been suggested by Fox (Fran Berman (Editor) 2003), prominent amongst which are Compute Centric Grids and Data Centric Grid Systems, which are described here in some detail:

B.1.2.1 Compute Centric Grids

Grid infrastructures, primarily utilizing computational cycles across the virtual organisations, are referred to as computational Grids. SETI@HOME, GLOBUS and CONDOR-G are examples. Most computational Grids provide job submission, monitoring and scheduling facilities as a means of access to the remote computational resources. A computational Grid may be formed by the resources owned by the resource provider (e.g. PBS combined with GLOBUS installation) or alternatively it may be formed by the free interaction between resource providers and consumers, for example SETI@HOME. There are many interesting open research issues in both types of computational Grids. While in the former types the issues pertaining to secured access and co-allocation have dominated the agenda, the latter have focused on creating computational economies and the mechanisms to design a self-sustaining computational infrastructure.

Traditionally, job submission on a distributed set of resources does not provide strict guarantees on the performance of the jobs being handled by the service. For example, both CONDOR and SETI have relied on providing the best effort scheduling capabilities. However, as is evident from recent studies (Nabrzyski, M., and Jan 2004), it is possible to statistically guarantee adherence to a multitude of objective functions for scheduling on a set of distributed resources.

B.1.2.2 Data Centric Grids (DCG)

Data Centric Grids provide the infrastructure for accessing, disseminating, archiving and provenance tracking over a set of distributed data resources. Most DCGs provide the means to discover, summarize data (metadata) and create access and transport mechanisms between data resources. An application may access multiple data resources may interact independently with each of the data resources. Alternately, an application may orchestrate the services provided by each of the data resources and in turn allow workflow optimisers to achieve data flow optimisation between data resources. Publish/subscribe based data resource management represents a data Grid system type which is relevant to the scope of this thesis. As a part of dataflow optimisation, resources in a data centric Grid may form a self-managing overlay network to reduce the data transport and management costs. The formation of such an overlay cannot be conceived by the designer of any one data resource provider and overlay will evolve from the complex interaction between data resources.

B.1.3 Peer-to-Peer computing (P2P)

P2P computing provides a novel distributed computing architecture and is characterized by its decentralisation of control. Each of the participating resources in the network is referred to as peer. In an ideal P2P system, each of the participating peers are considered to be uniform. Early P2P computing systems, like FreeNet (Clarke, Sandberg, Wiley, and Hong 2001) and Gnutella (GNUTELLA) were centralized repositories that aided discovery of resources between peers using a flooding protocol. Subsequently, a number of structured P2P overlay creation techniques were proposed, for example, Distributed Hash Techniques (DHT), to provide bounded average discovery paths, and resilient system performance in face of failure/recovery of peers. The various P2P approaches can be divided into three primary categories: firstly, structured P2P systems like CAN (Ratnasamy, Francis, Handley, Karp, and Schenker 2001), Tapestry (Zhao, Kubiatowicz, and Joseph 2001) and Chord (Stoica, Morris, Liben-Nowell, Karger, Kaashoek, Dabek, and Balakrishnan 2003); secondly, unstructured P2P systems like FreeNet (Clarke, Sandberg, Wiley, and Hong 2001), and Gnutella (GNUTELLA) and thirdly, semi structured P2P systems like JXTA (Qu and Nejdl 2001).

Most P2P system use messaging based communication protocols to allow operation on a set of heterogeneous resources. State-of-the-art P2P systems use dynamic resource discovery and binding to allow creation of peer-groups which are very similar to the concept of the virtual organisation illustrated in Grid computing environments. Software architecture for developing P2P computing systems has been the primary research focus of most P2P computing systems. Additionally, research has also focused on how to develop P2P systems that guarantee that resource providers and consumers have equal benefits from their mutual association and the means to develop trust between them.

B.1.4 Ad hoc network systems

Advances in mobile communications have enabled the creation of ad hoc networks, characterised by their ability to adapt to availability of resources. In ad hoc networks, wireless and/or mobile resources (also known as mobile nodes) are able to communicate with each other in the absence of a fixed communication infrastructure, in the absence of any centralized control. Multi-hop communication is achieved as nodes route packets on behalf of other nodes. The dynamic creation of a mobile communication infrastructure introduces many challenges in the areas of network management and data communication. Problems encountered in the network layer of ad hoc networks include topology control, data communication, and service access. Topology control problems include discovering neighbours, identifying position, determining transmission radius, establishing links to neighbors, scheduling node sleep and active periods, clustering, constructing the dominating set (each node either belongs to or has a neighbor from the dominating set), and maintaining the selected structure. Service access problems include Internet access, cellular network access, data or service replication upon detection or expectation of network partition, and unique IP addressing in merge or split-network scenarios.

Data communication problems include:

1. *routing*- sending a message from a source to a destination node,
2. *broadcasting*- flooding a message from a source to all other nodes in the network,
3. *multicasting*- sending a message from a source to a set of desirable destinations,
4. *geocasting*- sending a message from a source to all nodes inside a geographic region, and
5. *location updating*- maintaining reasonably accurate information about the location of other nodes.

For a state-of-the-art description of ad hoc systems and related issues, readers are advised to refer to the book by Perkins (Perkins 2001), who provides a detailed description of ad hoc routing protocols.

Analogies can be drawn between ad hoc networks and P2P networks, as both network types operate without any central co-ordinating authority, and in both cases, the nodes (peers) autonomously choose appropriate information and communication systems to interact with their sub-ordinate nodes (peers). The prime distinguishing features between P2P and ad hoc networks are: that ad hoc networks are limited to wireless communication networks, and, while P2P systems can impose a fixed topological structure such restrictions cannot be imposed on the mobility of nodes in an ad hoc network. These distinguishing characteristics impact the design methodologies and the resource management objectives in the two system types. While P2P systems are evaluated on the basis of their communication costs, the cost metrics for ad hoc networks also includes computational and energy costs.

B.1.4.1 Sensor networks

Sensor networks (Hill, Szewczyk, Woo, Hollar, Culler, and Pister 2000) are an emerging application area in ad hoc networks. Most sensor networks are designed to operate autonomously without any centralized control. While some sensor networks systems are designed to operate under a static environment, with complete knowledge of associated sensors and patterns of communication, most systems tend to require dynamic reconfiguration of network topology for efficient communication and data dissemination across sensor networks, in the situation of failing and intermittently available nodes. For a detailed description on sensor networks, refer to (Ilyas and Mahgoub).

B.1.5 Agent-based computing economies

Large scale distributed systems have been modelled as interactions between independent autonomous components capable of operating in dynamically-changing operating environments. Agent based systems can be modelled in various ways; most such models describes agent systems as consisting of three main constituent components:

1. *Agents* represent encapsulated computer systems and are capable of flexible and autonomous actions in their operating environments in order to meet design objectives.

2. *Interactions:* Agents will invariably use appropriate communication channels to interact with other agents and manage their inter-dependencies. These interactions may lead to co-operative, competitive or co-ordination based problem solvers.

3. *Organisations:* Agent interactions take place in some form of organisational context (e.g. a marketplace, an electronic institution).

