# INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

# The Distributed Utility Model Applied to Optimal Admission Control & QoS Adaptation in Multimedia Systems & Enterprise Networks

by

Md Mostofa Akbar

B.Sc. (Computer Science and Engineering), Bangladesh University of Engineering and Technology, Dhaka, 1996

M.Sc.( Computer Science and Engineering), Bangladesh University of Engineering and Technology, Dhaka, 1998

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

We accept this dissertation as conforming
to the required standard

---

Dr. E. G. Manning, Supervisor (Departments of Computer Science and Electrical & Computer Engineering)

---

Dr. G. C. Shoja, Supervisor (Department of Computer Science)

---

Dr. J. Ellis, Departmental Member (Department of Computer Science)

---

Dr. F. Gebali, Outside Member (Department of Electrical & Computer Engineering)

---

Dr. K. F. Li, Outside Member (Department of Electrical & Computer Engineering)

---

C. Hobbs,External Examiner (Nortel Networks)

Supervisors: Dr. E. G. Manning and Dr. G. C. Shoja

# ABSTRACT

Allocation and reservation of resources, such as CPU cycles and I/O bandwidth of multimedia servers and link bandwidth in the network, is essential to ensure Quality of Service (QoS) of multimedia services delivered over the Internet. We propose a *Distributed Multimedia Server System* (DMSS) configured out of a collection of networked multimedia servers where multimedia data are partitioned and replicated among the servers. We also introduce *Utility Model - Distributed* (UM-D), the distributed version of the Utility Model, for admission control and QoS adaptation of multimedia sessions to maximize revenue from multimedia services for the DMSS.

Two control architectures, a centralized and a distributed, have been proposed to solve the admission control problem formalized by the UM-D. In the centralized broker architecture, the admission control in a DMSS can be mapped to the *Multidimensional Multiple-choice Knapsack Problem* (MMKP), a variant of the classical 0-1 Knapsack Problem. An exact solution of MMKP, an NP-hard problem, is not applicable for the on line admission control problem in the DMSS. We therefore developed three new heuristics, M-HEU, I-HEU and C-HEU for solving the MMKP for on-line real-time admission control and QoS adaptation. We present a qualitative analysis of the performance of these heuristics to solve admission control problems based on the worst-case complexity analysis and the experimental results from different sized data sets.

The fully distributed admission control problem in a DMSS, on the other hand, maps to the *Multidimensional Multiple-choice Multi Knapsack Problem* (MMMKP), a new variant of the Knapsack Problem. We have developed D-HEU and A-HEU, two new distributed heuristics to solve the MMMKP. D-HEU requires a large number of messages and it is not suitable for a on line admission controller. A-HEU finds the solution with fewer messages but achieves less optimality than D-HEU.

We have applied the admission control strategy described in the UM-D to the set of Media Server Farms providing streaming videos to users. The performance of different heuristics in the broker has been discussed using the simulation results. We have also shown application of UM-D to *Distributed SLA* (*Service Level Agreement*) *Controllers* in Enterprise Networks. Simulation results and qualitative comparison of different heuristics are also provided.

**Examiners:**

Dr. E. G. Manning, Supervisor (Departments of Computer Science and Electrical & Computer Engineering)

Dr. G. C. Shoja, Supervisor (Department of Computer Science)

Dr. J. Ellis, Departmental Member (Department of Computer Science)

Dr. F. Gebali, Outside Member (Department of Electrical & Computer Engineering)

Dr. K. F. Li, Outside Member (Department of Electrical & Computer Engineering)

C. Hobbs,External Examiner (Nortel Networks)

# Table of Contents

# List of Figures

# List of Tables

# Glossary of Terms

| | |
|---|---|
| A-HEU | Distributed heuristic solving the MMMKP by arbitrating among the solvers |
| AMS | Adaptive Multimedia System |
| BB | Bandwidth Broker |
| BBLP | Branch and Bound Linear Programming algorithm to solve the exact solution of the MMKP |
| BW | Bandwidth |
| C-HEU | Heuristic for solving the MMKP using Convex Hull approach |
| CoS | Class of Service |
| D-HEU | Distributed heuristic to solve the MMMKP, a distributed version of I-HEU |
| DMSS | Distributed Multimedia Server System |
| DSC | Distributed SLA Controllers |
| D-SLAOpt | Java Simulation of Distributed SLA Controller for interconnected Enterprise Networks |
| DWDM | Dense Wavelength Division Multiplexing |
| EN | Enterprise Network |
| $f_k$ | Infeasibility factor |
| G-HEU | Heuristic for solving the MMKP using Greedy approach |

| | |
|---|---|
| HEU | Heuristic for solving the MMKP |
| I-HEU | Incremental HEU |
| ISP | Internet Service Provider |
| $K$ | Number of candidate paths considered in the SLAOpt and D-SLAOpt |
| $l$ | Number of items in each group of the MMKP |
| $m$ | Number of resource dimensions in the MMKP |
| $M$ | Number of solvers or servers in the MMMKP or DMSS respectively |
| MCKP | Multiple Choice Knapsack Problem |
| M-HEU | Modified HEU |
| MMKP | Multidimensional Multiple Choice Knapsack Problem |
| MMKP_HEU | Class of the heuristics solving the MMKP |
| MMMKP | Multidimensional Multiple Choice Multi Knapsack Problem |
| MPLS | Multi Protocol Label Switching |
| MRMD | Multiple Resource Multiple Dimension |
| $n$ | Number of groups in the MMKP or MMMKP |
| $N$ | Number of nodes in an Enterprise Network |
| O/S | Operating System |
| PCAR | Partial Change of Aggregate Resource and |
| PF | Partial Feasibility |

| | |
|---|---|
| QoS | Quality of Service |
| QoS-B | QoS with respect to Broker |
| SC | SLA Controllers |
| SLA | Service Level Agreement |
| SLAOpt | Java Simulation of SLA Controller for Enterprise Networks |
| TCAR | Total Change of Aggregate Resource and |
| TCP | Transmission Control Protocol |
| TF | Total Feasibility |
| UDP | User Data Protocol |
| UM | Utility Model |
| UM-D | Utility Model Distributed |
| UME | Utility Model Engine |
| VoD | Video On Demand |

# Acknowledgement

I would like to express my deepest appreciation to my supervisors Dr Eric G Manning and Dr Gholamali C Shoja for their thoughtful suggestions. I would like to thank the members of my dissertation committee Dr John Ellis, Dr Fayez Gebali and Dr. Kin Li. I would like to specially thank Dr John Ellis for his suggestions regarding heuristics for solving Knapsack Problems.

I want to thank all the members of PANDA lab. I really enjoyed working with John Foxgrod, Rob Watson, Tim Ducharme, Jian Pu, Eric Growland, Steven Shelford and Doug Johnson. I am especially grateful to Eric Gowland, Steven Shelford, Glenn Mahoney, Numaan Mehryer Huq and Chris Falk for reviewing my dissertation chapters. I also want to thank my friends and graduate fellows of UVic specially Dr Mohamed Watheq El-Kharashi, Shoreh Hadian and Dr Afjal Suleman for giving moral and technical support during my PhD program. Watheq was my big brother during my stay at UVic. Thanks to Isabel, Natasha, Zoria, Marion, Nancy and Sharon for making life in the Computer Science department easier.

My Bangladeshi friends, colleagues, students and teachers all over the world gave me continuous inspiration during my PhD program. Specially I would like to thank Dr Manzur Murshed, Dr Sayed Ansar Mohammad Tofail, Partha Pratim Pande, Abdul Mannan, Farhad Hossain, Mamunul Islam, Masud Karim Khan, Muhibur Rahman, Moniruzzaman Mazumdar, Hossain Tauhiduir Rahman, Jabed Faruque, Imtiaz Ahmed and Mohammad Meftauddin for their moral support.

I would like to express my gratitude to my parents Md Akbar Ali and Tahmina Khatun who always encouraged me to pursue a Ph.D. The inspiration and moral support from my brothers Md Shamim Akbar and Md Abid Akbar, my sister Laila Arjumand Banu, my

brother in law Munshi Golam Mostafa, my nieces Farah Diba and Farah Tajrin deserves special mention. I would like to thank all the farmers of my home village Topon Bhag for wishing me good luck during my study. I would like to thank specially Sirajul Islam Bhuiyan, Shaikh Akhtarul Islam and the late Dr A K M Shirajul Hoque who convinced me to apply for a Commonwealth Scholarship. I am also grateful to Dr M Rezwan Khan and Dr M Ali Chowdhury for recommending me for a Commonwealth Scholarship.

My Bangladeshi friends in Victoria were the source of endless joy and happiness. Thanks to Mohammad Khaliquzzaman for helping me to settle down and carry out daily life. Life without friends like Kuhel Faizul Islam, Mahmud Hasan and Humayun Kabir would have been really tough. We enjoyed our life together while sharing apartments. I always enjoyed friendly company of Aziz, Deena, Sabbir, Raunak, Sharmin, Tuhin, Faruk, Newaz, Suraia, Khaled, Sonia, Ushashi, uncle Mahfuz and aunt Tinna. Thanks to all of them. I used to watch Hindi movies during my Victoria days. Thanks to Yash Chopra, Karan Johar, Vindhu Vinod Chopra, Gulzar, Kamal Hasan, Subhash Ghai, Mehboob Khan, Kundan Shah, Mahesh Manjrekar and Sanjay Lela Bhansali for their hard work in delivering these quality movies. These movies were the source of my recreation.

Last but not the least, I would like to thank the International Council for Canadian Studies for the Commonwealth Scholarship and the New Media Innovation Centre (NewMIC), Vancouver, B.C., Canada for the Newmic Student Scholarship.

*To Mohammad Khaliquzzaman, my best friend in Victoria*

*and*

*to my grand parents*

*Mohammad Shafiuddin Mollah, Chutu Begam,*

*Mohammad Shamsur Rahman Sardar and Hafiza Khatun*

# Part I: Introduction and Literature Review

# 1. Introduction

The distribution of multimedia data over networks is an interesting problem in Computer Science, involving Communication Networks and Distributed Computing. The following are the issues related to distributed multimedia:

- How the multimedia information will be mapped to the various server sites
- How the multimedia streams will be carried from the location of the storage to the users
- How Quality of Service (QoS) of multimedia services will be ensured
- How prices will be determined

Our research addresses these issues, presenting partially and fully distributed admission control and QoS adaptation algorithms for a distributed system of servers, and also for a network.

## 1.1 Motivation

Delivery of multimedia streams with absolute guarantees of QoS from a single multimedia server has been proposed by Khan [44]. The delivery of multimedia streams through the links of a network by controlling admission based on Service Level Agreements has been proposed by Watson [40]. The following problems and prospects lead us to develop a distributed admission control and QoS adaptation scheme for multimedia servers and networks.

Multimedia refers to composite media, media that contain multiple information streams of various types, such as video, image, sound, text, etc. Some multimedia streams such as video, and audio require *absolute* QoS guarantees. There may be associated time constraints on data rates (e.g., NTSC video's 30 frames per second), latency (e.g., a real-time, interactive video conference must have latency of no more than a few hundred ms),

jitter, and inter stream synchronization (e.g., speech lip-synched with talking head images). The delivery of multimedia information with absolute QoS guarantees has presented Internet Service Providers (ISPs) with the following challenges and opportunities:

- Revenue from commercially attractive multimedia services like Video On Demand (VoD), high-quality videoconferencing, interactive video services, Internet Phone, and WebTV could help rescue the technology sector from the current recession (the so-called Tech Wreck in 2000-2001). However, ISPs must be able to provide guaranteed absolute QoS to realize these opportunities.
- Networks must be able to carry multimedia streams with guaranteed absolute QoS. However, the present Internet is based on best effort connectionless datagram service, without any QoS guarantee. This service without any guarantees, is not suitable for paid service.
- The introduction of some form of connection-oriented service atop the IP datagram service is necessary to solve this problem. The required bandwidth for the multimedia sessions must be reserved on a fixed path from the server to the user with low enough latency and jitter in order to guarantee absolute QoS over the network. All IP datagrams of the session must be routed along the fixed path because best effort datagram service does not guarantee timely delivery of multimedia streams on a particular path.
- Similarly, the necessary CPU cycles, I/O bandwidth, and memory in multimedia server(s) must be reserved, to ensure guaranteed delivery of multimedia streams from the multimedia server.
- The resources in the multimedia server (CPU cycles, I/O bandwidth and memory) and in the network (link bandwidths) are finite. If these resources are overbooked then QoS may not be maintained. Therefore, some form of admission control is

necessary, to reject some of the applicant sessions when insufficient resources are available to serve all of them. Maximization of revenue by admitting profitable sessions is an important objective of the admission controller.

## 1.2 Problem Definition and Previous Work

Guaranteed absolute QoS for multimedia service requires *end-to-end* guarantees, covering the server, network and client. More precisely:

- The user's machine must have enough CPU cycles, I/O bandwidth, memory and hard disk space to play the multimedia stream with guaranteed absolute QoS.
- A connection of sufficiently high bandwidth is required from server to client. A multimedia stream must follow a fixed path from server to client, and each link must have enough bandwidth, with low enough latency & jitter, to carry the multimedia stream with the required absolute level of QoS.
- The server serving the multimedia stream must have the capability to deliver multimedia streams to all admitted users with guaranteed absolute QoS. Sufficient server resources (CPU cycles, I/O BW and memory) must be reserved for each multimedia session.

The *Admission Controller* works as a *resource manager* by allocating the resources of the server such as CPU cycles, I/O BW, memory, etc. to the user when her multimedia session starts. It also does *QoS adaptation* dynamically by upgrading or downgrading a session in progress. Each prospective user offers a price for each level of QoS for the multimedia service. This bid price is the basis of admission control and QoS adaptation when there is resource contention in the system. The *Utility Model* (UM), proposed in [46], presents the admission control and QoS adaptation of a single server multimedia service provider, to maximize the utility (revenue earned) from the bids offered to a multimedia service provider by the users. Some economists recommend bidding as a good approach for resource allocation when there are finite resources for a service [57].

A particular amount of bandwidth must be reserved on a path from the server to the user for transmission of multimedia streams such as audio and video with particular delay and jitter bounds. An *SLA* (*Service Level Agreement*) between the user and network owner for delivery of the multimedia streams must be specified, with different data rates, delays and jitter bounds (one for each QoS level) from the source (the node connected to the multimedia server) to the destination (the node connected to the user). An *Enterprise Network* (*EN*) is a network with limited nodes and links administered by a single organization or an autonomous subsidiary of an organization. An *SLA Controller*, an engine doing admission control and QoS adaptation for SLAs in Enterprise Networks, works as a resource manager for the bandwidth on the links of the network. A Java simulation of a centralized SLA Controller has been developed by Watson [40] for single Enterprise Networks.

With the dramatic increase of communication link capacities and decrease in unit cost of bandwidth due to Dense Wavelength Division Multiplexing (DWDM) optical fibre transmission [89], it will be possible to transmit multimedia streams very cheaply, and hence the demand for multimedia streams over internets is expected to be very large. To ensure the quality of such services, high performance multimedia servers, which contain the digital real time data of these multimedia services, must be deployed. However, the amounts of data to be delivered, and delivered with real time constraints, are so huge, and in many cases the anticipated participants or viewers are so many, that it will be clearly impossible to maintain all the data in just one server. The limited capacities of servers, the desire to exploit geographic locality of reference, and the need for fault tolerance, are all reasons to partition or replicate different components of multimedia streams across multiple servers, which is the basic principle of distributed multimedia service.

The users of a multimedia application are not distributed evenly all over the world. Actually they are concentrated in the big cities. We can exploit this geographic locality of reference by providing servers in different locations, which are closer to the groups of users. Let there be servers in the big cities of each continent for a particular movie

released in North America. The servers in Asia can serve the users of Asia and the servers in North America can serve the users in North America. Thus the cost of bandwidth from North America to Asia can be saved. The delay and jitter of transmission can be reduced as well. Besides these, if the servers of North America are fully loaded in the evening and the servers of Asia are lightly loaded at that time, due to 12 hours local time difference, then additional users in North America can be served using the servers in Asia. Thus the difference in local times of the server sites can improve the scalability and fault tolerance for online multimedia service.

To demonstrate the partition of the multimedia streams we can take an example of a video server farm. Let there be several servers serving the on-line video of the recent movies. The users are also allowed to browse reviews of the movies containing texts and images. As the size of this text and images is not large compared with the MPEG video of the movie, one data server is enough. The number of required video servers depends on the current load of users enjoying the movies. Thus partitioning of multimedia components makes efficient use of multimedia servers.

There might be multiple interconnected Enterprise Networks in an organization, each of which is administered by an autonomous subsidiary. SLA controllers in the ENs must run distributed algorithm to do admission control and QoS adaptation of the SLAs. Figure 1.1 shows a typical example of two SLA Controllers working together. This is more manageable, scalable and fault tolerant architecture for SLA controller. In this dissertation, we present a new, distributed version of the Utility Model. This *Utility Model- Distributed* (UM-D) is applicable to do admission control and QoS adaptation in a *Distributed Multimedia Server System* or a group of *interconnected Enterprise Networks*.

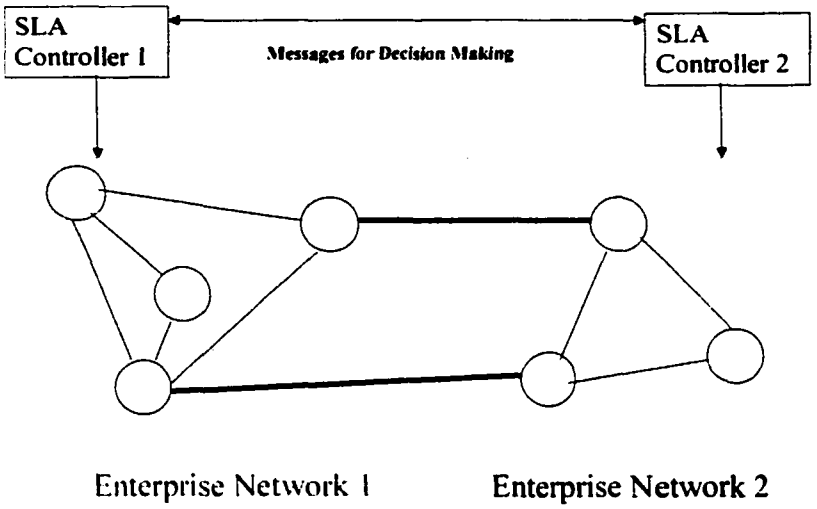


*Figure 1.1 Group of SLA Controllers working together*

## 1.3 Scope and Focus

The main focus of this dissertation is to present models, architectures and algorithms for distributed multimedia service over the network. Admission control problems to be discussed in this dissertation can be mapped to variants of the Knapsack Problems, which are NP- hard problems. We apply heuristics to solve these problems for on line admission control and QoS adaptation. Development of exact algorithms is beyond the scope of this dissertation. To analyse the performance of an admission controller, we developed a discrete event simulation of the admission controller. On the other hand to demonstrate the working methodology we simulated a system in Java with a graphical user interface. Implementation or developing the prototype of the working system is beyond the scope of this dissertation.

## 1.4 Outline

The dissertation is organized in three parts and nine chapters as shown in the following figure.

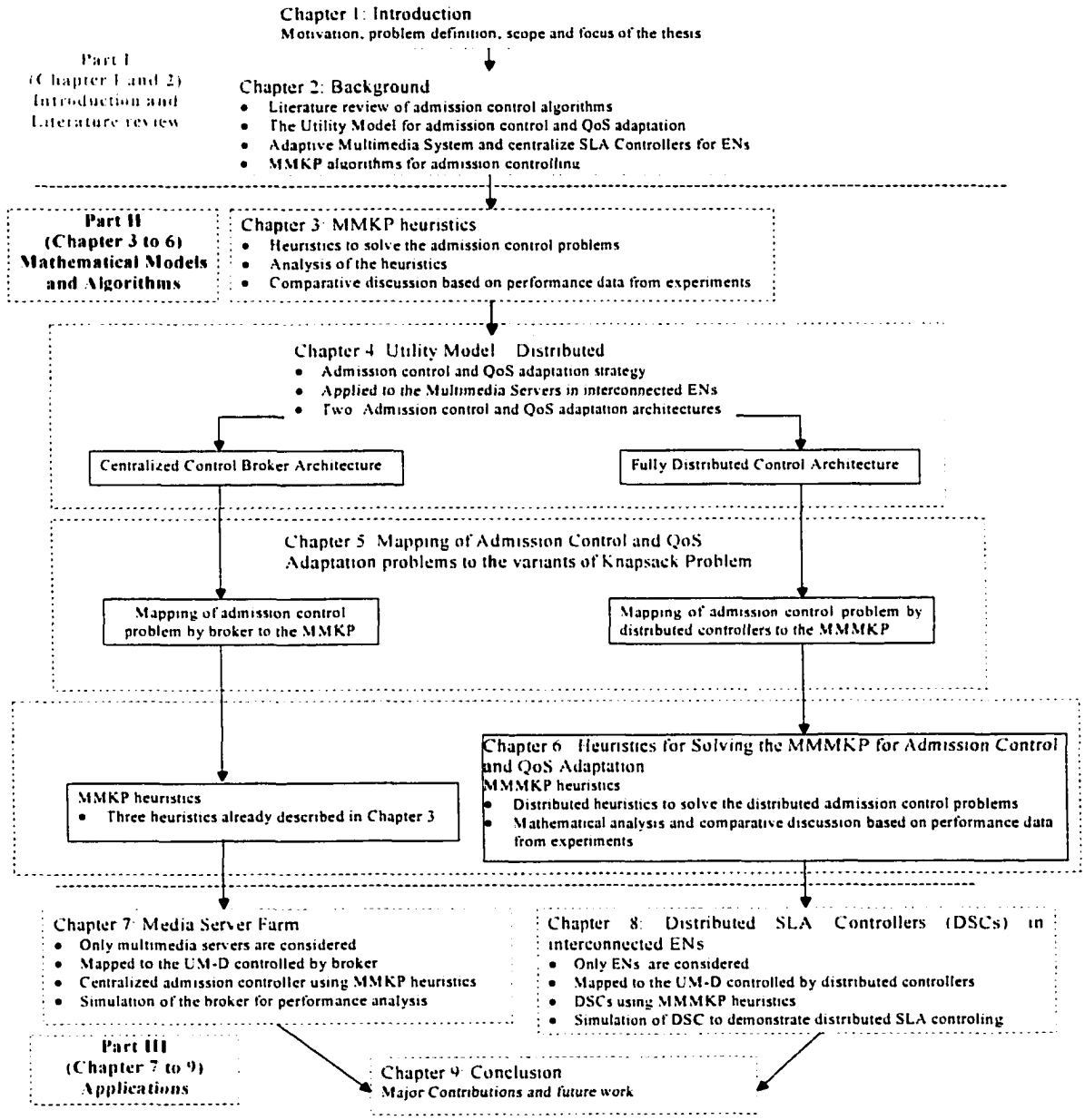


*Figure 1.2 Organization of the dissertation chapters*

# 2. Background

Different admission control approaches, which are currently being used in the Internet, telephone networks and multimedia server systems, will be reviewed in this chapter. The Utility Model and its application to the Adaptive Multimedia Problem and Enterprise Networks to do the admission control and QoS adaptation problem will be described briefly. The admission control and QoS adaptation using the Utility Model can be mapped to a variant of Knapsack Problem. A detailed introduction to Knapsack Problems and the algorithms to solve these problems will be described in the following sections.

## 2.1 Literature Review of Admission Control and QoS Adaptation Techniques

Admission control is a useful technique in telecommunication networks to achieve quality of the service. Different strategies are adopted to do admission control in different types of network. In circuit switching networks, a circuit is not established if there is not enough free bandwidth to create that circuit. On the other hand, the Internet, which is basically a packet switched network, uses traffic shaping for QoS adaptation during congestion.

### 2.1.1 Admission Control in Telephone Network

The telephone network, which carries audio between a source (the caller terminal) and a destination (the called terminal), is the best example of a circuit switched network. When a user dials a number, the telephone switch finds a free end-to-end voice circuit to carry the audio transmission. If such a path is available then the call is set up, otherwise the user gets the busy tone. All users are guaranteed to receive uninterrupted service if a call is set up. Throughput maximization in a telephone network depends on the algorithm used for selecting particular paths for new circuits. Plotkin [3] presented online competitive routing and admission control strategies for both throughput maximization

and congestion minimization models, motivated by competitive analysis [1]. In both models each link in the network is defined by a *length* $c_e$, which is defined exponentially with regard to the current congestion of the link. Let there be a request for bandwidth between nodes $s$ and $t$. In the throughput maximization model, the flow is routed if there exists a path $P(s,t)$ such that the flow is profitable with respect to this length $c_e$. If it is not, the flow is rejected. In the congestion minimization model, a rejection edge *rej* is created and a flow is routed along the shortest path with respect to the length $c_e$. The flow is then considered rejected if it is routed along *rej*, or if the path selected has insufficient capacity. Otherwise, it is considered accepted.

Because of only one QoS level in telecommunication networks, there is no opportunity for downgrading or upgrading the QoS level by changing the allocated bandwidth or the already selected path. Therefore, there is no way to optimize operating conditions dynamically. This is true for the sessions with almost invariable length or for the permanent calls. When the session lengths are unknown, it is important to admit a set of sessions that give maximum throughput for the system. This would require future prediction of the length of the sessions. Statistical distributions can be used to estimate the duration of a particular call. Admission decision with a *risk factor* [1] is also a good technique for prediction.

### 2.1.2 Resource Reservation in IP Data Networks

Currently, the Internet is based on a best-effort datagram service model: this model does not require (and generally does not permit) resource reservation prior to data transmission. When a packet arrives at a router, and sufficient resources (such as time and buffer-space on the outgoing link) are available, the packet is forwarded to the next router. However, if the necessary resources are not available, the incoming packet may be delayed, or even dropped. It is therefore difficult to predict, let alone guarantee, the bandwidth or latency experienced by a stream of packets under best-effort datagram services. And since each packet of a session is forwarded through the network

independently, packets may experience variable and unpredictable delays, and may arrive at the destination out of order. This service has many advantages, but it is unworkable for real-time multimedia applications requiring absolute standards of performance such as continuous bandwidth during a Video on Demand session with maximum delay and jitter constraints. Hence a best-effort datagram service model is not considered suitable for the Internet2 [75], which proposes to offer the end-to-end quality-of-service guarantees similar to that of the telephone network.

Class-based forwarding proposals such as DiffServ [61], where the packets of applications requiring guaranteed QoS are assigned higher priority classes than the best-effort traffic, are at best partial solutions: they provide superior service to the QoS-sensitive application in the *relative* sense, i.e., relative to the best-effort traffic, but they cannot guarantee *absolute* standards of QoS to applications, including telephony and interactive video communication, which require such standards.

RSVP [60] is a protocol used to reserve resources i.e., link bandwidth over the Internet. It requires reservation in each switch from the source to the destination, which clearly requires the determination of a fixed path on which all datagram of the flow are carried. This protocol works for real-time audio and video transmission, and in that sense could provide a basis for guaranteed QoS, but scalability is a problem.

MPLS [59] provides a mechanism for sending data independent of the IP routing tables in the routers. In this mechanism each packet is routed through a predefined path, which is determined before data transmission. A label is added to the packet, and this label is used for table look up in the router for forwarding packets to the next router with another label. The label forwarding table is created at the time of fixing the path and it contains additional information such as *Class-of-Service* (CoS) values that can be used to prioritize packet forwarding. As MPLS is a lower layer protocol than IP and UDP, real-time multimedia transmission using IP or UDP over MPLS is considered plausible.

less than a threshold. Calls are downgraded by bit dropping in real time data transmission, or by coarser video encoding to maximize the sharing of resources among the calls. Srikant [74] proposed the central limit theorem for the approximation of probability of service interruption.

Scheduling algorithms are used to do admission control in optical and wireless networks where bandwidth is shared by the users of the network. Dasylva [77] presents a polynomial algorithm to produce optimal schedules under certain conditions of traffic metrics. Shakkottai [73] discusses the problem of real-time streams with deadlines over a shared channel using different scheduling algorithms.

Asynchronous Transfer Mode (ATM) is a packet switch technique where the data is divided into 53 bytes cells and multiplexed on time slotted channels [94]. ATM switches use Virtual Circuits (VCs) and all the packets of a call follow the same route. When a cell arrives at a switch, the switch determines an outgoing link looking at the VC number in the header of the ATM cell. Courcoubetis [93] designed an admission control algorithm for call acceptance that guarantees a bound of cell loss because of buffer overflow.

In the smart market scheme introduced by Varian [91], the actual price for each packet is determined based on the current state of network congestion. Users offer a bid for their packets. The packets whose bids are more than a threshold will be admitted and the rest are dropped or buffered. Clark [91] suggested an economic model where users pay for the their privilege of using the network capacity when needed. Singh [90] proposed a dynamic capacity contracting model which is implementable in the differentiated services architecture of the Internet.

## 2.2 Knapsack Problems

The classical 0-1 Knapsack Problem (KP) is to pick up items for a knapsack for maximum total value, so that the total resource required does not exceed the resource constraint $R$ of the knapsack. The 0-1 classical KP and its variants are used in many

resource management applications such as cargo loading, industrial production, menu planning and resource allocation in multimedia servers. Let there be *n items* with *values* $v_1,v_2,\ldots,v_n$ and let the corresponding resources required to pick the items be $r_1,r_2,\ldots,r_n$ respectively. The items can represent *services* and their associated values can be values of *revenue* earned from that service. In mathematical notation, the 0-1 Knapsack Problem is to find $V = \text{maximize}\sum_{i=1}^{n} x_i v_i$, subject to the constraint $\sum_{i=1}^{n} x_i r_i \le R$ and $x_i \in \{0,1\}$. The 0-1 Knapsack Problem is an NP-Hard problem [47]. There is a pseudo polynomial algorithm with $O(nR)$ computational complexity by using the concept of dynamic programming.

The Multidimensional Multiple-choice Knapsack Problem (MMKP) is a variant of the classical 0-1 KP. Let there be $n$ *groups of items*. Group $i$ has $l_i$ items. Each item of the group has a particular value and it requires $m$ resources. The objective of the MMKP is to pick exactly one item from each group for maximum total value of the collected items, subject to $m$ resource constraints of the knapsack. In mathematical notation, let $v_{ij}$ be the value of the $j$ th item of the $i$ th group, $\vec{r}_{ij} = (r_{ij1}, r_{ij2}, \cdots, r_{ijm})$ be the required resource vector for the $j$ th item of the $i$ th group and $\vec{R} = (R_1, R_2, \cdots, R_m)$, be the resource bound of the knapsack. Now, the problem is to find

$V = \text{maximize}\sum_{i=1}^{n}\sum_{j=1}^{l_i} x_{ij} v_{ij}$, (objective function),

so that, $\sum_{i=1}^{n}\sum_{j=1}^{l_i} x_{ij} r_{ijk} \le R_k$ (resource constraints),

where, $V$ is the *value of the solution*, $k = 1, 2,\ldots, m$, $x_{ij} \in \{0,1\}$ are the picking variables, and

$$\sum_{j=1}^{l_i} x_{ij} = 1 \ .$$

Figure 2.1 illustrates an MMKP. We have to pick exactly one item from each group. Each item has two resources, $r_1$ and $r_2$. The objective of picking items is to maximize the total

value of the picked items subject to the resource constraints of the knapsack, that is $\sum(r_1 \text{ of picked items}) \leq 17$ and $\sum(r_2 \text{ of picked items}) \leq 15$.

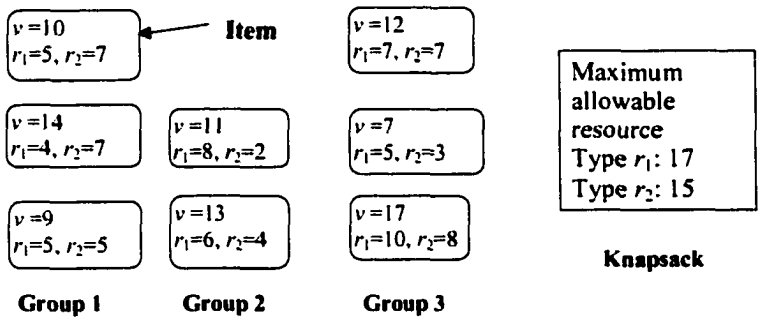


*Figure 2.1 Multidimensional Multiple-choice Knapsack Problem.*

## 2.3 Related Research on KP

There are various algorithms for solving variants of Knapsack Problems [47]. The Multidimensional Knapsack Problem (MDKP) is one kind of KP where the resources are multidimensional, i.e. there are multiple resource constraints for the knapsack. The Multiple Choice Knapsack Problem (MCKP) is another KP where the picking criteria for items are restricted. In this variant of KP there are one or more groups of items. Exactly one item will be picked from each group. Actually, the MMKP is the combination of the MDKP and the MCKP.

There are two methods of finding solutions for an MMKP: one is a method for finding exact solutions and the other is heuristic. Finding exact solutions is NP hard. Using the Branch and Bound with Linear Programming (BBLP) technique, Kolesar [36], Shih [50], Nauss [41] and Khan [44] presented exact algorithms for 0-1 KP, MDKP, MCKP and MMKP respectively.

The branch and bound algorithm for the MMKP involves the iterative generation of a *search tree*. A *node* of the tree is expanded by selecting an item of a particular group, called *branching group*. At a node, if the items of a group are not selected then the group is called *free group*. Initially there is only one node in the tree where all the groups are

free. Applying linear programming technique on the free groups of a node we can determine an *estimate of optimum total value* as well as the branching group of a node. The use of linear programming to determine the branching group reduces the time requirement in the average case. In each iteration the node with the highest upper bound is explored. The nodes, which do not give any solution value using linear programming are considered as infeasible. These nodes are deleted from the tree.

A solution is found when a node without any free group has the maximum estimated total value. Though the search space for a solution in MMKP is smaller, because of more restriction of picking items from a group, than the search space in other variants of KP, it is not applicable to our on line admission control problem. Experimental results in Section 3.6.2 presents the time requirements for BBLP algorithms. Please see [46] for a detailed explanation of BBLP with examples.

A greedy approach has been proposed [44][47][53] to find near optimal solutions of Knapsack Problems. For a 0-1 KP as described in the previous subsection, items are picked from the top of a list sorted in descending order on $v_i/r_i$ (value per unit resource) because these items seem to be the valuable and profitable items [47].

To apply the greedy method to the MDKP Toyoda proposed a new measurement called *aggregate resource consumption* [53]. Khan [46] has applied the concept of aggregate resource consumption to pick a new candidate item in a group to solve the MMKP. Aggregate resource of the *j*th item of the *i*th group is defined by $a_{ij} = \sum_k r_{ijk} \times C_k \Big/ |C|$, where $C_k$= amount of the *k*th resource consumption and $|C| = \sqrt{\sum C_k^2}$. His heuristic HEU selects the lowest-valued items by utility or revenue of each group as initial solution. It then upgrades the solution by choosing a new *candidate item* from a group, which has the highest positive $\Delta a_{ij}$, the change in aggregate consumed resource (the item which gives the best revenue with the least aggregate resource). If no such item is found then an item

with the highest $(\Delta v_{ij})/(\Delta a_{ij})$ (maximum value gain per unit aggregate resource expended) is chosen. Here,

$$\Delta a_{ij} = \sum_k \left(r_{i\rho[i]k} - r_{ijk}\right) \times C_k \text{, the increase in aggregate consumed resource.}$$

$r_{ijk}$= amount of the $k$th resource consumption of the $j$th item of the $i$th group.

$\rho[i]$=index of selected item from the $i$th group and $\Delta v_{ij} = v_{i\rho[i]} - v_{ij}$, is the gain in total value.

We do not use the constant $|C|$ in the expression of $\Delta a_{ij}$, as it does not vary during the selection of a candidate item.

This heuristic for MMKP provides solutions with total value on average equal to 94% of the optimum, with a worst-case time complexity of $O\left(mn^2(l-1)^2\right)$. Here, $n$= number of groups, $l$ =number of items in each group (assumed constant for convenience of analysis) and $m$ =resource dimension.

Magazine and Oguz [29] proposed another heuristic based on Lagrange Multipliers to solve the MDKP. Moser's [27] heuristic also uses the concept of graceful degradation from the most valuable items based on Lagrange Multipliers to solve the MMKP. This algorithm is also suitable for real time applications.

Tabu Search [97], Simulated Annealing [98] and Genetic Algorithms [96] can be applied to solve the variants of Knapsack Problem. The Genetic algorithm has the exponential worst case complexity - it can explore all of the items. Tabu search and simulated annealing is based on looking at the neighbours. These are costlier than the greedy approach used in HEU. HEU uses a two way interchange approach and searches candidates in the neighbourhood which yield better revenue, and changes one selection to another. But in Tabu Search, Simulated Annealing and Genetic Algorithm approach current solution is moved to another solution by upgrading some and downgrading some.

This upgrade and downgrade at the same step requires more time because we have to search all neighbouring combinations of current solution.

## 2.4 Adaptive Multimedia System (AMS)

We define a multimedia service provider, which provides multimedia service to users with guaranteed QoS levels and supports QoS adaptation an Adaptive Multimedia System(AMS) [46]. Each user submits its session request together with a set of QoS levels such as Gold (colour video and CD quality audio), Silver (B/W Video with telephone quality audio), or Bronze ( phone quality audio). Depending on the availability of the resources of the AMS such as CPU cycles, I/O bandwidth and memory, a session can be admitted to enjoy a particular QoS level, or rejected. There must be an engine working as an admission and QoS adaptation controller in the AMS. For brevity we call it an *Admission Controller*. It keeps track of all the allocated resources of the system. When a new user places a request for a multimedia service, or when a session leaves, the controller can dynamically adapt the QoS level of any running session to allocate some resource or to re-allocate newly- released resources. Considering multiple QoS level makes the problem of admission control and QoS adaptation more complex. But this also gives the opportunity to upgrade to a higher level and earn more revenue when some resources are released from the system.

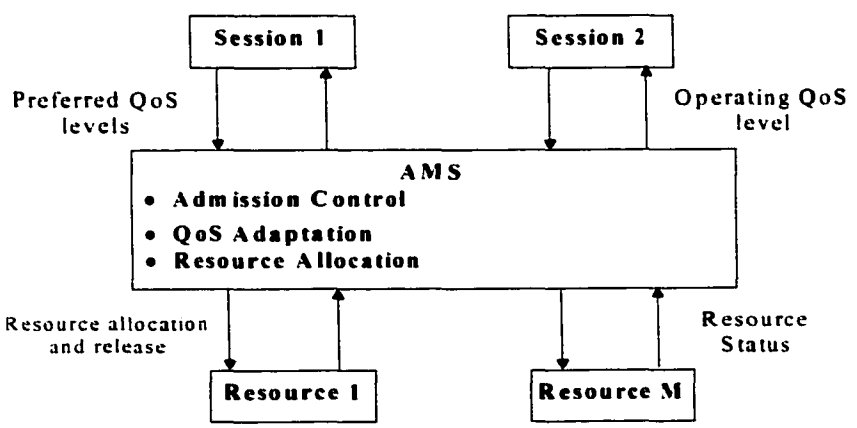


*Figure 2.2 Adaptive Multimedia System*

## 2.5 Literature Review of Admission Control and QoS Adaptation Methodology for AMS

There has been lots of interesting work in recent years on reservation-based management of resources, like CPU cycles and network bandwidth for multimedia service providers [6][22][21][40][32].

The *Benefit Model* for adaptation of quality attributes of a single user multimedia application has been proposed by Schreier and Davis [24]. The quality of the service is expressed by the video frame rate, audio/video quality and audio/video synchronization. Each of these quality parameters has an associated benefit function. The objective is to maximize the benefit of the service by adjusting the quality parameters. Chatterjee [42] proposed a *Logical Application Stream Model* (LASM) to capture the structure of a distributed multimedia application, with relevant resource requirement as well as end-to-end QoS parameters. Moser [28] presented *an Optimally Graceful QoS Degradation Model* (OGQD) where a single session's quality is gracefully degraded to meet resource constraints. For a multimedia session the system calculates the set of services for maximum utility subject to resource constraints using a heuristic for solving the MMKP [27].

However, the Benefit Model, LASM and OGQD discuss the adaptation of QoS for a single multimedia session. They do not address the problem of adaptation in a multi session environment with a predefined objective like revenue or utility maximization.

Venkatasubramanian [33] proposed an economic framework for a multi-user multimedia service provider with different objectives for the users and for the service provider. In this framework a user's objective is to maximize QoS with respect to paid price but the service provider's objective is to maximize revenue with respect to resource usage of the system. This principle does not ensure the maximum utilization of resources of the service provider.

Kawachiya [21] has proposed a processor execution model for a multimedia operating system. It reserves the resources for the sessions and changes the quality of the sessions during contention according to downgrading options set by the users. It finds a suitable operating quality for each session at a particular state of the system.

Lee [7] applied the concept of convex hull to solve the QoS management of a MRMD (Multiple Resource Multiple QoS Dimension) system. Each QoS of a multimedia session request is transformed to a point in a two dimensional space. A convex hull frontier is constructed with the points representing the QoS levels of each requested session. Admission or rejection, as well as QoS adaptation of a session, is based on the list of all the segments of the convex hull frontiers sorted in ascending order according to their slopes.

## 2.6 The Utility Model

In this dissertation we define *utility* as the revenue earned by the server from a user enjoying a service. The term utility can also be defined by different kinds of satisfaction of the user (response time, quality of the service etc.), but we are not considering those in this dissertation. The Utility Model, proposed by Khan [43][44][46] demonstrates the admission control and QoS adaptation principles of the AMS. Let there be $n$ session requests $s_1, s_2, \ldots \ldots s_n$ from $n$ users in a multimedia service provider. Session $s_i$ has $l_i$ QoS levels $\vec{q}_{i1}, \vec{q}_{i2}, \ldots\ldots, \vec{q}_{ij}, \ldots, \vec{q}_{il_i}$. Each QoS level $\vec{q}_{ij}$ requires $m$ resources (CPU cycles, memory, I/O and network bandwidth etc.) from the server system, which can be denoted by the vector $r(\vec{q}_{ij}) = (r_{ij1}, r_{ij2}, \ldots, r_{ijm})$. We assume that all the resources are additive. The offered price (utility) for QoS level $\vec{q}_{ij}$ is $u_{ij}$. The principle of selecting QoS levels of the sessions can be expressed in mathematical notations as follows:

*Objective:* $U = \sum_{i=1}^{n} \sum_{j=1}^{l_i} x_{ij} u_{ij}$ is maximized (i.e., maximum revenue is earned),

Where, $x_{ij} = \begin{cases} 1, \text{iff QoS } j \text{ of session } i \text{ is selected} \\ 0, \text{otherwise} \end{cases}$

*Constraints:* $\sum_{j=1}^{l_i} x_{ij} = 1$ (i.e., a single QoS level is selected at any particular instant of time)

and $\sum_{i=1}^{n} \sum_{j=1}^{l_i} x_{ij} r_{ijk} \le R_k$ (i.e., the sessions must consume less resources than the total capacity), where, $R_k$ is the total amount of the $k$th resource in the server.

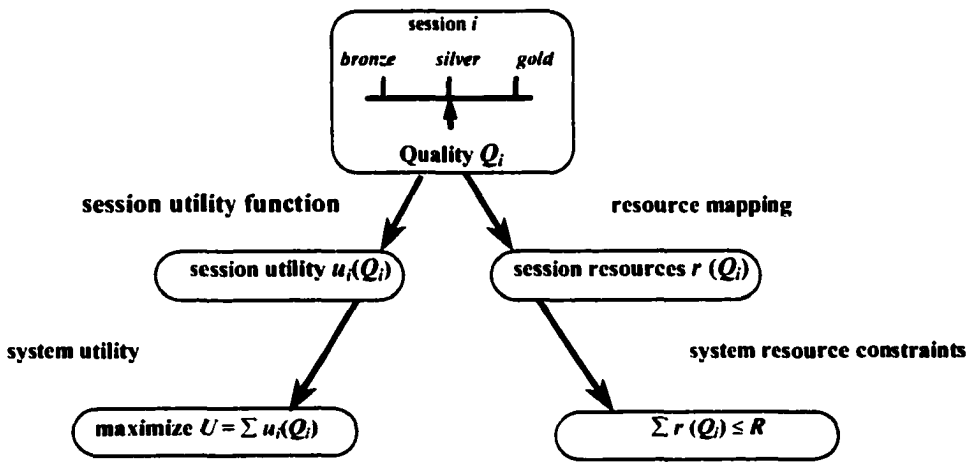


*Figure 2.3 Relations between system and session utilities, and between resource mappings and constraints*

This model exactly fits an MMKP, where sessions are mapped to groups, sessions at QoS levels to items and associated utilities to the values of the items. Figure 2.3 shows this mapping clearly. The target of the Utility Model is to maximize the utility; we do not consider maximization of resource utilization in this model.

## 2.7 Working Principle of Admission Controller

Admission control and QoS adaptation is done on *batches* of sessions. The heuristics for solving the MMKP are applied to a batch of sessions once in a regular time interval which is called an *epoch*. In each epoch some of the old sessions leave and some new session requests are batched for admission. Thus there is a change of resource usage in the

system. The other sessions in the batch are previously admitted sessions, which currently enjoy a particular QoS level. For the new session requests, we add a null QoS level to the QoS profile. This is actually a dummy QoS level with no offered price and no resource requirements. The heuristics find the set of QoS levels for the old sessions and newly admitted sessions once in each epoch. The new solution may have new QoS levels for one or more already-admitted sessions. The new QoS levels can be interpreted as follows:

- If a new session request has a non null QoS level then the user starts enjoying multimedia service. Thus the session gets admission with that QoS level.
- If the QoS level of a new session remains null then the request is rejected. It may wait for another epoch to get admission, it may raise its offered price, or it may simply withdraw.
- The QoS level of an existing session may be upgraded or downgraded to a higher or lower QoS level. For reasons of good customer relations, a session is not downgraded to a lower QoS level until and unless the associated customer permits us to do so.