Agent-based Computational Economies (ACE) represent an organisational model for agent-based systems and derive from artificial intelligence and economic theory. State-of-the-art techniques and the description of research issues in the field of ACEs can be found in (Tesfatsion 2002). In ACEs, agents represent the buyers and sellers of resources traded in electronic market places. Agents learn about the behaviour of other agents and the general behaviour of the market place and initiate appropriate actions in order to maximize objective functions. Agent economies have been shown to be self-organizing and self sustaining under varying market conditions.

The application of market based resource management for developing large scale distributed systems has been explored by (Abramson, Buyya, and Giddy 2002), and (?). ACEs present an approach towards developing self-managing computational infrastructure because of their ability to adapt to time varying operating conditions.

B.1.6 Discussion

A number of different paradigms for developing large scale computing systems were described in the preceding sections and provide specialized solutions for their target application domains. Increasingly, applications derive from the properties of one or more paradigms to create complex computing systems. The adoption of Services based architecture by Grid computing and P2P computing, Agent-based computational Economies for Grid computing, the merger of P2P techniques and ad hoc networks represents the trend for an evolving complex computational infrastructure. It may be argued that, over a period of time, these paradigms will merge to form a pervasive, adaptive computing infrastructure that will augment current WWW capabilities, to provide seamless integration of data and computational resources.

Instead of adopting a technology or the paradigm aligned approach an approach that investigates the common requirements across the above mentioned system types is advocated. As described above, almost all the above mentioned system types consist of resource providers that collaborate to share computational resources. The overwhelming requirement of scheduling computational resources in an online fashion needs further investigation. Additionally, as in most systems a resource is required to collaborate with a set of resources within the network, the choice of the subset and its relation to the objective functions needs further investigation. Additionally, when considering sensor networks and ad hoc networks the interplay with the network topology assumes prominence. In addition to the above two problem types, one other significant trend is the use of a single resource to provide support for multiple service requests. Usually, multiple service requests are supported on a limited resource. Concurrency of tasks introduces potential resource sharing. Adaptive system resource management with QoS warranties needs to be investigated. In addition, the above three trends in a fore-mentioned systems need to be investigated for the case of applications that have the common characteristics stated below.

B.2 Common characteristics

Common characteristics across systems:

1. Decentralisation of control.
2. Partial and varying visibility of system state.
3. Different operating conditions than those perceived at design time.
4. Online nature of the environment.
5. Continual re-optimisation.

B.3 Summary

Evolving systems have further refined the definition of resource management. While traditional resource management deals with allocation, utilisation and management of resources, recent advances have introduced the concept of self organisation and overlay.

Appendix C

Appendix: PSQL - Extended Query Language for Streams

An expressive query specification language enables a database management system to determine the common subexpressions between multiple query definitions. Such sub expressions are commonly used to identify possible resource sharing between multiple queries. Multi-query optimisation based on sub-expressions is common in relational databases (Date 1995). A similar sub-expression based resource sharing can also be applied to queries in stream data management systems. However, to date the language extensions to SQL - for example, CQL (Arasu, Babu, and Widom 2003) and ATLaS (Wang and Zaniolo 2003) - used to specify the queries in a stream management system do not incorporate the notion of similarity. Although CQL incorporates the notion of equality between queries it remains short of supporting similarity (see section C.1.1 for details). This appendix, introduces an extension to the SQL, known as PSQL, which incorporates the notion of similarity for definition in a stream management system.

The remainder of this appendix is organized as follows: The next section describes the language constructs. Subsequently, it describes object-relational mapping between the various constructs of the language and provides a representation of the same. It concludes with a list of examples that describe the capability of the language extensions.

C.1 Stream query language (PSQL)

The PSQL language extends SQL constructs and consists of an additional data type ("stream"). The stream data type is defined using a relational schema and consists of an ordered set of tuples. Each tuple has an associated timestamp, which determines its temporal validity. Theoretically, a stream represents an infinite set of tuples, however the scope of the tuples is determined by the window.

Definition: Every stream 'S' has a schema Rs and contains an ordered set of tuples Os adhering to the relational schema Rs. The stream 'S' follows 'append only' semantics. Each tuple in the stream is associated with a timestamp, which represents the time the tuple was appended to the stream. A stream represents a temporal stream if the set of tuples (Os) is ordered in accordance to its associated time stamp.

Definition: A query Q is defined in terms of relational operators on a set of streams. A query has access to unbounded but finite items of the stream. The bounds on number of data items accessed by a query Q are described by means of a window operation on individual streams.

A stream data management system may maintain the entire historical record of all the data items ever encountered by the stream. However, limited main memory and the higher latency costs of accessing secondary memory requires prioritized access to the limited memory resources. Sliding window specifications on streams provide preliminary estimates on the amount of memory required by each of the query definitions. Cumulative memory requirements need to be ascertained from a set of active continuous queries. The query language needs to be descriptive to provide necessary information to determine the number of data tuples that should be maintained in DSMS main memory for each of the streams. Such cumulative memory requirements can be determined if the window specifications support similarity and subsuming operators. It should be noted that most present implementations of DSMS do not consider tuple sharing between multiple queries and have therefore not built the language constructs to represent the subsuming of the stream windows.

Data management systems may choose to maintain additional tuples either in the main memory or in secondary storage or a combination of the two. Alternatively, it may maintain synopsis on the historical data tuples that are out of the scope of the union of window specifications. However, for the scope of the experiments discussed in this thesis considered that DSMS maintains the streams exclusively in the main memory. However, the PSQL allows storage attributes at the time of the creation of the stream.

PSQL supports relational operators over streaming data and relation data. It supports traditional data storage types like tables and views, while the streaming data is stored in the stream data container. A typical query expression may associate the query operations between the streaming data and data held in traditional data containers like tables and views. PSQL does not distinguish between various data containers while specifying the relational operators. The only notable difference is that the query referencing the stream container specifies the scope of the query by means of the windowing operations. The query language does not impose any restrictions on the type of operations performed by the operators, but necessitates that the operators that allow streams as input produce a stream as an output. That is to say that if either of the inputs to the operator is a stream, the output of the operator is considered to be a stream. However, the operators that exclusively operate on a static data container produce static outputs and are considered to be static datasets. The classification was carried out with the aim of allowing independent optimisation of the static and dynamic parts of the query. In some specific cases, the query scrambling techniques may result in two distinct parts of the query tree, where the root node evaluates the static and stream relations.

As the PSQL distinguishes between the streams and static relations on the basis of the windowing operations, the query executable plans are produced in order to:

1. Use the relational semantics to specify the operations between the data streams and data tables.
2. Produce plans that reduce computational costs by evaluating static relations for minimal number of times and cache the static result set for operation against with the streaming data.
3. Produce query plans that can be represented as a series of operator executions.
4. Reduce the computational costs from updates to static relational data tables.
5. Allow the use of standard join operations and multiple join operations.
6. Identify tuple-level resource sharing and permit sharing of intermediate results between multiple queries where possible. The resource sharing may require modifications to query scheduling and the language should allow identification of such correlation between query specifications.