The null QoS level is removed from the session's QoS profile if it gets admission with a non-null QoS level. The heuristic must be able to determine the level of QoS for the sessions in the next epoch.

## 2.8 SLA Controller for an Enterprise Data Network

The Utility Model can be easily applied to admission and QoS adaptation control of Service Level Agreements (SLAs) for allocation of link bandwidths in a data network. To transmit components of multimedia streams like video, and audio we need guaranteed bandwidth as well as latency over the network. So, the application of the Utility Model is necessary to support the multimedia servers, by providing the necessary bandwidth while maximizing revenue for the network operator.

### 2.8.1 Enterprise Network

An *Enterprise network* (EN) is a private network, usually with fewer than 100 nodes (switches), and administered by a single organization. This network can be used by the company for data transmission. For example, a company can set up an Enterprise Network among all of its offices in North America. Teleconferencing or videoconferencing may be the applications over this Enterprise Network. Similarly, a telecommunication company can set up an EN especially for multimedia stream transmission. A multimedia user will submit her request to the owner of the EN for bandwidth from a source node to a destination node of the EN. Figure 2.4 shows a small network with five nodes and three SLAs from node $S$ to node $D$, represented by the dotted lines, thick lines and thick broken lines respectively.

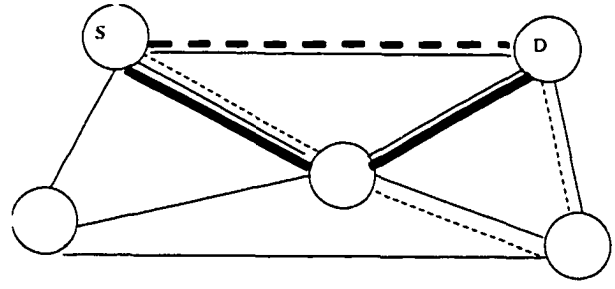


*Figure 2.4 A simple Enterprise Network with three SLAs*

### 2.8.2 SLAs in an Enterprise Network

The $i$th SLA in the list of submitted SLAs is defined as follows:

**Source and Destination:** Nodes of the network labelled $S_i$ and $D_i$.

**Delay Bound:** The propagation delay and the switching delay in the intermediate links and switches between $S_i$ and $D_i$ must be bounded above by $d_{ij}$ for the $j$th level of QoS for $SLA_i$.

**Bandwidth Requirement:** QoS level $j$ of $SLA_i$ requires bandwidth $B_{ij}$ between $S_i$ and $D_i$. Thus, all the links on the path chosen between $S_i$ and $D_i$ must reserve bandwidth $B_{ij}$ for $SLA_i$, when operating at QoS level $j$. We can loosen this

requirement, allowing overbooking of a link, only if we do not guarantee the promised level of QoS. In our research we reject that alternative.

**Duration:** Duration of the multimedia session to be enjoyed by the user.

**Rate:** There must be a time rate of revenue or utility earned by the SLA. This rate will be in dollars per unit time.

**QoS levels:** Users are allowed to submit different options of desired QoS level and bid price. So, an SLA specifies a set of QoS levels determined by the required bandwidth and delay constraint. Note that there may be more than one path from source to destination, with different values of delay.

**Flow Attributes :** An SLA also specifies several attributes of the data flow from source to destination. These are as follows:

***Path Restriction Flag**: This flag specifies whether or not the path can be changed after the transmission of data starts. If the flag is set we cannot re-route this SLA in order to upgrade or downgrade the QoS level of this SLA or other SLAs, or to admit a new SLA.*

***Down Flag**: If this flag is set then the level of QoS of an SLA can be downgraded after admission. (This is not always advisable for psychological reasons, just as it is not always advisable to downgrade a business class airline passenger to economy.)*

***Up Flag**: If this flag is set, then upgrading of the QoS level is permitted after the admission of an SLA.*

If we control the admission of multimedia traffic in the EN, the switches will not be overloaded beyond their capacity. So the network will not face any significant jitter. That is why we can ignore the jitter bound in this definition of SLA.

### 2.8.3 Application of the Utility Model to an SLA Controller

The Utility Model can be applied for admission control and QoS adaptation of the SLAs in an Enterprise Network. The Engine that runs the Utility Model for maximization of revenue by admission control and QoS adaptation is called the *SLA Controller* (SC). The basic working principle of an SC can be described in Figure 2.5.

***K* shortest paths finder**: This part of the SLA controller does all the preprocessing of SLAs (QoS to resource mapping) to make them applicable to the Utility Model Engine (UME). The *K* shortest paths finder gets delays in the links and switches of the network. The switches of the network send all the statistics to this module of the SLA controller. For each new SLA, the *K* shortest paths, which follow the delay bound of the QoS levels of the SLA are determined. The delay of a particular end - to - end path is calculated by simply adding their associated delays. We assume a static delay on each link, i.e., the number of hops from source to destination defines the total delay on a path. An SLA can be routed in any of these paths. There has been considerable work on *K* shortest paths problems [8]. A path-deviation style algorithm [12][13] has been applied by Watson [40].

**Utility Model Engine (UME):** The Admission control and QoS adaptation problem by the SLA controller can be easily mapped to the Utility Model as follows:

- Each SLA is equivalent to the sessions in the AMS. A path that ensures a QoS level with delay bound gives revenue to the Enterprise Network.
- The link bandwidths are the resources. The bandwidth used by all the SLAs on a link must be less than the capacity of the link. A particular amount of bandwidth is reserved for each SLA.
- The objective of the SLA controller is to maximize the revenue for the Enterprise Network from the SLAs.

The admission control and QoS adaptation algorithm is exactly the same as AMS. Watson [40] implemented SLAOpt, a Java simulation of an optimal admission controller for the SLAs in an EN. This implementation uses N-HEU, a modified heuristic using path nearness among the SLAs to do admission control and QoS adaptation. Extra checking is required to determine whether an upgrade or downgrade complies with the path restriction, up, and down flags. This requires comparison of each QoS level of an SLA

with the QoS level and path selected in the previous epoch. If there is a change in network status due to link or node failure we do QoS adaptation by running I-HEU, but the flags specified in the SLAs are totally ignored. Please see [11] for detailed explanation and algorithms of admission control by SCs.

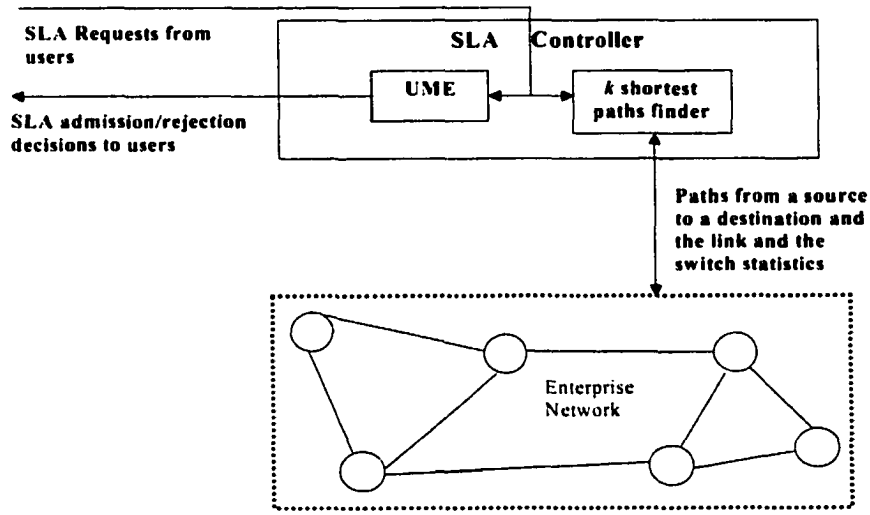


*Figure 2.5 Working principle of SLA controller for an Enterprise Network*

## 2.9 Implementation Considerations for an SLA Admission Controller

The admission controller for an Enterprise Network functions as (part of) the *Bandwidth Broker* (the controller which allocates bandwidth to the users), which in turn is a part of the *Policy Server* (the server defining the policy of the network such as the routing algorithm used, the number of shortest paths to be determined, etc.) [65][75]. Here we discuss the implementation of the admission controller for two cases: to control admissions to a single Enterprise Network and to control admissions to an interconnected set of Enterprise Networks.

### 2.9.1 Controlling a Single Enterprise Network

Figure 2.6 shows the Bandwidth Broker (BB) in the architecture of a single Enterprise Network to support real time multimedia services with absolute QoS guarantees. The

users (denoted by workstations) and servers are connected to the switches of the network. The BB is actually an SC connected to a switch of the Enterprise Network. The users will put a *multimedia session request* to the multimedia server. The multimedia server submits SLAs to the BB for bandwidth between a source switch and a destination switch. The switches, which connect the server and the user to the EN, are the source and the destination of these SLAs, respectively. These SLAs are called *proposed SLAs*. The BB determines whether the SLAs will be admitted or rejected. The accepted SLAs by the BB are called *active SLAs*. The link with the broken line in the Figure 2.6 shows the connection between a user and the BB; TCP/IP connections through the Enterprise Network can be used for these virtual connections. The Bandwidth Broker determines the path and QoS level for the active SLAs. That is, the Utility Model will be applied only once in the BB. The intermediate nodes will not do any path calculations. We assume that an end to end virtual path is created in the packet switching network using MPLS. The BB sets the MPLS labels in the switches of the Enterprise Network for the path with a particular Class of Service (CoS). The CoS in the MPLS table represents the QoS level of the SLAs. Then a *multimedia session* starts transmitting multimedia stream using UDP over MPLS. The CoS entry in the MPLS routing table is used to guarantee the latency by prioritizing a UDP packet for multimedia session with high QoS level or delaying a packet for multimedia session with low QoS level.

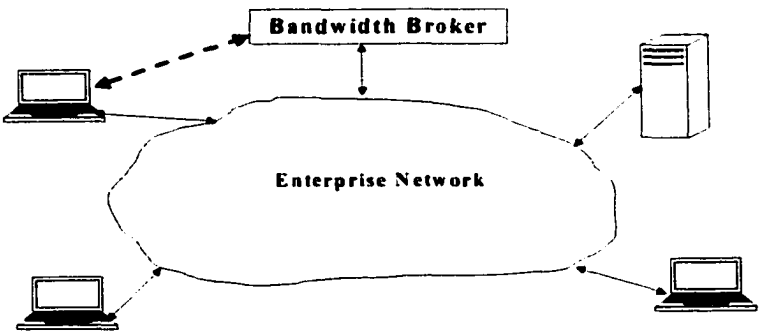


*Figure 2.6 Bandwidth Broker in an Enterprise Network.*

## 2.9.2 Distributed Admission Control for Multiple Enterprise Networks

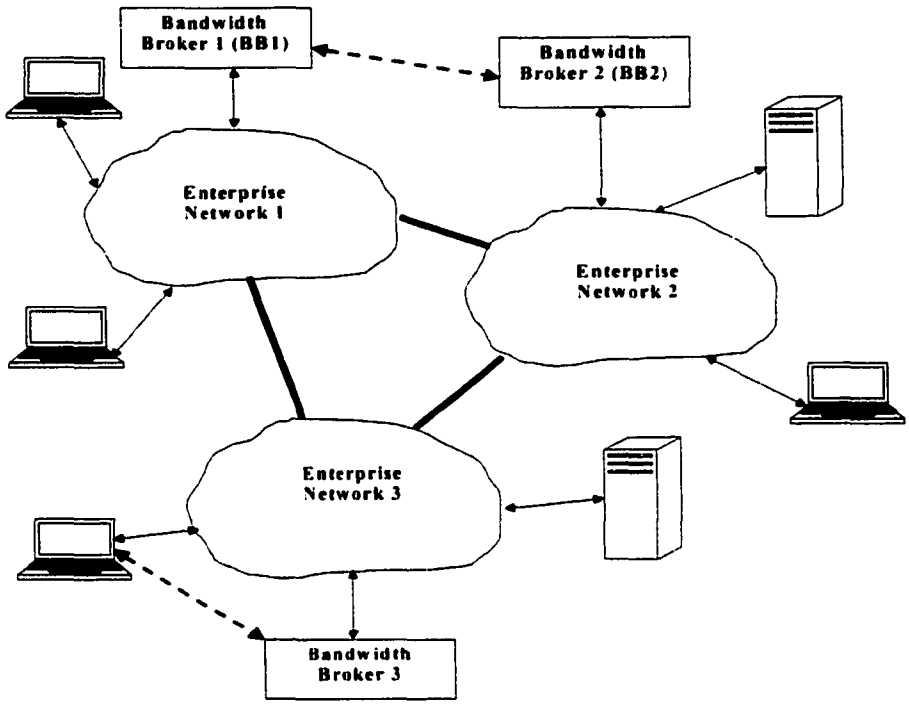


*Figure 2.7 Three Enterprise Networks connected by BGP links.*

Several Enterprise Networks can be connected as shown in Figure 2.7. We assume a BB including an SLA Admission Controller for each Enterprise Network, for reasons of scalability and fault tolerance. The thick lines represent *external links* between pairs of Enterprise Networks. Now, there might be two kinds of SLAs. *Intra-EN SLAs* have the source and destination in the same EN. *Inter-EN SLAs* have the source and destination in two different ENs. For inter-EN SLAs the BBs of the Enterprise Networks must exchange messages among themselves to do admission control. Hence we need a distributed version of SLAOpt for the BBs of the Enterprise Networks, which implies distributed versions of the heuristics to solve the Knapsack Problem. Chapter 8 presents this distributed admission control problem. There are two different kinds of multimedia sessions. A multimedia session, which requires a multimedia stream from a server in an EN to a user in another EN is called an *inter-EN multimedia session*. If the user and

server are in the same EN then the user enjoys an *intra-EN multimedia session*. The BBs set the MPLS labels with a particular CoS in the switches, and multimedia streams can be sent using UDP over MPLS.

## 2.10 Chapter Summary

In this chapter we have reviewed the existing admission control and QoS algorithms in the telephone networks and the Internet. A brief description of the Utility Model has been presented. MMKP heuristics are used to do admission control and QoS adaptation using the Utility Model. Recent works on Knapsack Problem algorithms have also been described. Applications of the Utility Model for admission control in multimedia servers and Enterprise Networks have been introduced. We will discuss the distributed version of these admission control techniques in the rest of the dissertation. Plausible applications of admission controlling to provide multimedia services with guaranteed QoS using the existing data transmission protocols has been presented. The next chapter starts the mathematical models and algorithms, the second part of the dissertation. We present new algorithms for solving the MMKP in Chapter 3. The performance of these algorithms to solve the Utility Model is better than existing algorithms in terms of time requirement and achieved optimality.

# Part II: Mathematical Models and Algorithms

# 3. New Heuristics to Solve Multidimensional Multiple Choice Knapsack Problems

In this chapter we introduce M-HEU (Modified version of HEU by Khan [46]), a new heuristic for solving the MMKP using the concept of aggregate resource. I-HEU, the incremental version of M-HEU is introduced as well. C-HEU, a heuristic for solving the MMKP using the concept of convex hull is developed. Analysis of the algorithms is presented followed by experimental results. The chapter concludes with a comparative discussion among the different heuristics for solving the MMKP.

## 3.1 Modified HEU (M-HEU)

We sort the items in each group of the MMKP in non-decreasing order according to the value associated with each item. Hence, we can say that in each group the bottom items are *lower-valued items* than the top ones. The items at the top can be defined as *higher-valued items* than those in the bottom. Picking a higher-valued or lower-valued item than the currently selected item in a group is called an *upgrade* or a *downgrade* respectively. In the heuristic we need to find an upgrade or downgrade frequently. That is why we need the items of each group sorted according to the associated values of the items. If a particular pick of items (one from each group) does not satisfy the resource constraints, we define that solution as *infeasible.* A *feasible solution* is a solution that satisfies the resource constraints. For any resource $k$, *infeasibility factor* $f_k$ is defined as $C_k / R_k$. The $k$th resource is feasible if the infeasibility factor $f_k \leq 1$, otherwise it is infeasible.

## 3.2 Problems with HEU as an MMKP Solver

We define the selection of the lowest valued item in each group as the *initial solution* of an MMKP. If this solution is not feasible then HEU terminates notifying "No Solution Found". However, there may be a solution using higher-valued items that requires fewer

resources. Thus we should add a new step to find a feasible solution if the initial solution is infeasible.

HEU finds a solution by only upgrading the selected items of each group. There might be some higher-valued items in the MMKP, selection of which make the solution infeasible, but if some of the other groups are downgraded we may obtain a feasible solution. This method, of upgrading followed by downgrading, may increase the *solution value*, the summation of the values of the selected items of the groups of the MMKP.

Thus M-HEU modifies HEU by adding a pre-processing step to find a feasible solution and a post-processing step to improve the total value of the solution with one upgrade followed by one or more downgrades.

### 3.2.1 Description of the Algorithm

The heuristic consists of the three following steps:

- Finding a feasible solution by upgrading the selected items of the groups.
- Improving the solution value by selecting a feasible higher-valued item from the groups subject to resource constraints, i.e., by feasible upgrades.
- Improving the solution value by one infeasible upgrade followed by one or more downgrades.

Following are the variables and the procedures to describe the steps of the algorithm.

$n$: The total number of groups in the MMKP.

$m$: The total number of resources in the MMKP.

$i$: A variable indicating the group number

$l_i$: The number of items in the $i$th group.

$j$: A variable indicating the item number

$r_{ijk}$: The $k$th resource requirement of the $j$th item of the $i$th group.

$v_{ij}$ : The value of the $j$th item of the $i$th group.

$C_k$: The amount of $k$th resource consumed by the selected items of the groups.

$R_k$: The total amount of the $k$th resource in the MMKP.

$f_k$: $C_k / R_k$ , infeasibility factor of the $k$th resource

*current_sol, saved_sol* : The solution vector containing the indexes of the selected items from each group.

$\rho[i]$ : The index of the currently selected item of the $i$th group.

*save_solution*( ): Saves the array of indexes of the selected items of the groups in an array *saved_sol.*

*revive_solution*( ): Changes the selected item of the groups to the saved index of the groups in *saved_sol.*

The change of aggregate resource consumption, $\Delta a_{ij} = \sum_k \left(r_{i\rho[i]k} - r_{ijk}\right) \times C_k$ (if the $j$th item is selected during an upgrade of the $i$th group)

The change of revenue, $\Delta v_{ij} = v_{i\rho[i]} - v_{ij}$ (if the $j$th item is selected during an upgrade or downgrade of the $i$th group)

The scaled change of resource consumption with respect to available resource, $\Delta a'_{ij} = \sum_k \frac{r_{i\rho[i]k} - r_{ijk}}{R_k - C_k}$

The scaled change of resource consumption with respect to over consumed resource, $\Delta a''_{ij} = \sum_k \frac{r_{i\rho[i]k} - r_{ijk}}{C_k - R_k}$

*gain*: the increase of total value for an infeasible upgrade.

*Utility*( ): A function returning the total value of the selected items.

*change_selection*($i$ , $j$): $\rho[i] \leftarrow j$ and returns the increase of total value for this selection. Positive increase denotes upgrade and negative increase denotes downgrade.

*feasible_change*($i$ , $j$): Returns whether *change_selection*( $i$ , $j$ ) satisfies all resource constraints.

*feasible*( ): Returns whether current solution satisfies all resource constraints

*find_feasible_item_by_upgrade*( ) : This procedure returns a higher valued item of a group than the selected item of the group, which does not make any infeasible resource requirement more infeasible, any feasible resource requirement infeasible and gives the highest $\Delta a_{ij}$ .

*find_upgrade_feasible_item*( ): This procedure returns a feasible higher valued item of a group than the selected item of the group, which gives the highest positive $\Delta a_{ij}$ . If no such item is found then it returns the item, which gives the highest $\Delta v_{ij} / \Delta a_{ij}$ .

*find_upgrade_infeasible_item*( ): This procedure returns an infeasible higher valued item of a group than the selected item of the group, which gives the highest $\Delta v_{ij} / \Delta a'_{ij}$ .

*find_downgrade_item*( ): This procedure returns a lower valued item of a group than the selected item of the group, which gives the highest $\Delta a''_{ij} / \Delta v_{ij}$ and keeps *gain* (achieved by infeasible upgrade) still positive.

*Procedure M-HEU ( )*

/*This is the main procedure to solve the MMKP using heuristic.*/

```
1.  current_sol ← lowest-valued items from each group
2.  if find_feasible_solution_by_upgrade( )=false then          // Step 1
3.      No feasible solution found for the MMKP
4.      return;
5.  endif
6.  do
7.      feasible_upgrades( )                                    // Step 2
8.      save_solution()                                         // Step 3 begins
9.      if infeasible_upgrade( )=false
10.         Solution found
11.         return
12.     endif
13.     do
14.         status ← feasible_downgrade( )
15.     while status = INFEASIBLE_DOWN
16.     if status = NO_DOWN then                                // Step 3 could not upgrade
17.         revive_solution( )                                  // No more upgrading
18.         Solution found
19.         return
20.     endif                                                   // Step 3 ends
21. while true                                                  // Step 3 improves solution, so try Step 2 again.
22. end procedure
```

*Begin Procedure find_feasible_solution_by_upgrade( )*

/*This procedure finds a feasible solution of the MMKP. If it fails to find any feasible solution it returns with *false* otherwise it returns *true*.*/

```
1.  k_m ← The resource which has the highest infeasibility factor.
2.  if f_{k_m} ≤ 1 then    // current solution is feasible
3.      return true;
4.  end if
// current solution is not feasible. Following loop looks for a feasible solution
5.  do
6.      (candidate_group, candidate_item) ← find_feasible_item_by_upgrade( )
7.      if candidate_group = null then      //No suitable item available
8.          return false
9.      else
10.         change_selection( candidate_group , candidate_item )    //upgrading
11.     end if
12. while not feasible( )   //still infeasible try again
13. return true
14. end procedure
```

*Begin Procedure feasible_upgrades( )*

/*This procedure finds a feasible solution of the MMKP by upgrading the items of the groups.*/

```
1.  do
2.      (candidate_group, candidate_item) ← find_upgrade_feasible_item ( )
3.      if candidate_group ≠ null then
4.          change_selection( candidate_group , candidate_item ) //upgrading
5.      endif
6.  while candidate_group ≠ null //try again for next feasible upgrade
7.  end procedure
```

*Begin Procedure infeasible_upgrade( )*

/*This procedure finds an infeasible solution of the MMKP by upgrading one item of one group. If no such upgrade is possible then returns false.*/

```
1.  (candidate_group, candidate_item) ← find_upgrade_infeasible_item( )
2.  if candidate_group = null then      //No suitable item available
3.      return false
4.  else
5.      gain ← change_selection( candidate_group , candidate_item ) //Infeasible upgrade. Setting the
        global variable gain.
6.  end if
7.  return true
8.  end procedure
```

*Begin Procedure feasible_downgrade( )*

/*This procedure tries to find a feasible solution of the MMKP by downgrading an item of a group no more than *gain* from an infeasible solution. It returns status of downgrade denoted by the following constants:

*FEASIBLE_DOWN*: A downgrade found that makes the solution feasible keeping *gain* still positive.

*INFEASIBLE_DOWN*: A downgrade found keeping *gain* still positive and the solution still infeasible.

*NO_DOWN*: No downgrade found keeping *gain* positive. */

```
1.  (candidate_group, candidate_item) ← find_downgrade_item( )
2.  if candidate_group ≠ null then    //An item to downgrade is available
3.      gain ← gain + change_selection( candidate_group , candidate_item )
4.      if feasible( ) then
5.          return FEASIBLE_DOWN      //feasible solution found
6.      else
7.          return INFEASIBLE_DOWN    //solution is still infeasible
8.      end if
9.  end if
10. return NO_DOWN    // No downgrade found keeping gain positive
11. end procedure
```

### 3.2.2 Discussion

Our approach in this heuristic is to find the optimal or near-optimal solution by upgrading the feasible solution. The algorithm will work if we start upgrading from any feasible solution. Moser's heuristic [27] finds a feasible solution starting from the highest-valued items. Experimental results show that this approach does not perform well for practical problems like AMS or SLA admission control where the resource consumption by the items follow the *monotone feasibility property* (i.e., the item with higher resource consumption gives higher value of revenue) in almost every case. However, it is easy to find a feasible solution by checking the lowest valued items in the cases where the resource consumption follows the monotone feasibility property. So we start from the lowest valued items and try to find a feasible solution by upgrading the solution if the current one is not feasible.

Our target in *find_feasible_solution*( ) is to find a feasible solution. Finding items with less resource consumptions than the selected items is the main technique here. We do this by selecting an item which releases some of the infeasible resources and does not make any feasible resource infeasible. There might be more than one item like this. To select the best one we use change in aggregate resource consumption, $\Delta a_{ij}$ as the criterion. Evaluation with this parameter penalizes those items that consume more resources than the currently selected items.

In *feasible_upgrade*( ) we first consider items, which provide additional revenue while decreasing aggregate resource consumption. The item with the highest positive $\Delta a_{ij}$ is such a candidate. This is plausible as it leaves resources free for later use. Secondly, we look for an item, which gives the highest revenue per unit of change in aggregate resource consumption. To do this we use $(\Delta v_{ij})/(\Delta a_{ij})$ as the selection criterion. This penalizes the items which are giving small increments of value with higher resource requirements.

This is plausible as we are trying to maximize value, and we would like to leave as much resource as possible free for later use.

In *find_feasible_solution*( ) and *feasible_upgrade*( ) we upgrade the solution value by selecting a higher-valued item than the previously selected item. At a particular time we will find no other upgrade through *feasible_upgrade*( ), due to the resource constraints. Thus we proceed to the next step and try iterative improvement of the solution using one upgrade followed by one or more downgrades in each iteration. Upgrading can be done by more than one infeasible upgrades followed by downgrades. This requires more computation and makes the overall complexity higher. We use the simplest one as our target is to design a heuristic to do on line admission control. This is an attempt to avoid getting stuck on a local maximum. If the heuristic fails to improve the solution in one iteration, it returns the current solution, and terminates. The upgrade is based on the value of $(\Delta v_{ij})/(\Delta a'_{ij})$, the change of total value per unit change of scaled resource consumption $(\Delta a'_{ij})$. In this step we are dealing with a small amount of available resources, as no other feasible upgrade is possible. For this reason, we scale the change of resource with the available resource. The downgrade is based on the value of $(\Delta a''_{ij})/\Delta v_{ij}$, the change of scaled resource consumption per unit change of total value. Here the change of resource consumption $\Delta a''_{ij}$ is scaled by the over-consumed resource.

## 3.2.3 Examples Demonstrating Steps of M-HEU

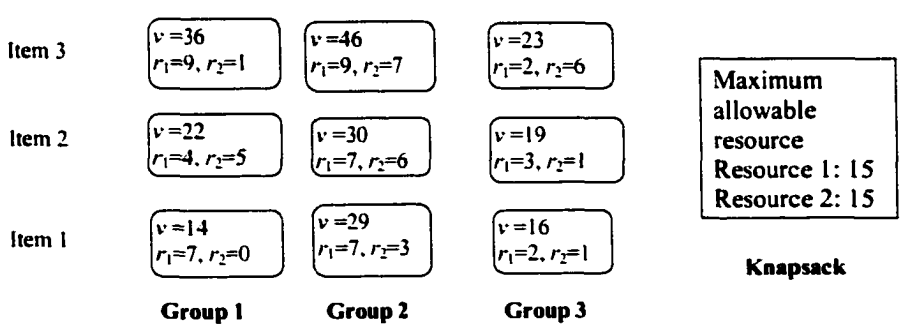


*Figure 3.1 Example of an MMKP*

**Demonstrating Step 1 (when the initial solution is infeasible)**

Figure 3.1 is an MMKP with 3 groups. Each group has 3 items (indexed from 1 to 3 ) sorted according to their associated values. The resource of each item is two dimensional. Item 1 of each group is initially selected. Thus the solution vector can be denoted by (1, 1, 1) where the $i$th element denotes the index of the selected item from the $i$th group. For this MMKP the initial selection is not feasible. The resource usage vector is (16, 4) where 16 and 4 denote the resource consumption by Resources 1 and 2 respectively. So Resource 1 is the most infeasible resource i.e., $k_m$=1.

Now for resource conservation we find $\Delta a_{ij}$ for all possible upgrades. The only upgrade possible is to decrease Resource 1 is Group 1 from item 1 to 2 with $\Delta a_{12}$=28. The other upgrades make the consumption of Resource 1 more infeasible. The computation of $\Delta a_{12}$ can be shown as follows:

$$\Delta a_{12} = \sum_{k=1}^{2} (r_{11k} - r_{12k}) \times C_k$$
$$= (r_{111} - r_{121}) \times C_1 + (r_{112} - r_{122}) \times C_2$$
$$= (7-4) \times 16 + (0-5) \times 4$$
$$= 48 - 20 = 28$$

So the feasible solution will be (2, 1, 1) with total value 67 (= 22+29+16 ) and resource usage (13,9).

**Demonstrating Steps 2 and 3 (when the initial solution is feasible)**

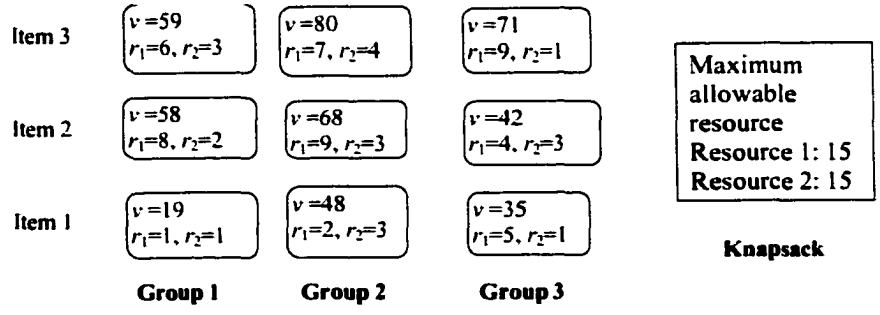


*Figure 3.2 Example of an MMKP*

Figure 3.2 shows an MMKP. Initial selected solution is (1, 1, 1). This is feasible. The resource usage vector is (8, 5). So no iteration is required in Step 1.

*Feasible upgrades* (*Step* 2)

Values of $\Delta a_{ij}$ and $\Delta v_{ij}/\Delta a_{ij}$ for all the feasible upgrades from the currently selected items are as follows.

$\Delta a_{12}$ =-61 $\Delta v_{12}/\Delta a_{12}$ =0.64 $\Delta a_{13}$ = -50 $\Delta v_{13}/\Delta a_{13}$ =0.80

$\Delta a_{22}$ = -56 $\Delta v_{22}/\Delta a_{22}$ =0.36 $\Delta a_{23}$ = -45 $\Delta v_{23}/\Delta a_{23}$ =0.71

$\Delta a_{32}$ = -2 $\Delta v_{32}/\Delta a_{32}$= 3.5 $\Delta a_{33}$ =-32 $\Delta v_{33}/\Delta a_{33}$ = 1.125

Where $\Delta v_{12} = v_{1,p[1]} - v_{12}$

There is no positive $\Delta a_{ij}$. The highest increase of value per unit aggregate resource is $\Delta v_{32}/\Delta a_{32}$. Thus Group 3 is upgraded from Item 1 to Item 2. We get current solution (1, 1, 2) with resource usage (7,7).

Similarly we calculate $\Delta a_{ij}$ and $\Delta v_{ij}/\Delta a_{ij}$ as follows.

$\Delta a_{12}$ =-56 $\Delta v_{12}/\Delta a_{12}$=0.70 $\Delta a_{13}$ = -49 $\Delta v_{13}/\Delta a_{13}$ =0.82

$\Delta a_{22}$ =-49 $\Delta v_{22}/\Delta a_{22}$ = 0.41 $\Delta a_{23}$ = -42 $\Delta v_{23}/\Delta a_{23}$ = 0.76

$\Delta a_{33}$ =-21 $\Delta v_{33}/\Delta a_{33}$ =1.38

We upgrade the solution to (1, 1, 3) and the resource usage is (12,5)

There is no other feasible upgrade possible in Groups 1 or 2 as every upgrade makes the resource usage infeasible.

*Infeasible upgrade followed by downgrades* (*Step* 3)

At first the current solution (1, 1, 3) with its total value 138 (=19 + 48 + 71) is saved.

Now we upgrade ignoring resource constraints. The values of $(\Delta v_{ij})/(\Delta a'_{ij})$ for possible upgrades are:

$(\Delta v_{12})/(\Delta a'_{12})$=16.04 $(\Delta v_{13})/(\Delta a'_{13})$=21.43

$(\Delta v_{22})/(\Delta a'_{22})$=8.57 $(\Delta v_{23})/(\Delta a'_{23})$=18.11

Where, $\Delta a'_{12} = \sum_k \frac{r_{1\rho[1]k} - r_{12k}}{R_k - C_k} = \frac{r_{111} - r_{121}}{R_1 - C_1} + \frac{r_{112} - r_{122}}{R_2 - C_2} = \frac{1-8}{15-12} + \frac{1-2}{15-5} = -2.43$

So Item 3 of Group 1 will be selected. The infeasible solution vector is (3, 1, 3) with resource usage (17,7).

Now we will look for one or more downgrades so that the solution becomes feasible and revenue becomes higher than 138. To do this we find $(\Delta a''_{ij})/\Delta v_{ij}$ where downgrades give total values better than 138. We find two cases:

$(\Delta a''_{31})/\Delta v_{31}$ = 0.06 $(\Delta a''_{32})/\Delta v_{32}$ =0.09.

Where, $\Delta a''_{31} = \sum_k \frac{r_{3\rho[3]k} - r_{31k}}{C_k - R_k} = \frac{r_{331} - r_{311}}{C_1 - R_1} + \frac{r_{332} - r_{311}}{C_2 - R_2} = \frac{9-5}{17-15} + \frac{1-1}{7-15} = 2.0$

So we can downgrade Group 3 to from Item 3 to Item 2. Current solution is (3, 1, 2). The resource usage is (12, 9), which is feasible and the total value is 149.

Now we look for upgrading using Step 2. There is no feasible upgrade from this solution. So Step 3 will be executed again.

Save the current solution (3, 1, 2) with total value 149.

The values of $(\Delta v_{ij})/(\Delta a'_{ij})$ for possible upgrades are:

$(\Delta v_{22})/(\Delta a'_{22})$=8.57 $(\Delta v_{23})/(\Delta a'_{23})$=17.45

$(\Delta v_{33})/(\Delta a'_{33})$=21.75

So we upgrade to (3, 1, 3) with resource usage (17, 7). Every downgrade gives worse solution than the saved solution. So we revive the saved solution (3, 1, 2).

## 3.3 Incremental Solution of the MMKP (I-HEU)

If the number of groups in the MMKP is very large then it is not possible to run M-HEU once every few seconds, as a real time system, (for example a multimedia system with 10,000 sessions) might well require. An *incremental* solution is a necessity to achieve better computation speed. By changing the technique of finding feasible solution we can use M-HEU to solve the MMKP *incrementally*, starting from an already solved MMKP. This approach of incremental solution is similar to reinforcement learning [100]. In I-HEU a feasible solution is searched by selecting a lower valued item at first. If no feasible solution is found by searching lower valued item then we look for higher valued items like M-HEU. Hopefully, we can re-use most of the solution at hand and thus obtain the new solution with less effort. The changed procedures are as follows.

*find_feasible_item_by_downgrade*( ): This procedure returns a lower valued item of a group than the selected item of the group, which does not make any infeasible resource requirement more infeasible and any feasible resource requirement infeasible and gives the highest $\Delta a_{ij}$ .

*Begin Procedure find_feasible_solution_by_downgrade*( )

This procedure finds a feasible solution of the MMKP by downgrading. If it fails to find any feasible solution it returns *false* otherwise it returns *true*.

1. $k_m \leftarrow$ The resource which has the highest infeasibility factor $f_{k_m}$ .
2. *if* $f_{k_m} \leq 1$ *then*
3. *return true;*
4. *endif*
5. *do*
6. (*candidate_group, candidate_item*) ← *find_feasible_item_by_downgrade*( )
7. *if candidate_group = null then* //No suitable item available
8. *return false*
9. *else*
10. *change_selection*( *candidate_group* , *candidate_item* ) // downgrading
11. *end if*
12. *while not feasible*( ) //still infeasible try again
13. *return true*
14. *end procedure*

*Procedure I-HEU* ( )

```
/*This is the main procedure to solve the MMKP incrementally using heuristic.*/
1.  current_sol ← previously selected item from the already solved MMKP and the lowest-valued items
    from each new group                                    // Selecting initial solution
    // Starting of Step 1
2.  if find_feasible_solution_by_downgrade( )=false then   //trying by downgrading selected items
3.      if find_feasible_solution_by_upgrade( ) = false then  //trying by uograding selected items
4.          No feasible solution found for the MMKP
5.          return;
6.      endif
7.  endif                  //end of Step 1
8.  line 6-21 of M-HEU     //Step 2 and 3
9.  end procedure
```

I-HEU will require almost the same amount of time as M-HEU if it is to find the near optimal solution after doing a lot of upgrading and downgrading in the older groups as well as new groups. It will give the best performance when the solution is determined by upgrading or downgrading only the new groups of items. In the average case, it does not require abrupt changes in the already solved groups after finding a feasible solution. Thus we can expect some time advantage with this incremental method over M-HEU. We use I-HEU to do admission control and QoS adaptation in the SLA controller of an Enterprise Network [11]. The steps of I-HEU using upgrades and downgrades are equivalent to path changes, upgrading and downgrading QoS of the SLAs as performed by the SLA controller using N-HEU in an Enterprise Network [40].

## 3.4 Analysis of the Algorithm

We need to do thorough statistical analysis of the steps of the heuristics to determine the average case complexity of the steps of M-HEU and I-HEU heuristics. For the convenience of analysis we present the worst-case analysis of computational complexity in this dissertation. We also show that the steps of these algorithms never regenerate a solution state, so there is no chance of an infinite loop. We do not know how approximate the solution is to the optimal solution for the average case. A lower bound of the total value of the items picked by M-HEU shows the performance of the algorithm in terms of the achieved optimality.

### 3.4.1 Non Regenerative Property

If the two solutions of an MMKP have different total values or different resource consumptions it is obvious that the items picked for these two solutions are not the same. The obvious reasons for this convergence property are as follows:

Step 1 never makes any feasible resource requirement infeasible or infeasible resource requirement more infeasible. Let there be two infeasible solutions $S_i$ and $S_j$ at the $i$th and the $j$th iteration of Step 1. These might be from two consecutive iterations or from two iterations separated by more than one iteration. The resource consumptions by these solutions can be expressed as ($C_{i1}$, $C_{i2}$, ... ... $C_{im}$) and ($C_{j1}$, $C_{j2}$, ... ... $C_{jm}$). Let the total value earned from these two solutions be $V_i$ and $V_j$ and $V_i$=$V_j$. So these solutions might be the same if $C_{ik}$=$C_{jk}$ where $k$=1,2, ... ..., $m$. This situation might happen if and only if the infeasible resources in $S_i$ are the same as the infeasible resources in $S_j$. Now the intermediate iterations must select one infeasible resource to decrease its resource consumption by selecting new item for a group. This decreased resource consumption must not be increased on the way to the $j$th iteration. So the amount of infeasible resource must be different for at least one resource. Thus Step 1 never regenerates a previously generated solution.

Step 2 always upgrades the solution vector with increased total value. Let the solution vectors and revenues in the iterations be $\{(S_1,u_1),(S_2,u_2), \ldots \ldots (S_i,u_i) \ldots \ldots\}$. As we know that $u_1<u_2< \ldots \ldots <u_i< \ldots \ldots$ we can easily conclude that $S_1 \neq S_2 \neq \cdots \neq S_i \neq \cdots$.

Step 3 upgrades one item followed by downgrading one or more items for an increase in total value, thereby excluding the possibility of regenerating a previously determined solution or infinitely looping.

### 3.4.2 Computational Complexity

Sorting of the items in each group using the selection sort algorithm requires $O(l^2)$ computation in the worst case. So we need $O(nl^2)$ computations for sorting $n$ groups in the MMKP.

The computational complexity of M-HEU in Step 1 will be worst when there is no feasible solution or there is a feasible solution located at the highest valued item of each group and at each iteration only one item of one group moves one level up. Initially the selected items of each group are the lowest-valued item of each group. So the total number of upgrades in Group $i$ is $(l_i - 1)$ and the total number of upgrades in Step 1 is $\sum_{i=1}^{n}(l_i - 1)$. For convenience of analysis we assume that all groups contain the same number of items, i.e., $l_i=l$. So the *do* loop (line 5-12) in *find_feasible_solution_by_upgrade*( ) will be executed $(nl-n)$times.

Line 1 in *find_feasible_solution_by_upgrade*( ) requires $m$ divisions and $m$ comparisons to find the feasibility factors and the most infeasible resource.

During iteration $(j+1)$ of the *do* loop (line 5-12) of *find_feasible_solution_by_upgrade*( ) to determine the candidate group and item (line 6) a *for* loop (not shown in the pseudo code) will be executed $(nl-n-j)$times. Each iteration of the loop requires the following computations:

- $m$ multiplication, $m$ subtraction and $m$ addition operations to determine $\Delta a_{ij}$.
- $m$ addition, $m$ subtraction and $m$ resource comparison operations to find the feasibility of selecting an item.
- 1 comparison to determine the highest $\Delta a_{ij}$.

After selecting an item we need $m$ addition and $m$ subtraction operations to find the new $\bar{C}$.

The total number of floating point operations in Step 1 is

$$\sum_{j=0}^{(l-1)n-1}\{(nl-n-j)\times(6m+1)+2m\}=\left\{\frac{n^2(l-1)^2}{2}+\frac{n(l-1)}{2}\right\}\times(6m+1)+2m\times n(l-1)$$

So, complexity in Step 1 is $O\left(mn^2(l-1)^2\right)$.

Step 2 requires the highest number of computations when there is a feasible solution using the lowest item from each group (initial solution). The *do* loop (line 1-6) in *feasible_upgrade*( ) upgrades one level of one group in every execution and upgrading continues until it reaches the highest valued item of each group. The complexity analysis is almost the same as Step 1. There are $(nl-n)$ possible upgrades in Step 2.

During iteration $(j + 1)$ of the *do* loop (line 1-6) of *feasible_upgrades*( ) to determine the candidate group and item (line 2) a *for* loop (not shown in the pseudo code) will be executed $(nl-n-j)$times. Each iteration of the loop requires the following computations:

- $m$ multiplication, $m$ subtraction and $m$ addition operations to determine $\Delta a_{ij}$.
- 1 division, 1 subtraction operations to determine $\Delta v_{ij}/\Delta a_{ij}$
- $m$ addition, $m$ subtraction and $m$ resource comparison operations to find the feasibility of selecting an item.
- 2 comparisons to determine the highest $\Delta v_{ij}/\Delta a_{ij}$ or the highest $\Delta a_{ij}$.

After selecting an item we need $m$ addition and $m$ subtraction operations to find the new $\vec{C}$.

The total number of floating point operations in Step 2 is

$$\sum_{j=0}^{(l-1)n-1}\{(nl-n-j)\times(6m+4)+2m\}=\left\{\frac{n^2(l-1)^2}{2}+\frac{n(l-1)}{2}\right\}\times(6m+4)+2mn(l-1)$$

So, Complexity in Step 2, $O\left(mn^2(l-1)^2\right)$

The available resource is very small in Step 3 compared to Step 2. We expect that this step requires less iteration than the previous steps. This step is analogous to the hill climbing approach in a plateau, for classical AI problems [14]. We present the worst-case complexity required to escape from a local maxima using one upgrade followed by one or more downgrades (line 9-20 in M-HEU).

In the procedure *infeasible_upgrade*( ) of Step 3 to determine the candidate group and item (line 1) a *for* loop (not shown in the pseudo code) can be executed at most $n(l-1)$ times. Each iteration requires:

- $(m+1)$ division, $(2m+1)$ subtraction and $m$ addition operations to determine $\Delta v_{ij}/\Delta a'_{ij}$
- 1 comparison to find the highest $\Delta v_{ij}/\Delta a'_{ij}$.

So, the number of floating-point operations in the procedure *infeasible_upgrade*( ) is $n(l-1)\times(4m+3)$

The *do* loop (line 13-15) in *M_HEU*( ) can be executed $(n-1)(l-1)$ times because the already upgraded group will not be downgraded. To determine the candidate group and item for downgrade (line 1) in *feasible_downgrade*( ) a *for* loop (not shown in the pseudo code) can be executed at most $(n-1)(l-1)$ times. Each iteration requires:

- $(m+1)$ division, $(2m+1)$ subtraction and $m$ addition operations to determine $\Delta a''_{ij}/\Delta v_{ij}$ .
- 1 addition and 1 subtraction operations for checking the total value
- 1 comparison to find the highest value of $\Delta a''_{ij}/\Delta v_{ij}$ .
- $m$ addition, $m$ subtraction and $m$ comparison operations for checking the feasibility

So, the number of floating-point operations to find all the downgrades in the worst case situation is $(n-1)(l-1)\{(n-1)(l-1)(7m+5)\}$.

Thus the worst case complexity to escape from local maxima is

$n(l-1)\times(4m+3)+(n-1)(l-1)\{(n-1)(l-1)(7m+5)\}$, which is $O(m(n-1)^2(l-1)^2)$

The complexity of M-HEU will be the worst when a sequence of one upgrade followed by downgrades occurs. In the scenario that Steps 1and 2 are not be able to upgrade any group, then Step 3 will upgrade one group, and transfers the control to Step 2. Step 2 will fail again to upgrade any solution. This is a very unlikely situation. We do not present the complexity analysis of M-HEU using only Step 3 in this dissertation. We consider Step 3 as an escaping technique for escaping from local maxima, as used in different local search heuristics. So, an analysis of the combined complexity of Step 1 and Step 2 might be indicative of the actual computational behaviour of the algorithm.

All the steps of M-HEU require more computations than the computations required by sorting the items. So $O(nl^2)$ complexity of sorting items will not change the overall complexity of the heuristic. Step 1 finds the feasible solution and also upgrades the solution. Step 2 upgrades this feasible solution. The orders of the computation complexity in each iteration of Step 1 and Step 2 are the same. Therefore, the same kind of computation is shared by Step 1 and Step 2. Again, Step 2 starts right after Step 1 and with the feasible solution calculated from Step 1. Thus the combined worst-case complexity of Step 1 and Step 2 is $O(mn^2(l-1)^2)$.

I-HEU requires extra computation by *find_feasible_solution_by_downgrade*( ) in line 2. The complexity will be the worst when a new group joins an MMKP where the highest valued items are initially selected in each group and each iteration downgrades one item of one group down towards a final selection of all the lowest valued items in the groups. Let there be $n$ groups including a new group in the MMKP. So the total number of downgrades in the worst case is $(n-1)(l-1)$. The computations in each iteration are similar to *find_feasible_solution_by_upgrade*( ). Thus the total number of floating point operation in *find_feasible_solution_by_downgrade*( ) is $\sum_{j=0}^{(n-1)(l-1)-1}[\{(n-1)(l-1)-j\}\times(6m+1)+2m]$

which is $O(mn^2(l-1)^2)$. So the orders of computational complexities in the worst case for all the steps of I-HEU are the same as M-HEU.

### 3.4.3 Lower Bound of Total Value by M-HEU

Here we present a lower bound of the total value of the items picked by the heuristic M-HEU for an MMKP with single resource, i.e., MCKP. The achieved total value will not be lower than this bound if we run M-HEU to solve an MCKP.

Consider a problem where all the lowest-valued items have zero resource with zero value. We assume that all higher-valued items have resources greater than zero and exhibit the monotone feasibility property. Let the initial solution be item 0 with zero value in each group. This is the optimal solution when total resource constraint is zero. Now we execute Step 2 of the heuristic. We find that Step 2 always selects the optimal solution in each iteration for the resource it consumes. The following shows the inductive proof of this statement.

Finding the initial optimal selection is straightforward. When all the selected items are yielding zero total value and consuming no resource the item with the highest value per unit resource is the optimal selection if there is available resource.

Let the current selections of the items give value $V_i$ with resource $R_i$. This selection gives the optimal value for the used amount of resource. We select an item which gives $p$ extra value with $r$ extra resource in Step 2. This is an upgrade expressed by the pair ($p$ , $r$). This is the item which gives the highest value increase per unit increase of resource (for the single resource problem, increase of aggregate resource is equivalent to the increase of resource for the newly selected item) among all the possible upgrades. We have to prove that total value ($V_i + p$) is optimal if the resource constraint is ($R_i + r$).

Let there be other upgrades that yield a higher total value for the resource constraint ($R_i$ + $r$). This might be obtained by the selection of several other smaller upgrades. Let the upgrades be ($p_0$ , $r_0$), ($p_1$ , $r_1$), ($p_2$ , $r_2$)..... ($p_n$ , $r_n$). These upgrades may be from the

same group as upgrade ($p$, $r$) or from other groups. Assume these upgrades are selected from the smaller upgrades inside the same group as upgrade ($p$, $r$). Now the total increase of resource consumption for these upgrades is $\sum r_i = r$. For this increase of resource the increase of total value is definitely $\sum p_i = p$. Thus the alternative solution will yield the same total value. Consider the case where smaller upgrades are not from the same group as the group of upgrade ($p$, $r$). We can prove that upgrade ($p$, $r$) is the optimal one as follows:

$\frac{p_i}{r_i} < \frac{p}{r}$ [upgrade ($p$, $r$) provides the highest value increase with per unit increase of resource]

$$\Rightarrow p_i < \frac{pr_i}{r} \Rightarrow \sum p_i < \sum \frac{pr_i}{r} \Rightarrow \frac{\sum p_i}{\sum r_i} < \frac{\sum \frac{p}{r} r_i}{\sum r_i} \Rightarrow \frac{\sum p_i}{\sum r_i} < \frac{p}{r} \Rightarrow \sum p_i < p$$ [as $r = \sum r_i$ and assuming $r > 0$]

Now consider an upgrade ($P$ , $R$) larger than the upgrade ($p$ , $r$), i.e., $P > p$. We have to prove that selection of ($p$ , $r$) will give better value than ($V_i + P$) if the selection leads to a resource usage of ($R_i + R$). Let the alternate upgrades be ($p$, $r$) and ($p'$, $R$-$r$). These two upgrades result in a resource consumption of ($R_i + R$). According to the algorithm, the increase of value per unit change of resource consumption is higher for the item which is selected for upgrade than the items which are not yet selected.

Therefore, $\frac{p'}{R-r} > \frac{P}{R}$. Again $\frac{p}{r} > \frac{P}{R}$. From these two inequalities we find the following relation among $P$, $p$ and $p'$.

$$p' > \frac{P}{R}(R-r) \Rightarrow p' + p > P - \frac{P}{R} \times r + p \Rightarrow p' + p > P - \frac{p}{r} \times r + p \Rightarrow p' + p > P$$

Thus the upgrade ($p$ , $r$) is the optimal selection and Step 2 of our algorithm leads to the optimal selection when resources are available.

The heuristic continues to pick the optimal items while we have resources available for any upgrade. When the optimal upgrade does not meet resource constraints it tries to select an upgrade which does not violate these constraints. The worst case is a situation when there is no upgrade possible with the limited amount of remaining resources. We can find an upper bound of the difference between the optimal total value and the total value of the selected items before reaching this condition.

Let the difference between total resource constraint and consumed resource in the worst case be $r_d$ and the difference between optimal total value and current total value be $p_d$. $r_i > r_d$, where the infeasible upgrades in this situation are denoted by $(p_i,r_i)$.

The upgrade $(p_{max},r_{max})$ gives the highest increase in value per unit resource among all the possible upgrades in the MCKP. Obviously $p_{max} > p_d$. Thus the lower bound of the algorithm is $V_{lower}=(U_{optimal} - p_{max})$, where $p_{max}$ is the change of total value for any upgrade, i.e., maximum difference of values of any two items of any group. This is the same as lower bound of the Lee's heuristic for solving the MMKP using the convex hull approach [7]. We expect approximately the same lower bound for the MMKP with multidimensional resources.

Step 3 of the algorithm can improve the solution quality by upgrading followed by one or more downgrades. These steps do not increase the lower bound of the total value but increases the solution value in the average cases.

The algorithm will definitely show better results for a larger MMKP, i.e., if the number of groups and total amount of resources are increased. $U_{optimal}$ increases with a greater number of groups as we get more selected items. Since $p_{max}$ remains the same and it depends on the characteristics of the distribution of the items, so the ratio $V_{lower}/U_{optimal}$ will increase.

## 3.5 Solving the MMKP by Constructing Convex Hulls

The *convex hull* of the points in a two dimensional space is defined by the smallest convex polygon containing those points. There are two different groups of line segments in the convex hull connecting the bottom most and the topmost points. Each of these groups of line segments are called *frontier* of the convex hull. The QoS management problem of the MRMD (Multiple Resource Multiple Dimension) system by Lee [7] can be easily mapped to an MMKP. So the solution algorithm proposed by Lee can be applied to solve the MMKP. The MRMD system has some restrictions between required resources for a QoS level and associated utility. The QoS levels follow a special monotone feasibility order, i.e., a QoS level with higher utility must require higher resource requirements. So this algorithm is not applicable for those MMKPs where some higher-valued items require less resource consumptions than lower valued items. The QoS controller of the MRMD system transforms each multidimensional resource to a single dimension by multiplying a penalty vector as described in Section 3.5.1. Now the quality of service level for each session represents a point in the two dimensional space. The offered bid price represents the $y$ co-ordinate and the transformed resource represents $x$ co-ordinate. A convex hull is constructed for each session with the points represented by the QoS levels.

*Gradient* is a vector that always points in the direction of maximum change, with a magnitude equal to the slope of the tangent to the curve at the point. Admission control and QoS adaptation of a session is done based on the gradient of the segments of the *efficient convex hull frontier* [58]. The frontier, which earns more revenue in terms of resource usage is considered as efficient convex hull frontier. For the MRMD system the gradients of these segments never become negative because of the monotone feasibility property. We use different sorting criterion to make it applicable for the MMKP. We sort the segments according to the angle between the positive $x$-axis and the segment. Thus the negative gradients are mapped to the angles higher than 180 degrees. These segments

are always in the beginning of the list and these are selected for upgrading in the beginning. This makes the selection criterion reasonable because the items with higher utility and lower resource requirements are always preferable for utility maximization. Figure 3.3 shows two convex hulls. The dotted lines show the efficient convex hull frontier. The efficient convex hull frontier of Group 1 has some segments with negative gradients, which give more utility with less resource. On the other hand the efficient convex hull frontier for Group 2 follows the monotone feasibility property. Every segment on this convex hull frontier has a positive and non-increasing gradient. The use of angles instead of gradient removes all the problems of picking items in the generalized MMKP. The next subsection presents the algorithm for solving the MMKP using the convex hulls.

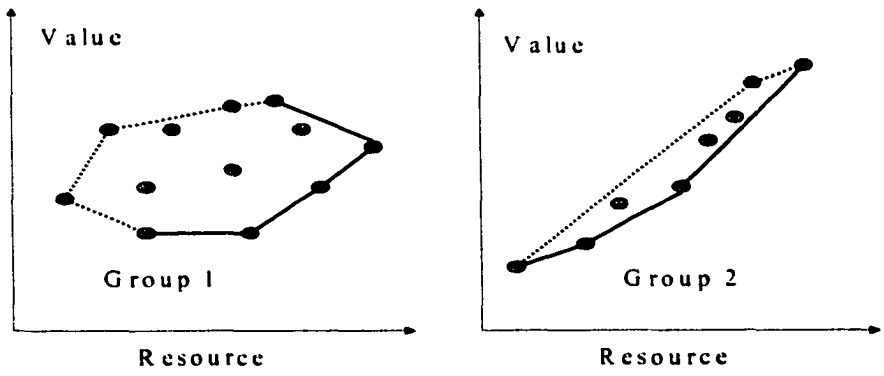


*Figure 3.3 Convex hulls of the items of two groups.*

## 3.5.1 Heuristic Algorithm for Solving the MMKP using Convex Hull Approach (C-HEU)

*current_sol, saved_sol* : The solution vector containing the indexes of the selected items

*penalty*: The transformation vector to transform from multidimensional resource to single dimension.

*Utility*( ): A function returning the total value earned from the selected items.

*initial_penalty*( ) : returns the initial penalty vector.

*adjust_penalty*( ) : returns the penalty vector based on the current resource usage.

*list_of_frontier_segments, ch_frontier:* Lists of segments. A segment is a vector with two items representing a straight line.

$p_1$, $p_2$: Items representing points in two dimensional space. The associated value represents the $y$ co-ordinate and transformed single dimensional resource represents $x$ co-ordinate.

*Begin Procedure C_HEU ( )*

1. *current_sol* ← The lowest item from each group;
2. *penalty* ← *initial_penalty*();
3. *for repeat* ← 1 to 3 *do* //only three iterations for finding solution
4. *saved_sol* ← *current_sol* //saving the current solution
5. $u$ ← *Utility*(*current_sol*) //saving utility
6. *for* each group in the MMKP *do*
7. Transform each resource consumption vector of each item using vector *penalty*.
8. *ch_frontier* ← efficient convex hull frontier of the items of the group
9. *list_of_frontier_segments* ← *list_of_frontier_segments* + *ch_frontier*
10. *end for*
11. Sort the segments of *list_of_frontier_segments* in descending order according to the angle of each segment with the $x$ axis.
12. *for* each segment in the *list_of_frontier_segments do*
13. $p_1$,$p_2$ ←The items associated with the segment.
14. *current_sol* ← The new solution after selecting $p_1$ at first and then by selecting $p_2$. Resource constraint must be followed during the upgrades.
15. *end for.*
16. *if Utlity*()<$u$ *then* // New solution is inferior than the saved one
17. *saved_sol* is the final solution.
18. *endif*
19. *penalty* ← *adjust_penalty*( ) //adjust penalty for the next iteration
20. *end for*
21. *current_sol* is the final solution.
22. *end Procedure*

Please see [7] for the detailed description of function *initial_penalty*() and *adjust_penalty*().

## 3.5.2 Lower Bound and Computational Complexity

The worst-case complexity of finding the solution of an MMKP with $n$ groups and $l$ items in each group is $O(nl \log nl + nlm)$, where $m$ is the dimension of resource consumption. The

lower bound of this heuristic is $(U_{optimal} - \theta)$, where $\theta$ is the maximum difference between any two items of a group. Please refer to complexity analysis of SRMD and MRMD systems in [7] for detailed analysis.

## 3.6 Experimental Results

In order to study the run-time performance of M-HEU, I-HEU and C-HEU we implemented these algorithms along with three other algorithms:

(1) BBLP, an optimal algorithm based on a branch-and-bound search using linear programming

(2) *Moser's heuristic*, based on Lagrange relaxation.

(3) *Greedy approach*, based on a linear search of all the items of each group. It picks the highest-valued feasible item from each group. In this dissertation we call this heuristic G-HEU.

BBLP is infeasible in practical applications for larger data sets. Each iteration of BBLP estimates the total value using the linear programming approach. We run the first iteration to determine $V_{opt\_est}$, an *estimate of the optimal solution*. This estimate must be higher than the optimal solution. Only one iteration using linear programming determines this estimate whereas indefinite number of iterations determines the optimal solution. The subsequent iterations generate estimates closer to the optimal total solution value. We implemented all the algorithms using the Visual C++ programming language. For simplicity of implementation, we assumed that each group has the same number of items, i.e., $l_1 = l_2 = \cdots\cdots = l_n$. We used the Simplex Method code from [51] for linear programming. We ran the algorithms on a Pentium III IBM Think Pad 700 MHz with 192 MB of RAM running Windows 2000. We have performed experiments on an extensive set of problem sets. We used randomly generated and correlated MMKP instances for our tests. For each set of parameters *n*, *l* and *m*, we generated 10 MMKP instances which are correlated with resource consumptions and 10 random MMKP instances which are not

correlated with resource consumptions. Please see [45] for the files containing these MMKP instances. We ran BBLP, Moser's heuristic, M-HEU and C-HEU on all 10 instances, and tabulated the averages of solution-value and execution time. To compare I-HEU and M-HEU we have applied both algorithms to an already solved problem set after dropping some existing groups and adding some new groups.

### 3.6.1 Test Pattern Generation

The data sets for testing the performance of different heuristics were initialized as follows:

$R_c$ = Maximum amount of a resource consumption by an item

$P_c$ = Maximum cost per unit resource

$R_i$ = Total amount of the $i$th resource = $n \times R_c \times 0.5$, Here we assume $R_c \times 0.5$ amount resource for each session.

$P_i$ = cost of the $i$th resource= $Random(P_c)$ = A uniform discrete random number from 0 to $(P_c - 1)$.

$r_{ijk}$ = The $k$th resource of the $j$th item of the $i$th group = $Random(R_c)$. As the total resource for each session is $R_c \times 0.5$, we can say approximately 50% solutions are infeasible because there is a chance that items with resource consumption between $R_c \times 0.5$ to $R_c$ may be infeasible.

$v_{ij}$ = Value of the $j$th item of the $i$th group $= Random\left(m \times \frac{R_c}{2} \times \frac{P_c}{2}\right) \times \frac{j+1}{l}$, when the item values are not correlated with the resource requirement.

$v_{ij} = \sum r_{ijk} \times P_k + Random\left(m \times 3 \times \frac{R_c}{10} \times \frac{P_c}{10}\right)$, when there is a linear correlation between the resource consumption and item values

To measure the performance of I-HEU we initialize an MMKP with a particular size. Let there be $n$ groups. This MMKP is solved by M-HEU. Now $b$ groups are dropped out from the MMKP from the top of the group list and a batch of $b$ new groups are added. The values and resources of the new items of the new groups can be initialized using the above-mentioned functions. We can solve the new MMKP by two ways:

- Applying M-HEU directly to the initial problem set.
- Applying I-HEU to the previously determined solution.

The results from these two methods have been analyzed.

### 3.6.2 Test Results

Table 3.1 and Table 3.2 show the performance of the algorithms for correlated and uncorrelated (random) data sets respectively with the number of groups ranging from 5 to 30. Here we find the optimality of G-HEU, C-HEU, Moser's heuristic and M-HEU by normalizing the total value of the items picked by the heuristics with the optimal value (as found by BBLP). Table 3.3 shows the time requirements in milliseconds for the same problem set. The computation times for the heuristics are not significant (less than a millisecond) with respect to BBLP. We used the *ftime*( ) function of Visual C++ to measure the elapsed time used by the algorithms. This function gives the current system time with millisecond accuracy.

The graphs of Figure 3.4 to Figure 3.6 compares the optimality achieved by the different heuristics with the increase in number of groups, number of items in each group and number of resource dimensions. For the experiments done with larger data sets the computation times for the optimal solution by BBLP are too large for practical interest, as it takes too long (days or years) on the average to do the computation. So the solution values of the heuristics have been normalized by the estimated optimum total value. The graphs of Figure 3.7 to Figure 3.9 compares the time required by different heuristics. We have not plotted the time required by G-HEU as it takes very insignificant time for this

range of data sets. The worst case has been plotted following a quadratic function derived from the worst complexity analysis. Figure 3.10 compares the performance of M-HEU and I-HEU in terms of the total values achieved and the time required to solve the problem. All the plotted data in the above mentioned graphs are the average of 10 problem sets. To verify the consistency of the results we present graphs showing $V_{opt\_est}$ and total values from different heuristics for 15 correlated and uncorrelated data sets in Figure 3.11 and Figure 3.12.

*Table 3.1 Performance comparison of BBLP, HEU, Moser, C-HEU and G-HEU for correlated data sets. G, C, L, M and B indicate G-HEU, C-HEU, Moser's heuristic (Lagrange's polynomial approach), M-HEU and BBLP respectively. $V_X$ indicates the total value of the items picked by heuristic X and $X_s$ indicates the number of times we got a solution for the MMKP by using heuristic X.*

| $n$ | $l$ | $m$ | $\frac{V_G}{V_B}\times 100$ | $\frac{V_C}{V_B}\times 100$ | $\frac{V_L}{V_B}\times 100$ | $\frac{V_M}{V_B}\times 100$ | $V_B$ | $G_s$ | $C_s$ | $L_s$ | $M_s$ | $B_s$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 5 | 5 | 90.40 | 93.69 | 92.60 | 96.94 | 551.80 | 6 | 10 | 8 | 10 | 10 |
| 5 | 7 | 5 | 90.79 | 88.77 | 90.40 | 95.96 | 609.70 | 3 | 10 | 8 | 10 | 10 |
| 5 | 9 | 5 | 91.25 | 89.21 | 92.96 | 94.07 | 586.40 | 8 | 10 | 9 | 10 | 10 |
| 10 | 5 | 5 | 91.65 | 89.43 | 93.45 | 95.98 | 1126.90 | 6 | 10 | 8 | 10 | 10 |
| 10 | 7 | 5 | 92.13 | 89.57 | 92.95 | 94.93 | 1339.30 | 8 | 10 | 10 | 10 | 10 |
| 10 | 9 | 5 | 91.66 | 91.37 | 93.38 | 94.93 | 1374.20 | 8 | 10 | 10 | 10 | 10 |
| 15 | 5 | 5 | 93.92 | 91.80 | 93.70 | 95.06 | 2003.10 | 8 | 10 | 10 | 10 | 10 |
| 15 | 7 | 5 | 93.37 | 88.22 | 94.35 | 95.29 | 1751.10 | 6 | 10 | 10 | 10 | 10 |
| 15 | 9 | 5 | 92.35 | 90.02 | 94.57 | 95.97 | 2074.80 | 10 | 10 | 10 | 10 | 10 |
| 20 | 5 | 5 | 94.12 | 90.67 | 95.67 | 96.82 | 2864.90 | 9 | 10 | 10 | 10 | 10 |
| 20 | 7 | 5 | 94.26 | 91.45 | 95.32 | 97.25 | 2893.80 | 9 | 10 | 10 | 10 | 10 |
| 20 | 9 | 5 | 92.80 | 90.21 | 95.11 | 96.45 | 2305.20 | 9 | 10 | 10 | 10 | 10 |
| 25 | 5 | 5 | 94.45 | 92.81 | 95.13 | 96.92 | 3311.90 | 8 | 10 | 10 | 10 | 10 |
| 25 | 7 | 5 | 94.64 | 90.14 | 95.01 | 96.16 | 3286.30 | 9 | 10 | 10 | 10 | 10 |
| 25 | 9 | 5 | 92.85 | 90.01 | 94.53 | 96.50 | 2685.70 | 10 | 10 | 10 | 10 | 10 |
| 30 | 5 | 5 | 93.12 | 91.98 | 94.95 | 96.81 | 3631.50 | 9 | 10 | 10 | 10 | 10 |
| 30 | 7 | 5 | 93.55 | 90.38 | 95.02 | 95.70 | 3498.20 | 10 | 10 | 10 | 10 | 10 |
| 30 | 9 | 5 | 94.34 | 90.09 | 94.74 | 96.42 | 3946.60 | 10 | 10 | 10 | 10 | 10 |

*Table 3.2 Performance comparison of BBLP, HEU, Moser, C-HEU and G-HEU for uncorrelated data sets. The symbols carry the same meaning as Table 3.1.*

| $n$ | $l$ | $m$ | $\frac{V_{G}}{V_B}\times 100$ | $\frac{V_{C}}{V_B}\times 100$ | $\frac{V_{L}}{V_B}\times 100$ | $\frac{V_{M}}{V_B}\times 100$ | $V_B$ | $G_s$ | $C_s$ | $L_s$ | $M_s$ | $B_s$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 5 | 5 | 74.45 | 92.44 | 98.34 | 98.53 | 274.64 | 10 | 10 | 9 | 10 | 10 |
| 5 | 7 | 5 | 68.17 | 91.79 | 94.06 | 96.58 | 296.54 | 10 | 10 | 8 | 10 | 10 |
| 5 | 9 | 5 | 72.51 | 93.56 | 93.48 | 98.24 | 324.94 | 10 | 10 | 9 | 10 | 10 |
| 10 | 5 | 5 | 76.21 | 96.65 | 95.69 | 99.94 | 587.56 | 10 | 10 | 10 | 10 | 10 |
| 10 | 7 | 5 | 78.76 | 94.19 | 92.84 | 98.92 | 633.83 | 10 | 10 | 10 | 10 | 10 |
| 10 | 9 | 5 | 78.99 | 90.31 | 94.26 | 97.32 | 617.48 | 10 | 10 | 10 | 10 | 10 |
| 15 | 5 | 5 | 77.94 | 94.86 | 92.93 | 99.06 | 903.52 | 10 | 10 | 10 | 10 | 10 |
| 15 | 7 | 5 | 86.31 | 93.60 | 94.31 | 99.15 | 988.21 | 10 | 10 | 10 | 10 | 10 |
| 15 | 9 | 5 | 82.05 | 96.44 | 92.83 | 98.91 | 1001.29 | 10 | 10 | 10 | 10 | 10 |
| 20 | 5 | 5 | 79.80 | 96.87 | 91.76 | 99.52 | 1274.62 | 10 | 10 | 10 | 10 | 10 |
| 20 | 7 | 5 | 84.85 | 97.96 | 96.15 | 99.57 | 1393.10 | 10 | 10 | 10 | 10 | 10 |
| 20 | 9 | 5 | 86.76 | 96.42 | 94.62 | 98.99 | 1374.04 | 10 | 10 | 10 | 10 | 10 |
| 25 | 5 | 5 | 84.07 | 97.19 | 94.28 | 99.03 | 1455.58 | 10 | 10 | 10 | 10 | 10 |
| 25 | 7 | 5 | 80.43 | 96.55 | 92.92 | 99.13 | 1579.44 | 10 | 10 | 10 | 10 | 10 |
| 25 | 9 | 5 | 86.93 | 96.74 | 91.49 | 99.29 | 1713.74 | 10 | 10 | 10 | 10 | 10 |
| 30 | 5 | 5 | 88.78 | 98.45 | 97.51 | 99.86 | 1892.68 | 10 | 10 | 10 | 10 | 10 |
| 30 | 7 | 5 | 86.77 | 97.66 | 93.87 | 99.80 | 2093.57 | 10 | 10 | 10 | 10 | 10 |
| 30 | 9 | 5 | 87.95 | 96.82 | 94.75 | 99.00 | 2153.96 | 10 | 10 | 10 | 10 | 10 |

*Table 3.3 Time requirements by BBLP for solving the MMKP with correlated and uncorrelated data sets.*

| $n$ | $l$ | $m$ | $t_{BBLP}$ in ms (for correlated data sets) | $t_{BBLP}$ in ms (for uncorrelated data sets) |
|---|---|---|---|---|
| 5 | 5 | 5 | 11.00 | 2.00 |
| 5 | 7 | 5 | 16.00 | 5.00 |
| 5 | 9 | 5 | 22.00 | 6.00 |
| 10 | 5 | 5 | 242.00 | 16.00 |
| 10 | 7 | 5 | 1087.00 | 33.00 |
| 10 | 9 | 5 | 18713.00 | 61.00 |
| 15 | 5 | 5 | 13726.00 | 44.00 |
| 15 | 7 | 5 | 35887.00 | 126.00 |
| 15 | 9 | 5 | 126147.00 | 147.00 |
| 20 | 5 | 5 | 78115.00 | 197.00 |
| 20 | 7 | 5 | 980619.00 | 275.00 |
| 20 | 9 | 5 | 471723.00 | 835.00 |
| 25 | 5 | 5 | 353385.00 | 276.00 |
| 25 | 7 | 5 | 452098.00 | 955.00 |
| 25 | 9 | 5 | 141629.00 | 2009.00 |
| 30 | 5 | 5 | 84722.00 | 593.00 |
| 30 | 7 | 5 | 725852.00 | 1276.00 |
| 30 | 9 | 5 | 2958402.00 | 3240.00 |

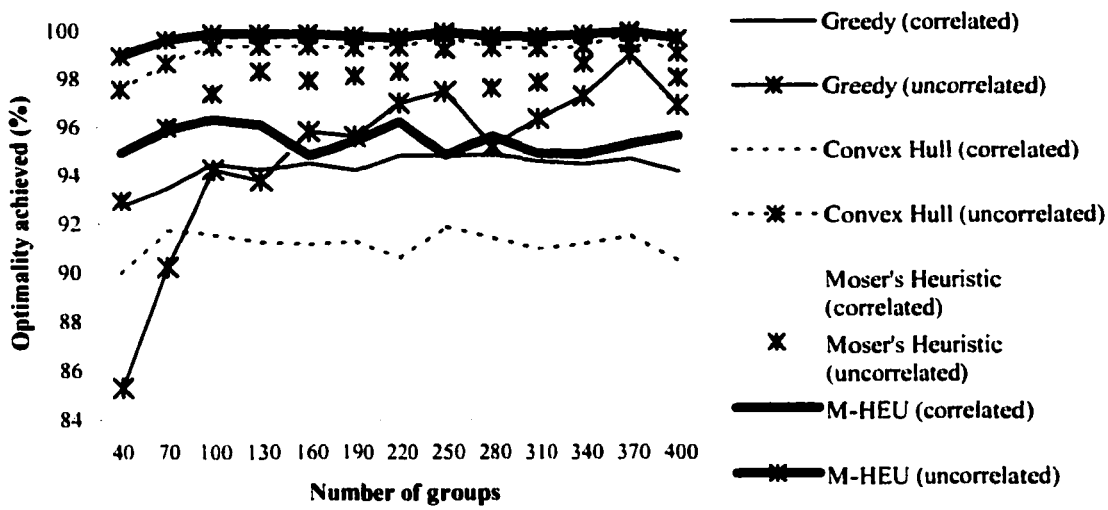


*Figure 3.4 Performance of different heuristics normalized with the estimated optimal total value for the MMKP data sets with l=10 and m=10.*

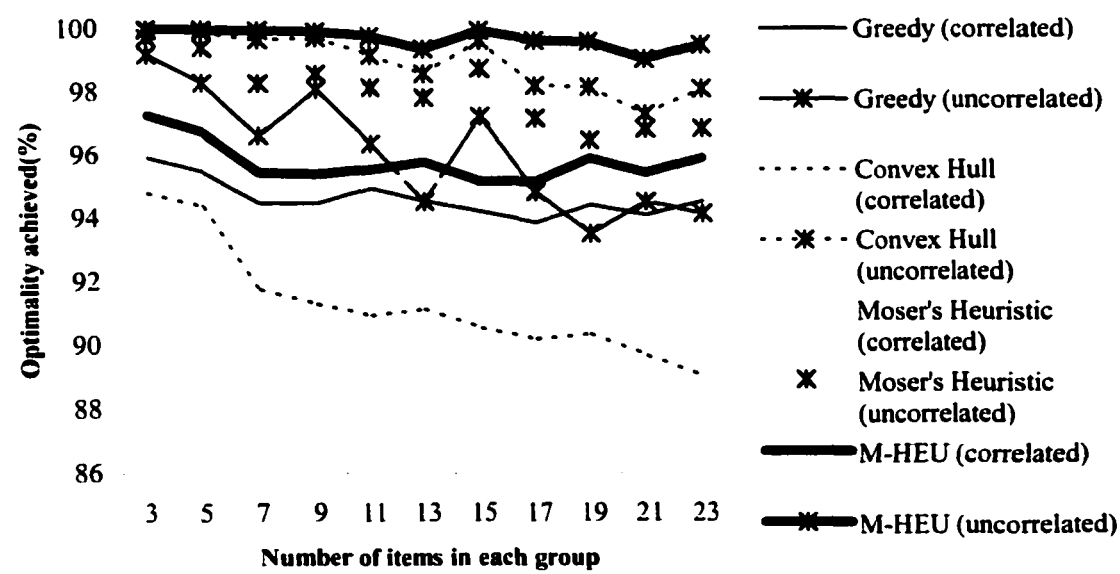


*Figure 3.5 Performance of different heuristics normalized with the estimated optimal total value for the MMKP data sets with n=200 and m=10.*

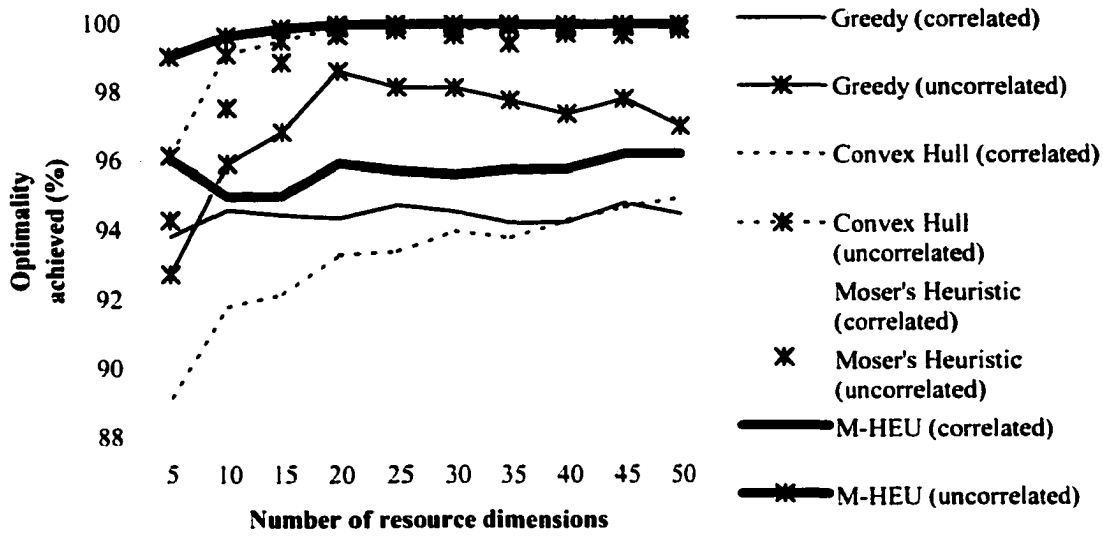


*Figure 3.6 Performance of different heuristics normalized with the estimated optimal total value for the MMKP data sets with n=200 and l=10.*

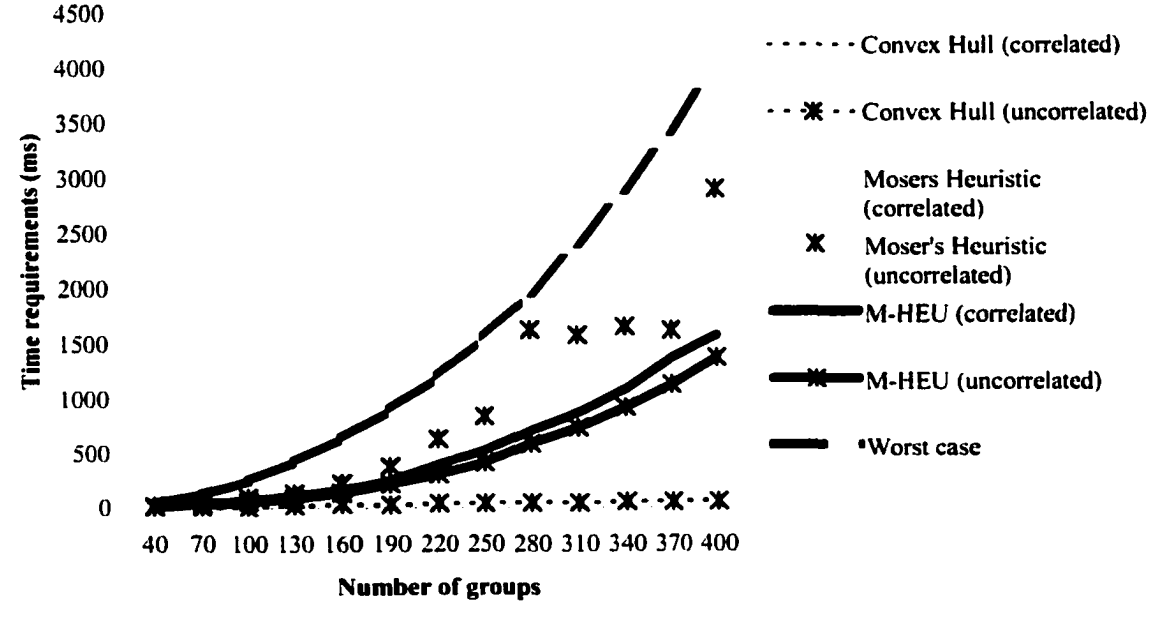


*Figure 3.7 Time required by different heuristics for the MMKP data sets with m=10 and l=10.*

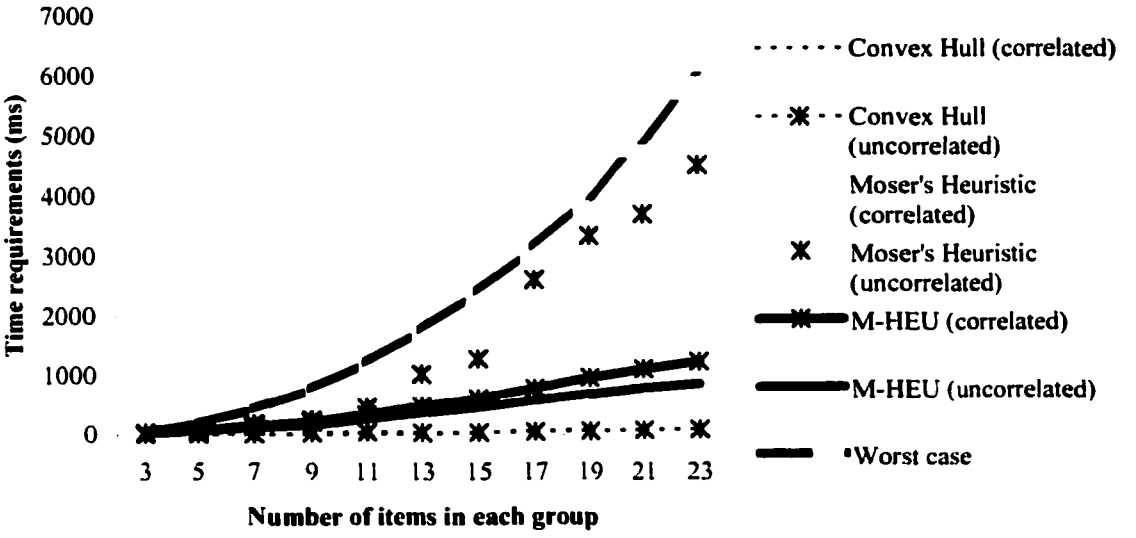


*Figure 3.8 Time required by different heuristics for the MMKP data sets with n=200 and m=10.*

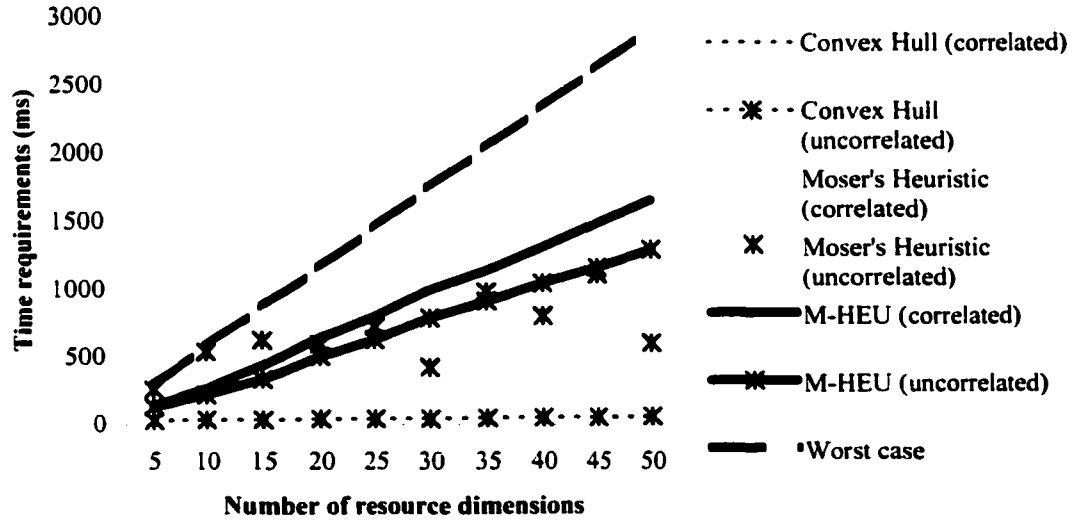


*Figure 3.9 Time required by different heuristics for the MMKP data sets with n=200 and l=10.*

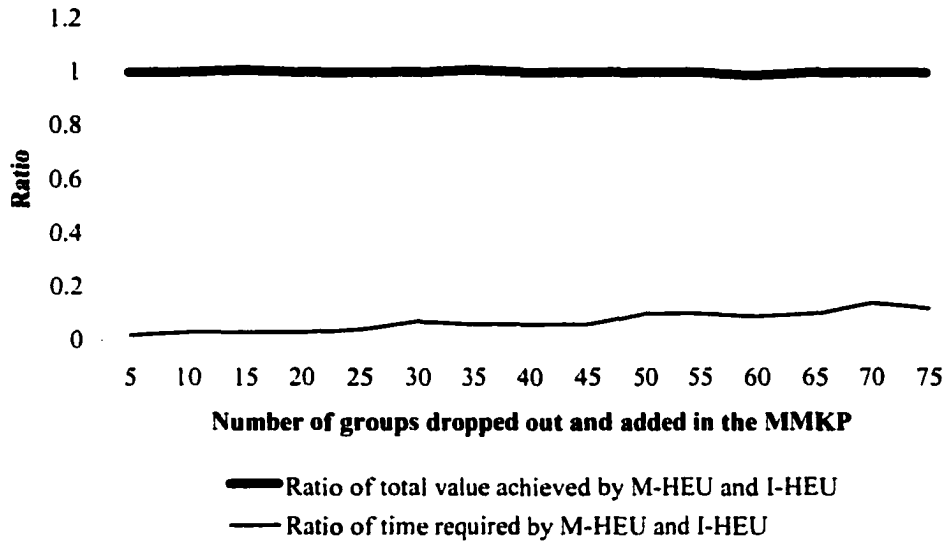


*Figure 3.10 Performance comparison of M-HEU and I-HEU in terms of total value achieved and the time requirements for a problem set with n=200, l=10 and m=10.*

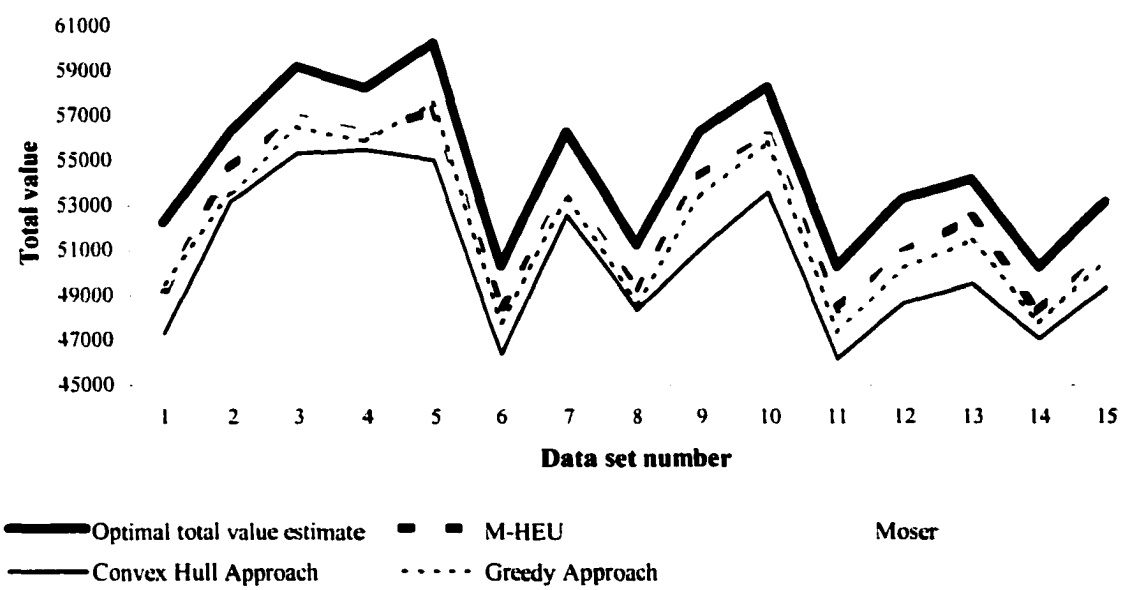


*Figure 3.11 Comparison of the total values of the items picked by different heuristics for 15 correlated problem sets with n=200, l=10 and m=10.*

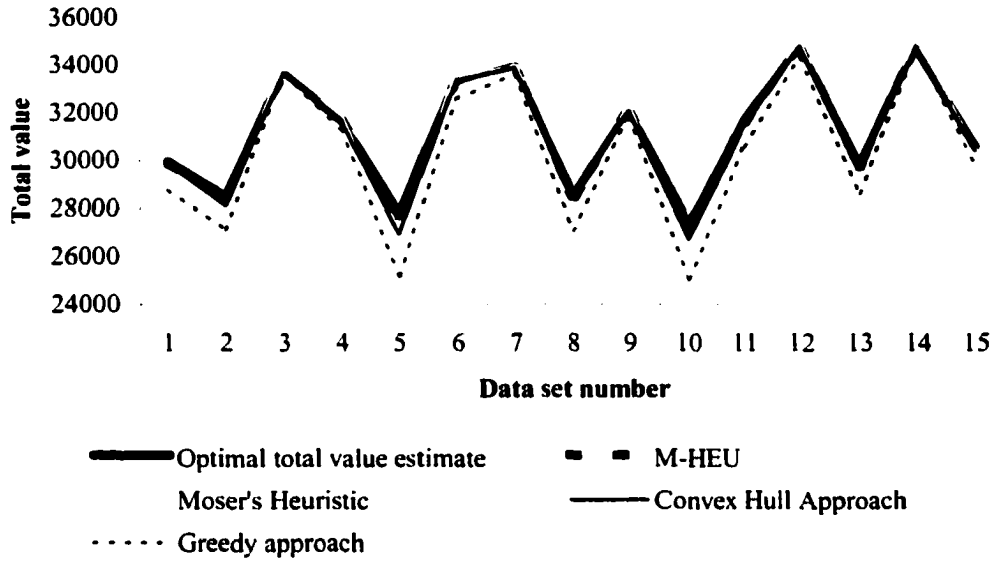


*Figure 3.12 Comparison of the total values of the items picked by different heuristics for 15 uncorrelated problem sets with n=200, l=10 and m=10.*

### 3.6.3 Observations

One can make the following observations from the presented tables and figures.