The above stream execution strategy allows us to specify the continuous queries. A continuously executing query is valid for some temporal interval during which the streams are monitored and the query results evaluated, and therefore associating the temporal constraints with each of the query specifications. The temporal constraints are normally specified in terms of the wall clock time, but could also be signalled by means of events. The temporal constraints on the query execution remain optional. DSMS like StreamDB(Arasu, Babcock, Babu, Datar, Ito, Motwani, Nishizawa, Srivastava, Thomas, Varma, and Widom 2003) and TelegraphCQ (Chandrasekaran, Cooper, Deshpande, Franklin, Hellerstein, Hong, Krishnamurthy, Madden, Raman, Reiss, and Shah 2003) assume the queries to be valid from the time of submission to time of deletion.

There is a number of performance criteria for optimizing the query performance in a DSMS. For example, certain queries may require strict warranties on timeliness of query response and permit partial evaluation of the query. Alternatively some queries may require complete evaluation at the cost of permissible delays. While the DSMS maintains ultimate control in determining the exact order of query evaluations, PSQL allows queries to specify the performance optimisations.

C.1.1 Similarity features of PSQL

A sequence of data items representing the stream can be compared on the basis of the schema and their temporal characteristics, while those in the static tables need only be equated on the basis of their schema. To enable tuple level sharing between queries in stream management systems, the system needs to identify the overlaps between the data sets used by each query. Consider that each shared data item is represented by a stream S_s, and is shared between the number of queries. S_s has a schema R_s and sequence of O_s. The shared data items can be used by queries that have a schema $S_i = \prod S_s$ and the window moments generate a stream such that each item in the stream $O_y \in O_s$ and in the same causal ordering. For example, considering the two window specification Window-1 $[STOCKWINDOWROW(100)ONDATAARRIVAL]$, and Window-2 $[STOCKWINDOWROW(5($ on stream "STOCK". In this case, if the schema of Window-1 and Window-2 are equal, the window specifications allow tuple sharing between the streams. In the above example Window-2 can be subsumed by the Window-1.

PSQL allows identification of such similarity expressions as it takes into account both the data schema and window moment semantics. Additionally, the optional temporal clause coupled with scheduling hints, such as periodic, allow for synchronizing the window moments across the queries.

C.2 Comparison with other languages

One of the main distinguishing features of the PSQL is that it does not distinguish between the streaming and the non-streaming datasets. It considers each of the datasets and the intermediate results as a snapshot of the state at a particular time. A state that remains unaltered over an interval becomes a potential candidate for retention in the limited cache space. The language allows the data management system to uniquely identify and specify such states and the temporal ordering in which they are evaluated.

C.2.1 CQL

Continuous Query Language (CQL) (Arasu, Babu, and Widom 2003) was developed for StreamDB a stream data management system based on two classes of operators, the stream operators and the relation operators. In CQL terminology - a stream represents an unbounded bag of tuples with 'append only' semantics, while a relation is defined as time varying bag of updatable tuples. CQL primarily converts the stream into relations to take advantage of the standard relational operators, and finally converts the relation into the stream for continuous query semantics. The current specification of the CQL relies on time based sliding windows, whereas PSQL provides a wider range of sliding window semantics. CQL has ability to detect equivalence between the sliding windows can be exploited to detect common components between multiple queries and is not designed to consider tuple sharing between multiple queries. The CQL model to create the relation from streams prohibits it from detecting tuple level sharing between multiple queries. PSQL on the other hand does not distinguish between data containers, but instead depends on the datasets. This extended capability of PSQL allows it to incorporate the semantics of both SQL and CQL. In addition, PSQL contains explicit scheduling hints that determine the liveliness of the query, a feature that is not catered for in CQL.

C.2.2 ATLaS

ATLaS (Wang and Zaniolo 2003) adds to SQL the ability to define new User Defined Aggregates (UDA) and table functions for data mining applications, which accept stream inputs and produce output in the form of data streams. ATLaS provides semantics for expressing UDAs with both traditional blocking aggregates and non-blocking aggregates - such as online aggregates and the continuous aggregates used for time series - in a syntactic framework that makes it easy to identify non-blocking aggregates. ATLaS defines SQL extensions, and describes three distinct blocks, namely initialisation, aggregate definition and the termination block. The initialisation block is executed immediately at query submission, while the iterate block is executed for each of the query evaluation and the termination block is executed at the end of the query interval. The ATLaS structure imposes strict scheduling constraints on the responsiveness of the query, which makes it unsuitable for direct application in DSMS, which usually control the operator scheduling characteristics. The iterate block is evaluated for each execution step and needs to be appropriately described to restrict its evaluation to newly arrived data items in the stream. Also, the presence of the initialisation and termination blocks provides possibilities of maintaining state information while executing the query. Maintaining state information on an individual query basis is bound to either restrict the capability of the query engine or increase the complexity of sub-expression matching in a query optimizer.

C.2.3 Tapestry

Tapestry queries (Terry, Goldberg, Nichols, and Oki 1992) are expressed using SQL syntax. At time t, the result of a Tapestry query Q contains the set of tuples logically obtained by executing Q as a relational SQL query at every instant $t' < t$ and taking the set union of the results. The semantics for Q is equivalent to the CQL query operations on relations. Tapestry does not support sliding windows over streams or any relation-to-stream operators.

CQL remains the most closely related language to PSQL, which was inspired by the former. PSQL extends the capability of CQL to allow the possibilities of incorporation of tuple sharing between the query plans. It departs from CQL's notion of streams and relations and is based on the notion of temporal datasets instead. However, PSQL adopts the basic window specification semantics of CQL and extends it to associate the scheduling criteria with the window specifications. It also supports additional window definitions like landmark and snapshot windows.

C.3 Language - yacc representation

The following is the listing of the yacc implementation of PSQL. The yacc representation is provided as an alternative to BNF form of the language representation, the keywords appearing in CAPITALS represent tokens.

```
start
        : psqlcommand ';' {parse_tree = $1;  YYACCEPT;}
        ;

    psqlcommand
        : ddl
            {$$ = ddl($1);}
        | dml
            {$$ = dml($1);}
        ;

    ddl
        : createstream
            {$$ = $1;}
        | dropstream
            {$$ = $1;}
        | modifystream
            {$$ = $1;}
        ;

    createstream
        : KW_CREATE KW_STREAM DV_STRING '(' type_attribute_list ')'
            {$$ = create($3, $5, 0);}
        | KW_CREATE KW_STREAM '(' storage_type ')' DV_STRING '(' type_attribute_list
            {$$ = createspecificstorage($6, $4, $8, 0);}
        | KW_CREATE KW_TABLE DV_STRING '(' type_attribute_list ')'
```

```
            {$$ = create($3, $5, 1);}
        | KW_CREATE KW_TABLE '(' storage_type ')' DV_STRING '(' type_attribute_list '
            {$$ = createspecificstorage($6, $4, $8, 1);}
        ;

    dropstream
        : KW_DROP KW_STREAM DV_STRING
            {$$ = drop($3);}
        | KW_DROP KW_TABLE DV_STRING
            {$$ = drop($3);}
        ;

    modifystream
        : KW_MODIFY KW_STREAM DV_STRING KW_ADD '('type_attribute_list')'
            {$$ = modify($3,$6, true);}
        | KW_MODIFY KW_STREAM DV_STRING KW_DELETE '(' type_attribute_list')'
            {$$ = modify($3,$6, false);}
        | KW_MODIFY KW_TABLE DV_STRING KW_ADD '('type_attribute_list')'
            {$$ = modify($3,$6, true);}
        | KW_MODIFY KW_TABLE DV_STRING KW_DELETE '(' type_attribute_list')'
            {$$ = modify($3,$6, false);}
        ;

    storage_type
        : ST_PRIMARY
            {$$ = ST_PRIMARY;}
        | ST_HYBRID
            {$$ = ST_HYBRID;}
        | ST_SECONDARY
            {$$ = ST_SECONDARY;}

    type_attribute_list
        : attribute_spec ',' type_attribute_list
            {$$ = schema($1, $3);}
        | attribute_spec
            {$$ = $1;}
        ;

    attribute_spec
        : DV_STRING D_INTEGER
            {$$ = defineInt($1);}
    | DV_STRING D_FLOAT
        {$$ = defineFloat($1);}
    | DV_STRING D_CHAR ’[’ DV_INT ’]’
        {$$ = defineString($1, $4);}
    | DV_STRING D_BYTE ’[’ DV_INT ’]’
        {$$ = defineByte($1, $4);}
    | DV_STRING D_TIME
        {$$ = defineTime($1);}
    ;



dml
    : query
        {$$ = query($1);}
    ;

query
    : selectfromwhere
        {$$ = $1;}
    ;

selectfromwhere
    : selectclause fromclause optionalwhereclause optionalgroupbyclause optionalo
        {$$ = selectnode($1, $2, $3, $4, $5, $6, $7);}
    ;

selectclause
    : KW_SELECT KW_DISTINCT nonmd_projterms_list
        {$$ = selectclause(true, $3);}
    | KW_SELECT nonmd_projterms_list
        {$$ = selectclause(false, $2);}
    | KW_SELECT KW_DISTINCT ’*’
        {$$ = selectclause(true, 0);}
    | KW_SELECT ’*’
        {$$ = selectclause(false, 0);}
    ;

fromclause
    : KW_FROM nonmd_relation_list
        {$$ = $2;}
    ;
```

```
optionalwhereclause
    : KW_WHERE nonmd_condition_list
        {$$ = $2;}
    | nothing
    ;

optionalgroupbyclause
    : KW_GROUP KW_BY nonmd_attribute_list
        {$$ = $3;}
    | nothing
    ;

optionalorderbyclause
    : KW_ORDER KW_BY nonmd_attribute_list
        {$$ = $3;}
    | nothing
    ;

validityclause
    : KW_FOR timeclause
        {$$ = valid($2);}
    | nothing
    ;


windowlist
    : '[' windowclause ']'
        {$$ = $2; }
    | nothing
        {$$ = 0;}
    ;

windowclause
    : windowspecification KW_ON schedulingcriterion ',' windowclause
        {$$ = windowlist(window($1,$3),$5);}
    | windowspecification KW_ON schedulingcriterion
        {$$ = window($1,$3);}
    | windowspecification  ',' windowclause
        {$$ = windowlist(window($1,0),$3);}
    | windowspecification
```

```
        {$$ = $1;}
    ;

windowspecification
    : DV_STRING KW_WINDOW WT_ROWS '(' DV_INT ')'
        {$$ = windowrows($1,$5);}
    | DV_STRING KW_WINDOW WT_TIME '(' DV_STRING ',' DV_STRING ')'
        {$$ = 0;}
    | DV_STRING KW_WINDOW WT_LANDMARK '(' datetimestamp ')'
        {$$ = windowlandmark($1, $5);}
    | DV_STRING KW_WINDOW WT_SNAPSHOT
        {$$ = windowsnapshot($1);}
    | DV_STRING KW_WINDOW WT_NOW
        {$$ = windownow($1);}
    | DV_STRING KW_WINDOW WT_UNBOUNDED
        {$$ = windowunbounded($1);}
    ;


schedulingcriterion
    : SC_DATAARRIVAL
        {$$ = schedule(SC_DATAARRIVAL);}
    | SC_SNAPSHOT
        {$$ = schedule(SC_SNAPSHOT);}
    | SC_PERIODIC
        {$$ = schedule(SC_PERIODIC);}
    | SC_OVERFLOW
        {$$ = schedule(SC_OVERFLOW);}
    ;

timestamp
    : DV_INT '::' DV_INT '::' DV_INT
        {$$ = time($1, $3, $5);}
    | DV_INT T_HOUR DV_INT T_MIN DV_INT T_SECONDS
        {$$ = time($1, $3, $5);}
    ;

datetimestamp
    :  DV_INT '/' DV_INT '/' DV_INT ',' timestamp
        {$$ = datetime($1,$3,$5,$7);}
    ;
```

```
timeclause
    : '(' assignment_operation ';' condition ';' arithmetic_operation ')'
    ;

assignment_operation
    : D_TIME DV_STRING C_EQ DV_STRING
    ;



nonmd_projterms_list
    : projectionterm ',' nonmd_projterms_list
        {$$ = projectlist($1, $3);}
    | projectionterm
        {$$= $1;}
    | nothing
    ;

projectionterm
    : arithmetic_operation
        {$$ = $1;}
    | aggregation_operator
        {$$ = $1;}
    ;

aggregation_operator
    : FN_MIN '(' attribute ')'
        {$$ = function(MIN, $3);}
    | FN_MAX '(' attribute ')'
        {$$ = function(MAX, $3);}
    | FN_COUNT '(' attribute ')'
        {$$ = function(COUNT, $3);}
    | FN_COUNT '(' '*' ')'
        {$$ = function(COUNT, 0);}
    | FN_SUM '(' attribute ')'
        {$$ = function(SUM, $3);}
    | FN_SD '(' attribute ')'
        {$$ = function(SD, $3);}
    | FN_MEAN '(' attribute ')'
        {$$ = function(MEAN, $3);}
    ;

arithmetic_operation
    : attribute
       {$$= $1;}
    | constant
       {$$ = $1;}
    | datetimestamp
       {$$ = $1;}
    | arithmetic_operation '+' arithmetic_operation
       {$$ = arithOp(ADD, $1, $3);}
    | arithmetic_operation '-' arithmetic_operation
       {$$ = arithOp(SUB, $1, $3);}
    | arithmetic_operation '*' arithmetic_operation
       {$$ = arithOp(MUL, $1, $3);}
    | arithmetic_operation '/' arithmetic_operation
       {$$ = arithOp(DIV, $1, $3);}
    | '(' arithmetic_operation ')'
       {$$ = $2;}
    ;


nonmd_relation_list
    : relation ',' nonmd_relation_list
       {$$ = relationlist($1, $3);}
    | relation
       {$$ = $1}
    ;

nonmd_condition_list
    : '(' nonmd_condition_list ')'
    | condition KW_AND nonmd_condition_list
       {$$ = conditionlist(AND, $1, $3);}
    | condition KW_OR nonmd_condition_list
       {$$ = conditionlist(OR, $1, $3);}
    | condition KW_NOT nonmd_condition_list
       {$$ = conditionlist(NOT, $1, $3);}
    | condition
       {$$ = $1;}
    ;
```

```
nonmd_attribute_list
    : attribute ',' nonmd_attribute_list
        {$$ = attribtolist($1, $3);}
    | attribute
        {$$ = attriblist($1);}
    ;



relation
    : DV_STRING
        {$$ = relationnode($1);}
    | DV_STRING KW_AS DV_STRING
        {$$ = relation($1, $3);}
    ;

condition
    : arithmetic_operation C_LT arithmetic_operation
        { $$ = condition(CO_LT, $1, $3);}
    | arithmetic_operation C_LE arithmetic_operation
        { $$ = condition(CO_LE, $1, $3);}
    | arithmetic_operation C_EQ arithmetic_operation
        { $$ = condition(CO_EQ, $1, $3);}
    | arithmetic_operation C_NE arithmetic_operation
        { $$ = condition(CO_NE, $1, $3);}
    | arithmetic_operation C_GE arithmetic_operation
        { $$ = condition(CO_GE, $1, $3);}
    | arithmetic_operation C_GT arithmetic_operation
        { $$ = condition(CO_GT, $1, $3);}
    ;

attribute
    : DV_STRING '.' DV_STRING
        {$$ = attributenclass($1, $3);}
    | DV_STRING
        {$$ = attribute($1);}
    ;

constant
    : DV_QSTRING
        {$$ = constantstring($1);}
    | DV_INT
```

```
            {$$ = constantint($1);}
        | DV_FLOAT
            {$$ = constantfloat($1);}
        ;

    nothing
        :
        ;
```