- The heuristics produce solutions which are close to the optimal solutions provided by the algorithm BBLP. M-HEU produces the solution nearest to the optimal solution among all the heuristics.
- We find from Table 3.1, Table 3.2 and Figure 3.4 to Figure 3.6 that M-HEU, Moser's heuristic and C-HEU give better results for uncorrelated data sets than correlated data sets. C-HEU gives better results than Moser's heuristic for uncorrelated data sets. This is remarkable as C-HEU with $O(nl \log nl + nlm)$ is giving better results than Moser's heuristic with $O\left(mn^2(l-1)^2\right)$.
- We also find from Table 3.3, Figure 3.7 to Figure 3.9 that for correlated data sets, all algorithms take more time than uncorrelated data sets.

- We can give a plausibility argument of the behavioural differences between correlated and uncorrelated data sets in solving the MMKP by M-HEU. When the data are fully correlated, the items of a group lie on a straight line. The items with high resource consumption and high values are picked first. So the algorithm goes to Step 3 when there are significant amounts of resources available. Step 3 tries to improve the solution with an upgrade followed by downgrades. This could potentially be computationally expensive. But if the data sets are random then the picking of items will not be biased (both high and low valued items will be picked with the same probability) and we get nearly optimal solutions with a comparatively large number of iterations in Step 2 and a few iterations in Step 3. When a data set is correlated there is a chance that almost every combination is feasible. In an uncorrelated data set, we generally get more infeasible picking constraints than correlated data sets. We do not need to calculate the aggregate resources for those items. Therefore we can get a solution with fewer iterations for random data sets than for correlated data sets. Recall that the picking criterion of Step 2 is much more efficient for maximization of total value than that of Step 3. This is likely the reason that M-HEU has better optimality for uncorrelated MMKP data sets than correlated MMKP data sets.

- In Moser's heuristic and C-HEU the most profitable items with large values and a small resource requirement are given priority for picking, so we obtain the same behaviour for these heuristics as M-HEU. G-HEU does not use any picking criterion like this. This explains the random nature of the amount of time required by G-HEU.

- The execution time requirements for BBLP in Table 3.3 show that BBLP is practically infeasible for applications requiring real time decision making such as online admission of customers to networks and multimedia servers.

- M-HEU, Moser's heuristic, C-HEU, G-HEU may fail to find a feasible solution where BBLP succeeds. However, for the same set of problem instances, the number of instances where M-HEU or C-HEU failed to find a feasible solution is much smaller

than the failed instances in Moser's Heuristic or G-HEU. The columns denoted by $G_s$, $C_s$, $L_s$, $M_s$, $B_s$ in Table 3.1 and Table 3.2 show this. This might be due to the extra steps used by M-HEU and C-HEU to find feasible solutions in the higher-valued items with lower resource consumptions.

- Figure 3.4 shows that for smaller problem sets the optimality achieved by different heuristics increases with the increase in problem set size. But for larger problem sets the optimality remains almost stable. We find almost the same trend for an increase in the number of resource dimensions [Figure 3.6].

- Figure 3.5 shows that the achieved optimality decreases with an increase in the number of items in each group. This degradation is the worst for C-HEU. This is likely because we completely ignore some items from the search space of each group by constructing convex hulls. In other heuristics we ignore some items while picking items but not at all.

- Figure 3.7 and Figure 3.8 show that the time requirements of M-HEU and Moser's heuristic increase quadratically with an increase in group size (number of groups) and the number of items in each group. The time required by M-HEU is 50% to 70% of that required by Moser's heuristic and also shows less variation than Moser's heuristic. The time required by all these algorithms in the experiments is less than the estimated time requirement (derived from the worst case complexity) for the worst case. The time requirement for C-HEU is much less than for the heuristics of quadratic complexity such as M-HEU and Moser's heuristic, implying that it is not quadratic and it is $O(nl \log nl + nlm)$. Figure 3.9 shows how the time requirements increase linearly with an increase in the number resource dimensions of the MMKP.

- Figure 3.10 shows that the total values of the items picked by M-HEU and I-HEU are almost the same. But I-HEU solves the MMKP quicker than M-HEU. The speed-up factor depends on the batch size of the items added to the MMKP. Although the worst-case complexities are the same for both heuristics, observation suggested we

can expect faster responses by I-HEU in the average case. M-HEU is necessary to solve an MMKP where there is no previously calculated solution.

- If we observe the difference between estimated optimal total value and the total value of the items picked by different heuristics of the MMKP in Figure 3.11 and Figure 3.12, M-HEU appears to be more consistent than any other heuristics applied to solve the MMKP.

- We find some irregularities in the data for computation time. As the computation time required by the heuristic and BBLP depends on the contents of data set, it may happen that smaller data sets take longer than larger data sets. The time requirements for $n$=25 in Table 3.3 show this unlikely situation in practice. If the data set consists of very few feasible solutions it might take less time to get a solution. We find the same irregularity in Figure 3.7 ($n$ = 280 to 370, uncorrelated data sets) and Figure 3.9 ( $m$ = 25 to 40, uncorrelated data sets) for Moser's heuristics.

## 3.7 Chapter Summary

We have presented M-HEU, I-HEU and C-HEU, three heuristics of different computational complexity to solve the MMKP in this chapter. I-HEU and C-HEU are very suitable for applications such as on line admission control in networks and multimedia servers. Different heuristics can be used for different proposes. If the system requires very quick response time then C-HEU and G-HEU are appropriate choices. I-HEU can be used for better revenue when time constraints to do the admission control are more relaxed. The next chapter presents the Distributed Multimedia Server System, a description of the problem of distributing multimedia to users over communication network.

# 4. The New Distributed Utility Model for Distributed Multimedia Server Systems

A new model for admission control and QoS adaptation of Distributed Multimedia Server Systems (DMSS) is presented in this chapter. Different control architectures for DMSS are also proposed together with their accompanying assumptions.

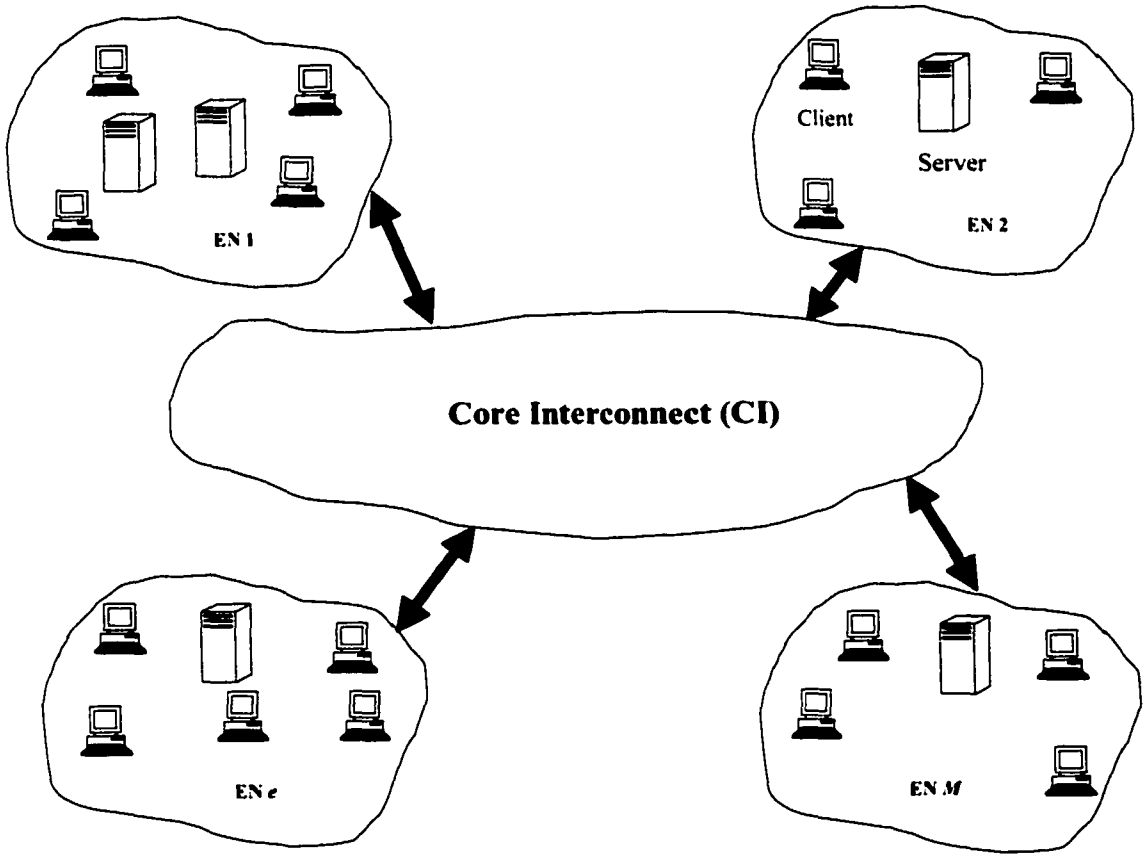


*Figure 4.1 A typical DMSS.*

## 4.1 Distributed Multimedia Server System (DMSS)

The main purpose of the DMSS is to provide multimedia streams to users with guaranteed QoS parameters, such as transmission bit rate, delay and jitter. The servers of

the Distributed Multimedia Server System (DMSS) are a collection of multimedia servers which can be located anywhere in the world and connected through communication networks and owned by a single organization. They distribute responsibility for the provision of multimedia streams and the content of multimedia streams amongst themselves. Figure 4.1 shows a DMSS. Each multimedia server is accessible to the clients through an EN (Enterprise Network)[Section 2.8.1]. Each EN is connected to all other ENs through the *Core Interconnect* (*CI*). The CI carries the multimedia stream when a user of an EN receives multimedia streams from a server of another EN.

## 4.2 Requirements of DMSS

To generalize the Distributed Utility Model, we consider a DMSS with the following requirements.

- Multimedia data have multiple components such as text, audio, video, and images, each with different characteristics. We assume that not all components of multimedia data are necessarily retrieved from the same server i.e., the content of the multimedia stream can be partitioned. For modelling simplicity we also assume that any two components can come from different servers. (There must be another module in the DMSS that is able to synchronize the components of the multimedia streams when they are retrieved from separate servers. This problem is beyond the scope of admission controller and hence of this dissertation. In reality, considering current multimedia technology, certain pairs of components must come from the same server. The video and the accompanying audio of a movie - e.g. a person's talking face and the speech - must come from the same source, as the resulting synchronization problem is otherwise formidable.)
- There may be replication of data in multiple servers. For example, video may be replicated in two servers while text is stored in only one server [Figure 4.2].

- Different servers may serve different types of multimedia streams (music, movie etc.). Each server advertises its multimedia streams (the name of the music or movie) to the other servers.
- There must be one or more admission controllers (not shown in Figure 4.1) in the DMSS doing admission control and QoS adaptation of the submitted requests for multimedia sessions.

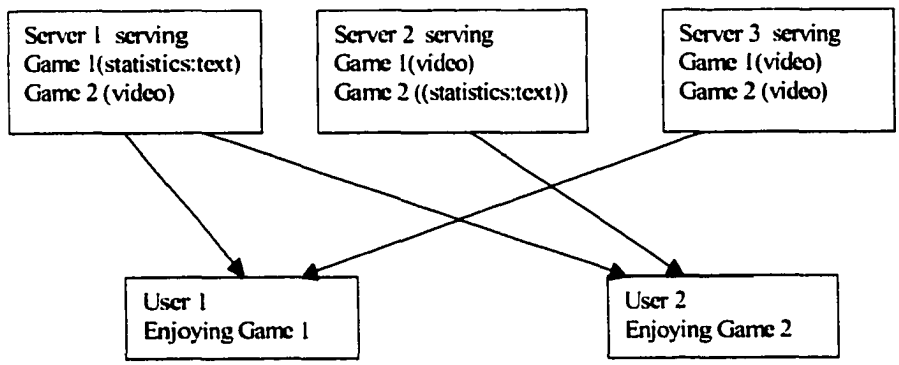


*Figure 4.2 A typical example of Distributed Multimedia Service*

## 4.3 Admission Control and QoS Adaptation in the DMSS

The admission controller in the single server version of the multimedia service provider determines a particular QoS level for each multimedia session. In the DMSS, in addition to this, the admission controller also determines which server a session will use for each component of the multimedia stream. As DMSS resources are limited, it is necessary to select optimal QoS levels and servers for the sessions to maximize the revenue earned by providing multimedia services to the users, while not over consuming any resource. The following example demonstrates the necessity of admission control and QoS adaptation to achieve optimal revenue while avoiding over-consumption of resources.

Consider two servers in the DMSS. Server 1 serves B/W video and Server 2 serves colour video. One server cannot provide video to more than one user at a time. User 1 submits

its request for multimedia service followed by user 2. The requests are shown in Table 4.1.

*Table 4.1 Requests with options and offered prices for Multimedia service.*

| Quality Option | Bid value ($) | |
|---|---|---|
| | User 1 | User 2 |
| B/W video | 5 | 5 |
| Colour Video | 10 | 15 |

*Table 4.2 Quality adaptation and change of server .*

| Users | Server for user 1 | Server for user 2 | Revenue |
|---|---|---|---|
| User 1 | Server 2 (Colour) | | 10 |
| Users 1 and 2 | Server 1 (B/W) | Server 2 (Colour) | 20 |

User 1 is willing to pay less for colour video than user 2. User 1 is downgraded to B/W quality and forwarded to server 1. Thus the DMSS earns better revenue from User 2 by serving colour video from Server 2 (we ignore the costs of supplying service in this simple example, and assume that gross revenue equals net revenue). Hence, a separate module must be employed to determine the admitted sessions and their optimal levels of quality of service in Distributed Multimedia Server Systems.

## 4.4 Utility Model - Distributed (UM-D)

The Utility Model - Distributed provides a strategy for selecting an optimal set of sessions (with particular QoS levels and selection of servers to provide multimedia stream components) from the submitted requests for multimedia sessions. The goal is to maximize the revenues earned from the selected sessions, subject to the resource constraints imposed by the servers and networks of the DMSS. The UM-D mainly describes the admission control and QoS adaptation strategy for the multimedia sessions in mathematical notation. The following assumptions are made for modelling purposes.

### 4.4.1 Assumptions

- Users are required to communicate with (one of) the admission controllers of the DMSS to submit their requests, and to receive admission or rejection decisions made by the controllers. The servers of the DMSS notify the admission controllers about the amounts of available resources. The controllers may communicate among themselves to make a decision regarding a session's admission. The underlying network must be able to accommodate the messages required for the admission control and QoS adaptation processes.
- The utility obtained and resources required may not be the same for a given service on two different servers, owing to the different costs of providing the service and the different technology (e.g., operating system or compression algorithms) used in different servers. The utility is therefore defined as *net revenue*, i.e. the bid price less the cost of service; it is assumed to always be positive.
- The amount of available resources is not identical in different servers. The I/O bandwidth and CPU cycles are the resources modelled for the servers of the DMSS. The link bandwidths in the ENs and Cl are the resources modelled for the networks of the DMSS.
- Metrics such as available bandwidth, delay and jitter for multimedia transmission from one EN to another EN are collected by a network performance data collection facility. The values of these metrics are available to the admission controllers.

### 4.4.2 Mathematical Formulation

Let the following functions be available:

$D(s,i)$= Maximum latency suffered by a multimedia stream transmitted on a path from the $s$th server to the $i$th user of the DMSS.

$J(s,i)$= Maximum jitter on the path from the $s$th server to the $i$th user of the DMSS.

There are $n$ sessions in the DMSS. For each session the user specifies one or more acceptable levels of QoS. The $n$ sessions can be represented by $n$ vectors of quality profiles, as shown below. Session $i$ contains $l_i$ quality profiles. Let the bid price offered by user $i$ for quality $j$ be $b_{ij}$. Then

$$\vec{q}_1 = (\vec{q}_{11}, \vec{q}_{12}, \ldots\ldots\ldots\ldots\ldots\ldots, \vec{q}_{1l_1})$$

$$\vec{q}_2 = (\vec{q}_{21}, \vec{q}_{22}, \ldots\ldots\ldots\ldots\ldots, \vec{q}_{2l_2})$$

$$\vdots$$

$$\vec{q}_n = (\vec{q}_{n1}, \vec{q}_{n2}, \ldots\ldots\ldots\ldots\ldots, \vec{q}_{nl_n})$$

Here, $\vec{q}_{ij}$ = the $j$th level of quality of the $i$th session.

Let there be $N_c$ component streams in the composite multimedia stream. The quality vector $\vec{q}_{ij}$ has $P_{ij}$ components, where $0 < P_{ij} \leq N_c$. A component of $\vec{q}_{ij}$, $\vec{q}_{ijk}$, is the level of quality desired for component $k$ of the multimedia stream for the $j$th QoS level of the $i$th session. Note that Session $i$ at QoS level $j$ may not require all of the data components. Thus, $\vec{q}_{ijk} = 0$ implies that component $k$ is not required for QoS level $j$ of Session $i$. The associated delay and jitter constraints for $\vec{q}_{ij}$ are $D_{ij}$ and $J_{ij}$.

Quality component $\vec{q}_{ijk}$ must be offered by at least one of the $M$ servers in the distributed system, or the session will be rejected out of hand. The offered bid price $b_{ij}$ will be distributed among the quality components by any suitable mapping function $f_{db}$ (distribute bid). Then the bid price $b'_{ijk}$ will be mapped to utility (net revenue earned from the component) $\vec{u}_{ijk}$ by another mapping function $f_{bu}$ (bid to utility). These two mapping functions will be more complex when the DMSS is owned by different organizations. Maximization of utility for multiple organizations is beyond the scope of this dissertation. Here, the utility is a vector. The $s$th component of the vector $\vec{u}_{ijk}$ is $u_{ijks}$, which indicates

the utility if the *s*th server provides the *k*th component of the *j*th QoS level of the *i*th session.

$$f_{db}(b_{ij}) = \vec{b}'_{ij} = (b'_{ij1}, b'_{ij2}, \ldots\ldots\ldots\ldots, b'_{ijP_n})$$
$$f_{bu}(b'_{ijk}) = \vec{u}_{ijk}$$
$$\vec{u}_{ijk} = (u_{ijk1}, u_{ijk2}, \ldots\ldots, u_{ijks} \ldots\ldots\ldots\ldots, u_{ijkM})$$

The utility of Component *k*, at QoS level *j*, from Server *s*, will be null (i.e., $u_{ijks} = 0$) for the following cases.

- Server *s* does not provide component *k* of the multimedia stream.
- $D(s,i) > D_{ij}$ or $J(s,i) > J_{ij}$, i.e., if the multimedia stream is transmitted from Server *s*, then the delay or jitter bound cannot be satisfied.

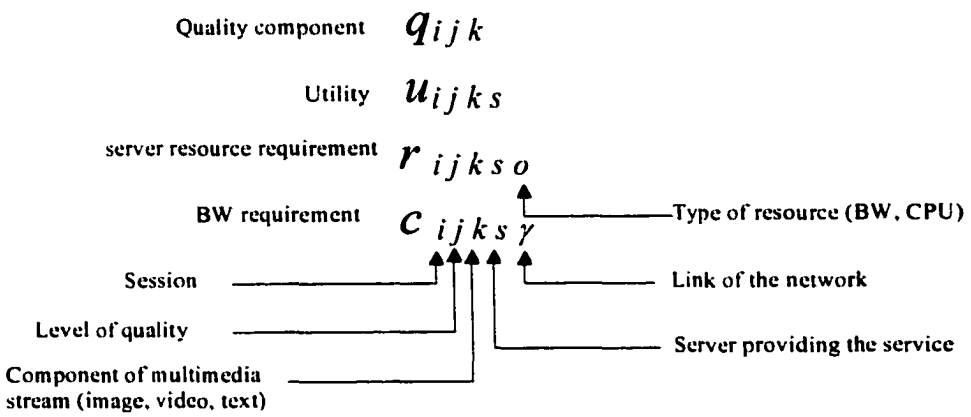


*Figure 4.3 subscript definitions*

The users should be completely unaware of the distribution of quality and utility among the servers. That is, the subscript *s* is to be made invisible to the users. If we map quality $\vec{q}_{ijk}$ into the quantities of the $m_s$ resources from the *s*th server, then the resource vector can be expressed as follows

$$r_s(\vec{q}_{ijk}) = (r_{ijks1}, r_{ijks2}, \ldots\ldots, r_{ijkso} \ldots\ldots\ldots\ldots, r_{ijksm_s})$$

Similarly, to deliver multimedia quality component $\vec{q}_{ijk}$ from the $s$th server, the bandwidth requirement can be expressed as follows

$$C_s(\vec{q}_{ijk}) = (c_{ijks1}, c_{ijks2}, \ldots\ldots, c_{ijksy} \ldots\ldots\ldots\ldots\ldots, c_{ijks\mu})$$

where

$\mu$ = total number of network links in the DMSS.

The vector of available resources of the servers in the DMSS can be expressed as follows:

$$\vec{R} = (\vec{R}_1, \vec{R}_2, \cdots, \vec{R}_s, \ldots\ldots\ldots, \vec{R}_M)$$

Here $\vec{R}_s$ indicates the resources provided by the $s$th server. The $m_s$ resource components of this server can be expressed as follows.

$$\vec{R}_s = (R_{s1}, R_{s2}, \ldots\ldots\ldots\ldots, R_{sm_s})$$

The total capacity of the links is expressed as follows:

$$\vec{C} = (C_1, C_2, \ldots\ldots\ldots\ldots, C_\mu)$$

Now the problem can be formulated as follows

***Objective:*** $\text{maximize} U = \sum_{i}^{n} \sum_{j}^{l_i} \sum_{k}^{P_{ij}} \sum_{s}^{M} x_{ij} y_{ijks} u_{ijks}$ , subject to the constraints listed below.

Here, $x_{ij}$ is the binary picking or selection variable associated with session $i$ and level of quality $j$, namely:

$$x_{ij} = \begin{cases} 1, \text{ if the } j\text{th quality level of the } i\text{th session is selected} \\ 0, \text{otherwise} \end{cases}$$

$y_{ijks}$ is the binary-valued variable associated with the selection of the $s$th server for multimedia component $k$ to provide the $j$th QoS level to the $i$th session.

The *constraints* applied to this optimization problem are:

***Constraint* 1:** For every session $i$ only one level of quality $j$ will be granted. i.e., $\sum_{j}^{l_i} x_{ij} = 1$ where, $x_{ij} \in \{0,1\}$. The 0th QoS level consumes no resources with no revenue. This is called null QoS level. When this level is selected the request is assumed to be rejected.

***Constraint* 2:** Any component of any level of quality for any session will be supplied by resources from one server. i.e., $\sum_{s}^{M} y_{ijks} = 1$ where, $y_{ijks} \in \{0,1\}$

This means that there is no need to provide one component of any multimedia stream, from more than one server.

***Constraint* 3:** All the components of a multimedia stream at a particular level of quality will be stored in the servers of the system. No server is to be selected for any stream component $\vec{q}_{ijk}$ where utility $u_{ijks} = 0$. This constraint ensures the integrity of the DMSS agreement that a user will enjoy all the components of the multimedia stream specified in the granted QoS level i.e., $\sum_{s}^{M} y_{ijks} u_{ijks} > 0$.

***Constraint* 4:** (Distributed resource constraints)

- For server resources $\sum_{i}^{n} \sum_{j}^{l_i} \sum_{k}^{P_{ij}} x_{ij} y_{ijks} r_{ijkso} < R_{so}$ , where $o = 1,2,.....m_s$ and $s = 1,2,.....M$

This means that any server $s$ cannot provide more of the *oth* type of resource (bandwidth, memory, CPU) than it possesses ($R_{so}$). Rigorous observation of this constraint and the next one is, of course, essential to provide guaranteed levels of absolute QoS.

- For link bandwidths, $\sum_{i}^{n} \sum_{j}^{l_i} \sum_{k}^{P_{ij}} \sum_{s}^{M} x_{ij} y_{ijks} c_{ijks\gamma} < C_{\gamma}$ , where $\gamma = 1,2, \ldots \ldots \mu$

This means the total allocation, to all admitted sessions, of the bandwidth of a link must not exceed its capacity $C_\gamma$.

## 4.5 Admission and QoS Adaptation Controller Architectures for DMSS

The users of the DMSS receive multimedia streams at particular QoS levels , determined by the policy described in the UM-D. There are several possible arrangements or architectures of UMEs (Utility Model Engines), working as admission controller(s) to solve the UM-D in a DMSS cooperatively. Table 4.3 describes different architectures of UMEs.

*Table 4.3 Different architectures of UMEs*

| **Architectures** | | **Resource location** | |
|---|---|---|---|
| | | **Distributed** | **Centralized** |
| **Computation** | **Distributed** | There must be multiple UMEs and servers in different locations. The UMEs will exchange messages to run the distributed version of the admission control and QoS adaptation algorithm. | The machine running the UME has multiple processors with shared memory. The available resources of the multimedia server are shared by the processors running the parallel algorithm for optimal admission control and QoS adaptation. |
| | **Centralized** | This is called the *broker architecture.* There is one UME solving the problem of admission control and QoS adaptation where resources are distributed among multiple servers. | This is the simplest model. UME and resources are located in one server. This case has been studied by Khan [44]. |

## 4.6 Candidate Architectures

Here we present two possible architectures or arrangements for constructive solutions of the UM-D; the broker architecture and the fully-distributed architectures.

### 4.6.1 Broker Architecture

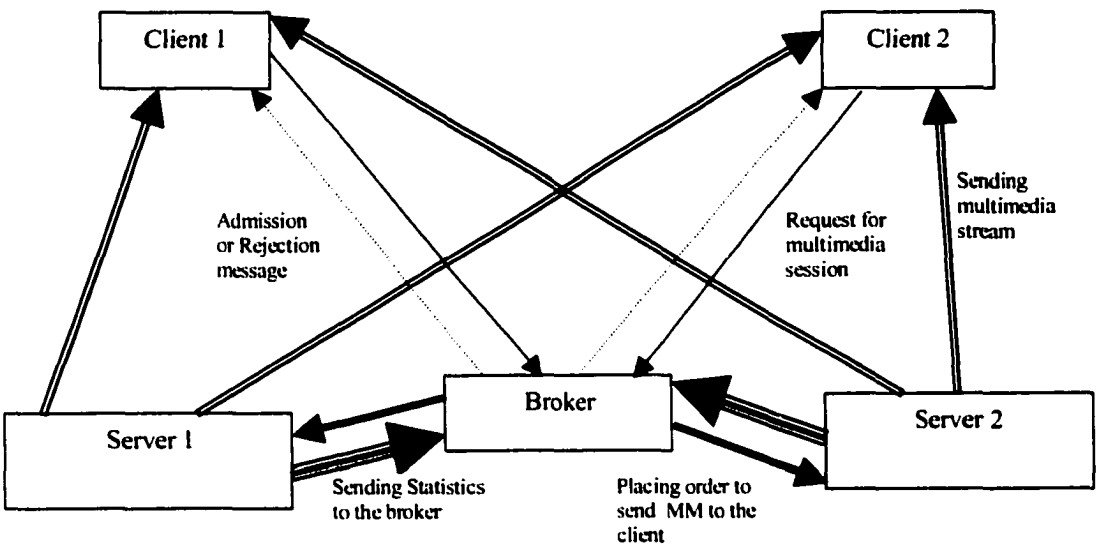


*Figure 4.4 Broker for the DMSS. Similar arrowed lines indicate the same type of request, message or data transfer.*

This architecture utilizes centralized computing performed by a broker. The broker is interposed between the clients and servers of the system. The architecture of the DMSS with broker is shown in Figure 4.4 for two clients and two servers. [These two clients and servers might be from different regions.] The functions of the broker are as follows:

- It accepts requests from the clients (normal arrows).
- It maintains state information of resources, services and qualities offered to the clients. Each server sends this state information to the broker (arrows with triple lines).

- It determines the set of the selected QoS levels and the server for each component of the multimedia streams, to maximize utility of the system without violating any QoS constraint.
- It notifies clients of the approval or rejection of requests and negotiates with them (dotted arrows).
- It orders servers to send data directly to the clients (thick arrows).

This is a suitable architecture to demonstrate the optimal selection of resources from a set of servers, a current problem in content switching and routing [85]. This architecture is also suitable for a centralized SLA Controller in an EN doing admission control and QoS adaptation of SLAs in an EN. SLAOpt, implemented by Watson [40] is a good example of such a system.

### 4.6.2 Fully Distributed DMSS architecture

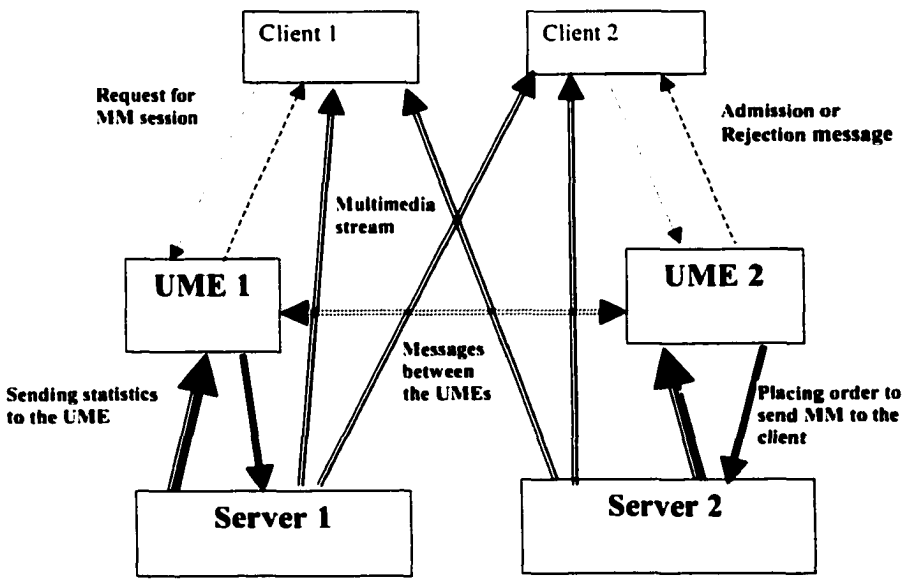


*Figure 4.5 Fully distributed DMSS Architecture. Similar arrow lines indicate the same type of request, message or data transfer.*

In this approach there is no centralized broker, as shown in Figure 4.5. All UMEs can receive requests from clients and all UMEs run parts of the UM-D collectively to maximize the revenue in the distributed system. This requires a *distributed algorithm* for solving the UM-D. The following features make this fully distributed architecture more attractive than centralized broker control architecture:

- It is more manageable. Each server in the DMSS can run an instantiation of the UME.
- It is scalable, because the admission control is distributed among the UMEs.
- We get potentially better fault tolerance, because the failure of a UME does not disrupt the service for all users.

So, this architecture is attractive for admission control of multimedia sessions in a DMSS or SLAs in interconnected ENs, where multiple instantiations of the UME are deployed to do admission control.

## 4.7 Chapter Summary

We have presented UM-D, a new model for admission control and QoS adaptation of requested multimedia sessions in Distributed Multimedia Server Systems (DMSS). This model gives the optimal strategy to select servers and QoS levels for a set of multimedia session requests. Two control architectures (centralized and distributed) to solve the UM-D have been described, and their relative advantages and disadvantages have been discussed. The next chapter describes how the admission control and QoS adaptation problem in the DMSS using different control architectures maps to the variants of Knapsack Problems.

# 5. Mapping of Admission Control and QoS Adaptation Problems to the Variants of Knapsack Problem

This chapter presents the mapping of the admission control and QoS adaptation problem in a DMSS to the variants of Knapsack Problems based on the optimal strategies described in the UM-D. Admission control and QoS adaptation by the centralized broker architecture can be mapped to the MMKP. But admission control and QoS adaptation using fully distributed controllers is mapped to the MMMKP, a new variant of the Knapsack Problem. We introduce this new variant of the Knapsack Problem mathematically. In this way the algorithms for solving the variants of the Knapsack Problem are rendered applicable for admission control and QoS adaptation in the DMSS.

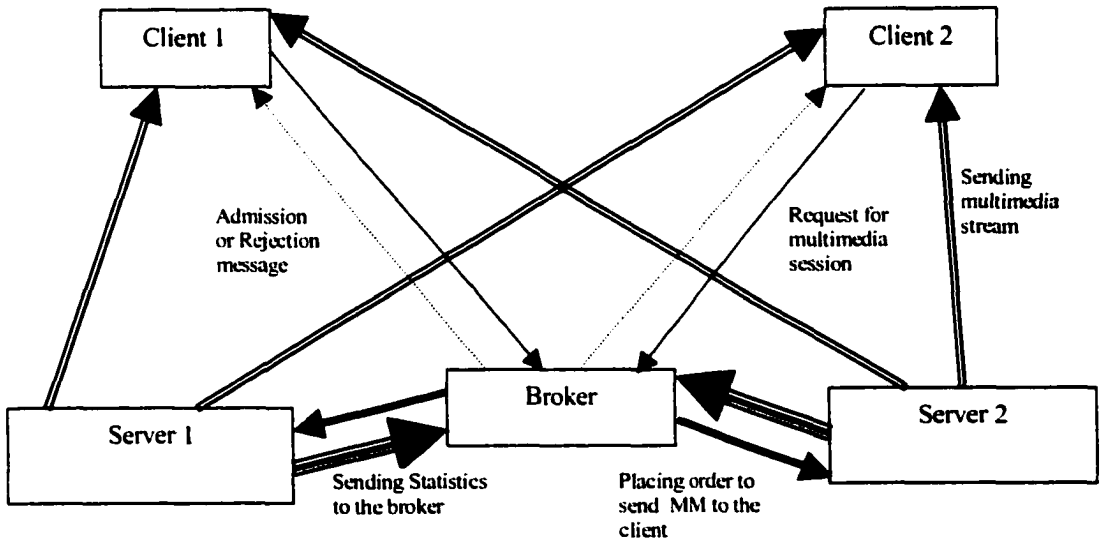


*Figure 5.1 Broker for the DMSS. Similar arrowed lines indicate the same type of request, message or data transfer. Reproduction of Figure 4.4.*

## 5.1 Mapping of the UM-D by Centralized Broker to the MMKP

Each server in the DMSS represents one knapsack with multidimensional resources such as CPU cycles, I/O and Memory. We ignore the link bandwidths for the simplicity of

mapping in this chapter. Knapsack Problem specifies resource management. Our policy is to do admission control by allocating resources. That is why Knapsack Problem is suitable to represent the admission control problem. Admission control and QoS adaptation in the DMSS can be mapped to the transportation problem [99] if we consider only one QoS level in the session requests. The following subsections present the mapping of sessions, QoS levels and resources of the servers defined in the UM-D to the groups, items and resources of the MMKP.

### 5.1.1 Mapping of Sessions to the Groups

Each multimedia session request with a set of QoS levels submitted to the broker represents a group in the MMKP.

### 5.1.2 Mapping of the QoS levels of a Session to the Items of a Group

The *j*th QoS level of the *i*th session, $\vec{q}_{ij}$, can be provided by one or more servers, because the components of the multimedia stream can be partitioned and replicated in multiple servers. We can find all the combinations of servers that can serve a particular level of QoS of a particular requested multimedia session. Let there be $M$ servers, each serving $P_{ij}$ components of $\vec{q}_{ij}$. Now this level of QoS can be provided by $M^{P_{ij}}$ combinations of servers [Figure 5.2].

These combinations might have different utilities. These are indistinguishable from the user's point of view, but are very different from the DMSS broker's point of view. Thus one QoS level is transformed to multiple options. Each of these options is defined as *QoS with respect to Broker* (*QoS-B*) and represents an item of a group in the MMKP. Let there be $l'_{ij}$ QoS-B levels for a QoS level $\vec{q}_{ij}$. The *j'*th QoS-B level of the *j*th QoS level is defined by $\vec{q}_{ijj'}$ and the server providing the *k*th multimedia stream component is $s_{ijj'k}$, a number lying between 1 and $M$. It would be computationally expensive (exponential complexity) to find all QoS-B levels of the sessions submitted to a DMSS with large

number of servers. However, The search space for QoS-B levels can be easily pruned if the almost fully-loaded servers (the servers with small amounts of available resources) are pruned from the list of those servers which can provide a component of the multimedia stream for the new admitted sessions. When there is no contention in the system we arbitrarily prune some of the servers from the list of servers as admission control is not necessary when there are avaialble resources in the system.

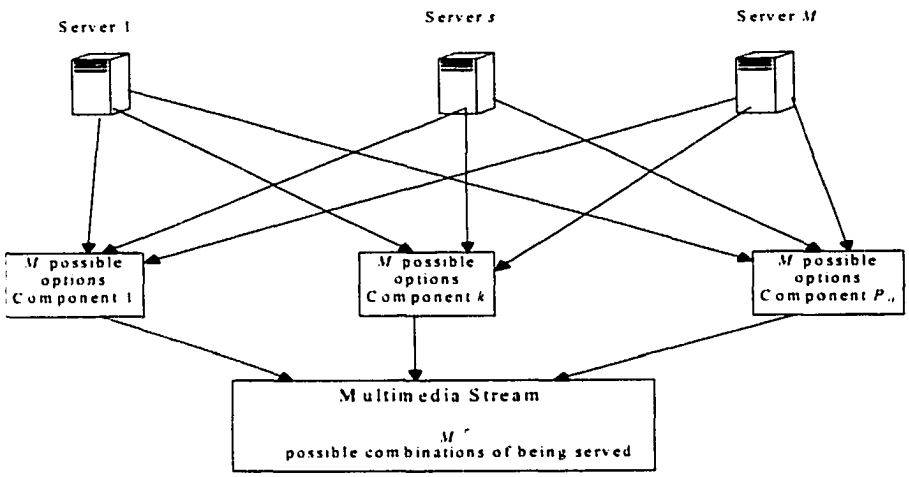


*Figure 5.2 Different options of selecting servers*

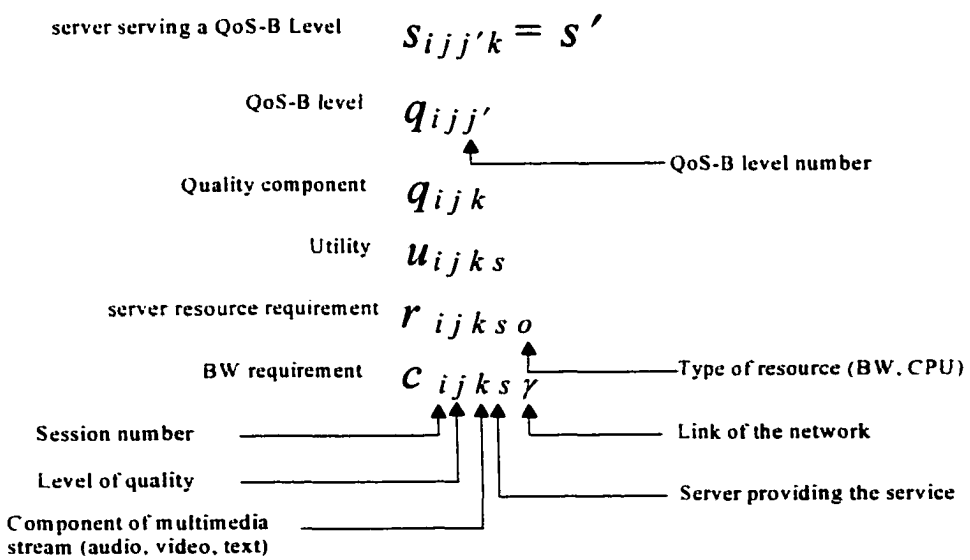


*Figure 5.3 Subscript definitions*

*5.1.2.1 Values and Resource requirements of the Items*

As the UM-D distributes the utility of a QoS level among the components of the multimedia stream, the utilility of a QoS-B level can be expressed by the summation of the component utilities. The resource requirements in the servers and the network are defined by the vector summations of the resource requirements of the components.

In mathematical notation, the utility, server and network resource requirements of a QoS-B level $\vec{q}_{ijj'}$ can be expressed as follows:

$$u(\vec{q}_{ijj'}) = \sum_{k=1}^{P_{ij}} u_{ijks'}, r(\vec{q}_{ijj'}) = \sum_{k=1}^{P_{ij}} \vec{r}_{ijks'} \text{ and } c(\vec{q}_{ijj'}) = \sum_{k=1}^{P_{ij}} \vec{c}_{ijks'}, \text{ where, } s' = s_{ijj'k}$$

## 5.1.3 Mapping of Server Resources to the Resources of the Knapsack

If there are $m_c$ resource dimensions in each of $M$ multimedia servers, then we can think of a *merged server* with $M \times m_c$ resources; it is in effect the union of the physical servers, and its resources belong to the cross-product space (server, resource). In this way we can map multiple servers to one merged server and hence to one knapsack. The links of the network are considered as resources of this knapsack.

## 5.1.4 Objective of the Broker

Now, our problem is to find a QoS-B level for each session for maximum utility of the DMSS, subject to resource constraints of the merged server, which is treated as an MMKP. An example will help to clarify this scenario; one is provided below.

## 5.1.5 An Example of Mapping UM-D to the MMKP

Figure 5.4 depicts a DMSS with two servers, two sessions, two multimedia stream components ($f$ and $g$) and two resources ($p$ and $q$) in each server. $p_i$ and $q_i$ represent the resources in the $i$th server. The link bandwidths of the underlying network have been ignored for simplicity.

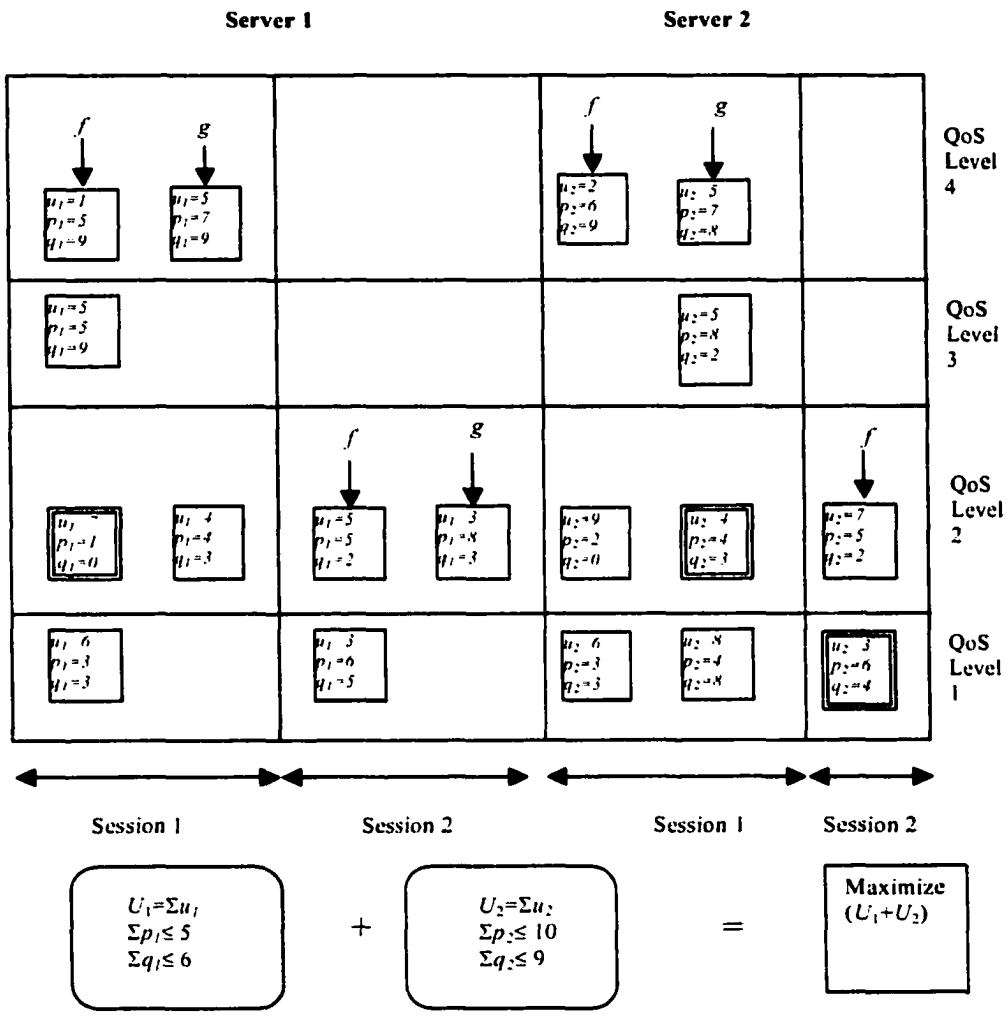


*Figure 5.4 Resources and sessions in a DMSS*

In Figure 5.4, the smaller rectangular boxes represent the service as well as the quality components of the multimedia stream provided by a server for a session, and $u_i$ is the value of utility for the $i$th server. Session 1 has 4 levels of QoS and Session 2 has 2 levels of QoS. The sessions can get component $f$ or component $g$ of the multimedia streams from either server, iff that server provides the specified component. As explained earlier in Section 4.4.1, the resource requirement for a particular session might be different if the session is served by different servers. For example, Session 1 has been admitted at QoS level 2 and Session 2 at QoS level 1 (shown by the boxes with doubled borderlines).

Session 1 is getting component *f* from Server 1 and component *g* from Server 2. So the total utility from Session 1 is (7+4)=11. Session 2 has only component *f* for QoS level 1. This component is served by Server 2 with utility 3. So the sessions consume ($\Sigma p_1$=1, $\Sigma q_1$=0) resources from Server 1 and ($\Sigma p_2$ = 4+6 , $\Sigma q_2$ = 3+4) resources from Server 2. The maximum units of resources provided by the two servers are (5, 6) and (10, 9) respectively. The objective of the selection is to maximize $U = \sum U_1 + \sum U_2$ , i.e., maximize the total utility of the two servers; we assume that the utility of the constellation of servers is the sum of their individual utilities, which is a plausible assumption if they are owned by a single owner.

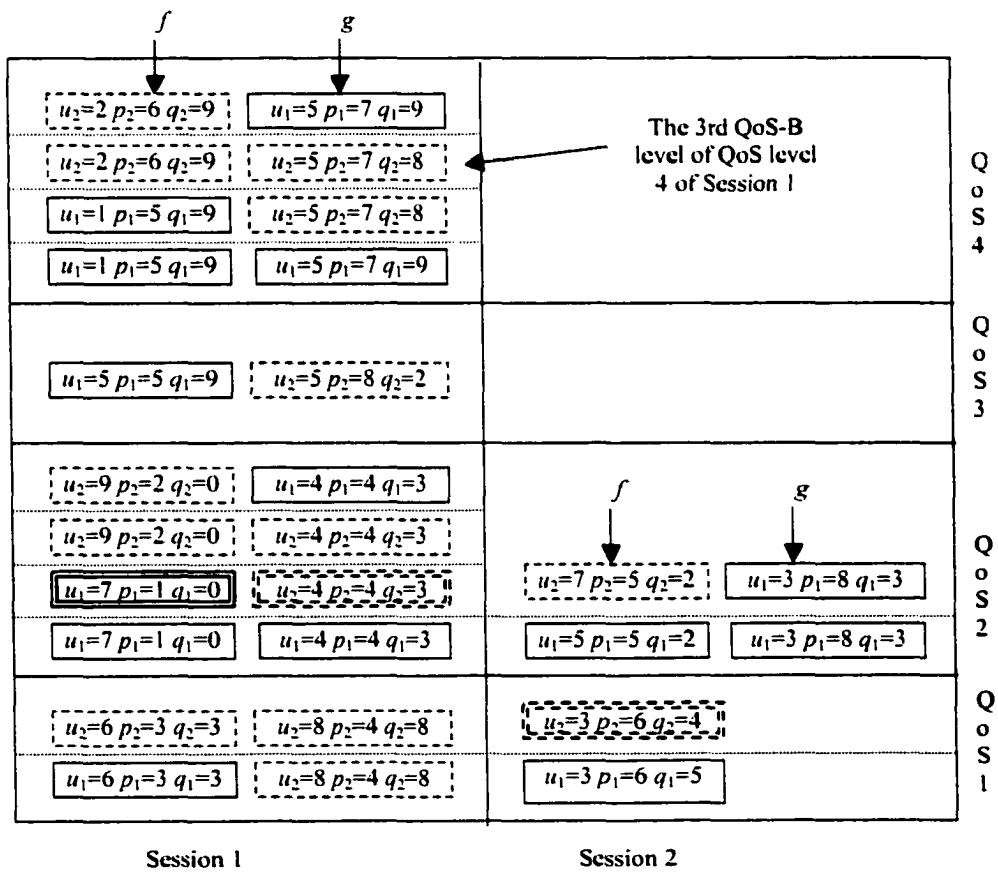


*Figure 5.5 Mapping of broker to an MMKP.*

Now to map the problem into an MMKP, the two servers are merged. The QoS-B levels are shown in Figure 5.5. A QoS level has one or more rows of QoS-B levels separated by dotted lines indicating choices of the servers within a QoS level. The resource consumption by the first and second server has been identified by the boxes of solid and broken borderlines respectively. Now this is an MMKP with 2 groups and 4 resources. The first group has 11 items and the second one has 4 items.

## 5.2 Mapping of the UM-D to a New Variant of Knapsack Problem

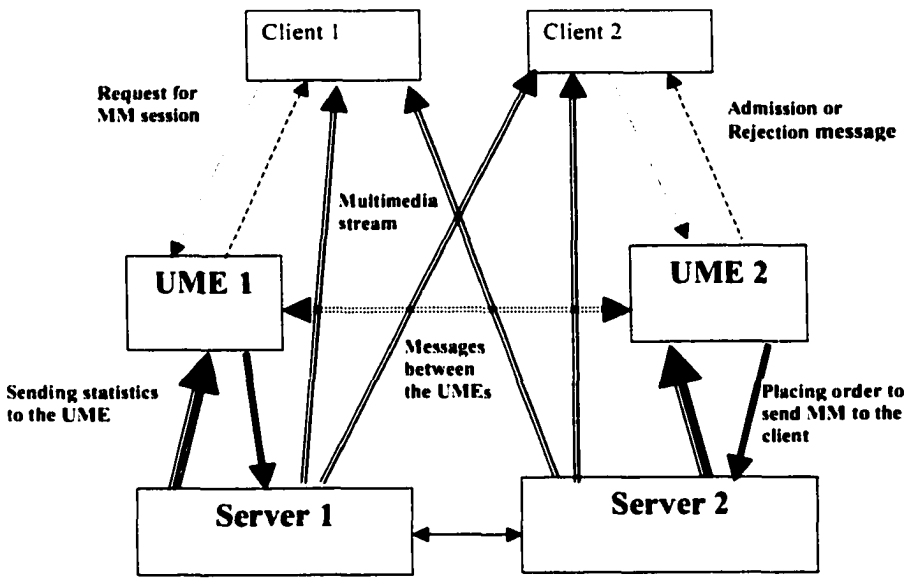


*Figure 5.6 DMSS architecture controlled by fully distributed controllers. Reproduction of Figure 4.5.*

We have already mentioned in Section 4.6.2 that using fully distributed UMEs in the DMSS requires a distributed algorithm to solve the UM-D. The following points describe how this problem maps to a new variant of Knapsack Problem.

- Each server in the DMSS maps to a knapsack and the UME in each server maps to a *solver* picking items.

- The resources of each server are the CPU, I/O and memory. We can assign a link between two servers in the DMSS to a particular UME, i.e., a particular UME allocates or releases bandwidth on that link for the particular session.
- Each multimedia session submitted to a UME maps to a group in the solver of a knapsack.
- Each QoS level can be served by several combinations of servers. Each of these combinations represents an item in a group.

The Knapsack Problem represented by Figure 5.7 is different from the MMKP, as it consists of more than one knapsack amongst which the resources are distributed. Solving the problem of maximizing the total revenue earned by all the knapsacks requires communication between the solvers. So, distributed computing techniques must be applied to solve this problem. The next section introduces the MMMKP (Multidimensional Multiple-choice Multi Knapsack Problem), a new variant of Knapsack Problem.

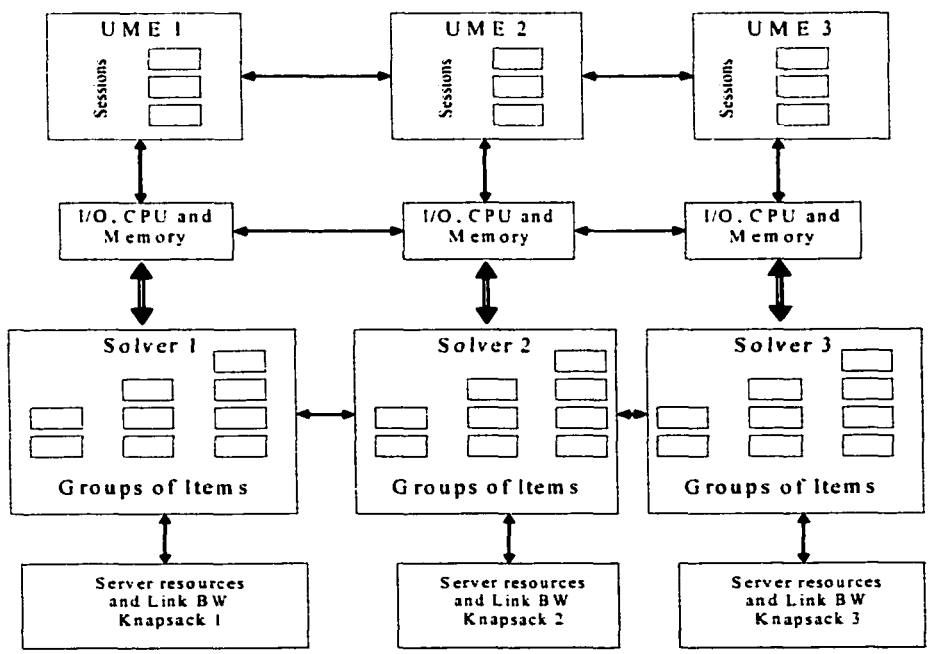


*Figure 5.7 Mapping of the UM-D to the MMMKP.*

## 5.3 Definition of the MMMKP

We define the MMMKP as a distributed version of the MMKP, where the resources are distributed among knapsacks. There is a *solver* associated with each knapsack for picking the items from the group. So, distributed computing techniques will be required for picking items. The following diagram shows an example of the MMMKP.

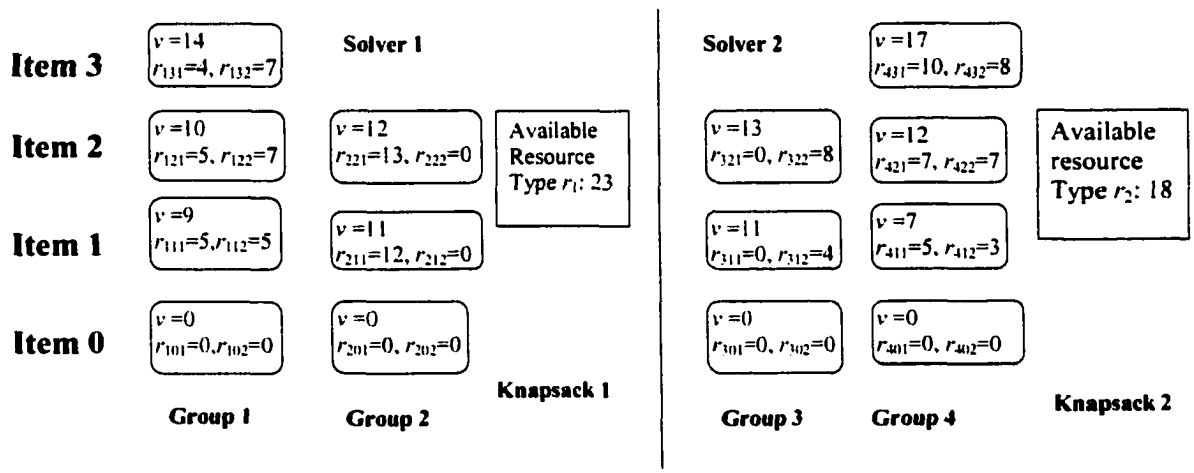


*Figure 5.8 An MMMKP with 2 knapsacks and one resource in each knapsack*

Group 1 and Group 2 have been submitted to Solver 1 and Group 3 and Group 4 have been submitted to Solver 2. Knapsack 1 contains 23 units of resource $r_1$ and Knapsack 2 contains 18 units of resource $r_2$. The groups submitted to a particular solver are labelled as *local groups* for that solver. A solver makes final decisions about picking items from its local groups. Each item of the groups submitted to any of the solvers has a utility value, and consumes specified amounts of resources $r_1$ and $r_2$. Groups 2 and 3 are *purely local* as they consume only one kind of resource: $r_1$ and $r_2$ respectively.

The state of the variable *available resource* in a knapsack is available only to its associated solver for both reading and updating. In this sense, the two knapsacks are autonomous; only Solver $s$ can allocate resources of Knapsack $s$. The joint objective of

the solvers is to pick exactly one item from each group so that the total value of the picked items is maximized subject to resource constraints.

The following points clearly explain why this is not a simple problem of two MMKPs running independently.

- The solvers can do their picking independently if and only if all the items of the groups in all the solvers are purely local.
- We can divide each item into two sub items, one for each kind of resource consumption. For example an item represented by $v$=14, $r_{131}$= 4 $r_{132}$=7 (group 1 item 3) can be divided into two sub items as $v_1$=14, $r_{131}$=4 and $v_2$=14, $r_{132}$=7. Now these two items can be submitted to Solver 1 and Solver 2 respectively. The solvers will not be able to work independently, as they must interact to pick the same sub item from each group.
- A solver could be allowed to pick items from its local groups to maximize revenue by itself, subject to resource constraints in all knapsacks of the MMMKP. The example in Table 5.1 shows that we cannot maximize the total value earned by using this strategy.

*Table 5.1 Selection of the items using different strategies*

| Groups | Picked Items (Solver 1 is picking first for maximizing revenue by Solver 1) | Picked Items (Solver 2 is picking first for maximizing revenue by Solver 2) | Picked Items (Solver 1 and Solver 2 are picking jointly) |
|---|---|---|---|
| Group 1 | 3 | 0 | 3 |
| Group 2 | 2 | 2 | 1 |
| Group 3 | 2 | 2 | 1 |
| Group 4 | 1 | 3 | 2 |
| Total Value | $V_1$=26 $V_2$=20 | $V_1$=12 $V_2$= 30 | $V_1$=25 $V_2$=23 |
| Resource consumption | $\sum r_1 = 22$<br>$\sum r_2 = 18$ | $\sum r_1 = 23$<br>$\sum r_2 = 16$ | $\sum r_1 = 23$<br>$\sum r_2 = 18$ |
| Observations | Maximum profit from Knapsack 1 but not an overall profit maximization. | Maximum profit from Knapsack 2 but not an overall profit maximization. | The overall maximum profit has been achieved. |

To define the MMMKP mathematically we need the following assumptions about the problem.

- There are $M$ knapsacks. $M$ solvers (one for each knapsack) pick items from the groups.
- The dimension of resources in Knapsack $s$ is $m_s$ and it provides resources labelled as $\mu_s$ to $\mu_s + m_s - 1$ inclusive. The total set of resources of the $s$th knapsack is defined by $(R_{\mu_s}, R_{\mu_s+1}, \cdots\cdots\cdots, R_{\mu_s+m_s-1})$.
- Each solver is associated with exactly one knapsack. Only Solver $s$ knows the entire state of Knapsack $s$ and Solver $s$ is solely responsible for allocating the resources of Knapsack $s$. The state information of a knapsack, such as resources used or available, is completely private to that knapsack and its solver, unless explicitly communicated to another solver by messaging.
- There are $n$ groups of items. The $i$th group has $l_i$ items. The $j$th item of the $i$th group requires $r_{ijk}$ of the $k$th resource. Each solver knows which resource is served by which knapsack. The value of the $j$th item of the $i$th group is $v_{ij}$. $n$ groups are distributed among $M$ solvers. The number of groups in Solvers 1, 2, ...,..., $M$ are $n_1$, $n_2$, ...,$n_s$, ..., $n_M$ respectively. The resource consumptions and associated values of the items of the $n_s$ local groups of Solver $s$ will not be advertised fully to all the solvers. The *partial resource consumption* of an item for a knapsack is defined by the resource requirement of the item from that knapsack. Thus, partial resource consumption of the $j$th item of the $i$th group for the resources of Knapsack $s$ is expressed by the vector $(r_{ij\mu_s}, r_{ij(\mu_s+1)}, \cdots, r_{ij(\mu_s+m_s-1)})$. The partial resource consumption of each item for any knapsack is sent to its associated solver. The set of $M$ solvers will jointly execute a suitable distributed algorithm to pick exactly one item from each group, so that the

total value of the picked items for the entire set of solvers is maximized subject to the resource constraints of each knapsack.

In mathematical notation, the MMMKP can be described as follows.

Maximize $V = \sum_{i=1}^{n}\sum_{j=1}^{l_i} x_{ij}v_{ij}$ , total earned value from the picked items of the groups of all servers

such that, $\sum_{i=1}^{n}\sum_{j=1}^{l_i} x_{ij}r_{ijk} \le R_k$, and $\sum_{j=1}^{l_i} x_{ij} = 1$ .

Where,

- $k = \mu_1, \mu_1+1, \ldots, \mu_1+m_1-1, \ldots, \mu_s, \mu_s+1, \ldots, \mu_s+m_s-1, \ldots\ldots, \mu_M, \mu_M+1, \ldots, \mu_M+m_M-1$
- $x_{ij} \in \{0,1\}$ , the picking variables
- $i = 1, 2 \ldots n; j = 1, 2 \ldots l_i$.

For our example in Figure 5.8 we can express the problem as follows:

Maximize $V = \sum_{i=1}^{4}\sum_{j=1}^{l_i} x_{ij}v_{ij}$ , subject to $\sum_{i=1}^{4}\sum_{j=1}^{l_i} x_{ij}r_{ij1} \le R_1 = 23$, and $\sum_{i=1}^{4}\sum_{j=1}^{l_i} x_{ij}r_{ij2} \le R_2 = 16$

## 5.4 Chapter Summary

In this chapter we mapped the problem of admission control and QoS adaptation based on the UM-D to the variants of Knapsack Problem. We introduce MMMKP, a new variant of Knapsack Problem to formulize the admission control and QoS adaptation problem for fully distributed controllers. We need to run heuristics with lower time requirements to do on line admission control and QoS adaptation in the DMSS. We have already described the heuristics for solving the MMKP in Chapter 3. The next chapter presents heuristics to solve the MMMKP.

# 6. Heuristics for Solving the MMMKP for Admission Control and QoS Adaptation

In this chapter we describe heuristics for solving the MMMKP. Here we present two new distributed heuristics to solve the MMMKP. D-HEU, the distributed version of I-HEU, requires a large number of messages. A-HEU, the arbitrated heuristic, determines the solution of the MMMKP by arbitrating among the solvers with a lower number of messages. But the total value of the items picked by A-HEU is often less than that of D-HEU.

## 6.1 Distributed Incremental Heuristic Solution (D-HEU) of the MMMKP

The solvers of the MMMKP are connected to each other through network connections as shown in Figure 6.1. We have the following assumptions about the MMMKP solvers:

- The network connections are reliable.
- The solver machines are running compatible software including the operating system.

In D-HEU, each solver in the MMMKP runs the same steps of I-HEU, the incremental heuristic for solving the MMKP. The decision to upgrade or downgrade in I-HEU is based on the change in aggregate resource consumption and the feasibility of the items in the groups. As resources are distributed among the knapsacks in the MMMKP, the computation of aggregate resource consumption and feasibility is distributed among the solvers in D-HEU. This algorithm always produces the same result as I-HEU applied to an equivalent MMKP, if there are no floating point errors in the calculation of aggregate resource. Please see Lemma 8.1 later in this chapter for a proof.

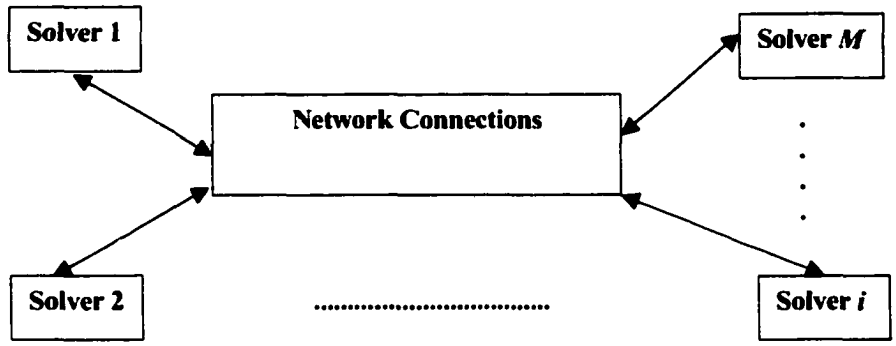


*Figure 6.1 Architecture of solvers in the MMMKP*

## 6.2 The Computations to be Distributed

In the presentation of D-HEU we use the following symbols that have already been introduced in previous chapters.

$C_k$ = the amount of the $k$th resource consumed by already picked items

$R_k$ = the total amount of the $k$th resource.

$k$ is the index of the resources. In an MMKP the index of $m$ resources are 1, 2, ..., $m$. In an MMMKP the indexes of $m_s$ resources of the $s$th knapsack are $\mu_s, \mu_s + 1 \cdots\cdots, \mu_s + m_s - 1$.

$\rho[i]$ =index of the selected item of the $i$th group.

$r_{ijk}$ = the consumption of the $k$th resource of the $j$th item of the $i$th group.

$v_{ij}$ = The value of the $j$th item of the $i$th group.

Aggregate resource consumption, $a_{ij} = \vec{r}_{ij}.\vec{C}$ = the scalar product of the resource requirement vector of an item and the total consumed resource vector of the knapsack.

The following subsections present how the computations performed by I-HEU are distributed in D-HEU to solve the MMMKP.

(TF) for an upgrade to the *j*th item of the *i*th group can be determined by the following expressions

$$TCAR_{ij} = \sum_{l=1}^{M} \Delta pa_{ijs} \text{ and } TF_{ij} = \prod_{l=1}^{M} pf_{ijs}$$

### 6.2.3 Finding an Item of a Group for Upgrading in Step 1 and Step 2

In Step 1, each solver can determine the item which gives the highest positive $TCAR_{ij}$ with $TF_{ij}$ =1 among all the items of its local groups. This is called a *local candidate for upgrading*. The item, which gives the highest $TCAR_{ij}$ with $TF_{ij}$ =1 globally among all the solvers can be determined after each solver notifies the others of its candidate for upgrading. We define it as the *global candidate for upgrading*. Each solver determines the global candidate for upgrade and updates the selection $\rho[]$ of the corresponding group. There is therefore replication of this computation in the solvers. In Step 2, the feasible items yielding the highest $\Delta v_{ij}/TCAR_{ij}$ can be similarly determined locally and globally by the solvers through checking TCARs and TFs. An alternative approach of finding local and global candidate might be to exchange complete resource states among servers. However, this would violate the requirement for information hiding – keeping resource data private to each solver - which we have imposed in the definition of the MMMKP.

### 6.2.4 Finding $\Delta a'_{ij}$ and $\Delta a^*_{ij}$ in Step 3

The partial and total ratio of increased resource requirement to available resource can be calculated by Solver *s* by the following formula

$$\Delta pa'_{ijs} = \sum_{k=\mu_s}^{\mu_s+m_s-1} \frac{r_{i\rho[i]k} - r_{ijk}}{R_k - C_k} \text{ and } TCAR'_{ij} = \sum_{s=1}^{M} \Delta pa'_{ijs}$$

These ratios are called *scaled PCAR* and *scaled TCAR for infeasible upgrade* , as this is the basis of finding a candidate item for an infeasible upgrade.

Similarly the partial and total ratio of decreased resource requirement to over-consumed resource are defined as follows:

$$\Delta pa''_{ijs} = \sum_{k=\mu_s}^{\mu_s+m_s-1} \frac{r_{ip[i]k} - r_{ijk}}{C_k - R_k} \text{ and } TCAR''_{ij} = \sum_{s=1}^{M} \Delta pa''_{ijs}$$

These ratios are termed *scaled PCAR* and *scaled TCAR for downgrade*, as this is the basis of finding a candidate item to downgrade in Step 3. PFs and TFs are also calculated in Step 3 to determine the feasibility of downgrades.

### 6.2.5 Finding an Item of a Group for Upgrading or Downgrading in Step 3

We can apply the same technique as in Steps 1 and 2 to find a *local* and *global candidate for upgrading or downgrading* in Step 3.

***Lemma 8.1*** *Let the total value of the items picked by D-HEU from an MMMKP be* $V_{D\text{-}HEU}$. *By merging the knapsacks this MMMKP can be transformed to an MMKP. Let the total value of the items picked by I-HEU from this MMKP be* $V_{I\text{-}HEU}$ . *If there is no precision error in floating point calculations then identical items are picked by D-HEU and I-HEU i.e.,* $V_{I\text{-}HEU} = V_{D\text{-}HEU}$.

***Proof.***

Due to precision error, two different results may be obtained by adding the same set of floating point numbers in two different orders. That is, $(x_1 + x_2 + x_3 + x_4)$ is not always equal to $(x_1 + x_2) + (x_3 + x_4)$. But, if we assume that there is no floating point precision error then $(x_1 + x_2 + x_3 + x_4) = (x_1 + x_2) + (x_3 + x_4)$. i.e., the order of addition will not affect the final result.

Now, if there is no floating point precision error in the computation of D-HEU and I-HEU we can derive the following expression:

$$\Delta a_{ij} = \sum_{k=\mu_1}^{\mu_M+m_{M_s}-1} (r_{ip[i]k} - r_{ijk}) \times C_k$$

$$= \sum_{k=\mu_1}^{\mu_1+m_1-1}\left(r_{i\rho[i]k} - r_{ijk}\right)\times C_k + \sum_{k=\mu_2}^{\mu_2+m_2-1}\left(r_{i\rho[i]k} - r_{ijk}\right)\times C_k + \ldots\ldots\ldots + \sum_{k=\mu_M}^{\mu_M+m_M-1}\left(r_{i\rho[i]k} - r_{ijk}\right)\times C_k$$

$$= \sum_{l=1}^{M} \Delta pa_{ijs} = TCAR_{ij}$$

As the feasibility is an integer, there will be no precision error in the calculation of partial and total feasibility. So, $f_{ij} = \prod_{k=\mu_1}^{\mu_M+m_M-1} pos\left(C_k + r_{i\rho[i]k} - r_{ijk}\right) = \prod_{l=1}^{M} pf_{ijs} = TF_{ij}$

Similarly, it can be shown that $\Delta a'_{ij} = TCAR'_{ij}$ and $\Delta a''_{ij} = TCAR''_{ij}$ when there is no floating-point precision error.

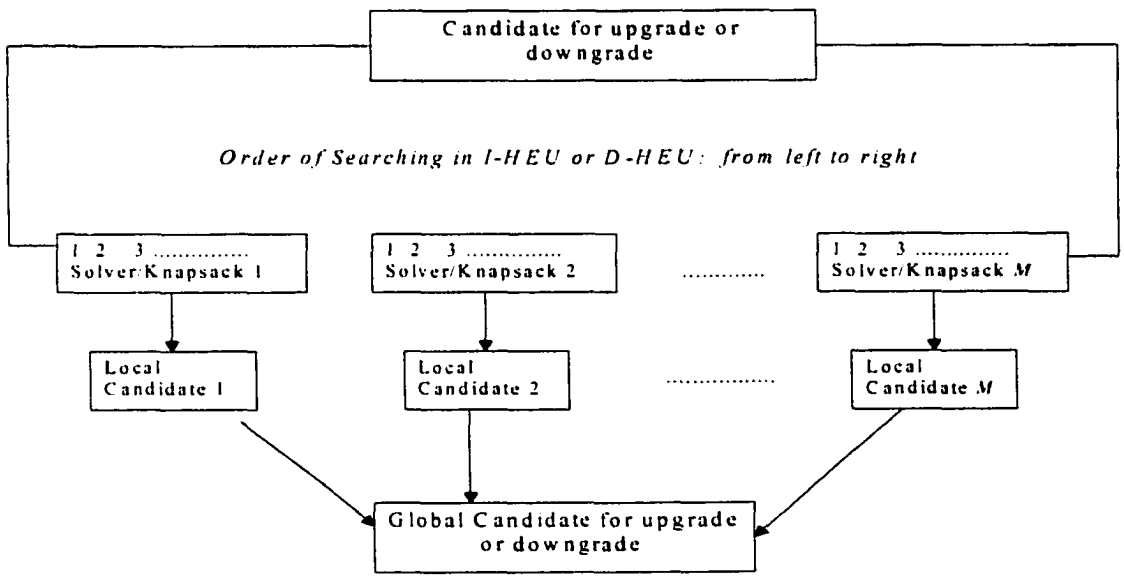


*Figure 6.2 Order of searching for candidate items in I-HEU and D-HEU*

Both I-HEU and D-HEU find candidate items with the maximum change of value per change of aggregate resource in Step 2. The change of value per unit change of aggregate resource can be expressed by $\Delta v_{ij}/\Delta a_{ij}$ in I-HEU and by $\Delta v_{ij}/ TCAR_{ij}$ in D-HEU. These values are equal if there is no floating-point precision error. Hence, the candidate items in I-HEU and D-HEU are the same if the candidate item search is done in the same order. Otherwise, there is a chance of selecting two different items with the same $\Delta v_{ij}/\Delta a_{ij}$ and

$\Delta v_{ij}/ TCAR_{ij}$. In I-HEU, groups are indexed by the knapsack number and group number. In D-HEU, groups are indexed by group number while finding local candidates, and local candidates are indexed by knapsack number while finding global candidates. Figure 6.2 shows this order. Thus I-HEU and D-HEU will generate identical candidates for upgrades or downgrades in Step 2. Similarly, we can show that the other steps of D-HEU and I-HEU pick identical items resulting in equal total values.

## 6.3 Example of an Upgrade by D-HEU

The solution of an MMMKP can be defined by the vector $(\rho[1], \rho[2], \cdots\cdots\cdots, \rho[n])$, where $\rho[i]$ is the index of the selected item of the $i$th group. The initial solution of the MMMKP shown in Figure 6.3 is (0, 0, 0, 0). Let the solution after one or more upgrades be (0, 1, 1, 0) as shown in Figure 6.4. The boxes with doubled borderlines represent the selected items. Following are the steps to do the next upgrade in Step 2.

- Each solver will find PCARs and PFs for each higher valued item in the knapsacks as shown in Figure 6.5. All upgrades are feasible here.
- Solver 1 will send PCARs and PFs of Groups 3 and 4 to Solver 2. Solver 2 will send PCARs and PFs of group 1 and 2 to Solver 1.
- Solvers 1 and 2 will calculate TCARs and TFs as shown in Figure 6.6.
- Solvers 1 and 2 find the local candidates represented by the boxes with thick borderlines in Figure 6.7. They exchange their local candidates.
- All TCARs are negative. So, there is no item giving better revenue with less aggregate resource consumption. Item 3 of Group 1 is therefore selected as global candidate for upgrade, as the change of value per unit TCAR is maximum for this item.

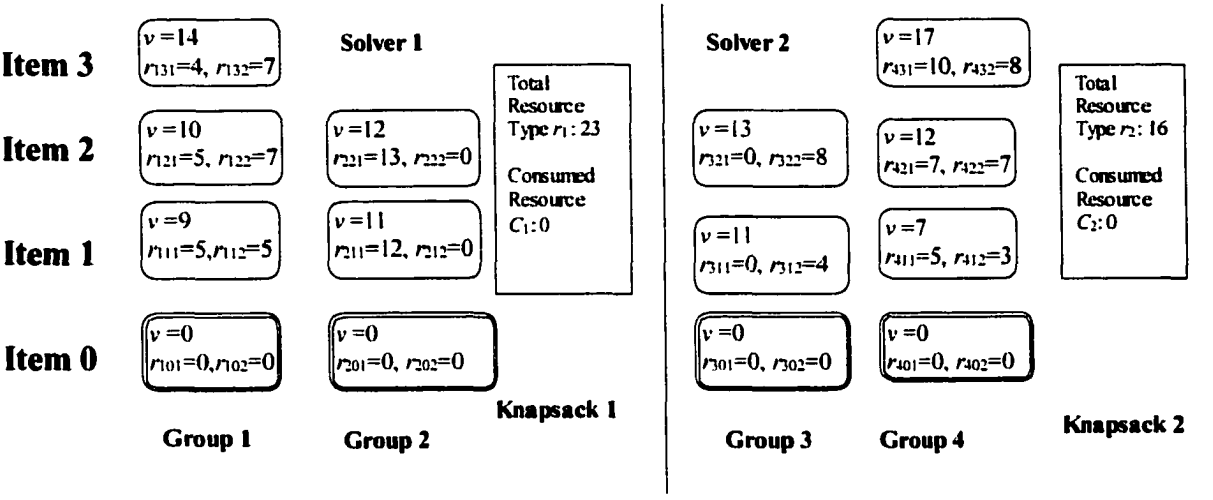


*Figure 6.3 An MMMKP with two knapsacks*

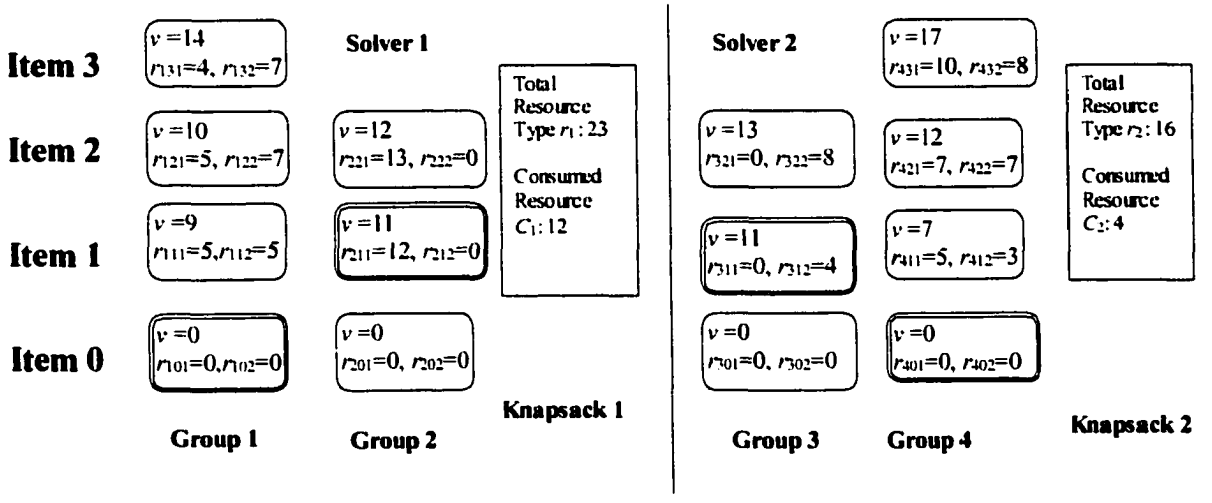


*Figure 6.4 An intermediate solution during D-HEU execution*

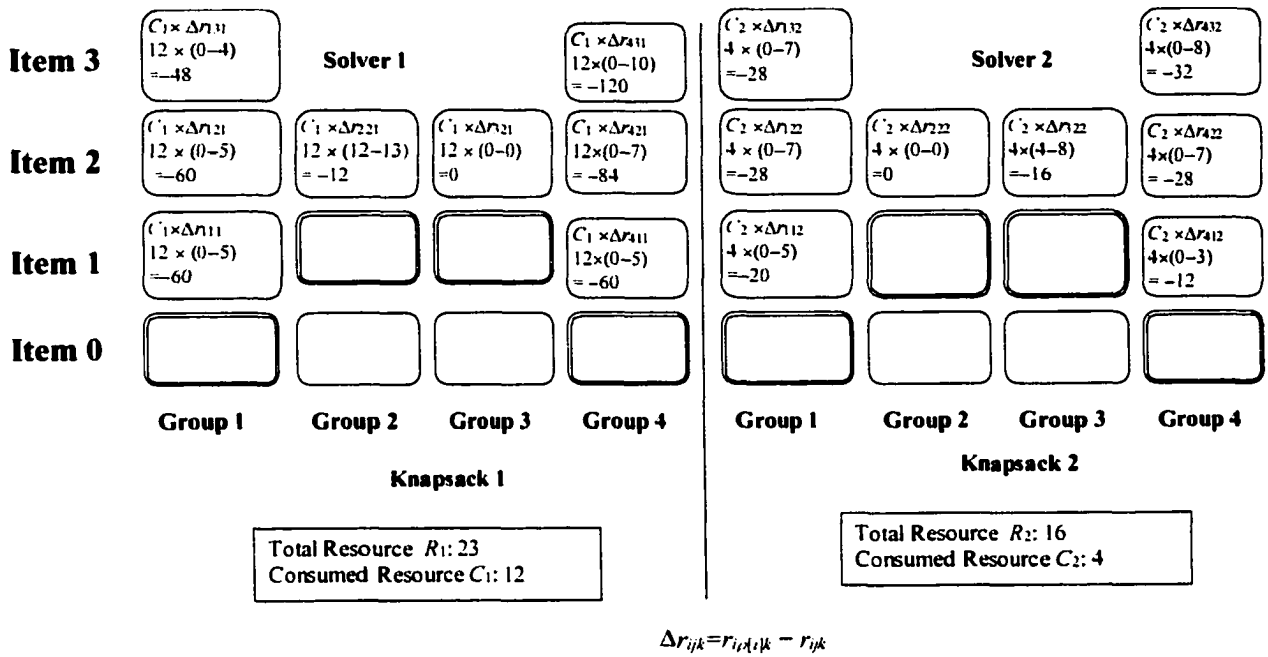


*Figure 6.5 Calculation of PCARs and PFs*

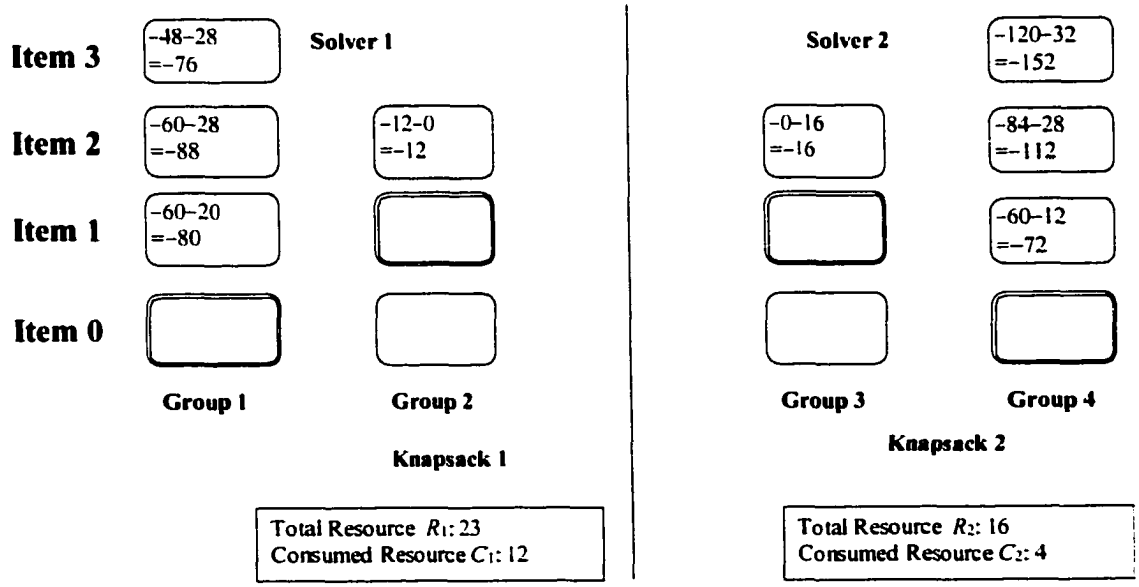


*Figure 6.6 Calculation of TCARs and TFs*

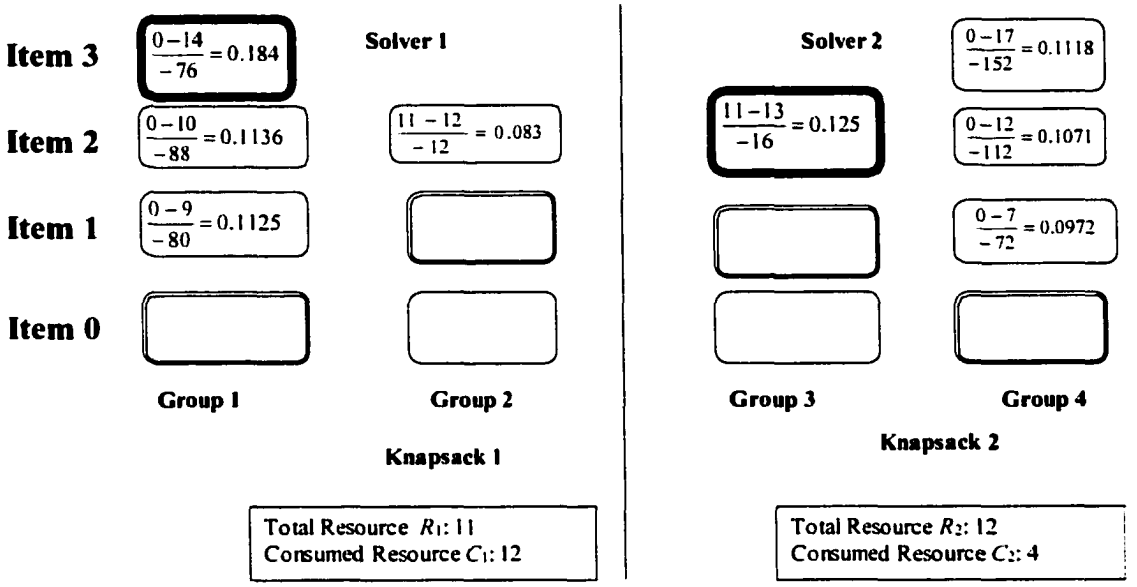


*Figure 6.7 Calculation of change of value per total change of aggregate resource ($\Delta v_{ij}$ / TCAR) and finding the local candidate items*

## 6.4 Description of D-HEU

There must be $M$ threads running in each solver of the MMMKP. There are two types of threads:

*Sender Thread*: There is only one thread of this type in each solver; it is used to send messages to the other solvers. This thread also does major computations such as calculating partial and total change of aggregate resources, and finding local and global candidates for upgrades or downgrades sequentially.

*Receiver Threads*: There are $(M-1)$ receiver threads corresponding to the $(M-1)$ other solvers of the MMMKP. The main jobs of these threads are to receive messages sent by other solvers and update the local state to reflect change of aggregate resources and local candidates for upgrades or downgrades.

### 6.4.1 Format of the Messages

All the messages start with the message identifier followed by the message body with predefined structures. The Required messages are described briefly in the following table:

*Table 6.1 Different messages used by D-HEU*

| Message Type | Description of the structure | Monitor counter associated with the message |
|---|---|---|
| Groups | This is a list of groups. Each group is defined by<br>(solver number, group number, number of items, partial resource requirements for the items) | *no_of_groups_msgs* |
| PCARs & PFs | This is a list of PCARs and PFs of the local groups. PCARs and PFs of a group is defined by the following vector:<br>(solver number, group number, number of items, PCARs and PFs of the items) | *no_of_pcars&pfs_msgs* |
| Local Candidate | The following vector is used to indicate all kinds of local candidates for downgrades and upgrades.<br>(solver number, group number, item number, change of value per TCAR, feasibility of the candidate) | *no_of_local_candidate_msgs* |
| Local Total Value | The vector (solver number, total value) indicates the total value of the items picked by a solver. | *no_of_local_total_value_msgs* |
| Local Most Infeasible Resource | It contains the following vector<br>(solver number, resource number, infeasibility factor) | *no_of_local_most_inf_res_msgs* |

We need the monitor counters to synchronize the sequences of the heuristics among the solvers. The algorithm can proceed to the next step after it gets the confirmation that it has received all messages from the other solvers. All the actions on these counters must be atomic, as shared access between receiver and sender threads might cause a deadlock.

### 6.4.2 Sequence of Events to Find a Global Candidate

Figure 6.8 shows the flow chart of the processes and events in D-HEU that take place while finding the global candidate for an upgrade or downgrade. Each solver allocates separate variables for the other solver's local candidates and PCAR & PFs of the items of

local groups. Variables for local candidate and an item's PCAR & PF from another solver are updated by only one receiver thread or sender thread. So, the actions 'saving PCARs & PFs' and 'saving local candidate' executed by the receiver threads need not be atomic or synchronized. The decision blocks in the flow chart check counters shared by all threads, so these decision blocks must be synchronized. Please see Section 10.1 in the appendix for the detailed pseudo code of D-HEU.

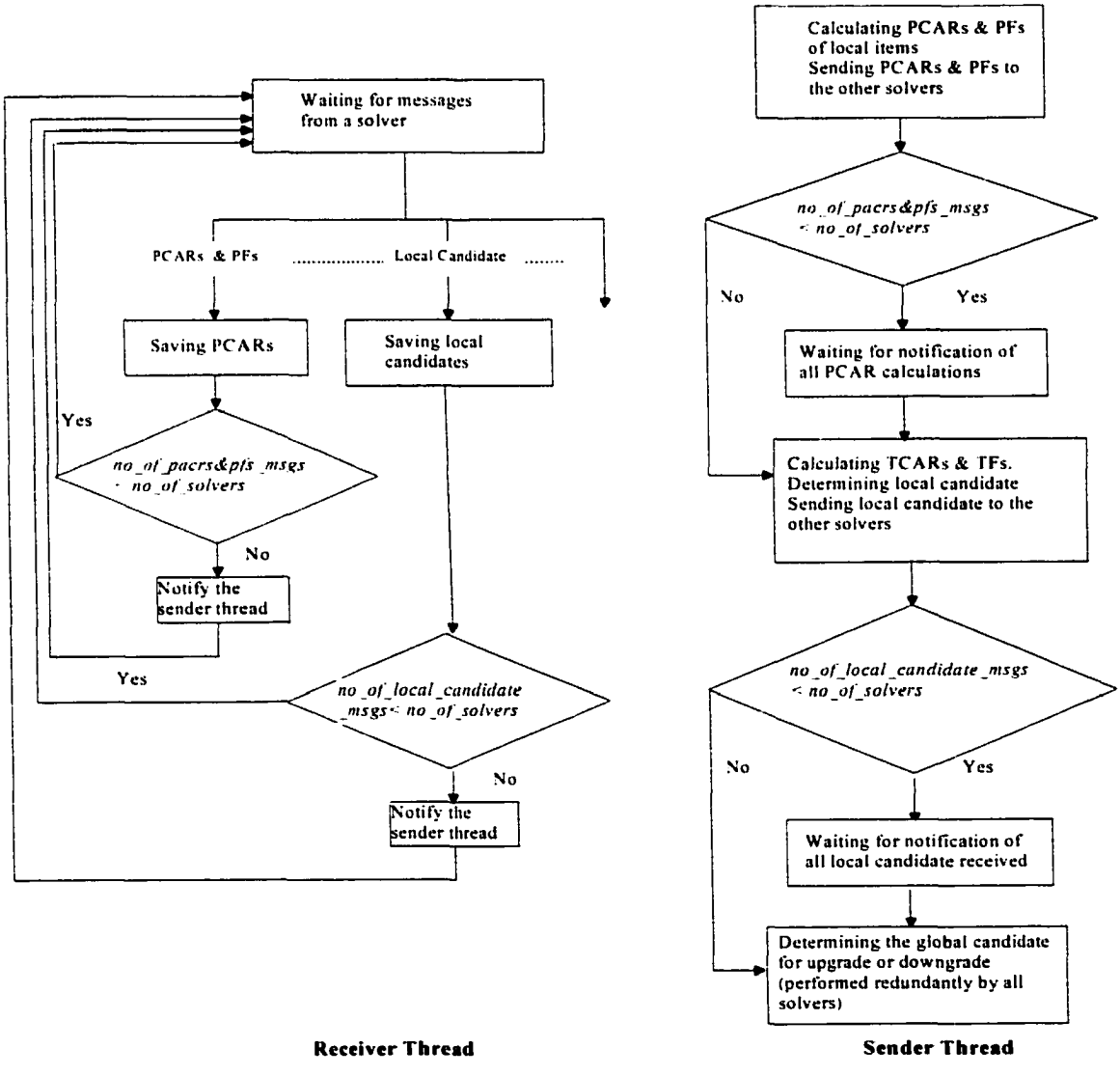


*Figure 6.8 Flow chart of distributed computation of global candidate.*

## 6.5 Complexity Analysis of D-HEU

For convenience of analysis we make the following assumptions:

- Each knapsack has the same resource dimension. That is, $m_1 = m_2 = \cdots = m_M = m/M$.
- Each solver is assigned the same number of groups. That is, $n_1 = n_2 = \cdots = n_M = n/M$.
- Each group of the MMMKP has the same number of items. That is, $l_1 = l_2 = \cdots = l_n = l$.