C.4 Query examples

PSQL specifies the semantics for creation, modification and deletion of data containers. It implements a restricted type system that consists of following data types:

INTEGER Integer data type, the size of 'int' data type in C

REAL Real data type, the size of 'float' data type in C

CHAR A single char data type, is also implemented as CHAR[] arrays.

BYTE An array of bytes.

TIME A time data type with a fixed format dd/mm/yyyy, HH::MM::SS.

Consider a stream application for data management in a sensor network system used for monitoring the stocks in a supermarket. The stock stream is obtained from items appended to the stream on addition/withdrawal of an item from the shelf. The type of possible items that can be stacked on the shelf are stored in a static table. Additionally, the sensor network periodically generates a temperature log. The following set of commands defines the procedure to create the streams and static tables.

CREATE STREAM STOCK (ShelfId INT, Barcode INT, ItemTypeId INT, ItemDesc CHAR[255], expiry DATETIMESTAMP);

CREATE TABLE TYPE (ShelfId INT, ItemTypeId INT, ItemTypeDesc CHAR[255], MaxTemperature INT);

CREATE STREAM TEMPLOG (ShelfId INT, Temperature INT);

A very brief list of probable queries are illustrated below:

Query 1: Monitor temperature within a range, for an unbounded time, translates to :

Select * from TEMPLOG where Temperature > 5 and Temperature < -2;

Query 2: Strictly monitor the temperature for ice cream shelf for next two days, report every 30 seconds, is represented as:

Select * from TEMPLOG where Temperature > 5 and Temperature < -2 FOR(t = NOW; t < NOW + 02/00/0000,00::00::00; t = t + 00/00/0000,00::00::30);

Query 3: Produce the list of items stocked in a shelf since yesterday, translates to:

Select * from STOCK where ShelfId = 3 [STOCK window Landmark(NOW-1)]

Query 4: Monitor Stock for wrongly placed items and report only for last 100 items, is formulated as:

Select S.ShelfId, S.BarCode, S.ItemTypeDesc from STOCK AS S, TYPE AS T where S.ItemTypeId = T.ItemTypeId AND S.ShelfId = T.ShelfId [S WINDOW ROWS(100) ON DATAARRIVAL];

The PSQL query language allows an extensive list of query expressions. The added advantage of defining the query delivery specification allows the PSQL to determine the intervals at which the result sets need to be created and delivered.

Appendix D

Appendix: Related Work

D.1 Publications

Zhou, J., Hall, W., De Roure, D. and Dialani, V. (2005) Supporting Collaborative Resource Sharing on the Web: A Peer-to-Peer Approach to Hypermedia Link Services. Submitted to ACM Transactions on Internet Technology.

Zhou, J., Dialani, V., De Roure, D. and Hall, W. (2003) A Distance-based Semantic Search Algorithm for Peer-to-Peer Open Hypermedia Systems. In Proceedings of The Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 7-11, Chengdu, China.

Zhou, J., Dialani, V., De Roure, D. and Hall, W. (2003) A Semantic Search Algorithm for Peer-to-Peer Open Hypermedia Systems. In Proceedings of The First Workshop on Semantics in Peer-to-Peer and Grid Computing, pp. 43-54, Budapest, Hungary.

Moreau, L., Miles, S., Goble, C., Greenwood, M., Dialani, V., Addis, M.,et.al. (2003) On the Use of Agents in a BioInformatics Grid. In Proceedings of Proceedings of the Third IEEE/ACM CCGRID'2003 Workshop on Agent Based Cluster and Grid Computing, pp. 653-661, Tokyo, Japan. Lee, S., Sekguchi, S., Matsuoka, S. and Sato, M., Eds.

Miles, S., Papay, J., Dialani, V., Luck, M., Decker, K., Payne, T. and Moreau, L. Personalised Grid Service Discovery. In Proceedings of 19th Annual UK Performance Engineering Workshop (UKPEW'03), pp. 131-140, University of Warwick, Coventry, England.

Miles, S., Papay, J., Dialani, V., Luck, M., Decker, K., Payne, T. and Moreau, L. Personalised Grid Service Discovery. IEE Proceedings Software: Special Issue on Performance Engineering 150(4):pp. 252-256.

Moreau, L., Avila-Rosas, A., Dialani, V., Miles, S. and Liu, X. (2002) Agents for the Grid: A Comparison with Web Services (part II: Service Discovery). In Proceedings of Workshop on Challenges in Open Agent Systems -(-), pp. 52-56, Bologna, Italy.

Moreau, L., Miles, S., Goble, C., Greenwood, M., Dialani, V., et.al(2002) On the Use of Agents in a BioInformatics Grid. In Proceedings of Network Tools and Applications in Biology (NETTAB'2002) — Agents in Bioinformatics -(-), Bologna, Italy.

Dialani, V., Miles, S., Moreau, L., De Roure, D. and Luck, M. (2002) Transparent fault tolerance for web services based architectures. In Proceedings of 8th International Europar Conference (EURO-PAR'02) 2400(-), pp. 889-898, Paderborn, Germany.

References

9075, ISO/IEC (1992, 11). Database language sql, international standard iso/iec 9075:1992. American National Standard X3.135-1992, American National Standards Institute, New York, NY 10036.

Abramson, David, Rajkumar Buyya, and Jonathan Giddy (2002). A computational economy for grid computing and its implementation in the nimrod-g resource broker. *Future Gener. Comput. Syst. 18*(8), 1061–1074.

Albers, Susanne (1997). Better bounds for online scheduling. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pp. 130–139. ACM Press.

Amsaleg, Laurent, Michael J. Franklin, and Anthony Tomasic (1998). Dynamic query operator scheduling for wide-area remote access. *Distrib. Parallel Databases 6*(3), 217–246.