We note that the computation and message passing techniques of D-HEU in Step 1 and Step 2 are almost identical. Hence the orders of the complexities will also be the same for these two steps. Here we present the analysis of Step 2. In this step we need $n(l-1)$ upgrades in the worst case. The $i$th upgrade requires the following messages and computations by one solver:

$\{n(l-1)-i+1\}\times\frac{6m}{M}$ floating point operations to determine PCARs & PFs.

$(M-1)$ messages to send PCAR&PF to others.

$\{n(l-1)-i+1\}\times M$ floating point operations to determine TCARs & TFs and the local candidate for upgrade.

$(M-1)$ messages to send the local candidate for upgrade to other solvers.

$M$ floating point operations to determine global candidate for upgrade.

The total number of floating point operations performed in Step 1 or 2 by each solver is

$$\sum_{i}^{n(l-1)}\left\{\{n(l-1)-i+1\}\times\frac{6m}{M}+\{n(l-1)-i+1\}\times M+M\right\} \cong n^2(l-1)^2\left(\frac{6m}{M}+M+1\right)$$

The total number of messages sent in Step 1 or 2 by each solver is $2n(l-1)(M-1)$

In Step 3 we might have $(n-1)(l-1)$ downgrades in the worst case after one infeasible upgrade to escape from a possible local maximum. A downgrade or an infeasible upgrade in this step requires the same number of messages as a feasible upgrade in Step 2. For an infeasible upgrade we need $\frac{4mn(l-1)}{M}$ floating point operations to determine the scaled PCARs & PFs. For each downgrade we need $\frac{7mn(l-1)}{M}$ floating point operations to determine scaled PCARs & PFs. The number of floating point computations required for calculating TCARs & TFs, local and global candidate for each infeasible upgrade are the same as a feasible upgrade.

Thus, the total number of floating points operations required in Step 3 by each solver to escape from local maxima is approximately $(n-1)(l-1)\times\left\{\frac{7mn(l-1)}{M}\right\}+\frac{4mn(l-1)}{M}$

$\cong (n-1)^2(l-1)^2\times\frac{7m}{M}$

The total number of messages required in Step 3 by each solver to escape from local maxima is $\{(n-1)(l-1)+1\}\times 2(M-1)\cong 2(n-1)(l-1)(M-1)$

Thus the message passing complexity by D-HEU is $O(Mnl)$. This is not practically feasible for real time decision making distributed system such as a distributed SLA controller or distributed multimedia server. For example, a fully distributed multimedia server system with 3 servers (one UME in each server) might have 5000 multimedia session requests, 3 QoS levels in each request. Hence the number of messages for admission control in the worst case is 39992. If each message takes 10 ms then time required for message passing in the worst-case is 400 sec, which is *much* higher than the usual required response time of a real time system.

## 6.6 Arbitrated HEU (A-HEU) for Solving the MMMKP

This method of solving MMMKP requires many fewer messages than the D-HEU algorithm; its message passing complexity is $O(M)$. The solver in each knapsack runs I-HEU independently. The candidate for upgrades and downgrades are calculated based on the value of PCAR. Hence, as a simplifying assumption, the resources in other knapsacks are completely ignored in this calculation. To find a feasible solution we first run Step 1 of I-HEU in each solver. If each solver finds a feasible solution and each solver satisfies the resource constraint of all the selected items from each group, then we find a feasible solution. Now, to find upgrades and downgrades, each solver sends its proposed selection of items, calculated by running Step 2 and 3 of I-HEU, to the other solvers. The proposed selection is sorted according to change of value per PCAR. The items in this sorted list, which satisfy the resource constraints of all the knapsacks, can be selected. Thus arbitration among the solvers is required to select items from the groups.

The same procedure can then be repeated until we run out of real time, to attempt to obtain better total values. Here we present an arbitration technique to select the items which requires only $O(M)$ message complexity. The following example shows only one arbitration step of A-HEU.

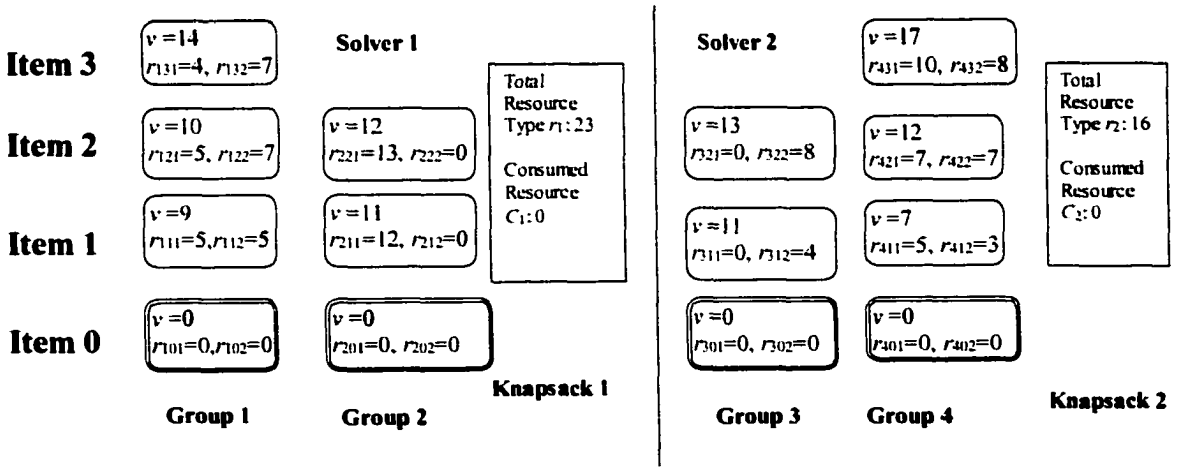


*Figure 6.9 An MMMKP with two knapsacks. (Reproduction of Figure 6.3)*

*Table 6.2 Items picked by Solver 1 and 2 by running I-HEU independently*

| Groups | Picked Items | Change of value per PCAR | Resource consumption |
|---|---|---|---|
| Group 1 | 3 | 3.5 | Resource consumption in Solver 1: $\sum r_1 = 17$ |
| Group 2 | 2 | 0.923 | |
| Group 3 | 2 | 1.625 | Resource consumption in Solver 2: $\sum r_2 = 16$ |
| Group 4 | 3 | 2.125 | |

If we run I-HEU in both the solvers of Figure 6.9 independently, the solution can be shown in Table 6.2. As the lowest valued items are all zero we need not find a feasible solution. Picking the $j$th item of the $i$th group can be expressed by $(i, j)$. The proposed lists by Solvers 1 and 2 are $\{(1,3),(2,2)\}$ and $\{(3,2),(4,3)\}$ respectively. These lists are exchanged between the solvers. The sorted global list after merging these proposed lists is $\{(1,3),(4,3),(3,2),(2,2)\}$. Now the feasible picks by Solver 1 are $\{(1,3),(4,3),(3,2)\}$ with $\sum r_1 = 14$. Similarly the feasible picks by Solver 2 are $\{(1,3),(4,3)\}$ with $\sum r_2 = 15$. So Solver 1 and 2 can satisfy the first 3 and 2 picks respectively from the proposed sorted list of selected items. They exchange this information and take the set intersection of their possible solutions; namely, they pick the first 2 items from the proposed list. Hence the solution after the first arbitration is $\{(1,3),(4,3)\}$ with $\sum r_1 = 14$, $\sum r_2 = 15$ and $V = 31$.

### 6.6.1 Format of the Messages

A-HEU requires messages "Groups" and "Local Total Value". These are already defined in the description of D-HEU. The other messages are listed as follows.

*Table 6.3 New messages required by A-HEU*

| Message Type | Description of the structure | Monitor counter associated with the message |
|---|---|---|
| Proposed Selected Item List | The following vector is used to define a proposed selected item<br><br>(solver number, group number, item number, value per PCAR) | *no_of_local_proposed_list_msgs* |
| Local Feasibility Index | The vector ($s$, $L$) indicates that the first $L$ items from the beginning of the global proposed selected item list are feasible with respect to the resources in Solver $s$. | *no_of_local_feasibilty_msgs* |
| Solution Not Found | This message indicates that a solver could not find any solution while determining proposed selected items for feasible solution. | Not applicable because the solver terminates if this message is received. |

### 6.6.2 Sequence of Events in A-HEU

Figure 6.10 shows the flow chart of the processes and events in A-HEU during each arbitration. We allocate separate variables for each solver's list of proposed selected items and local feasibility index. Thus, the actions 'saving proposed selected items' and 'saving local feasibility index' executed by the receiver threads need not to be atomic or synchronized. The decision blocks in the flow chart check counters shared by all threads,

so, these decision blocks must be synchronized. Please see Section 10.2 in the appendix for the detailed pseudo code of A-HEU.

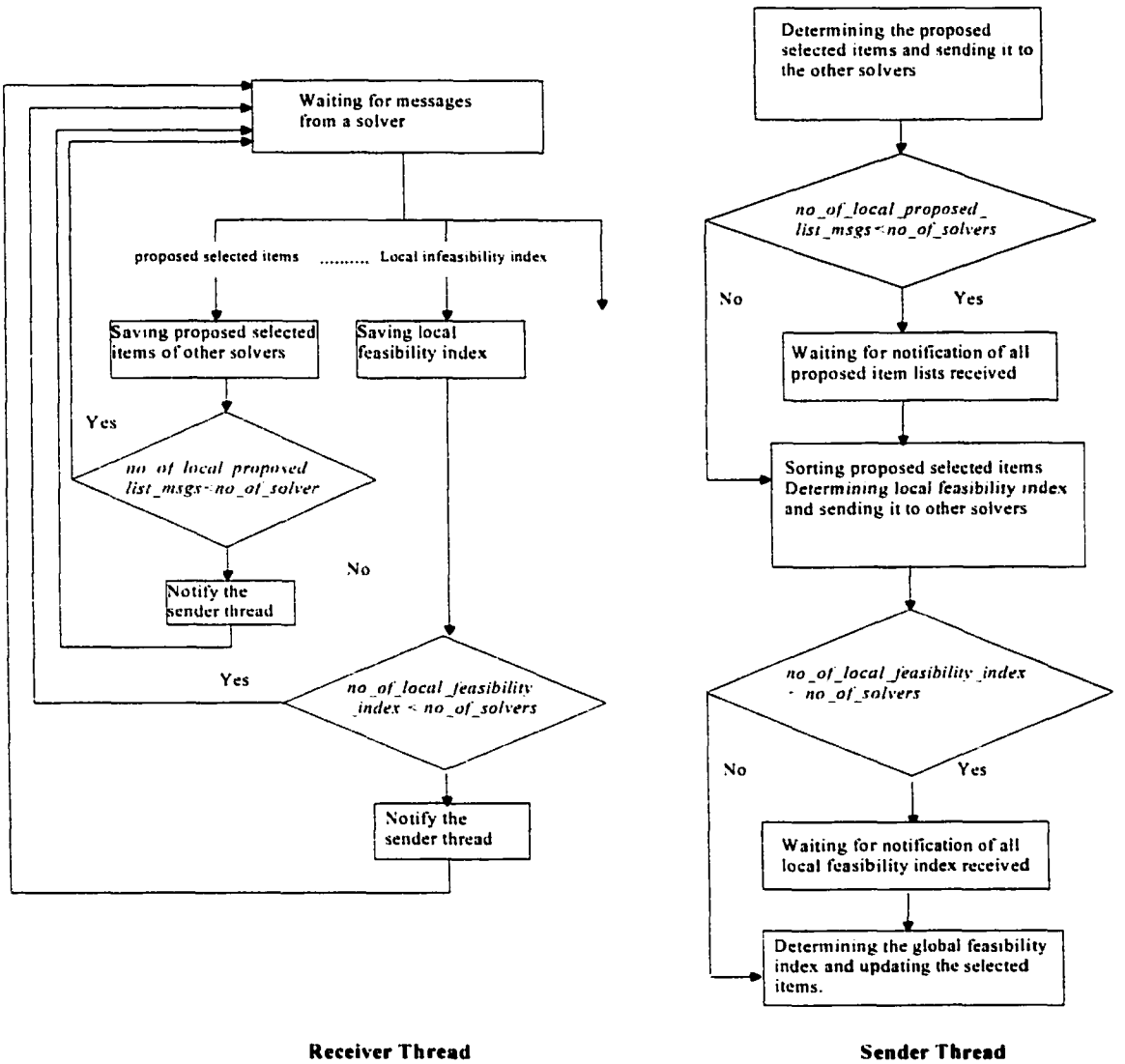


*Figure 6.10 Flow chart of distributed computation by A-HEU*

## 6.7 Complexity of A-HEU

Here we present the computational and message passing complexity by one arbitration of A-HEU as the first arbitration yields near optimal total value while the following iterations improve the solution with increased total values.

Computational complexity to run I-HEU on an MMKP with $n/M$ groups and $m/M$ resources is $\frac{3m}{M}\left(\frac{n}{M}\right)^2(l-1)^2$ in Step 2 and $\frac{7m}{M}\left(\frac{n}{M}-1\right)^2(l-1)^2$ to escape from local minima in Step 3.

We require the following messages and computations in each arbitration to find the candidate items for upgrading or downgrading:

$2(M-1)$ messages to send local proposed list of selected items.

$\frac{n(n-1)}{2}$ floating point operations to sort the locally proposed lists to determine the global proposed list.

$2\left(\frac{nm}{M}+M\right)$ comparisons to find the local and global feasibility index

Thus computation and message passing complexities in each solver are $O\left(n^2(l-1)^2\frac{m}{M^3}\right)$ and $O(M)$ respectively.

## 6.8 Experimental Results

In order to study the run-time performance of distributed algorithms to solve the MMMKP we implemented D-HEU and A-HEU along with I-HEU using Java. For simplicity of the implementation, we assume:

- Each group has the same number of items i.e., $l_1 = l_2 = \cdots, \cdots = l_n = 5$

- Each knapsack has the same number of resources i.e., $m_1 = m_2 = \cdots, \cdots = m_M = m = 4$.
- Each solver has the same number of groups i.e., $n_1 = n_2 = \cdots, \cdots = n_M = n_c$.

The algorithms were tested for an MMMKP with 3 knapsacks. Three different machines were used as three different solvers and a fourth machine was used as a generator of the MMMKP. The generator generates the groups of the MMMKP and sends them to the solvers. The generator machine also runs I-HEU on the transformed MMKP from the generated MMMKP.

*Table 6.4 Specifications of the solvers and generator of the MMMKP.*

| Machine name | Machine type | CPU speed | RAM | O/S | JDK Versions |
|---|---|---|---|---|---|
| Solver 1 | IBM PC compatible | 750 MHz | 256 MB | Windows 2000 | JDK 1.2.2 |
| Solver 2 | IBM Think Pad | 700 MHz | 192 MB | Windows 2000 | JDK 1.3.1_03 |
| Solver 3 | IBM PC compatible | 750 MHz | 256 MB | Windows 2000 | JDK 1.2.2 |
| Generator | IBM Think Pad | 700 MHz | 192 MB | Windows 2000 | JDK 1.3.1_03 |

### 6.8.1 Test Pattern Generation

The total amount of resources in the knapsacks, resource consumption by the items, and the values associated with the items are initialized as follows.

$R_c$ = Maximum amount of a resource consumption by an item

$P_c$ = Maximum cost per unit resource

$R_i$ = Total amount of the $i$th resource = $n_c \times M \times R_c$.

$P_k$ = Cost of the $k$th resource = $P_c \times$ *Random* (0.0, 1.0)

*Random* (0.0, 1.0) = A uniform continuous random number from 0.0 to 1.0.

Item 0 of each group has zero value with zero resource consumption, i.e., $r_{i0k} = 0.0$ and $v_{i0} = 0.0$. The other items of the groups are initialized by the following random functions:

$r_{ijk}$ = The $k$th resource of the $j$th item of the $i$th group = $R_c \times$ *Random* (0.0, 1.0)

For initializing item values we use the following functions:

$v_{ij}$ = Value of the $j$th item of the $i$th group $= \sum r_{ijk} \times P_k + Random(0.0,1.0) \times \left( m \times M \times \frac{R_c}{10} \times \frac{P_c}{10} \right)$

## 6.8.2 Test Results

The experiment was conducted for different values of $n_c$, from 100 up to 1000. The following data was collected from the experiments and is presented in Table 6.5, Figure 6.11, Figure 6.12 and Figure 6.13:

- Total values of the picked items and time required by D-HEU, A-HEU and I-HEU
- Number of messages required by D-HEU and A-HEU
- Required time, total value of the picked items and number of messages by one, two and three arbitrations of A-HEU

*Table 6.5 Ratio of total value of the items picked by A-HEU with respect to D-HEU. $V^i_{A-HEU}$ indicates the total value of the items picked by the ith arbitration of A-HEU. $V_{A-HEU}$ and $V_{D-HEU}$ indicates the total value of the items picked by A-HEU and D-HEU.*

| Number of groups in each solver | Percentage of total value in the *first* arbitration with respect to D-HEU $\frac{V^1_{A-HEU}}{V_{D-HEU}} \times 100$ | Percentage of total value in the *second* arbitration with respect to D-HEU $\frac{V^2_{A-HEU}}{V_{D-HEU}} \times 100$ | Percentage of total value in the *third* arbitration with respect to D-HEU $\frac{V^3_{A-HEU}}{V_{D-HEU}} \times 100$ | Percentage of total value in A-HEU with respect to D-HEU $\frac{V_{A-HEU}}{V_{D-HEU}} \times 100$ |
|---|---|---|---|---|
| 100 | 92.63 | 93.07 | 93.07 | 93.07 |
| 200 | 90.71 | 91.15 | 91.15 | 91.15 |
| 300 | 92.27 | 92.56 | 92.56 | 92.56 |
| 400 | 91.43 | 91.52 | 91.52 | 91.52 |
| 500 | 91.35 | 91.35 | 91.35 | 91.35 |
| 600 | 92.43 | 92.43 | 92.43 | 92.43 |
| 700 | 74.40 | 87.80 | 87.91 | 88.12 |
| 800 | 92.05 | 92.05 | 92.05 | 92.05 |
| 900 | 91.64 | 91.71 | 91.71 | 91.71 |
| 1000 | 92.57 | 92.77 | 92.80 | 92.80 |

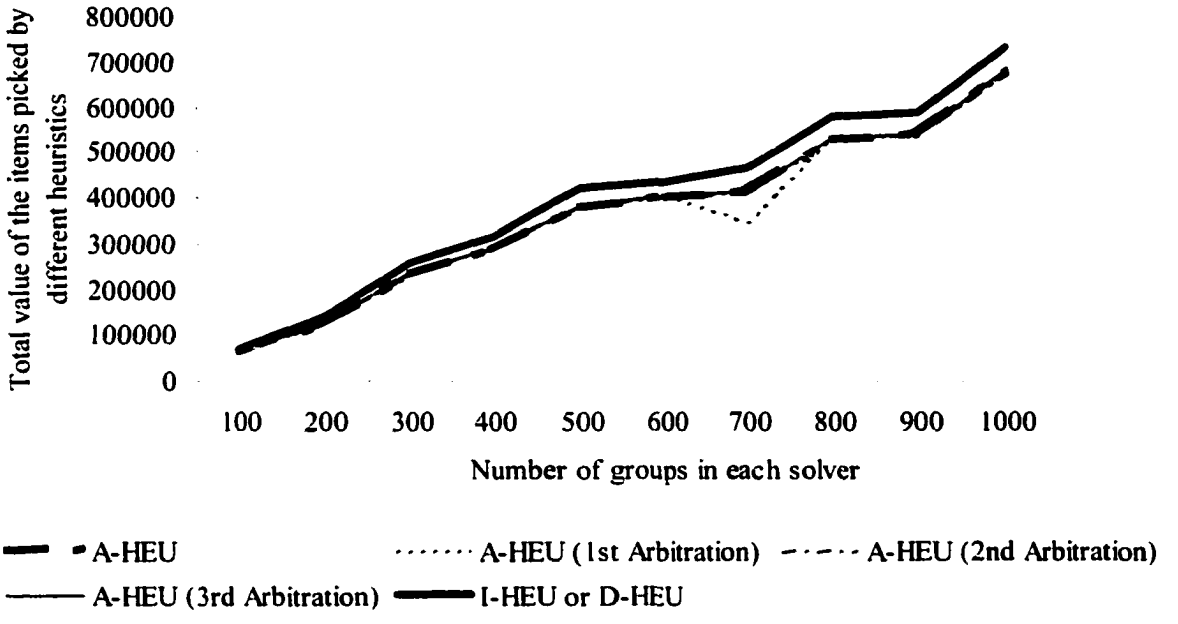


*Figure 6.11 Total value of the items picked by A-HEU, D-HEU and I-HEU*

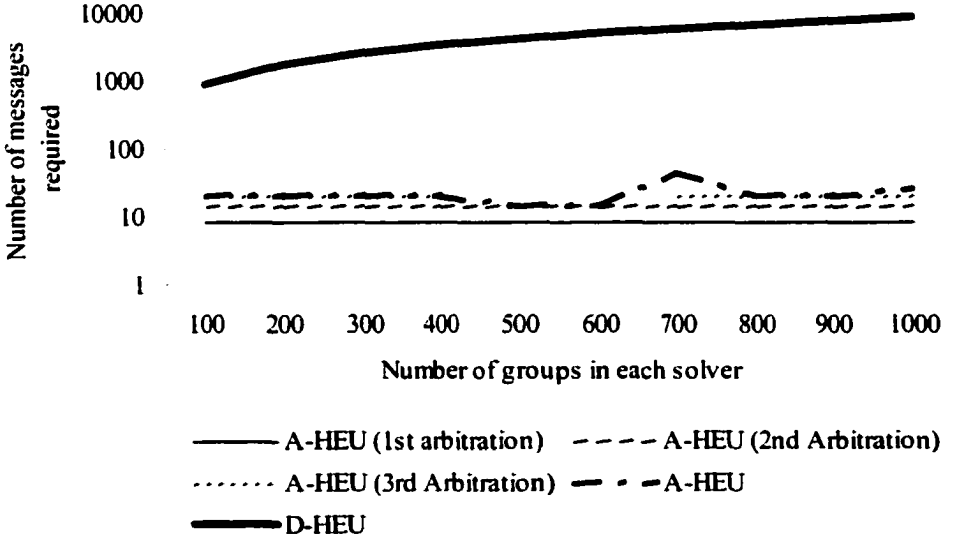


*Figure 6.12 Number of messages required by distributed algorithms to solve the MMMKP.*

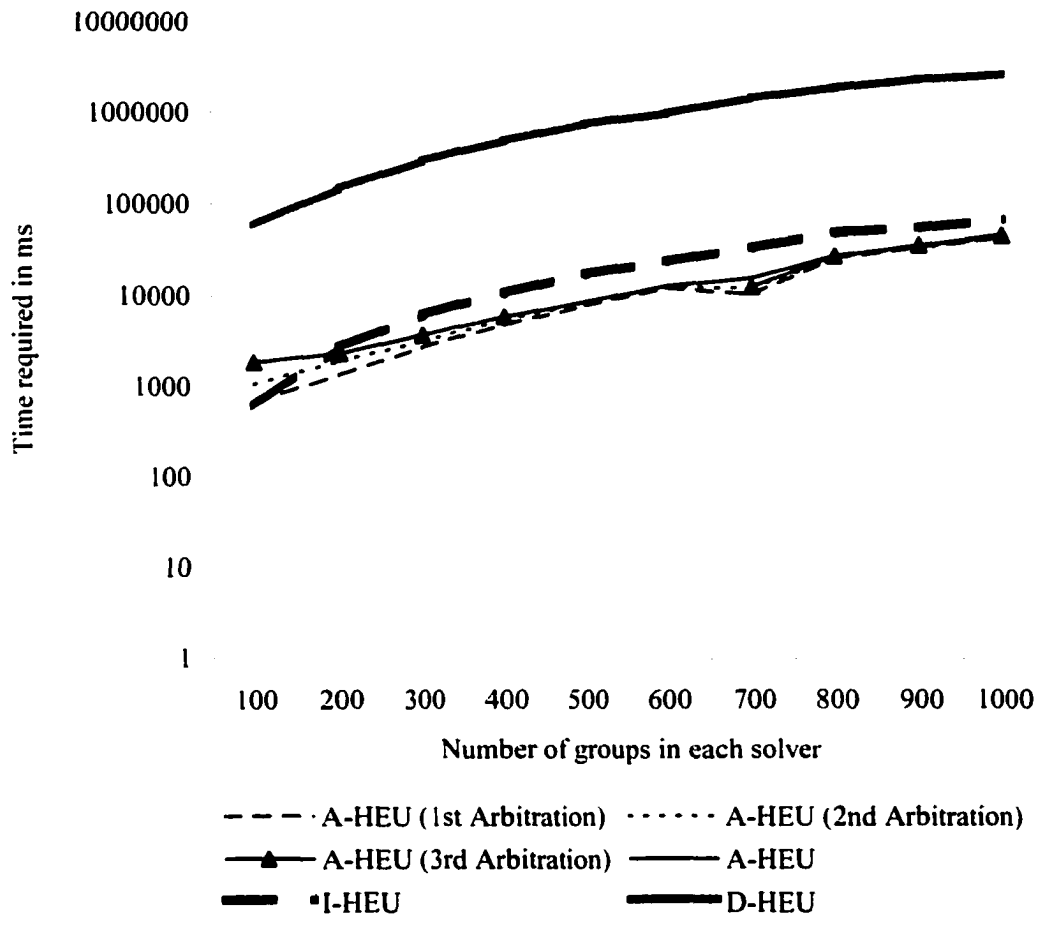


*Figure 6.13 Time required by different algorithms to solve the MMMKP and MMKP*

### 6.8.3 Observations

- The total values of the items picked by D-HEU were exactly the same as total values of the items picked by I-HEU for all generated MMMKP data sets in the experiment. There was no floating-point precision error in the computations of aggregated resource consumptions. This validates code written for D-HEU and I-HEU.
- The time and number of messages required to run D-HEU are much higher compared to A-HEU.

- The total value of the items picked by A-HEU is approximately 90% of the total value of the items picked by D-HEU, but A-HEU requires a negligible number of messages compared to D-HEU.
- A-HEU scales better than centralized I-HEU. Figure 6.13 shows the effect of fixed message passing time on the small MMMKP data sets (100 groups in each solver). In this case A-HEU takes longer than I-HEU. For larger MMMKP data sets (equal to or more than 200 groups in each solver) the effect of message passing time is less than computation time and A-HEU provides a faster response than I-HEU.
- For almost all the MMMKP data sets the total value of the items picked by first arbitration of A-HEU is more than 95% of the total value of the items finally picked by A-HEU.

## 6.9 Discussion of the Performance of A-HEU and D-HEU

- An arbitration in A-HEU requires far fewer messages than does D-HEU. So an iteration of A-HEU can be easily applicable for on line admission control algorithms. The admission control algorithm can execute further arbitration for better values if it has a more relaxed time constraint. Thus this algorithm is very suitable for an online system that requires quick decisions with a sub optimal total value, with the possibility of using more available time to improve the quality of the computation.
- The main reason for sub optimality in A-HEU lies in not considering all resource requirements in the preliminary selection.
- The arbitration technique for finding global feasibility index is another reason for sub optimality. We give up upgrading from the proposed list of selected items if we get one infeasible upgrade in the list. The very next item might be feasible. A

new arbitration technique with $O(nM)$ complexity might do that. However, if we do the arbitration again, with a newly calculated proposed selected item list for better total value, the competent items will get a chance to be selected. That is why we prefer multiple iterations of the arbitration, to a new arbitration technique with $O(nM)$ complexity.

- If the items of the groups consume all the resources of all knapsacks uniformly then A-HEU is unlikely to get better total value in the next arbitration, because a particular resource has already been exhausted in the previous arbitration.
- In practical cases, such as multimedia servers and Enterprise Networks, a QoS level of a session requires resources of a particular server or a particular network link. So there is a chance to allocate available resources to other selected items of the groups in the next arbitration.
- Since A-HEU achieves almost 90% optimality of D-HEU in $1/O(nl)$ of the time required by D-HEU, we do not see any apparent application of the D-HEU in real time systems. But this algorithm can be used for non real time applications where the system is allowed more time to solve the MMMKP for admission control and QoS adaptation. D-HEU is a logical step towards devising A-HEU. The approach used by D-HEU can be easily utilized to devise a parallel algorithm for the MMKP with better scalability. Large amounts of message passing is the main cause of the high time requirement by D-HEU. D-HEU might take less time if we run the heuristic using a switching network with faster message passing techniques. If the solvers are located in the same machine with multiple processors then the delays in the network could be avoided and the processors are used in parallel. We would expect better response time by D-HEU in this case.

## 6.10 Chapter Summary

We presented two distributed algorithms D-HEU and A-HEU for solving the MMMKP. D-HEU earns better revenue than A-HEU, but it requires a larger number of messages. That is why it is impractical to use D-HEU in real time admission controllers. On the other hand, A-HEU is scalable and it yields acceptable performance in earning revenues. We can also adjust the number of A-HEU arbitrations to achieve better scalability. A-HEU is thus applicable in real time admission controllers. We will apply these two algorithms in admission control and QoS adaptation of SLAs in Distributed SLA Controllers for the interconnected ENs in Chapter 8. Applications of the UM-D, Part III of the dissertation, starts from the next chapter where we apply the UM-D by a centralized broker to the content routing by a set of Media Server Farms.

# Part III: Applications

# 7. Application of UM-D to Optimal Server Selection for Content Routing

This chapter demonstrates that the Utility Model – Distributed (UM-D) can be applied successfully to the problem of content routing for multimedia traffic [84][85]. Multimedia content, such as video streams, audio streams and images, is generally provided at more than one server, at more than one location, to improve scalability and fault tolerance. The problem, then, is to select the optimal or a nearly-optimal server, from the set which hold copies of a particular item of content, to deliver that content to a particular customer, in order to maximize the revenue earned from the customers while fully respecting all QoS guarantees. Mechanisms for *content routing*, as this procedure is called, have been given by Johnson [83] and Barbir [86]. However, to our knowledge no one has preciously presented a policy - an algorithm to select the optimal server to deliver a given item to a given customer - to drive content routing mechanisms.

As the emphasis in this chapter is on servers and customers, the network which interconnects them is incidental and of little interest. Hence we will use the simplest architecture - the Broker architecture - for solving the UM-D in this work. We will assign large capacities and very low costs to the links and switches of the network so as to minimize their effect [Figure 7.2]. The performances of the broker using different heuristics will be discussed based on simulation of the DMSS. The simulations will illustrate the net-revenue-optimal selection of servers to deliver content while fully respecting all QoS Constraints.

## 7.1 Multimedia Content Routing in the DMSS

Figure 7.1 shows an example of DMSS where the users and media servers are connected to the switches of an interconnection network. Such a system can be used to provide various multimedia services to customers who pay money for the multimedia contents.

However, provision of multimedia content over networks is still in its infancy, and so we must rely on the following scenarios of plausible services to motivate our research.

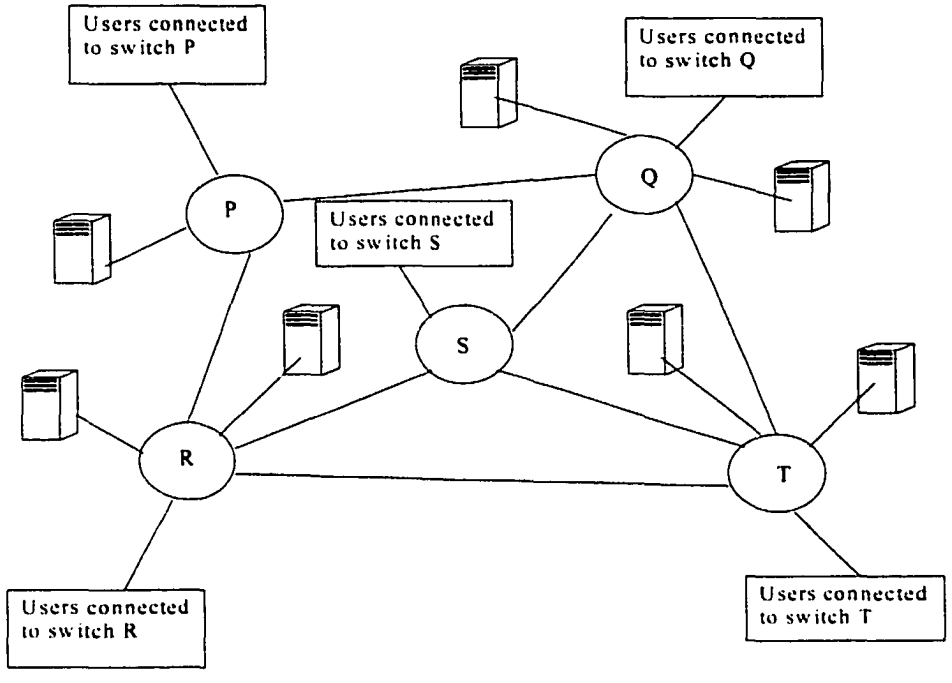


*Figure 7.1 Media servers in a DMSS. Circles represent switches, lines represent communication links, rectangular boxes represent groups of customers connected to switches.*

### 7.1.1 Sporting Events

As an example, consider a fan watching a professional league football game which is delivered to her TV set as a video stream or streams with accompanying audio streams. Because bandwidth and cameras are inexpensive, 30 different views of the game are available, taken from cameras located at midfield, behind each goal post, behind the players' benches, or even from the cameras cemented to the players' helmets. The fan can select one or two of these streams using the TV remote control. The fan can also select accompanying audio from a microphone collocated with the camera, or one of five

commentator streams delivered by sportscasters, or an audio reflector site where fans can hear one another's remarks.

Finally, the fan can designate any one player, perhaps by entering his jersey number or clicking on his image, and receive extensive material about that player in the form of text, images and video clips.

### 7.1.2 Buying a Car or Major Appliance

A prospective customer can view an introductory video clip about the car, showing its exterior and interior. The customer can then select detailed video clips illustrating interior options (e.g. leather seats) and exterior options (e.g. roof racks, alloy wheels). Any of these can lead to marketing information such as prices, availability, etc, represented as text plus images.

Finally, a seriously interested customer can initiate a videoconference with a sales person to discuss and perhaps conclude a purchase. This would entail video and audio streams and text documents.

### 7.1.3 A Tele-meeting

Immersive virtual reality and broadband networks offer the promise of very high quality videoconferences or e-meetings. It may be possible to create an experience of such fidelity and quality that the participants would notice little difference between the e-meeting and a traditional face-to-face meeting. This in turn raises a large number of unanswered research questions. However, it is clear that such an e-meeting would entail several video and audio streams to represent participants to one another, plus image streams (for presentations), images (e-whiteboards and posters) and text documents (meeting paperwork).

### 7.1.4 Movies

The media servers can offer Video on Demand (VoD) service to the users. The users can get movie streams from a media server through the network. This service is attractive to both the users and the owners of the servers and networks. A busy user can enjoy a recently released movie or any of a very large number of classic movies while sitting at home, and the owners of the servers and networks might be able to earn significant amounts of revenue by providing this service. We demonstrate the ability of UM-D to provide a server selection policy using the movie scenario as an example for simulation. The numeric values of parameters needed to drive the simulation will be chosen to be plausible for the movies scenario. Similar demonstrations could be undertaken for the other scenarios.

## 7.2 Media Server Farms to Deliver Movies to the Users

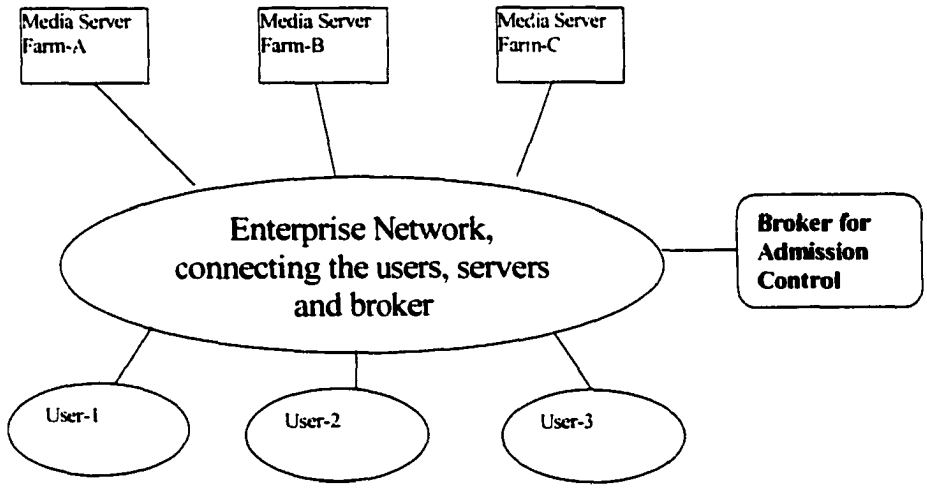


*Figure 7.2 A Media Server Farms for providing movies to customers*

The main purpose of the set of *Media Server Farms* is to provide videos to the users with guaranteed QoS. It is a DMSS owned by one company where the servers and users are connected to an Enterprise Network (EN) as shown in Figure 7.2. The media servers store a large collection of movies. The broker connected to the Enterprise Network works as an

admission controller for requests from the users connected to the EN. We assume that the Enterprise Network has sufficient bandwidth to carry multimedia streams for all users from the servers, so as to de-emphasize the impact of network constraints and stress the role of the servers. The broker therefore does admission control based solely on the available resources of the media servers.

It might be realistic to provide movies at different QoS levels as there are different kinds of video players with different qualities such as VHS, DVD and HDTV. The server resource requirements for these video qualities are not the same. So, the cost of providing different QoS levels must be different. Besides this, new movies (first round movies) are generally costlier than old movies (second round movies). Users would likely be willing to pay more to see a first round movie. In the simulation we simulate these variations in the resource requirements, cost and bid price for a given quality level of a movie by using random numbers.

Movies are replicated in the servers of the Media Server Farms. It is impractical to replicate a particular movie in all the video servers of the Media Server Farms, because each copy would require a substantial licensing fee. In the simulation, we have arbitrarily considered at least 3 replications of each movie in the system and at least two different movie copies on each video server. We have to store more than 2 copies of a movie on each server to ensure at least 3 replications when the number of servers is not larger than the total number of movies provided by the video server farm (10 movies and 5 servers, 3 replications of each movie and 6 copies in each server). Similarly, we have to replicate more than 3 copies of movies in different servers to ensure there are at least 2 movie copies on each server when the number of servers is larger than the total number of movies (10 movies 25 servers, 5 replications and 2 copies in each server).

### 7.2.1 Components of the Multimedia Stream Provided by the set of Media Server Farms

The main component of the multimedia stream provided by the Media Server Farms is a MPEG video stream. The users could also be allowed to browse the movie reviews, interviews with director, producer and casts and facts about making the movie. This *accessory information* would contain images and text, which is much less voluminous (bandwidth intensive) than video streams. We assume that the images and texts required for the accessory information does not exceed 0.5% of the video stream. Thus in a set of Media Server Farms with fewer than 200 servers, only one server with accessory information for all the movies would be enough to guarantee QoS. Figure 7.3 demonstrates the distribution and provision of movie content in a Media Server Farm with 3 video servers and an accessory information (image and text) server.

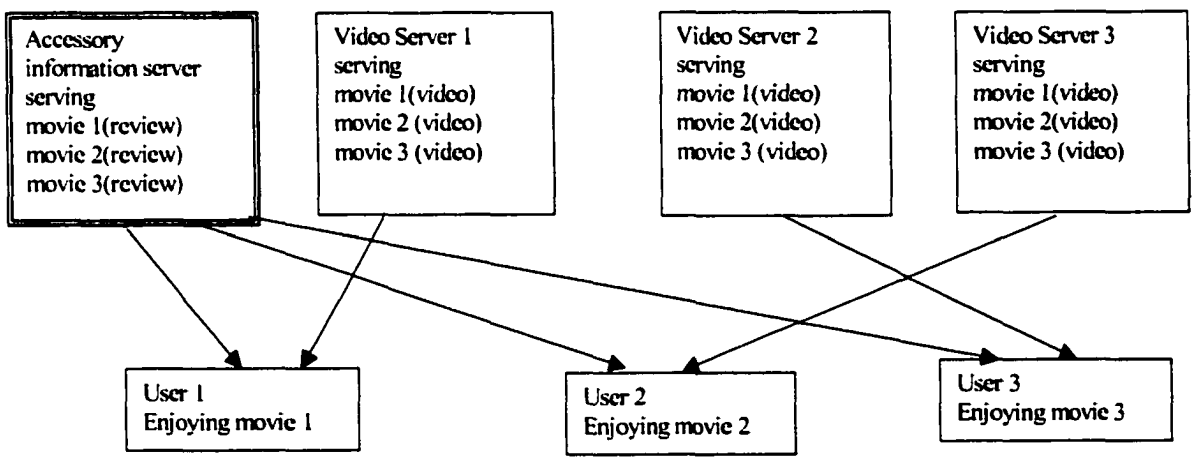


*Figure 7.3 Distribution of video streams and accessory information in a Media Server Farm.*

## 7.3 Controlling Algorithm of the Broker

In Section 5.1 we presented the mapping of the UM-D using broker to the MMKP. The broker for the set of Media Server Farms solves the MMKP to do real time admission

control and QoS adaptation. An exact algorithm for solving the MMKP, such as BBLP, is not suitable for real time decision making. We use heuristics such as I-HEU, C-HEU and G-HEU for solving the MMKP in the broker, as they compute near optimal solution in less time -- often much less time -- than the optimal algorithm. In the following sections we define the class of heuristics to solve the MMKP as *MMKP_HEU*. The following subsections present the admssion control and QoS adaptation methodology implemented by the broker.