Ankolekar, A. et.al. (2001). Daml-s: Semantic markup for web services. In *Proceedings of the International Semantic Web Working Symposium (SWWS).*

Arasu, Arvind, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Rajeev Motwani, Itaru Nishizawa, Utkarsh Srivastava, Dilys Thomas, Rohit Varma, and Jennifer Widom (2003). Stream: The stanford stream data manager. *IEEE Data Eng. Bull. 26*(1), 19–26.

Arasu, Arvind, Shivnath Babu, and Jennifer Widom (2003, October). The cql continuous query language: Semantic foundations and query execution. Technical Report http://newdbpubs.stanford.edu/pub/2003-67, Stanford University.

Arasu, Arvind and Jennifer Widom (2004). Resource sharing in continuous sliding-window aggregates. pp. 336–347. Morgan Kaufmann.

Atkinson, Malcolm, Ann Chervenak, Peter Kunst, Inderpal Narang, Norman Paton, Dave Pearson, Arie Shoshani, and Paul Watson (2004). *The Grid: Blue Print for a New Computing Infrastructure* (Second ed.)., Chapter Data Access Integration and Management. Morgan Kauffmann.

Avnur, Ron and Joseph M. Hellerstein (2000). Eddies: continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pp. 261–272. ACM Press.

Babcock, Brian, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom (2002). Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 1–16. ACM Press.

Babcock, Brian, Shivnath Babu, Rajeev Motwani, and Mayur Datar (2003). Chain: operator scheduling for memory minimization in data stream systems. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pp. 253–264. ACM Press.

Berners-Lee, T., J. Hendler, and O. Lassila (2001). *Scientific American*. Scientific American.

Borodin, Allan and Ran El-Yaniv (1998a, April). *Online Computation and Competitive Analysis* (1 ed.)., Chapter 2.Introduction to Randomized Algorithms, pp. 432. Cambridge University Press.

Borodin, Allan and Ran El-Yaniv (1998b, April). *Online Computation and Competitive Analysis* (1 ed.)., Chapter 7. Request-Answer Games, pp. 432. Cambridge University Press.

Brucker, Peter (2001). *Scheduling Algorithms*, Chapter 5. Parallel Machines, Section 5.1. Springer-Verlag New York, Inc.

Buyya, Rajkumar, David Abramson, and Jonathan Giddy (2001). Nimrod-g resource broker for service-oriented grid computing. *IEEE Distributed Systems 2*(7).

Carney, D., U. C etintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik (2002, August). Monitoring streams - a new class of data management applications. In *In Proc. of 28th VLDB Conference*, pp. 84–89.

Carr, L. A., D. C. De Roure, W. Hall, and G. J. Hill (1995). The distributed link service: A tool for publishers,authors and readers. In *Proceedings of the Fourth International World Wide Web Conference: The Web Revolution, Boston, Massachusetts*, pp. 647–656.

Carr, L. A., W. Hall, S. Bechhofer, and C. A. Goble (2001). Conceptual linking: Ontology-based open hypermedia. In *Proceedings of the Tenth International World Wide Web Conference, Hong Kong*, pp. 334–342.

Chakraborty, Dipanjan (2004, June). *Service Discovery and Compsition in Pervasive Environments*. Ph. D. thesis, University of Maryland, Baltimore County.

Chakraborty, Dipanjan, Anupam Joshi, Tim Finin, and Yelena Yesha (2004, July). Towards Distributed Service Discovery in Pervasive Computing Environments. *IEEE Transactions on Mobile Computing*.

Chandrasekaran, Sirish, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Vijayshankar Raman, Frederick Reiss, and Mehul A. Shah (2003). Telegraphcq: Continuous dataflow processing for an uncertain world. In *CIDR*.

Chandrasekaran, Sirish, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R. Madden, Fred Reiss, and Mehul A. Shah (2003). Telegraphcq: continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pp. 668–668. ACM Press.

Chandrasekaran, Sirish and Michael J. Franklin (2003). Psoup: a system for streaming queries over streaming data. *The VLDB Journal 12*(2), 140–156.

Chen, Jianjun, David J. DeWitt, Feng Tian, and Yuan Wang (2000). Niagaracq: a scalable continuous query system for internet databases. *SIGMOD Rec. 29*(2), 379–390.

Clark, David (2001). Face-to-face with peer-to-peer networking. *Computer 34*(1), 18–21.

Clarke, Ian, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong (2001). Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Lecture Notes in Computer Science 2009*, 46+.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Commun. ACM 13*(6), 377–387.

Dang, Viet Dung and Nicholas R. Jennings (2004). Generating coalition structures with finite bound from the optimal guarantees. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 564–571. IEEE Computer Society.

DasGupta, Bhaskar and Michael A. Palis (2000). Online real-time preemptive scheduling of jobs with deadlines. In *Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization*, pp. 96–107. Springer-Verlag.

Date, C. J. (1995). *An introduction to database systems* (6 ed.), Volume xxiii of *Addison-Wesley systems programming series*. Addison-Wesley Pub. Co.

De Roure, D., N. Walker, and L. Carr (2000). Investigating link service infrastructures. In *Proceedings of ACM Hypertext 2000*, pp. 67–76.

De Roure, D. C., L. A. Carr, W. Hall, and G. J. Hill (1996). A distributed hypermedia link service. In *Proceedings of the Third International Workshop on Services in Distributed and Networked Environments (SDNE96)*, pp. 156–161.

Dialani, Vijay, Dieter Gawlick, Cecile Madsen, Susan Malaika, and Shailendra Mishra (2005). Information dissemination in the grid environment. pp. 1–54.

Dialani, V., S. Miles, L. Moreau, D. De Roure, and M. Luck (2002). Transparent fault tolerance for web services based architectures. *Lecture Notes in Computer Sciences, EUROPAR(2002) 2400*, 889–898.

Eppstein, David, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig (1997). Sparsification - a technique for speeding up dynamic graph algorithms. *Journal of ACM 44*(5), 669–696.

FengTian and David J. De Witt (2003). Tuple routing strategies for distributed eddies. In Johann Christoph Freytag, Peter C. Lockemann, Serge Abiteboul, Michael J. Carey, Patricia G. Selinger, and Andreas Heuer (Eds.), *VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases, September 9-12, 2003, Berlin, Germany*, pp. 333–344. Morgan Kaufmann.

Foster, Ian, Carl Kesselman, Jeff Nick, and Steve Tuecke (2002). Grid services for distributed systems integration. *35*(6), 37–46.

Foster, Ian and Carl Kesselmann (1999). The grid. In Ian Foster and Carl Kesselmann (Eds.), *Blueprint for a new computing infrastructure*. Morgan Kauffmann.

Fountain, Andrew M., Wendy Hall, Ian Heath, and Hugh Davis (1990). MICROCOSM: An Open Model for Hypermedia with Dynamic Linking. In *European Conference on Hypertext*, pp. 298–311.

Fran Berman (Editor), Geoffrey Fox (Editor), Anthony J.G. Hey (Editor) (2003, April). *Grid Computing: Making the Global Infrastructure a Reality* (First ed.). Wiley Publishers.