### 7.3.1 Admission Control and QoS Adaptation Methodology

New session requests are collected into a batch over a time interval called an *epoch*. Each QoS level of the requested session can be satisfied by several combinations of servers. In Section 5.1.2 we defined each of these combinations as QoS-B (QoS level with respect to Broker) level. The broker finds the QoS-B levels for each new session. For the new sessions a dummy null level, which gives null revenue with null resource consumption, is added to the QoS-B profile. This null QoS-B level is the initial selected QoS-B level of each new session request. The QoS-B levels are sorted in ascending order according to their associated utilities. The change of a session's QoS level to a higher or lower QoS level is called an *upgrade* or *downgrade* respectively. The change of a session's QoS-B level without changing the QoS level is called a *crossgrade*; it indicates the selection of a new set of servers for the same QoS level. MMKP_HEU is applied to the new sessions as well as the already admitted ones, once in every epoch. MMKP_HEU finds the new set of QoS-B levels in the next epoch by upgrading, downgrading, crossgrading or without changing the QoS-B levels of the sessions, in order to maximize the total earned revenue summed over all sessions. If the selected QoS-B level of a new session in the new batch remains null after applying MMKP_HEU, it is rejected. The other sessions are admitted at their non null QoS-B levels. The sessions enjoy the assigned QoS-B level for at least the next epoch. When any session leaves, some resources are released. The next computation of MMKP_HEU for the next batch of requests will use these released

resources to upgrade some sessions and / or to admit more sessions. The following is the algorithm for admission control and QoS adaptation using pseudo code.

```
Procedure admit_sessions( )
1.  active_session_list: The list of sessions that already got admission.
2.  batched_session_list: sessions batched in the last epoch that are queued for admission.
3.  rejected_sessions( ): Returns the sessions with null current QoS-B level.
4.  accepted_sessions( ): Returns the sessions with non null current QoS-B level.
5.  waiting_sessions( ): Returns the rejected sessions which will wait for the next session to get admission.
6.  temp_session_list ← active_session_list + batched_session_list
        //The sessions on which MMKP_HEU will be applied
7.  batched_session_list ← null                 //ready for new sessions
8.  MMKP_HEU (temp_session_list)                //applying MMKP_HEU to all the sessions
9.  rejected_session_list ← rejected_sessions(temp_session_list) // determining the rejected sessions
10. active_session_list ← active_session_list + accepted_sessions(temp_session_list)
                                                // updating active session list
11. waiting_session_list← waiting_sessions(rejected_session_list) // determining waiting sessions
12. batched_session_list← batched_session_list+ waiting_session_list
                                                //updating the batch of the sessions seeking admission.
13. end procedure
```

## 7.3.2 QoS Adaptation when a Fault Occurs

If a server goes down then QoS adaptation is done as soon as the fault is detected. QoS adaptation after any kind of fault is performed as follows:

```
Procedure adapt_sessions( )
1.  MMKP_HEU (active_session_list)      //Trying to adapt sessions
2.  if no solution found from MMKP_HEU then     // Solution not found
3.      for each session in the active_session_list do
4.          current_QoS_level← bottom most non null QoS-B level //Making resources free
5.      end for
6.      MMKP_HEU (active_session_list) // Applying MMKP_HEU for adaptation
7.      if no solution found from MMKP_HEU then // again failed
            //The following loop makes all resources free to readmit all sessions
8.          for each session in the active_session_list do
9.              current_QoS_level← null //starting from scratch
10.         end for
11.         MMKP_HEU (active_session_list) // Applying MMKP_HEU for readmission
12.         rejected_session_list ← rejected_sessions(active_session_list) //determining rejected sessions
13.         active_session_list ← active_session_list − rejected_session_list //updating active session list
            //The following loop makes resources free to admit the rejected sessions
14.         for each session in the active_session_list do
15.             current_QoS_level← bottom most non null QoS-B level//Making resources free
16.         end for
17.         temp_session_list ← rejected_session_list //the sessions to be admitted again
```

```
18.        MMKP_HEU (temp_session_list) //trying to admit the rejected sessions
19.        rejected_session_list ← rejected_sessions(temp_session_list) //determining rejected sessions
20.        waiting_session_list← waiting_sessions (rejected_session_list) //determining the sessions
           waiting for the next batch
21.        batched_session_list← batched_session_list + waiting_session_list // updating the batch of
           the sessions seeking admission
22.        active_session_list←active_session_list + temp_session_list – rejected_session_list
                                                    // updating the list of active sessions
23.        MMKP_HEU (active_session_list) //Trying to earn more revenue with the admitted sessions
24.     endif
25. endif
26. end Procedure
```

## 7.4 Simulation of the Broker for the set of Media Server Farms

We simulate the broker of the set of Media Server Farms using I-HEU, G-HEU and C-HEU for admission control and QoS adaptation. The performance data has been collected by varying the epoch and the number of video servers. We also compare the revenue obtained by I-HEU to the estimated optimum revenue using the linear programming approach. We initialize the set of Media Server Farms with a small amount of the total resources (10% of the usual capacity) to do this comparison, as the calculation of estimated optimal revenue has exponential complexity.

### 7.4.1 Different Simulation Parameters

The following table presents the parameters of the set of Media Server Farms required for the simulation.

*Table 7.1 Different simulation parameters.*

| Parameter | Meaning | Value(s) for the experiment |
|---|---|---|
| *MOVIE_LENGTH* | Maximum length of movie. | 3 hours |
| *NO_OF_MOVIE* | Number of movies in the set of Media Server Farms. | 10 |
| *M* | Number of video servers in the set of Media Server Farms. | 5, 10, 15, 20, 25, 30, 35,40, 45, 50 |
| *NO_OF_REPLICATION* | Corresponding number of replications of MPEG video streams. | 3, 3, 3, 4, 5, 6, 7, 8, 9, 10 |
| *COPIES_PER_SRV* | Corresponding number of movie copies per server | 6, 3, 2, 2, 2, 2, 2, 2, 2, 2 |
| *ACC_FACTOR* | Ratio of resources required to deliver accessory information, to resource required to deliver video streams. | 0.005 |
| $L_{avg}$ | Users waiting in the queue | 10% of the total user enjoying the multimedia service |
| *NO_OF_DATA_SETS* | Repetitions of the simulation experiment. We present average results from these repeated experiments. | 10 |

### 7.4.2 Initialization of Server Resources

We simulate the media servers designed by Ninth House Corporations [82] in our experiments. The specification of this media server and initialization of total resources in our simulated video servers are as follows:

*Table 7.2 Initialization of servers in the Media Server Farms.*

| Type of resource | Ninth house server | Initialization function for a video server in the DMSS | Initialization function for the accessory information server in the DMSS | Assumptions |
|---|---|---|---|---|
| CPU | 400 MHz dual processor | $100 \times U(0.95, 1.05) \times 1000$ | $M \times 100 \times U(0.95, 1.05) \times 1000 \times ACC\_FACTOR$ | Total 800 MHz is equivalent to 100 cycles. |
| RAM | 256 MB | $120 \times U(0.95, 1.05) \times 1000$ | $M \times 120 \times U(0.95, 1.05) \times 1000 \times ACC\_FACTOR$ | 136 MB is used by the O/S. |
| I/O bandwidth | 640 Mbps | $600 \times U(0.95, 1.05) \times 1000$ | $M \times 600 \times U(0.95, 1.05) \times 1000 \times ACC\_FACTOR$ | 40 Mbps is reserved for the system. |

$U(lower, upper)$ is a uniform continuous random number generator between the numbers *lower* and *upper*.

Random numbers are used in these functions to simulate the fluctuation of the available resources in different servers due to O/S and different operating conditions.

Available resources are scaled by multiplying 1000 to apply fixed-point integer operations in the heuristics to solve the MMKP.

### 7.4.3 Different QoS Levels

The users enjoy three QoS levels defined by three different resolutions of video and image quality. The following table shows average resource requirements in the video server, offered prices and costs for different QoS levels.

*Table 7.3 Different QoS levels supported by the DMSS*

| QoS levels | Resolution | Average I/O bandwidth requirement | Average CPU cycles requirement | Average Memory requirement | Average offered price by the user | Average cost of providing media |
|---|---|---|---|---|---|---|
| QoS 1 (Bronze) | 320×240 | 1.5 Mbps | 0.25% | 0.3 MB | $1.0 | $0.75 |
| QoS 2 (Silver) | 640×480 | 3.0 Mbps | 0.50 % | 0.6 MB | $3.0 | $2.25 |
| QoS 3 (Gold) | 1024×786 | 4.5 Mbps | 0.75% | 0.9 MB | $5.0 | $3.75 |

The size of MPEG streams for different movies might vary drastically from the average resource requirements listed in Table 7.3. Offered prices would vary from one user to

another and the cost of providing multimedia streams would also vary from one movie to another. In order to simulate these variations we initialize the resource requirements, prices and costs as follows:

*Table 7.4 Initialization of resource requirements for different QoS levels*

| **Parameter** | **Initialization function** |
|---|---|
| I/O bandwidth requirement for the *k*th QoS level of the video stream | $1.5 \times k \times U(0.75, 1.25) \times 1000$ |
| Memory requirement for the *k*th QoS level of the video stream | $0.3 \times k \times U(0.75, 1.25) \times 1000$ |
| CPU cycles for the *k*th QoS level of the video stream | $0.25 \times k \times U(0.75, 1.25) \times 1000$ |
| Offered price by a user for the *k*th QoS level | $(2k+1) \times U(0.75, 1.25) \times 1000$ |
| Cost of providing *k*th QoS level of a movie | $(2k+1) \times 0.75 \times U(0.55, 0.95) \times 1000$ |
| Resource requirement for the *k*th QoS level of accessory information | *ACC_FACTOR* × Resource requirement for the *k*th QoS level of video stream. |

### 7.4.4 A Multimedia Session Request

- The movie enjoyed by the user: $UI(NO\_OF\_MOVIE) + 1$ , where $UI(i)$ is a random integer from 0 to $(i - 1)$.
- Duration of the movie session = $MOVIE\_LENGTH \times U(0.00 , 1.00)$
- Highest level of QoS requested by the user $= UI(NQOS\_LEVEL) + 1$. So the user will enjoy a QoS level from 1 to $UI(NQOS\_LEVEL) + 1$ if the user is accepted by the broker. For simplicity we assume that the users always get the same QoS level for the accessory information as the video stream.

### 7.4.5 Simulation Events

- *User arrival:* The interarrival times of users in service oriented systems such as telephony, post office generally follow the exponential random distribution, i.e., the number of new users requesting for service in each batch will not vary substantially. Here we assume that the interarrival times of the movie customers in the set of Media Server Farms follow exponential random number distribution. The capacity of the set of Media Server Farms can be estimated by the number of sessions enjoying Silver (average level of QoS) QoS at full load. This can be expressed as $n = M \times n_k$ . Where,

$n_k$ = estimated number of users enjoying Silver QoS from a server at full load $= \frac{\text{Total CPU cycles in a video server}}{\text{Average CPU cycles required for Silver quality video}}$. The average duration of a session can be defined by $1/\mu$ = *MOVIE_LENGTH*/2. Now the average arrival rate $\lambda$ can be determined from the equation $n = \frac{\lambda(1 + L_{avg})}{\mu L_{avg}}$, which has been derived using queueing theory [94]. Here, $\mu$ =service time of the queue and $L_{avg}$ is the proportion of the users waiting in the queue to the users enjoying the service. In this estimation, we assume poisson service time although in the simulation the duration of watching a movie is uniformly distributred.

- *Departure of the user:* If a user gets admission then she enjoys the application for the specified period of time. But if she does not get admission then she will continue waiting for the next batch with probability 0.5.

### 7.4.6 Simulation Environment

We have coded the simulation of the broker using the C++ programming language. To analyse the system at full load a sufficiently large batch is allowed to get admission by the broker initially to create contention of resources in the servers. Then we generate a small batch in each epoch with average arrival rate $\lambda$. We calculate the revenue earned from the users and the time required by the broker in each epoch to do admission control and QoS adaptation. The time required by the heuristics depends on the problem set. That is why we generate multiple small batches and calculate the average response time. In this experiment we generate small batches in each epoch from Second 0 to Second 3000 (arbitrarily selected) of the simulated clock. Summation of the revenues of all the epochs and average time requirement for admission control and QoS adaptation in an epoch are the measures of broker's performance in our simulation. We ran the simulation on a Quad 800 MHZ Pentium III UNIX Server with 2GB RAM.

## 7.5 Computational Complexity of the Broker for the set of Media Server Farms

The complexity of the broker depends on the size of the system, the number of users in the system and the algorithm used to solve the MMKP. We present the complexity analysis of the broker initialized here to run the simulation of the set of Media Server Farms.

The total number of users in this set of Media Server Farms with $(M + 1)$ servers is approximately $M \times n_k$., where $n_k$ =estimated number of users enjoying Silver QoS from a server at full load. There are $M$ video servers and one accessory information server. Each server has three resources, CPU cycles, Memory and I/O BW. So, the total number of resources = $3(M+1)$

Each video stream is replicated in more than one server. The number of replications can be expressed by $M \times 2/10$. On the average there are 2 QoS levels in each session. For each QoS level we prune half of the possible QoS-B level. So, the number of QoS-B levels for a session is approximately $M/5$. The MMKP solved by the broker has $n = M \times n_k$ groups, $l = M/5$ items in each group and $m = 3(M+1)$ resource dimensions. Using the complexity analysis of the heuristics of the MMKP we get the following worst-case complexities of the broker in Table 7.5.

*Table 7.5 Computational complexity in the broker using different heuristics*

| | | Time required in each batch (worst case estimation) |
|---|---|---|
| **Algorithm used in the broker** | I-HEU | $O\left(\frac{M^2 n_k^2 M^2 3(M+1)}{25}\right) \cong O(M^5)$ |
| | C-HEU | $O\left(\frac{M}{5} n_k M \log\left(\frac{M}{5} n_k M\right)\right) + O\left(\frac{M}{5} n_k M \times 3(M+1)\right) \cong O(M^3) + O(M^2 \log M)$ |
| | G-HEU | $O\left(\frac{M}{5} n_k M \times 3(M+1)\right) \cong O(M^3)$ |

## 7.6 Experimental Results

The following data were collected from the simulation and plotted.

- Revenue earned by different heuristics during the simulation.
- Average time requirements to do admission control and QoS adaptation at the end of an epoch.
- Average size of the batch in each epoch.
- Estimated optimal earned revenue and revenue earned using I-HEU by the broker of the smaller set of Media Server Farms.

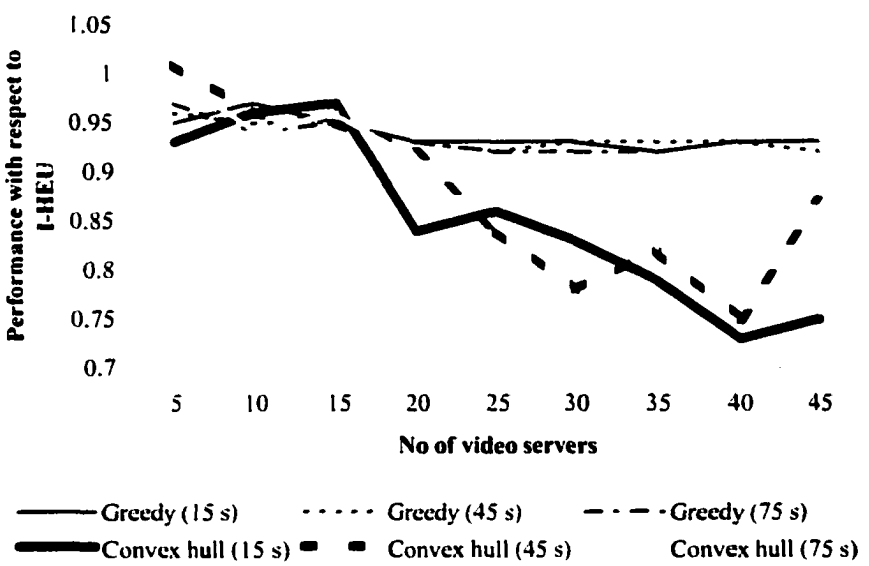


*Figure 7.4 Ratio of the earned revenues by the broker using different heuristics with respect to the earned revenues by the broker using I-HEU for different values of epoch. The numbers in the parentheses are epochs.*

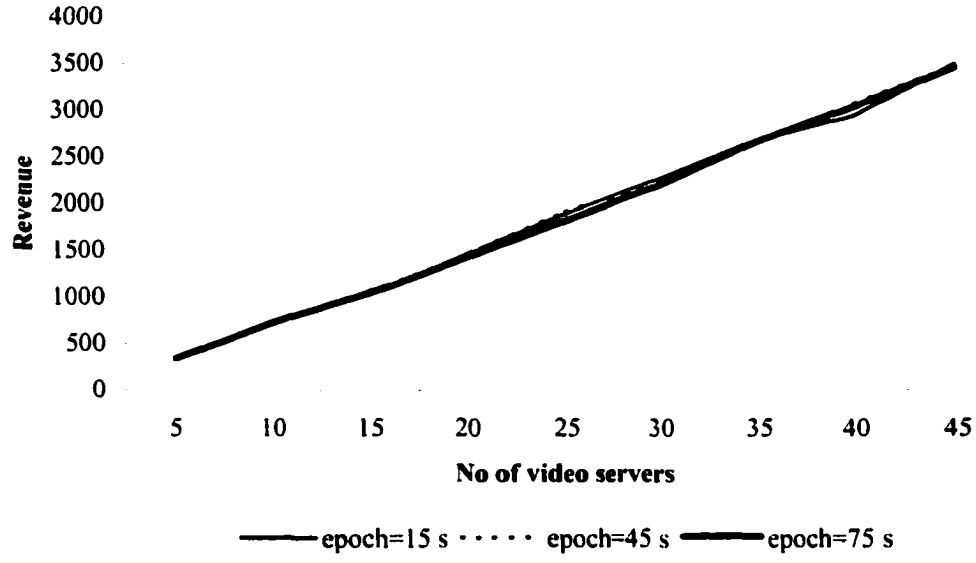


*Figure 7.5 Earned revenues by the broker using I-HEU for different values of epoch.*

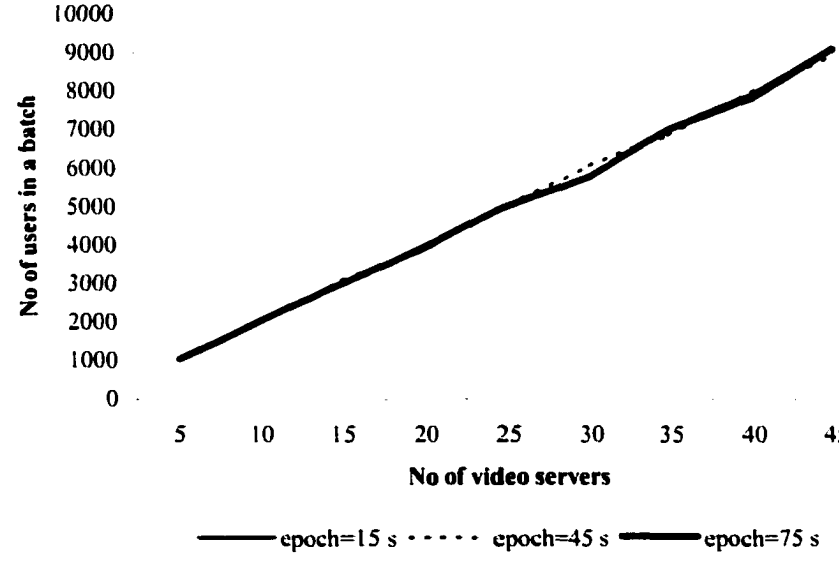


*Figure 7.6 Average number of users in each batch processed by the broker using I-HEU for different values of epoch.*

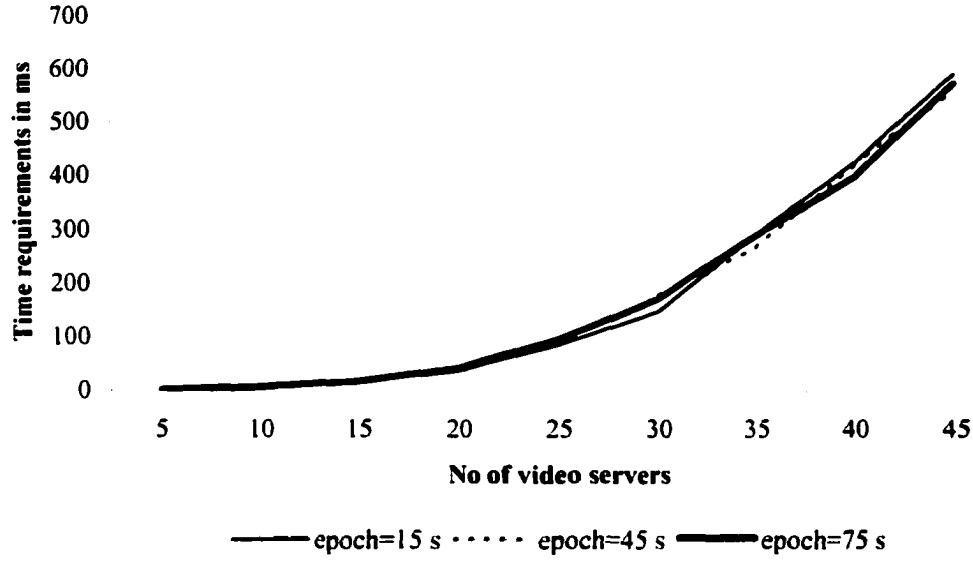


*Figure 7.7 Average time requirements to do admission control and QoS adaptation by the broker using G-HEU for different values of epoch.*

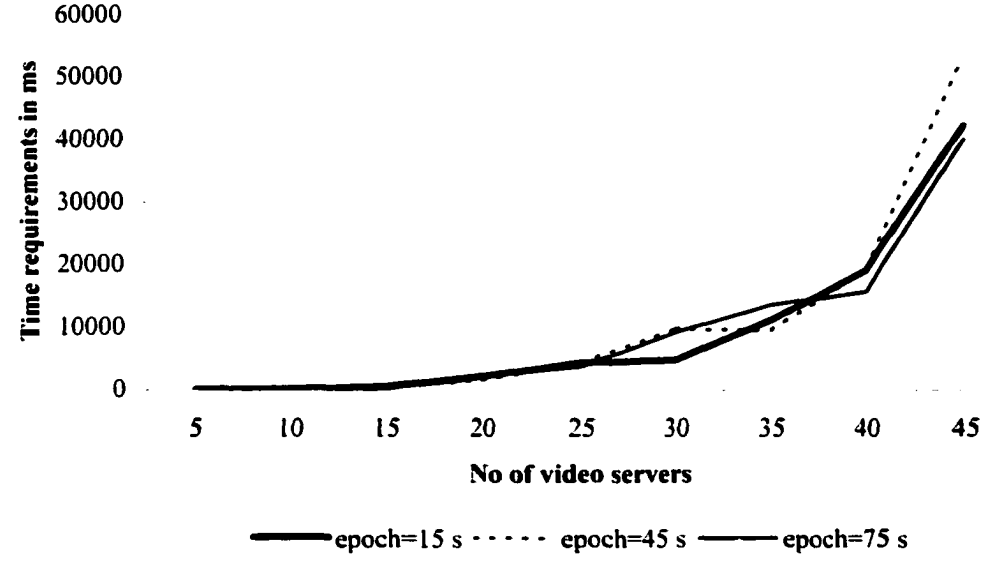


*Figure 7.8 Average time requirements to do admission control and QoS adaptation by the broker using I-HEU for different values of epoch.*

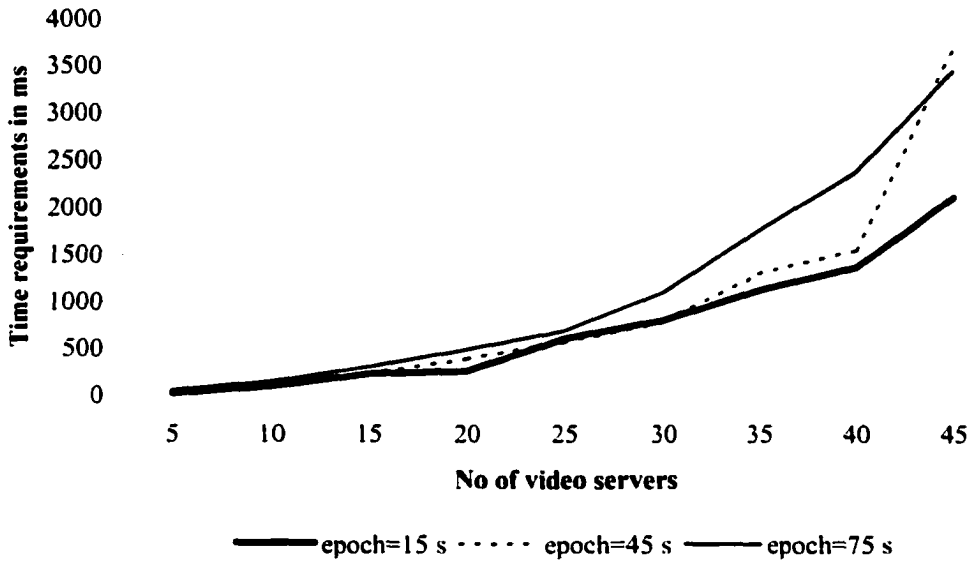


*Figure 7.9 Average time requirements to do admission control and QoS adaptation by the broker using C-HEU for different values of epoch.*

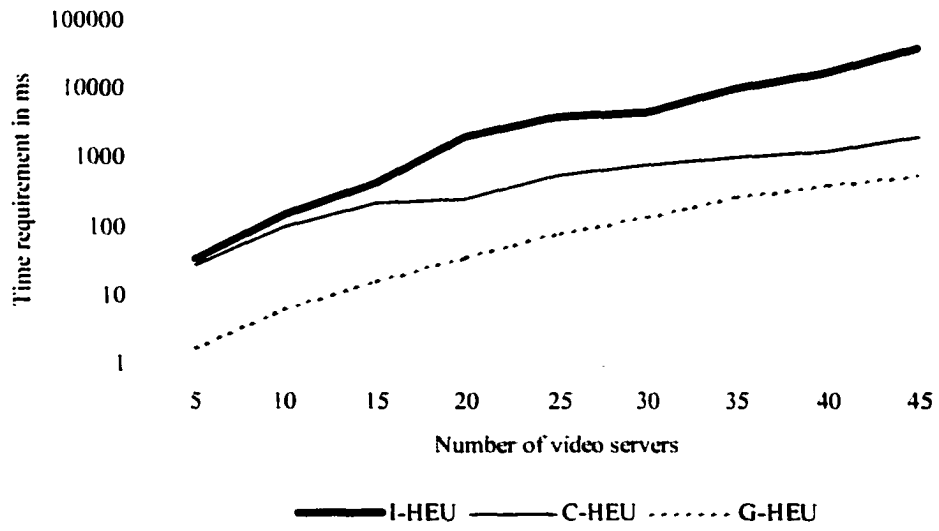


*Figure 7.10 Average time requirements to do admission control and QoS adaptation by the broker using different heuristics for 15 sec epoch for the estimated full load of customers enjoying Silver level of QoS.*

*Table 7.6 Ratio of revenue earned by the broker using I-HEU to the estimated optimal revenue for the set of Media Server Farms of different sizes and epoch lengths.*

| | | No of Servers | | | | | |
|---|---|---|---|---|---|---|---|
| | | 5 | 10 | 15 | 20 | 25 | 30 |
| epoch (sec) | 15 | 0.98 | 0.97 | 0.98 | 0.97 | 0.97 | 0.96 |
| | 45 | 0.99 | 0.98 | 0.98 | 0.98 | 0.98 | 0.97 |
| | 75 | 0.97 | 0.97 | 0.98 | 0.97 | 0.98 | 0.95 |

## 7.7 Validation of the Simulation

We have tested the sequence of generated random numbers by applying the $\chi^2$ test. This test verifies whether a sequence of numbers fits statistical distributions such as the uniform random distribution or the exponential random distribution. Consistency of the obtained results also validates the code written for simulation. We find that a video server can serve approximately 200 users with average (Silver) QoS levels. The data of average batch size in Figure 7.6 is consistent with this number. The time requirements by the broker using different heuristics to do admission control are also consistent with the complexity analysis of the broker. This also validates the code for the heuristics. We will discuss several observations on time requirements in the next section.

## 7.8 Observations

- I-HEU determines the operating QoS of the admitted session after analyzing all the QoS levels of the sessions (admitted and requested) in the system. Hence it gives better results than G-HEU, which is based on the FCFS (First Come First Serve) rule and does not search extensively to achieve better utilization of resources.
- C-HEU is good for a small DMSS (a set of Media Server Farms with fewer than 15 media servers). Figure 7.4 shows worsening performance of C-HEU in earning revenue than G-HEU with increase in the number of servers as it increases the

number of resource dimensions in the system. This might be due to the transformation of a multidimensional resource to a single dimensional resource by C-HEU.

- If we compare the earned revenue by I-HEU to the estimated optimal revenue we find that performance of I-HEU degrades with an increase in the number of video servers [Table 7.6]. But the degradation of the performance is not like G-HEU and C-HEU as shown in Figure 7.4.
- The response time of the broker using I-HEU in the DMSS is larger than the epoch for a large set of Media Server Farms with a large number of video servers (epoch =15 sec and $M$>35). We have to increase the epoch or use higher speed CPUs for the brokers.
- The time required for admission control by the broker for each batch is far less than the time required by the MMKP instances of equivalent size shown in Figure 3.7, Figure 3.8 and Figure 3.9. This is due to the fixed-point computations done by the broker in this experiment.
- The plotted curves of the time required by the broker using I-HEU are steeper than those for the broker using C-HEU or G-HEU [Figure 7.7 to Figure 7.9]. This is because of the different degree of polynomial complexity of these heuristics.
- The ratio of the degrees of the polynomial complexity of the broker using I-HEU, C-HEU and G-HEU is 5:3:3. We find almost the same ratio for larger DMSSs with large number of video servers in the logarithmic time requirement curves in Figure 7.10.
- Batch size (the number of users seeking admission plus the number of active multimedia sessions) depends on the size of the system. A larger system allows more users to get in.
- The time required for computation depends on the batch size, i.e., size of the MMKP. In Figure 7.7 to Figure 7.9 we find that time required in admission and adaptation increases with an increase in the number of servers and epoch size in almost every

case. I-HEU, C-HEU and G-HEU invariably depend on the characteristics of the problem set. If the problem size (batch size) is close, then we may find higher time requirements for smaller DMSSs or smaller epochs. This irregular behaviour shows up mostly in I-HEU and C-HEU.

- The revenue increases if we add more servers to the system and if there are proportionally more users requesting admission to the system [Figure 7.5]. We do not find any substantial change with the increase of epoch.
- If the broker is a very high-speed machine then a very small epoch can provide almost instantaneous services to the users. But if the response time is not a major issue then a larger epoch is acceptable.

## 7.9 Chapter Summary

This chapter presented several plausible scenarios of content routing of multimedia traffic. Delivery of movie streams from the servers of Media Server Farms to the users through the network is a plausible scenario for content routing. We collected the performance data for the admission controller of the set of Media Server Farms in the simulation. We find that I-HEU, the incremental heuristic of solving the MMKP earns better revenue with worse response time than the other heuristics. On the other hand, G-HEU, the greedy approach of selecting items in the MMKP has the quickest response time among all the heuristics. Thus the selection of a heuristic for the broker depends on the size of the set of Media Server Farms, number of users and the limit of response time set for the system. Specifically, we get a delay of approximately 0.1 sec, 1 sec and 10 sec by the broker for admission control and QoS adaptation of 5000 sessions using G-HEU, C-HEU and I-HEU respectively. The next chapter presents another application of the UM-D for admission control and QoS adaptation of SLAs by Distributed SLA Controllers in interconnected ENs.

# 8. Application of UM-D to Distributed SLA Controllers for Interconnected Enterprise Networks

This chapter introduces Distributed SLA Controllers (DSCs) for admission control and QoS adaptation of SLAs in a group of interconnected Enterprise Networks (ENs). We present a distributed heuristic to find the candidate paths for the SLAs using the DSCs. The admission control strategy of the DSCs is based on the DMSS with fully distributed controllers. The distributed algorithms for solving the MMMKP are applied to do admission control and QoS adaptation of the SLAs in D-SLAOpt, a Java simulator of a DSC. We present experimental data to compare the performance of different algorithms applied within D-SLAOpt.

## 8.1 Distributed SLA Controllers in a Network of Interconnected ENs

Interconnected ENs are generally used for communication among different parts or groups of an organization. Each EN, we assume, is fully controlled by a subsidiary part of the organization. The subsidiaries are assumed to be autonomous – one subsidiary does not disclose its internal information to another subsidiary of the organization. Autonomous management of ENs requires that significant control be kept within each EN. Distribution of control also offers the benefits of making the ENs more manageable, scalable, and fault tolerant. Distributed SLA Controllers are proposed to guarantee QoS of the SLAs in a network of interconnected ENs. Each EN contains a Distributed SLA Controller (DSC) for doing admission control and QoS adaptation of the SLAs submitted to it. These SLAs are called *local SLAs* to that DSC. Figure 8.1 shows the DSCs for three interconnected ENs.

We distinguish two kinds of SLAs, based on the source and destination of the SLA in the network of interconnected ENs:

- *Intra-EN SLA*, if the source and the destination are located in the same EN.

- *Inter-EN SLA*, if the source and the destination are located in two different ENs.

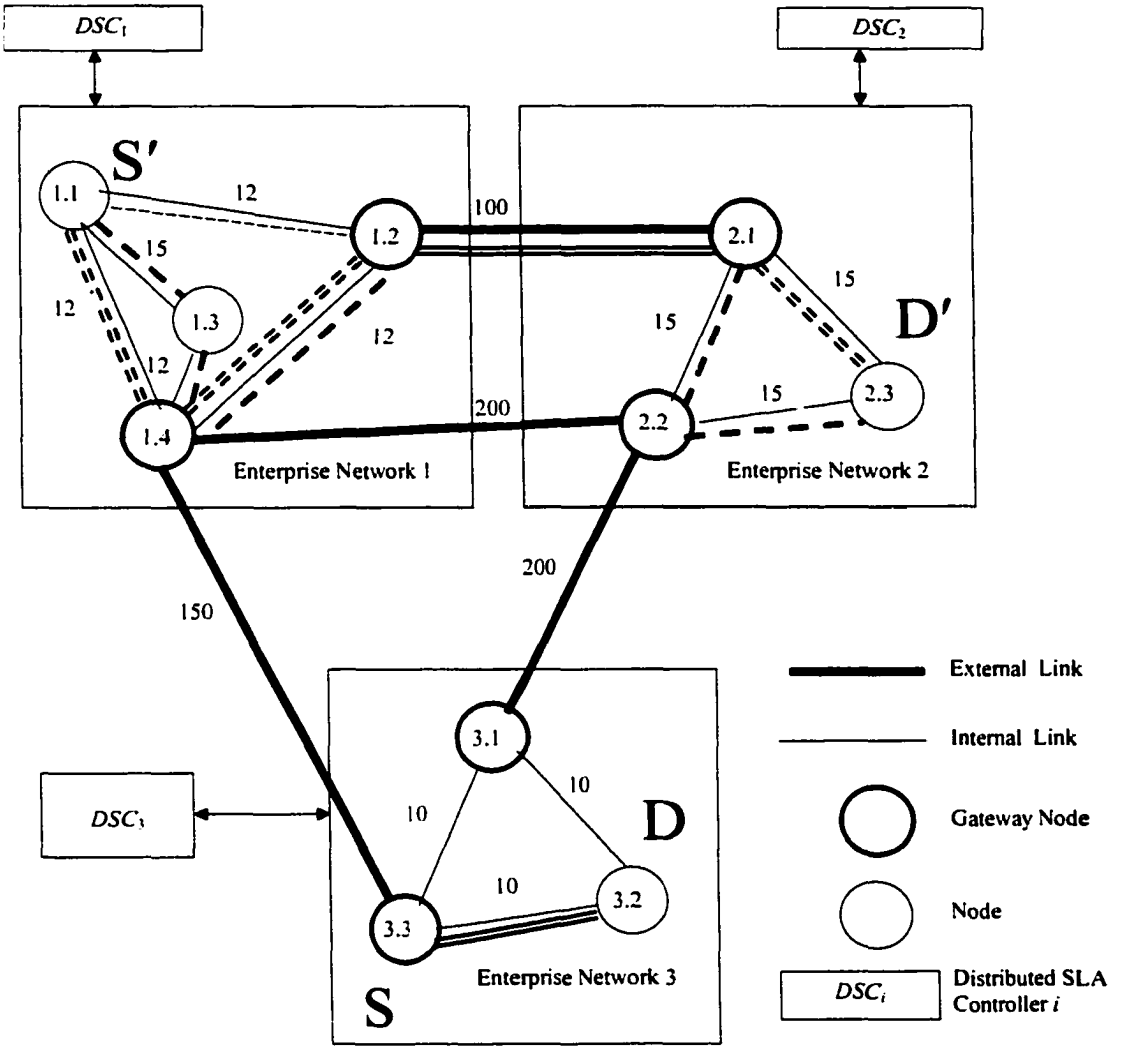


*Figure 8.1 Distributed SLA controllers in a network of with three interconnected ENs*

Two different types of links can be distinguished in the network of interconnected ENs:

- *Internal link*, a link that connects two nodes of an EN. An Intra-EN SLA uses links of this type.
- *External link*, a link that connects two nodes (gateways) of two different ENs. An Inter-EN SLA must use at least one link of this type.

An Intra-EN SLA might be routed through another EN using external links, if the paths through the internal links of the EN do not have enough free bandwidth. We ignore this possibility in this dissertation.

Inter-EN SLAs must consume resources of at least two ENs, i.e., the resource consumption is distributed among the ENs. So, the computation for finding the optimal QoS levels and paths of the inter-EN SLAs to maximize the earned revenue from the SLAs must be distributed among the SLA controllers, which must therefore communicate in order to make globally optimal decisions.

## 8.2 Assumptions for Admission Control and QoS Adaptation by DSCs

**Public Resources and Elements**

- The value of the metrics for external links, such as link bandwidth, delay and available bandwidth are public to every DSC of every EN and are periodically updated.
- The topology of the *global network* is public to every EN. This network consists of the external links and the gateways to the ENs.

**Private Resources and Elements**

- The topology and network parameters (link bandwidth, delay and available bandwidth) of each EN are private to its DSC.
- Each EN will keep all internal routing data private and will not share it with other subsidiaries.
- The values of SLA parameters are private to the DSC which received the SLA request. That DSC will make the final decision for admission and QoS adaptation of the SLA. However a DSC may send a part of an SLA to other DSCs to find the $k$ shortest paths for the SLA or to evaluate the feasibility of resource consumption of a QoS level.

**Communication among SCs**

- Each DSC communicates with other DSCs using the external and internal links through well-known gateways. [Any reliable protocol such as TCP over IP can be used for message passing among DSCs]
- There might be more than one gateway in an EN to transmit traffic to another EN. Each gate is connected to one or more gateways of other ENs.
- Each EN may query its local gateway about transit delays to other gateways in that EN to update the public information of external links.

## 8.3 Detailed Description of Figure 8.1 Describing Distributed SLA Controllers in a Group of 3 Interconnected ENs

In Figure 8.1 we follow a naming convention for the nodes: the $y$th node of the $x$th EN is numbered by $x.y$. This is useful for specifying source and destination of an SLA. The thin lines connecting the nodes of a particular EN represent internal links. The thick lines connecting two different ENs are external links. For example, the link 1.2→2.1 is an external link. $DSC_1$, $DSC_2$ and $DSC_3$ all have the statistics for delay, available and used bandwidth on link 1.2→2.1. But only $DSC_1$ knows the delay, available and used bandwidth on link 1.2→1.4. The numbers over the links are delays, and might plausibly be in milliseconds. The nodes with thick circles (1.2, 2.1, 1.4, 2.2, 3.1, 3.3) represent gateways of an EN to another EN.

An intra-EN SLA from source $S$ (node 3.3) to destination $D$ (node 3.2) can be routed through path 3.3→3.2 (thick doubled line) by $DSC_3$. An inter-EN SLA from source $S'$ (node 1.1) to destination $D'$(node 2.3) will be submitted to $DSC_1$. $DSC_1$ may disclose a part of the SLA to $DSC_2$ to determine the subpaths in EN 2 for this SLA. The sub paths denoted by thick double broken lines in EN 1 and EN 2 and connected by the external link 1.2→2.1 denote one of the possible paths for this SLA.

## 8.4 Mapping to the UM-D

The admission control and QoS adaptation problem in the DSCs of interconnected ENs can easily be mapped to the UM-D in a DMSS by fully distributed controllers. The following points describe this mapping.

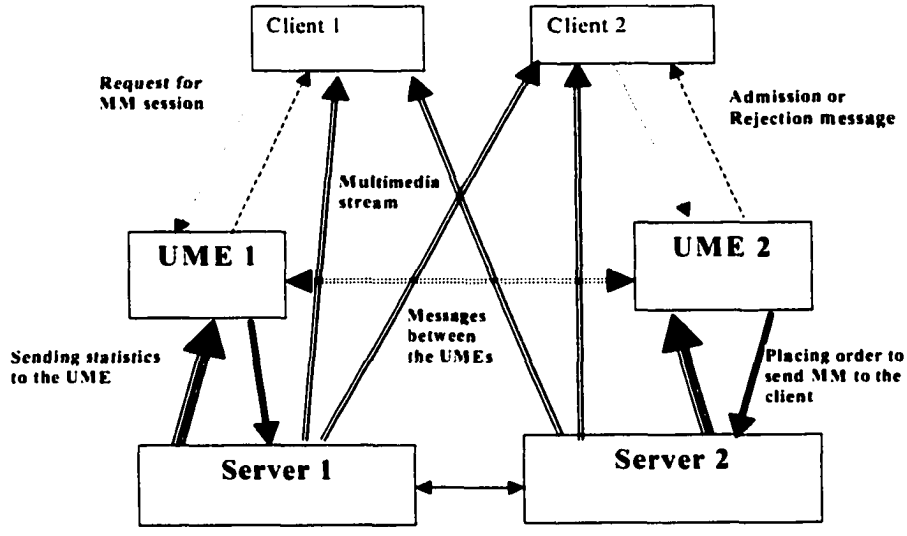


*Figure 8.2 DMSS Architecture controlled by fully distributed controllers. (Reproduction of Figure 4.5)*

- Each EN can be mapped to a multimedia server in the DMSS. This is not a physical mapping as a multimedia server is connected to a node of an EN in the DMSS. We can visualize this mapping from the service point of view. The ENs provide bandwidth to the users who submit their SLA requests through DSCs and multimedia servers provide multimedia streams to the users who submit their multimedia session requests through a UME.
- The internal link bandwidths in an EN correspond to the resources (CPU cycles, memory, I/O bandwidth etc.) in a multimedia server.
- The external links among the ENs are equivalent to the communication links among the multimedia servers.

- Each SLA is equivalent to a multimedia session request submitted to a server in the DMSS.
- The delay associated with a given QoS level of an SLA is equivalent to the delay allowed for a multimedia session at a specific QoS level.
- A path for an SLA consists of internal links from different ENs when the source and destination are not in the same EN. This is analogous to a composite multimedia stream (e.g., video and text) provided by more than one server connected through the links in a DMSS.
- Each DSC works as a Utility Model Engine (UME) determining the SLAs to be admitted at particular QoS levels such that revenue is maximized. Each UME will run a distributed heuristic to pick QoS levels for near optimal total revenue.

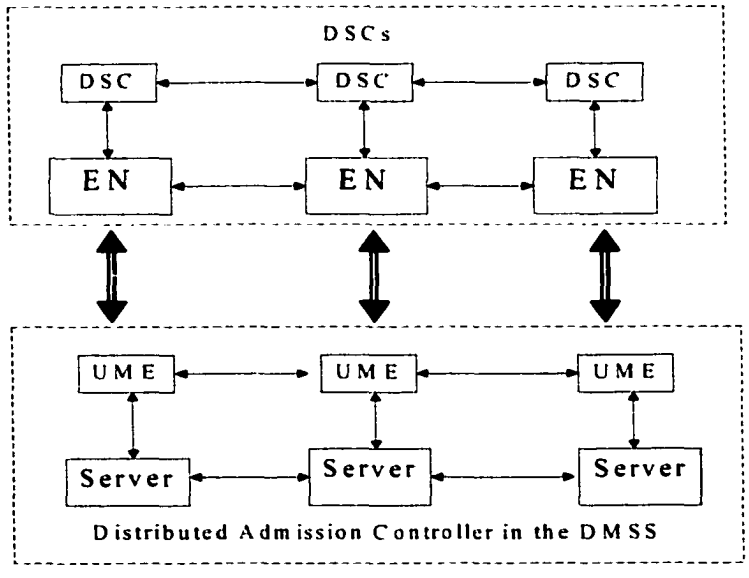


*Figure 8.3 Mapping of DSCs to UMEs. The single- lined two-way arrows represent communication among the components. The double- lined two-way arrows represent mapping.*

Thus each DSC is equivalent to a Utility Model Engine in a fully distributed controlled DMSS. Figure 8.3 shows the mapping of DSCs to the Distributed Admission Controllers (Utility Model Engines) for the DMSS.

## 8.5 Working Principle of a DSC

We implemented distributed versions of the computations done by a centralized SLA controller to do admission control and QoS adaptation in DSCs. The architecture in Figure 8.4 shows the two main computational components of a DSC.

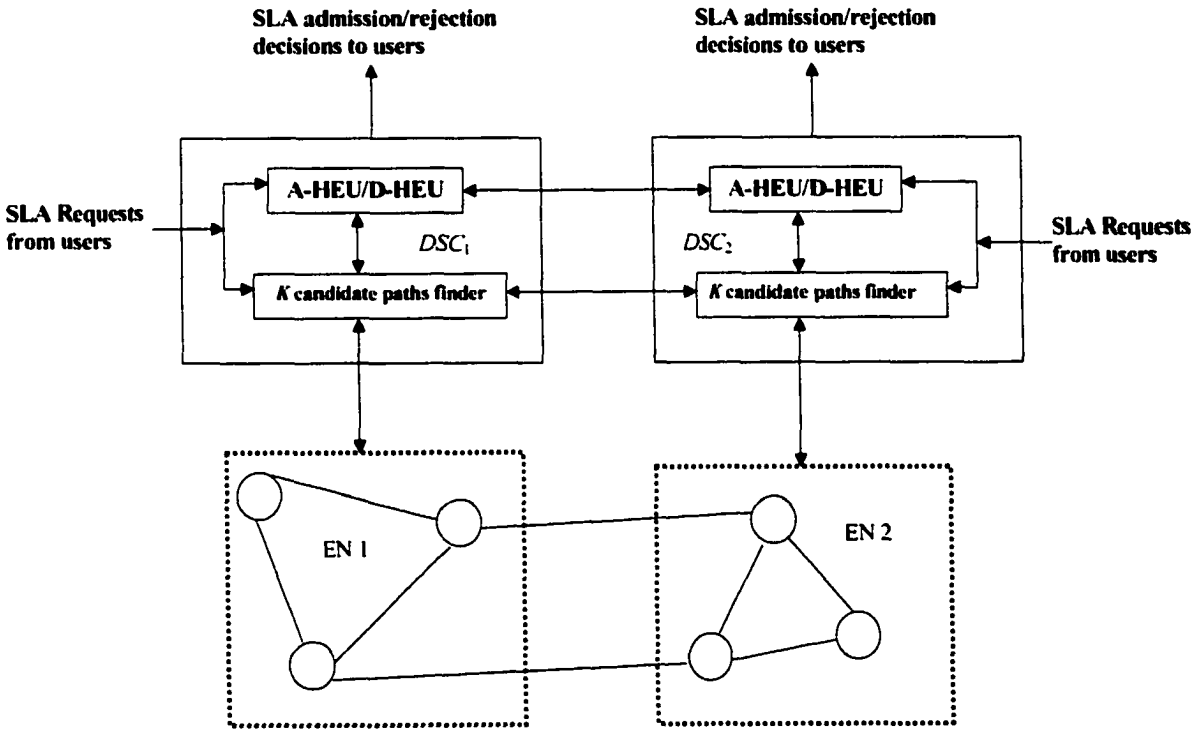


*Figure 8.4 Architecture of DSCs for two interconnected ENs*

The main job of the $K$ candidate paths finder in a DSC is to find $K$ paths from the source to the destination of each local SLA. The value of $K$ depends on the policy of the DSC. When there is contention in the network the shortest path might not be feasible to route SLAs. That is why we determine the $K$ shortest paths, so that DSCs can route the SLAs through the alternative paths satisfying the maximum delay specified in the QoS levels.