Frey, James, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke (2001). Condor-g: A computation management agent for multi-institutional grids. In *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01)*, pp. 55. IEEE Computer Society.

Garay, Juan A., Joseph (Seffi) Naor, Bulent Yener, and Peng Zhao. On-line Admission Control and Packet Scheduling with Interleaving.

Getta, J. R. (2000). Query scrambling in distributed multidatabase systems. In *Proceedings of the 11th International Workshop on Database and Expert Systems Applications*, pp. 647. IEEE Computer Society.

GNUTELLA. Gnutella. "http://gnutella.wego.com ".

Goble, Carole, Steve Pettifer, and Robert Stevens (2004). *The Grid: Blue Print for a New Computing Infrastructure* (Second ed.)., Chapter Knowledge Integration: In Silico experiments in bioinformatics. Morgan Kauffmann.

Goel, Ashish, Adam Meyerson, and Serge Plotkin (2001). Distributed admission control, scheduling, and routing with stale information. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pp. 611–619. Society for Industrial and Applied Mathematics.

Goldwasser, Michael H. (2003). Patience is a virtue: the effect of slack on competitiveness for admission control. *J. of Scheduling 6*(2), 183–211.

Goldwasser, Michael H. and Boris Kerbikov (2003). Admission control with immediate notification. *J. of Scheduling 6*(3), 269–285.

Gouda, Mohamed G. and Umeshwar Dayal (1981). Optimal semijoin schedules for query processing in local distributed database systems. In *Proceedings of the 1981 ACM SIGMOD international conference on Management of data*, pp. 164–175. ACM Press.

Graham, R, E.L.Lawler, J.K.Lenstra, and A.Rinnooy Kan (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. In *Discrete Optimization II*, Volume 5 of *Anals of Discrete Mathematics*, pp. 287–326.

Graham, Steve, Doug Davis, Simeon Simeonov, Toufic Boubez, Ryo Neyama, and Yuichi Nakamura (2001). *Building Web Services with Java: Making Sense of Xml, Soap, Wsdl, and Uddi.* Indianapolis, IN, USA: Sams.

Guha, Sudipto, Nick Koudas, and Kyuseok Shim (2001). Data-streams and histograms. In *ACM Symposium on Theory of Computing*, pp. 471–475.

Haas, Peter J. and Joseph M. Hellerstein (1999). Ripple joins for online aggregation. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pp. 287–298. ACM Press.

Haas, Z. and M. Pearlman (1998). The zone routing protocol (zrp) for ad hoc networks.

Hammad, Moustafa A., Michael J. Franklin, Walid G. Aref, and Ahmed K. Elmagarmid (2003). Scheduling for shared window joins over data streams. In Johann Christoph Freytag, Peter C. Lockemann, Serge Abiteboul, Michael J. Carey, Patricia G. Selinger, and Andreas Heuer (Eds.), *VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases, September 9-12, 2003, Berlin, Germany*, pp. 297–308. Morgan Kaufmann.

Hill, Jason, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister (2000). System architecture directions for networked sensors. *SIGOPS Oper. Syst. Rev. 34*(5), 93–104.

Ilyas, Mohammad and Imad Mahgoub. *Handbook of Sensor Networks.* CRC Press. [electronic resource] (Engnetbase) This is a record with electronic access and is only available through the Library's Web catalogue or through the Internet. Mode of access: Internet.

Ioanndis, Y.E. and Y.C. Kang (1990, May). Randomized algorithms for optimizing large join queries. In Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, San Francisco, CA. ACM.

Kahan, J. P. and A. Rapoport (1994). *Theories of Coalition Formation.* Hillsdale,NJ: Lawrence Erlbaum Associates.

Kemper, A, G Moerkotte, and K Peithner (1993, August). A blackboard architecture for query optimization in object databases. In *In the proceedings of the Conference on Very Large Data Bases*, pp. 543–554. Morgan Kaufmann Publishers Inc.

Kiran, Ali S (1998). *Chapter 21: Simulation and Scheduling, in Handbook of Simulation: Principles,Methodology,Advances,Applications and Practice* (1 ed.), Volume 1. John Wiley and Sons Inc.

Klusch, Matthias and Andreas Gerber (2002). Dynamic coalition formation among rational agents. *IEEE Intelligent Systems 17*(3), 42–47.

Kubiatowicz, J., D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao (2000, Nov). Oceanstore: An architecture for global-scale persistent storage. *In Proceedings of ACM ASPLOS*.

Kumaran, Ilango Ilango and S. Ilango Kumaran (2001, November). *Jini Technology: An Overview*. Pearson Education.

Lee, Chiang, Chi-Sheng Shih, and Yaw-Huei Chen (2001). Optimizing Large Join Queries Using A Graph-Based Approach. *Knowledge and Data Engineering 13*(2), 298–315.

Lee, Jae-Ha (2003). Online deadline scheduling: multiple machines and randomization. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pp. 19–23. ACM Press.

Lee, Jae-Ha (2004). Online deadline scheduling: Team adversary and restart. In Klaus Jansen and Roberto Solis-Oba (Eds.), *Approximation and Online Algorithms, First International Workshop, WAOA 2003, Budapest, Hungary, September 16-18, 2003, Revised Papers*, Volume 2909 of *Lecture Notes in Computer Science*. Springer.

Leung, Joseph Y-T. (2004, July). *Handbook of Scheduling*, Volume 1 of *Computer and Information Science Series*. Chapman and Hall.

Lipton, Richard J. and Andrew Tomkins (1994). Online interval scheduling. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*. ACM Press.

Litzkow, M.J., Miron Livny, and M.W.Mukta (1990). Condor - a hunter of idle workstations. In *In Proceedings of the IEEE Workshop on Experimental Distributed Systems*.

Lohman, G. and K. ONO (1990, August). Measuring the complexity of join enumeration in query optimization. In Proceedings of the 16th International Conference on Very Large Databases, Brisbane, Australia, pp. 149–159. VLDB Endowment, Berkeley, CA.

Luo, Gang, Curt J. Ellmann, Peter J. Haas, and Jeffrey F. Naughton (2002). A scalable hash ripple join algorithm. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pp. 252–262. ACM Press.

Lv, Qin, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker (2002). Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th international conference on Supercomputing*, pp. 84–95. ACM Press.

Madden, Samuel, Mehul Shah, Joseph M. Hellerstein, and Vijayshankar Raman (2002). Continuously adaptive continuous queries over streams. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pp. 49–60. ACM Press.

Michael and Jack Weast (2003, May). *UPnP Design by Example: A Software Developer's Guide to Universal Plug and Play.* Intel Press.

Miles, S., J. Papay, V. Dialani, M. Luck, K. Decker, T. Payne, and L. Moreau (2003a). Personalised grid service discovery. *In Proceedings of 19th Annual UK Performance Engineering Workshop (UKPEW'03)*, 131–140.

Miles, S., J. Papay, V. Dialani, M. Luck, K. Decker, T. Payne, and L. Moreau (2003b). Personalised grid service discovery. *IEE Proceedings Software: Special Issue on Performance Engineering 150*(4), 252–256.

Miller, Eric (2004). Weaving Meaning : An Overview of The Semantic Web. *Presented at the University of Michigan, Ann Arbor, Michigan USA.*

Nabrzyski, Jarek, Schopf Jennifer M., and Weglarz Jan (2004). *Grid Resource Management - State of the Art and Future Trends*, Volume 1 of *International Series in Operations Research and Management Science.* Kluwer Academic Press.

Ng, Kenneth W., Zhenghao Wang, Richard R. Muntz, and Silvia Nittel (1999). Dynamic query re-optimization. In *SSDBM '99: Proceedings of the 11th International Conference on Scientific on Scientific and Statistical Database Management*, pp. 264. IEEE Computer Society.

Norman, T J, A Preece, S Chalmers, N R Jennings, M Luck, V D Dang, T D Nguyen, V Deora, J Shao, W A Gray, and N J Fiddian (2003). Conoise: Agent-based formation of virtual organisations. In *Proceedings of 23rd SGAI International Conference on Innovative Techniques and Applications of AI*, pp. 353–366.

Oram, Andy (2001). *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology* (1 ed.). OReilly Associates.

Ozcan, Fatma, Sena Nural, Pinar Koksal, Cem Evrendilek, and Asuman Dogac (1997). Dynamic query optimization in multidatabases. *IEEE Data Eng. Bull. 20*(3), 38–45.

Perkins, Charles E. (2001). *Ad hoc networking: an introduction.* Addison-Wesley Longman Publishing Co., Inc.

Perkins, Charles E. and Pravin Bhagwat (1994). Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *SIGCOMM*, pp. 234–244.

Perkins, Charles E. and Elizabeth M. Royer (1999). Ad-hoc on-demand distance vector routing. In *WMCSA '99: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, pp. 90. IEEE Computer Society.

Plaxton, C. Greg, Rajmohan Rajaraman, Andr, and W. Richa (1997). Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, pp. 311–320. ACM Press.

Qu, Changtao and Wolfgang Nejdl (2001). Exploring JXTASearch for P2P Learning Resource Discovery.

Raman, V., A. Deshpande, and J. Hellerstein (2003, March). Using state modules for adaptive query processing. In *19th International Conference on Data Engineering*, pp. 353–367.

Ratnasamy, Sylvia, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker (2001). A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 161–172. ACM Press.

Ratsimor, Olga Vladi, Dipanjan Chakraborty, Sovrin Tolia, Deepali Khushraj, Anugeetha Kunjithapatham, Anupam Joshi, Tim Finin, and Yelena Yesha (2002, September). Allia: Alliance-based Service Discovery for Ad-Hoc Environments. In *ACM Mobile Commerce Workshop.*

Ritter, Jordan (2001). Why Gnutella Cant Scale. No, Really.

Rowstron, A. and P. Druschel (2001, November). Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 329–350.

Sandholm, Tuomas, Kate Larson, Martin Andersson, Onn Shehory, and Fernando Tohm (1999). Coalition structure generation with worst case guarantees. *Artif. Intell. 111*(1-2), 209–238.

Schwiegelshohn, Uwe and Ramin Yahyapour (2004). *Grid Resource Management - State of the Art and Future Trends*, Volume 1 of *International Series in Operations Research and Management Science*, Chapter Attributes for communication between grid scheduling instances. Kluwer Academic Press.

Selinger, Patricia G., Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price (1979a). Access path selection in a relational database management system. In Philip A. Bernstein (Ed.), *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, Boston, Massachusetts, May 30 - June 1*, pp. 23–34. ACM.

Selinger, P. Griffiths, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price (1979b). Access path selection in a relational database management system. In

Proceedings of the 1979 ACM SIGMOD international conference on Management of data, pp. 23–34. ACM Press.

SETI. Seti@home. "http://setiathome.ssl.berkeley.edu".

Sgall, Jiri (1998). Online scheduling – a survey, online algorithms: The state of the art. *Lecture notes in Computer Science* (1442), 196–231.

Shah, Mehul A., Joseph M. Hellerstein, Sirish Chandrasekaran, and Michael J. Franklin (2003). Flux: An adaptive partitioning operator for continuous query systems. In *ICDE*, pp. 25–36.

Steinburnn, M, G Moerkette, and A Kemper (1997, August). Heuristic and randomized optimization for the join ordering problem. *Very Large Data Bases Journal 6*(3), 191–208.

Stoica, Ion, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan (2003). Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw. 11*(1), 17–32.

Strom, Robert et al. (1998). Gryphon: An Information Flow Based Approach to Message Brokering. Technical report, IBM TJ Watson Research Center, 30 Saw Mill River Road, Hawthorne, NY 10532, USA. http://http://www.research.ibm.com/gryphon/issre98/ext-abstract.html.

Swami, A. and A.Gupta (1988, June). Optimization of large join queries. In the proceedings of the ACM SIGMOD conference on Management of Data, Chicago, IL, pp. 8–17.

Swami, Arun N. (1989). Optimization of large join queries: Combining heuristic and combinatorial techniques. In James Clifford, Bruce G. Lindsay, and David Maier (Eds.), *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, May 31 - June 2, 1989*, pp. 367–376. ACM Press.

Sycara, K., J. Lu, M. Klusch, and S. Widoff (1999). Dynamic service matchmaking among agents in open information environments. In *Journal ACM SIGMOD Record, Special Issue on Semantic Interoperability in Global Information Systems, Ouksel, A., Sheth, A.(ed.)*.

Terry, Douglas, David Goldberg, David Nichols, and Brian Oki (1992). Continuous queries over append-only databases. In *SIGMOD '92: Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, pp. 321–330. ACM Press.

Tesfatsion, Leigh (2002). Agent-based computational economics (ACE): Growing economies from the bottom up. In *Artificial Life*, Volume 8, pp. 55–82. The MIT Press.

Tsvetovat, Maksim and Katia Sycara (2000). Customer coalitions in the electronic marketplace. In *AGENTS '00: Proceedings of the fourth international conference on Autonomous agents*, pp. 263–264. ACM Press.

UDDI (2004). UDDI Specifications. Published by Oasis Working Group.

Urhan, Tolga, Michael J. Franklin, and Laurent Amsaleg (1998). Cost-based query scrambling for initial delays. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pp. 130–141. ACM Press.

Viglas, Stratis D. and Jeffrey F. Naughton (2002). Rate-based query optimization for streaming information sources. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pp. 37–48. ACM Press.

Wang, Haixun and Carlo Zaniolo (2003). Atlas: A native extension of sql for data mining. In *SIAM Data Management*.

Wang, Jie (Ed.) (2001). *On-Line Deadline Scheduling on Multiple Resources*, Volume 2108 of *Lecture Notes in Computer Science*. Springer.

Wiil, U. K. (1997). Open hypermedia: Systems, interoperability and standards. *Journal of Digital information 1*(2).

Wooldridge, M. and N. R. Jennings (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review 10*(2), 115–152.

Zhao, B. Y., J. D. Kubiatowicz, and A. D. Joseph (2001, April). Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB/CSD-01-1141, UC Berkeley.

Zhou, J., Vijay Dialani, D. De Roure, and W. Hall (2003). Parallel and distributed computing, applications and technologies, 2003. In *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies, 2003*, pp. 7–11.