For each SLA the link bandwidths are reserved for only the path, which are selected for that SLA. So there will be no provision of fault tolerance when a link failure occurs. For intra-EN SLAs we could easily apply, for example, the $K$ shortest paths algorithm proposed by Eppstein [8] to the graph representing the EN. But the $K$ shortest paths for inter-EN SLAs cannot be determined without the help of other DSCs as the topology of an EN is private to its DSC. We can arbitrarily pick links joining the ENs and then calculate $K$ shortest paths within each EN. Here we propose a more refined distributed heuristic to determine $K$ candidate paths for inter-EN SLAs by applying Eppstein's $K$ shortest paths algorithm hierarchically. This requires message passing among the $K$ shortest paths finder modules (shown by the bi-directional arrows in Figure 8.4). We present a detailed description of this algorithm in the next subsection.

We have mapped the problems of admission control and QoS adaptation by a DSC to the UM-D [Section 8.4] and we earlier mapped the UM-D using fully distributed controllers to the MMMKP [Section 5.2]. Thus admission control and QoS adaptation problem by DSCs finally maps to the MMMKP, where each EN represents a knapsack and the associated DSC represents a solver. A feasible path (one of the $K$ candidate paths) satisfying the constraints specified in a QoS level of an SLA represents an item of a group in the MMMKP. We solve this MMMKP using distributed algorithms such as D-HEU and A-HEU to determine the rejected SLAs and admitted SLAs with appropriate routes and QoS levels.

## 8.6 Heuristic to Find the $K$ Candidate Paths in a Group of Interconnected ENs

The central technique of this algorithm is to apply Eppstein's [8] $K$ shortest paths algorithm hierarchically in two levels. First we apply the algorithm in the *global network*, a network with the external links and gateways. Then we apply the $K$ shortest paths algorithm to find the sub paths in each DSC's EN. Finally these sub paths are merged to

choose the $K$ candidate paths. This approach of finding paths over the ENs is not new. The general idea, i.e., a complete path is composed by several path segments and each of them is a localized shortest path, is used in OSPF inter-area and inter autonomous system routing [79] [80]. However, with Autonomous Systems the Internet uses a distance-vector algorithm as $K$ shortest paths algorithm is considered unscalable and too expensive. Finding the candidate paths will require lots of messages if we find one path at a time. But if we calculate candidate paths for a batch of inter-EN SLAs it requires the same amount of messages as finding the candidate path of one inter-EN SLAs.

### 8.6.1 Finding Paths in the Global Network

Figure 8.5 shows the global network comprising the 3 interconnected ENs shown in Figure 8.1. The dotted lines in this figure are dummy links or paths inside the EN between two gateways of an EN. The topology of the global network is public to all the DSCs, so any DSC can easily determine the $K$ shortest paths in the global network for an inter-EN SLA. The delay for a dummy link between two gateways is unknown at this level: Only the delays in the external links are considered here. Paths, which can transmit the multimedia stream within the maximum delay bound specified by a SLA are determined.

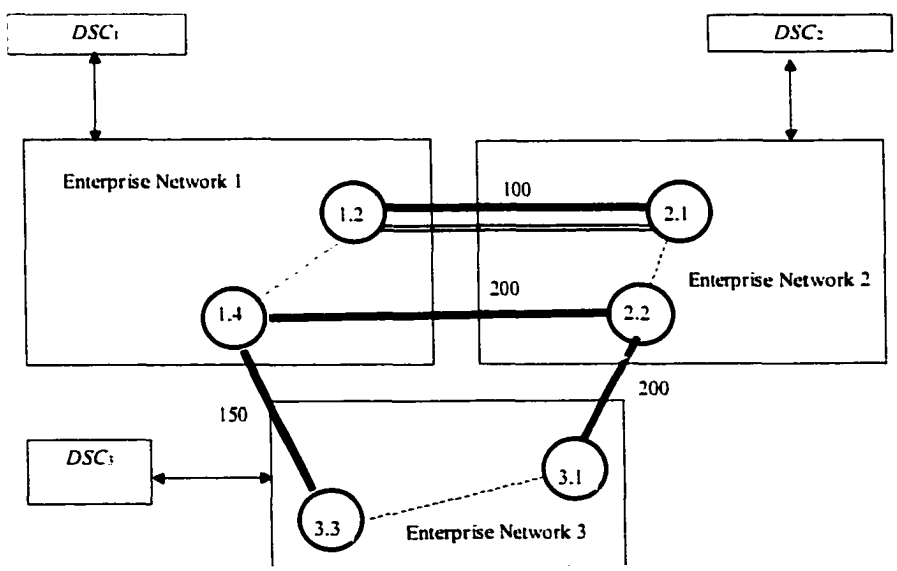


*Figure 8.5: A global network with external links and gateways.*

### 8.6.2 Finding Paths Inside the ENs

In this step we descend a level, and apply Eppstein's algorithm [8] to find the sub paths inside the EN. For a particular SLA, sub paths in different ENs are calculated by the corresponding DSC. Each DSC sends path request messages to the other DSCs and receives path messages from the other DSCs in order to calculate all the sub paths. The paths over the external links and the sub paths inside the ENs are merged to form paths from the source to the destination. Many paths will be formed if we consider all the combinations. To reduce the search space in large networks we set a limit of $K'$ paths when finding the paths in an EN or global network. We get $K'^{M+1}$ combinations of paths from $M$ ENs. $K$ shortest paths are chosen from this list of paths, which is expected to be not larger than $2K$. Thus the expected value of $K'$ is $\lceil (2K)^{\frac{1}{M+1}} \rceil$. A path will be considered as a candidate for routing multimedia streams if the sum of delays over the external links and internal links of the ENs is less than the delay specified by a QoS level of an SLA. We present the detailed pseudo code of our heuristic to determine the $K$ candidate paths for a batch of SLAs in the Appendix [Section 10.3].

### 8.6.3 An Example Demonstrating how $K$ Candidate Paths are Calculated

Consider an SLA from $S'(1.1)$ to $D'(2.3)$ in Figure 8.1, which requires a network path with delay less than 150 ms, 100 ms and 50 ms for QoS levels Bronze, Silver and Gold, respectively. We find only one path, 1.2→2.1, that satisfies the maximum delay constraint in the global network shown in Figure 8.5. A double line in this figure shows this path. As this network is small, we are finding all the sub paths inside each EN, rather than just $K'$ sub paths.

Now $DSC_1$ will determine the paths for 1.1→1.2 and $DSC_2$ will determine the paths for 2.1→2.3. These paths have been shown in Figure 8.1 by a single thin dashed line, single

- For intra-EN SLAs the heuristic does not find the paths using external links. If there are shorter paths through the external links for the intra-EN SLAs, then the two algorithms will definitely give two different sets of paths.
- The calculation of shortest paths in the global network and in an EN is different from the calculation of shortest paths in the merged EN. Floating point precision errors during these calculations might calculate different shortest path in the following cases for inter-EN SLAs if there are multiple shortest paths with the same delay.
    - The shortest paths in the global network may not use the same external links as the shortest paths in the merged EN, because both the paths are identical with respect to associated delay.
    - The shortest sub paths in an EN may not be the sub paths of the shortest paths in the merged EN.
- The searching algorithm for the next shortest path in the presented heuristic is different from the centralized $K$ shortest paths algorithm. So there is a chance of selecting two different sets of paths if there are paths with the same delay.

## 8.7 Admission Control and QoS Adaptation Heuristics in a DSC

We run D-HEU or A-HEU once at the end of each epoch using the new batched SLAs over the last epoch and the existing SLAs. To apply these heuristics on these SLAs we need the $K$ candidate paths for each new SLA. A feasible path satisfying the specified delay for a QoS level represents an item in a group (an SLA) of the MMMKP. Here, we determine the $K$ candidate paths once for each SLA. The links of an EN are completely controlled by the DSC associated with it. Allocation of bandwidth on an external link represented by $x_i.y_i \rightarrow x_j.y_j$ and we assume that it is controlled by the DSC of the EN numbered by $\min(x_i,x_j)$.

A null item with a null QoS level, a null path, and no value is added in the list of items for each new SLA submitted in the last epoch. The null item of a newly submitted SLA is

initially selected before running D-HEU or A-HEU. If the selected item of a new SLA remains null after solving the MMMKP, that SLA is considered to be rejected and may wait for the next epoch to reattempt admission. If a non-null item is selected by the heuristic then that SLA is considered to be admitted. The nuil item is removed from the list of alternatives when the SLA gets admitted with a QoS level on a particular path. During the calculation of partial feasibility of the items for upgrade or downgrades in D-HEU and A-HEU we need to check for the flags specified in the SLAs. If the upgrade or downgrade does not satisfy the up, down or path restriction flags then that upgrade or downgrade must be considered infeasible. This remark refers to the upgrade and downgrade done by D-HEU or A-HEU internally for calculating the final QoS level. An SLA will not be downgraded or upgraded for the next epoch when it gets admitted. If there is a change in network status due to link or node failure we do QoS adaptation by running D-HEU or A-HEU, but the flags specified in the SLAs are totally ignored. The pseudo code for admission control and QoS adaptation is as follows:

*active_sla_list*: The list of SLAs that are already admitted.

*batched_sla_list*: SLAs batched in the last epoch that are queued for admission.

*batched_QoS_list*$_i$: The list of QoS levels for the *i*th SLA of *batched_sla_list*.

*QoS_path_list*$_i$: The list of candidate paths satisfying different QoS levels of the *i*th SLA of *batched_sla_list*. Each element of the list is represented by (*candidate path*, *QoS level*) tuple.

*path_list* ← *determine_K_candidate_paths*(*batched_sla_list*)

// Determining candidate paths for those SLAs which were submitted in the last epoch. This requires distributed computing among the DSCs

*for i*←1 *to size*(*batched_sla_list*) *do*

    //Determining alternative paths for a QoS level

    *add* (*null,null*) *in the QoS_path_list*$_i$

    *for j*←1 *to size*(*batched_QoS_list*$_i$) *do*

        *for k*←0 *to size*(*path_list*$_i$) *do*

```
            if delay(path_list_ik) <delay(batched_QoS_list_ij) then        // Checking the delay constraint
                add (batched_QoS_list_ij, path_list_ik) in QoS_path_list_i
            endif
        endfor
    endfor
endfor
temp_sla_list ← active_sla_list + batched_sla_list                  // SLAs to be adapted
D-HEU (temp_sla_list) or A-HEU(temp_sla_list )
// Applying distributed heuristic to solve the MMMKP.
active_sla_list ← active_sla_list + accepted_slas(temp_sla_list)
// Determining active SLAs that have been admitted
rejected_sla_list ← rejected_slas(temp_sla_list)        // determining SLAs with null QoS level
wait_sla_list← wants_to_wait_for_the_next_batch(rejected_sla_list)
batched_sla_list← batched_sla_list – accepted_slas(temp_sla_list) – rejected_sla_list + wait_sla_list
                                                    // Determining new batch of SLAs
```

## 8.8 Complexity of DSCs

Consider $M$ interconnected ENs where each EN is controlled by a DSC. $n$ SLAs (each with $l$ QoS levels) have been submitted to the DSCs of the ENs. For convenience of analysis we make the following assumptions:

- There is no active SLA in the network. That is, the system is being initialized.
- $n$ SLAs are equally distributed among $M$ DSCs. There are $n/M$ SLAs seeking admission in each DSC.
- There are $M$ gateways in the global network (one for each EN).
- The network is fairly sparse, by which we mean that the number of links in the network is linearly proportional to the number of nodes. We assume $\frac{k_1 N}{M}$ internal links in each EN and $k_2 M$ external links connecting the gateways of ENs.

- Each candidate path of an SLA is routed through all the ENs. This is the worst case situation of distributed computing.

To find the $K$ candidate paths for $n/M$ SLAs by each DSC the following computations and messages are required:

- $O(n \log M) + O(K'n)$ floating point operations to find $K'$ shortest paths in the global network.
- $(M-1)$ messages to send requests to determine $K'$ shortest sub paths in other ENs by their DSCs.
- $O\left(n\frac{N}{M}\log\frac{N}{M}\right) + O\left(K'n\frac{N}{M}\right)$ floating point operations to determine the $K'$ shortest sub paths of $n$ SLAs ($n/M$ SLAs are local and $\frac{n}{M}(M-1)$ SLAs are from other DSCs).
- $(M-1)$ messages to send $K'$ shortest sub paths in an EN of $\frac{n}{M}(M-1)$ SLAs submitted to $(M-1)$ DSCs.
- $\frac{n}{M} \times 4K(M+1)$ floating point operations to determine delays of $2K$ paths for $n/M$ SLAs submitted to an EN. We get approximately $2K$ paths after joining the sub paths.
- $\frac{n}{M} \times O(K^2)$ floating point operations to find $K$ shortest paths from $2K$ paths for $n/M$ SLAs. We have to sort a list of $2K$ paths according to the delay on the paths.

To determine the operating QoS levels of the SLAs using D-HEU or A-HEU we need the following operations:

- $O\left(\frac{n}{M} \times Kl\right)$ floating point operations to determine the alternative paths for the QoS levels of the local SLAs.
- $O\left(n^2(Kl-1)^2\left(\frac{N}{M}+M\right)\right)$ or $O\left(\left(\frac{n}{M}\right)^2(lK-1)^2\frac{N}{M}\right)$ floating point operations to solve the MMMKP using D-HEU or A-HEU.
- $O(MnlK)$ or $O(M)$ messages to solve the MMMKP using D-HEU or A-HEU.

We can summarize the computational and message passing complexities in the following table.

*Table 8.1 Complexity of different types of SLA controller*

| Algorithms applied | Message Passing Complexity | Computational Complexity |
|---|---|---|
| A-HEU in DSC | $O(M)$ | $O\left(\left(\frac{n}{M}\right)^2(lK-1)^2\frac{N}{M}\right)+O\left(n\frac{N}{M}\log\frac{N}{M}\right)$ |
| D-HEU in DSC | $O(MnlK)$ | $O\left(n^2(Kl-1)^2\left(\frac{N}{M}+M\right)\right)+O\left(n\frac{N}{M}\log\frac{N}{M}\right)$ |
| I-HEU in centralized SLA controller | N/A | $O\left(Nn^2(lK-1)^2\right)+O(nN\log N)+O(KnN)$ |

## 8.9 Java Simulation of DSC Prototype

We have implemented D-SLAOpt, a Java simulation of DSC prototype using Sun's JDK 1.3.1 with Swing (Java's windowing toolkit), version 1.1.1. In addition, we used Sun's Java XML parser, JAXP, for message parsing. D-SLAOpt is the distributed version of SLAOpt, which is a Java simulation of a centralized SLA controller for a single Enterprise Network. Java sockets have been used extensively for communication among the D-SLAOpts (one for each Enterprise Network), to implement the heuristic to

determine $K$ candidate paths and the distributed algorithms for the MMMKP. In this section we present a brief description of D-SLAOpt. For a detailed description please see Shelford [88].

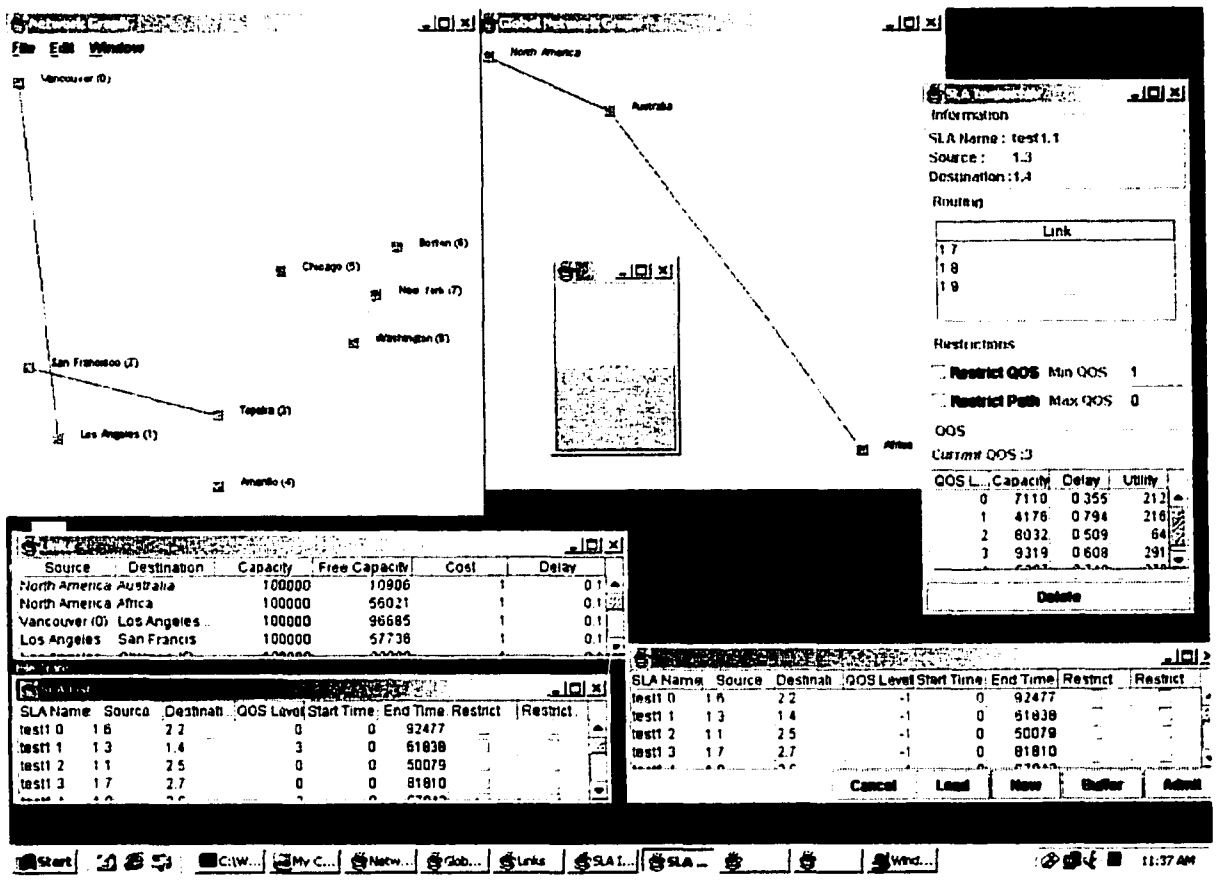


*Figure 8.6 Screenshot of D-SLAOpt for the EN with the nodes representing the cities of North America and the Global Network.*

D-SLAOpt consists of several components:

- The *Network Model* represents the Enterprise Network being controlled and the global network with the gateways of the ENs; it stores current capacities and node information.
- The *SLA Directory* stores active SLA profiles, including their QoS mappings.

- The *QoS Admission Controller Engine* implements D-HEU or A-HEU for admission control and QoS adaptation once in each epoch. As it is responsible for routing, in our implementation it also implements the heuristic to determine $K$ candidate paths for the SLAs.

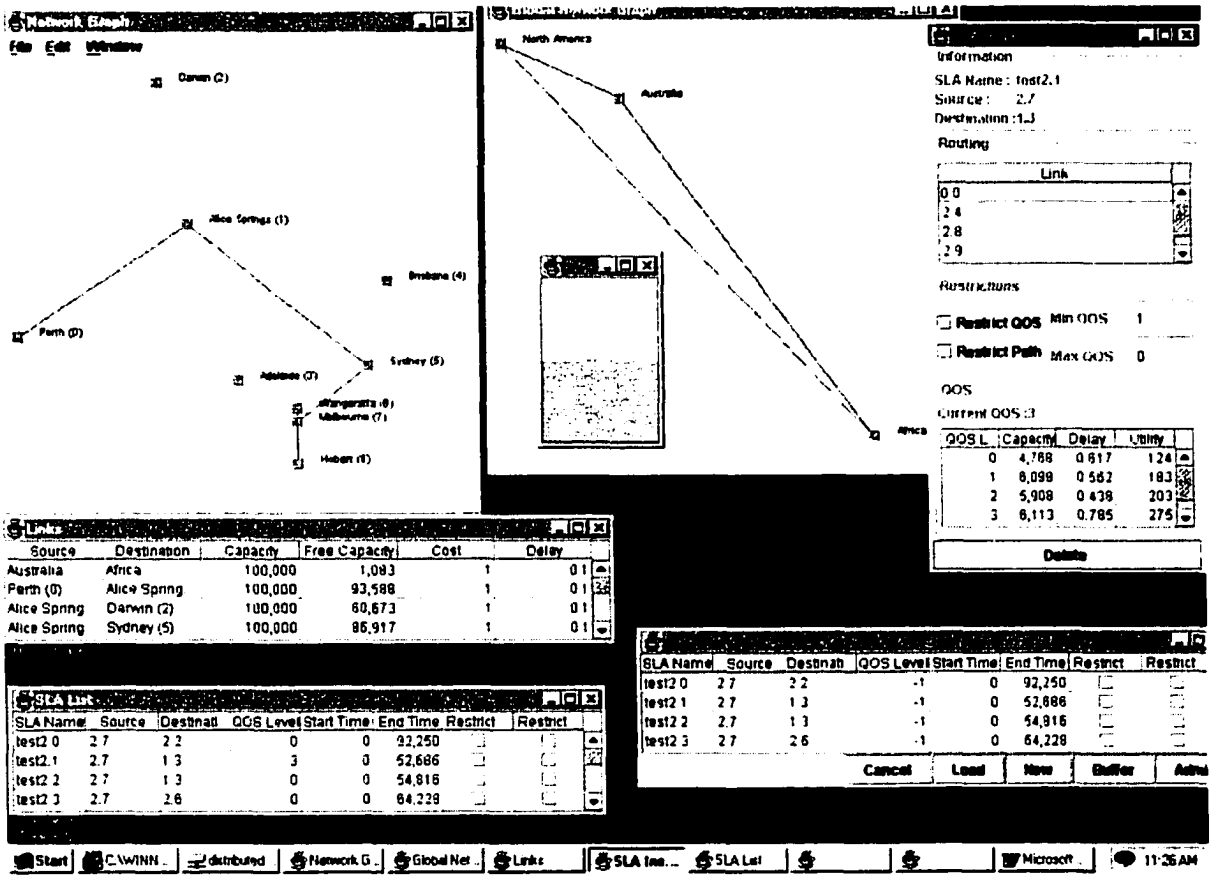


*Figure 8.7 Screenshot of D-SLAOpt for the EN with the nodes representing the cities of Australia and the Global Network.*

There are two types of interfaces:

- The *Graphical User Interface* displays the system's state to the network operator. It shows the Enterprise Network and global network as a linear graph. A simple display window called *Utility Meter* shows the total utility earned currently by the active SLAs. A table (Window *Links*) shows the condition (total capacity, free capacity and

delay) of each link of the Enterprise Network and the links of the global network controlled by D-SLAOpt.

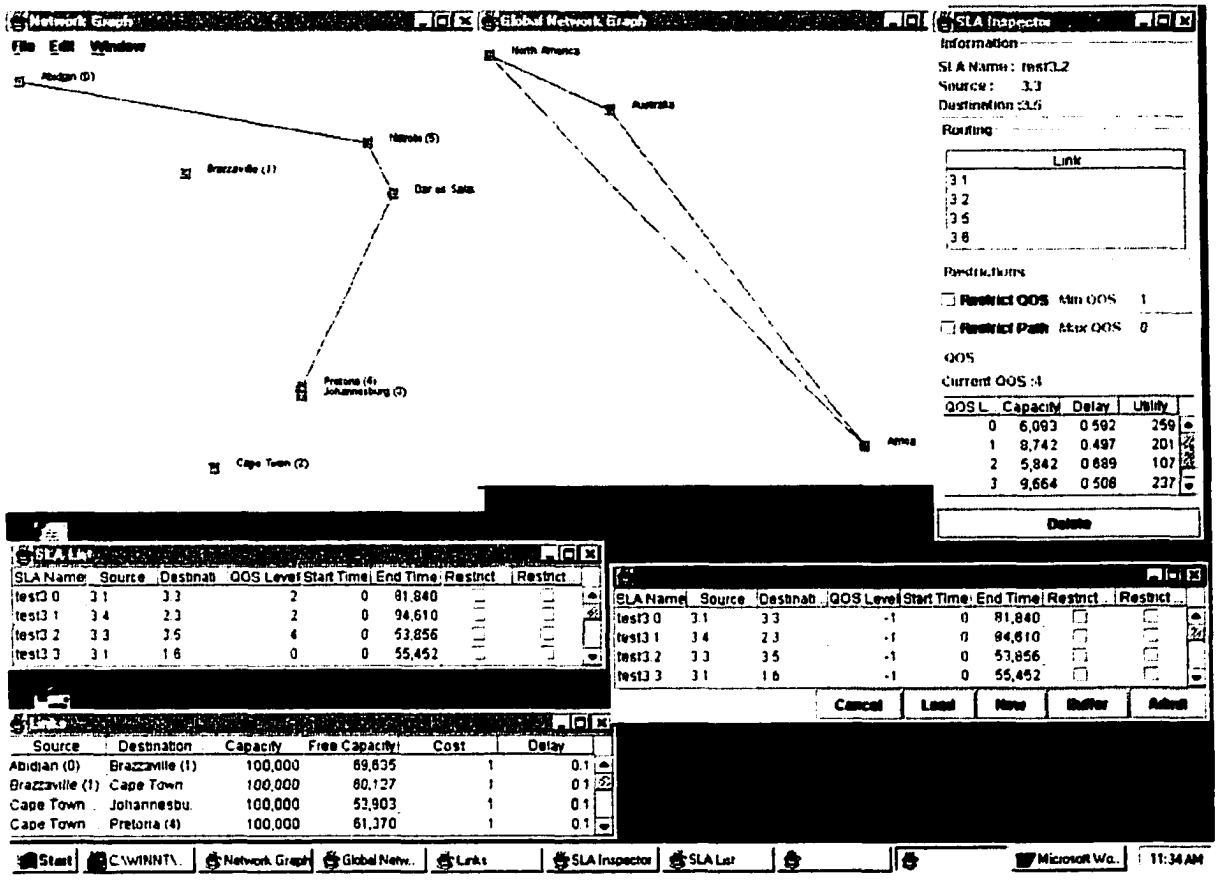


*Figure 8.8 Screenshot of D-SLAOpt for the EN with the nodes representing the cities of Africa and the Global Network.*

- The *SLA Interface* allows the operator to request SLA admissions manually or from an XML file (Window *SLA Admission Queue*). The active SLAs are shown in the window *SLA List*. The QoS levels of an SLA can be shown in a table (Window *SLA Inspector*).

### 8.9.1 Experimental Results

To measure the performance of the heuristics we tested them for a global network with component ENs representing cities of North America, Australia and Africa, as shown in Figure 8.6, Figure 8.7 and Figure 8.8. All links were identical and were given a token capacity of 100,000 Mbps and a latency of 0.1 ms. We calculated the 5 shortest sub paths in the ENs and the 5 shortest paths in the global network to find 5 candidate paths for each SLA. i.e., $K' = K = 5$.

Performance data were collected from randomly generated SLA batches of different sizes. Initially we submitted a batch of SLAs to the SLA admission controller. The initialized bandwidth requirements for a particular QoS level of an SLA varied with the size of the SLA batch in such a way as to assure that there was always contention for bandwidth in the network. Some SLAs were rejected for admission by the controller. We analyzed the performance of the admission controller over 20 (arbitrarily chosen) subsequent batches. We do not consider the results, which do not create contention for bandwidth in the network. For each of these batches we removed some (10% of the initial batch size) of the already admitted SLAs from the active SLA list and submitted some (12.5% of the initial batch size) new SLAs. These numbers were chosen arbitrarily to demonstrate the the performance of D-SLAOpt. The number of newly submitted SLAs was kept at 2.5% more than the number of SLAs leaving at the end of each epoch so that there was a contention for getting admission. In each batch approximately 50% were intra-EN SLAs. We ran three D-SLAOpts in the first three machines described in Table 6.4. To compare performance between the distributed and centralized version of the SLA controller, we ran SLAOpt for a network derived by merging the three networks representing the cities of North America, Australia and Africa plus the Global network on the 2$^{nd}$ machine (Solver 2) described in Table 6.4. We generated the same set of SLAs for both simulations. Thus the two simulation packages became comparable. The following data were collected from the experiment and are presented in Figures 8-9 to 8-13 and Table 8.2:

- Total revenue earned by the controllers,
- Time required to do admission control and QoS adaptation,
- Average number of SLAs in each batch,
- Average number of rejected SLAs after admitting a batch, and
- Average number of messages required doing admission control and QoS adaptation of an SLA batch.

The SLAOpt program is written in Java. Java as a language is currently unsuitable to writing time critical, highly processor dependent algorithms for general use. However, it makes an exceptional modelling language. It is considerably slower than a time-optimized implementation, however it provides reasonable facilities for testing trends of processing time. This becomes even more to the point in this environment, as the Java Virtual Machine must also support the GUI interface. We can expect much faster response if the algorithm is recoded in C. D-HEU might not improve as much as A-HEU by recoding as the execution time of D-HEU is more related to message passing.

*Table 8.2 Comparison of earned revenues by the heuristics used in SLAOpt and D-SLAOpt*

| Initial batch size | Revenue earned by D-HEU in D-SLAOpt (000) | Total Revenue by I-HEU in SLAOpt (000) | Total Revenue by A-HEU in D-SLAOpt (000) | Revenue earned by D-HEU in D-SLAOpt with respect to I-HEU in SLAOpt (%) | Revenue earned by A-HEU in D-SLAOpt with respect to I-HEU in SLAOpt (%) |
|---|---|---|---|---|---|
| 150 | 822 | 840.45 | 660.539 | 97.8 | 78.59 |
| 300 | 1527.2 | 1538.57 | 1184.416 | 99.26 | 76.98 |
| 450 | 2216.23 | 2204.98 | 1755.722 | 100.51 | 79.63 |
| 600 | 2963.57 | 3017.48 | 2352.375 | 98.21 | 77.96 |
| 750 | 3945.29 | 3940.32 | 3116.289 | 100.13 | 79.09 |
| 900 | 4496.43 | 4539.1 | 3582.628 | 99.06 | 78.93 |
| 1050 | 5502.97 | 5490.77 | 4321.71 | 100.22 | 78.71 |
| 1200 | 6170.24 | 6268.66 | 4816.357 | 98.43 | 76.83 |
| 1350 | 6844.26 | 6888.54 | 5516.362 | 99.36 | 80.08 |
| 1500 | 7602.03 | 7606.02 | 5728.825 | 99.95 | 75.32 |

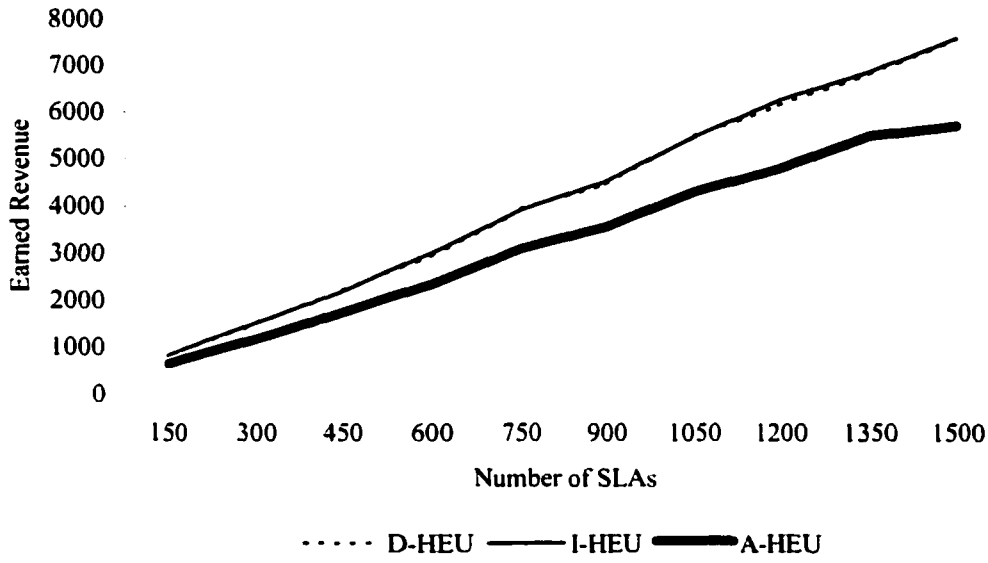


*Figure 8.9 Earned revenues by using different heuristics in SLAOpt and D-SLAOpt*

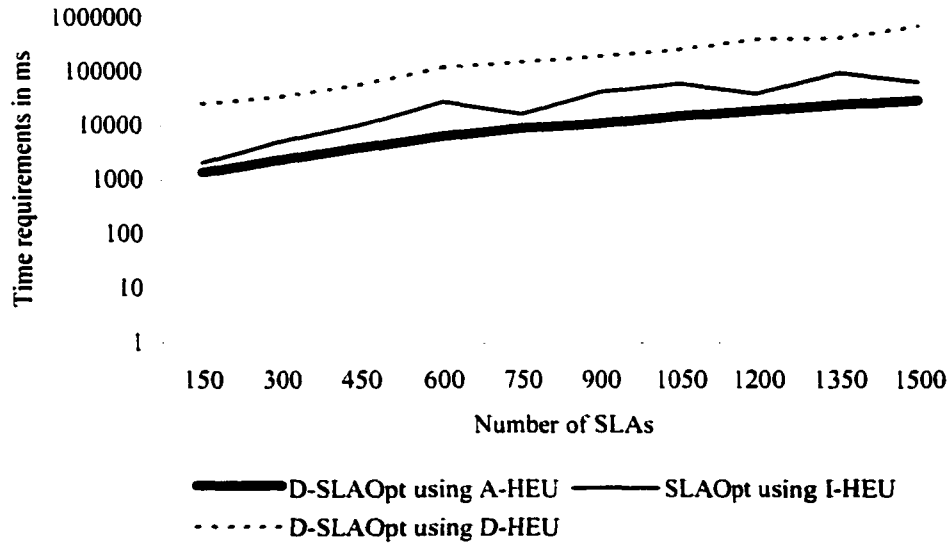


*Figure 8.10 Time required by SLAOpt and D-SLAOpt for doing admission control and QoS adaptation of new batch ( approximately 12.5% SLAs are new).*

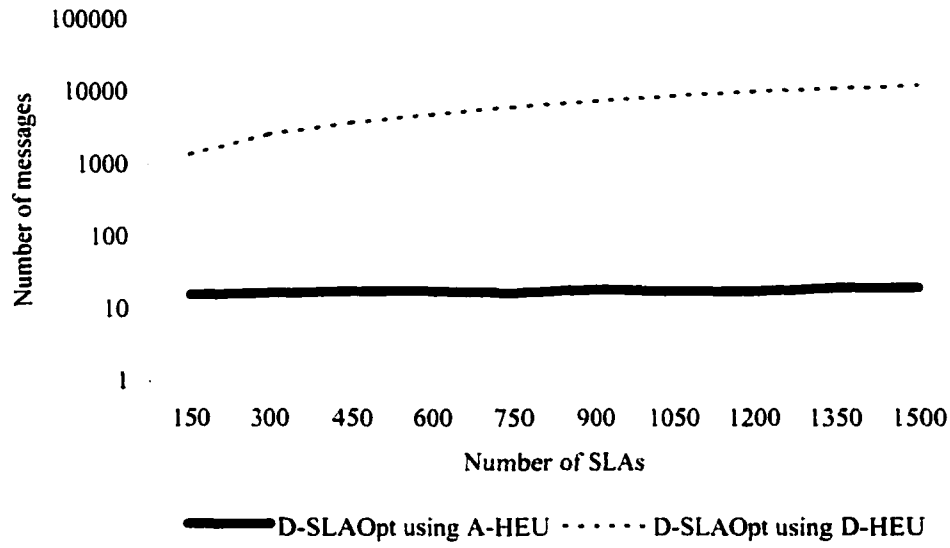


*Figure 8.11 Number of messages required by the D-SLAOpt for doing admission control and QoS adaptation of each new batch using two different distributed heuristics*

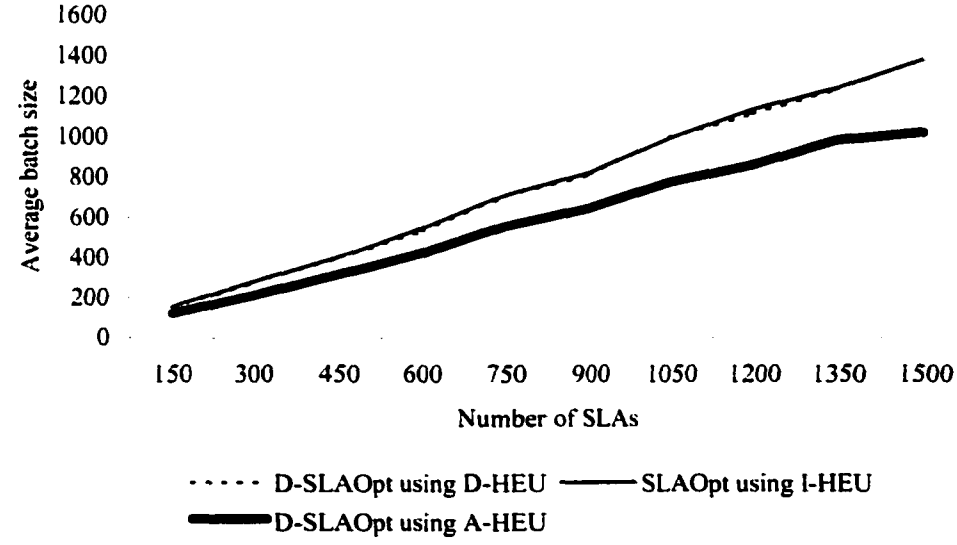


*Figure 8.12 Average batch size using different heuristics in SLAOpt and D-SLAOpt*

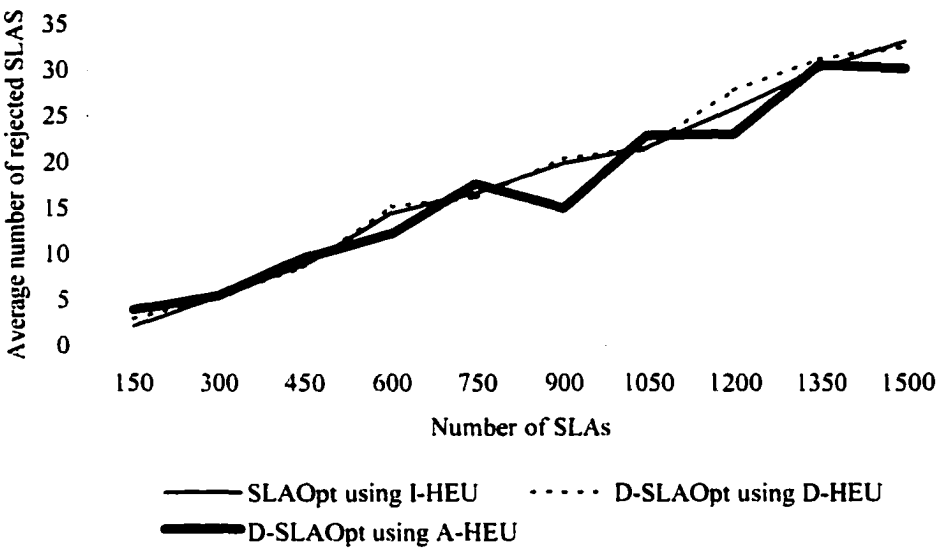


*Figure 8.13 Average rejected SLAs in each batch using different heuristics in SLAOpt and D-SLAOpt*

## 8.9.2 Observations

- The reasons for the difference in earned revenues by D-SLAOpt using D-HEU and SLAOpt using I-HEU are as follows:
  - The delays are assumed to be identical for each link (0.1 ms). There are multiple path sets from a source to a destination with the same delay in the experimental network. For a particular SLA, we may get a set of $K$ candidate paths in D-SLAOpt, which is different from the $K$ shortest paths in SLAOpt.
  - Floating point precision errors may contribute to differences in revenues earned by D-HEU and I-HEU.
- D-SLAOpt using A-HEU earns approximately 80% of the total revenue earned by D-SLAOpt using D-HEU in this experiment.

- The time required to do admission control and QoS adaptation in D-SLAOpt using D-HEU is impractical because of the large numbers of messages.
- As A-HEU requires a negligible number of messages compared to D-HEU, the time required by D-SLAOpt using A-HEU is practical for real-time SLA admission controllers.
- The time required by D-SLAOpt using A-HEU is less than that required by SLAOpt using I-HEU. This implies that distributed SLA controllers scale better than centralized SLA controllers, although we have to sacrifice revenue (approximately 20% according to this experiment).
- D-SLAOpt using A-HEU rejects more SLAs than D-SLAOpt using D-HEU or SLAOpt using I-HEU as A-HEU does not fully consider the resources of the other ENs.
- The following observations validate our experiment
  - The number of messages remains almost constant for A-HEU, but increases linearly for D-HEU with the increase in the number of SLAs in D-SLAOpt. This complies with the analysis of the algorithm.
  - Earned revenues and active batch sizes increase linearly with the increase in the number of SLAs.
  - The average number of rejected SLAs in subsequent batches, after the initial batch, by D-HEU, I-HEU or A-HEU is approximately 2.5% of the initial batch size as we deleted 10% SLAs and admitted 12.5% SLAs of the initial batch size in each batch.

## 8.10 Chapter Summary

This chapter presents necessary heuristics to implement D-SLAOpt. From the complexity analysis and experimental data we find that D-SLAOpt using A-HEU is appropriate for real-time controllers in interconnected ENs. D-SLAOpt using A-HEU can do admission control of 150 new SLAs and QoS adaptation of 1350 old SLAs (total 1500 SLAs) in 10 seconds. This gives us a real-time DSC with 15 second epochs taking admission and rejection decisions for 150 new SLAs in 10 seconds. The next chapter concludes this dissertation with recommended future research work and a summary of major contributions.

# 9. Conclusions

This chapter summarizes the major contributions of this dissertation and presents suggestions for future research.

## 9.1 Major Contributions

We presented a model for admission control and QoS adaptation of sessions in a Distributed Multimedia Server System. New algorithms for solving the model were presented and applied in both simulations and prototype implementations of optimal admission controllers. The following subsections summarize these contributions.

### 9.1.1 The Distributed Utility Model

The Utility Model – Distributed (UM-D) is an extension of the Utility Model proposed by Khan[46]. The UM-D captures the admission control and QoS adaptation strategy for multimedia session requests in a Distributed Multimedia Server System (DMSS). In this model we assume a DMSS with multiple servers providing multimedia service to multiple Enterprise Networks. The components of the multimedia streams are distributed over the servers and the components of a multimedia stream can be supplied to a user by different servers. CPU cycles, I/O bandwidth and memory in the servers and communication bandwidth in the network are considered as resources. The users issue requests with varying QoS levels, specifying audio or video quality, jitter, delay and offered price for the multimedia service. The UM-D demonstrates a strategy of selecting QoS levels and the servers providing multimedia stream components for each of the admitted sessions, such that maximum revenue is received from the users enjoying multimedia services.

We presented both centralized broker and fully distributed architectures for the admission controller of a DMSS. In the centralized broker approach the resources are distributed but

the computation is centralized on one machine. The admission control and QoS adaptation problems for this architecture can be mapped to the Multidimensional Multiple Choice Knapsack Problem (MMKP). In the other case we proposed multiple controllers to deal with fully distributed admission control and QoS adaptation. This situation can be mapped to the Multidimensional Multiple-choice Multi Knapsack Problem (MMMKP), a new variant of the knapsack problem.

### 9.1.2 Heuristic Algorithms for the MMKP

M-HEU, a new heuristic to solve the MMKP, was developed. This is actually a modified version of HEU proposed by Khan [46]. I-HEU, the incremental version of M-HEU, was also presented in this dissertation. This heuristic finds the solution from a previously determined solution, thus requiring less computation than M-HEU. The order of the worst-case computational complexities for HEU, I-HEU and M-HEU are identical. I-HEU is applied to do admission control on a new batch of requests and QoS adaptation of the previously admitted requests in SLA controllers for Enterprise Networks and brokers of DMSSs. We also developed another heuristic, C-HEU, by applying the convex hull approach. The worst-case computation complexities of I-HEU and C-HEU are $O(mn^2(l-1)^2)$ and $O(nl \log nl + nlm)$ respectively ($n$ = number of groups, $l$ =number of items and $m$ = number of resource dimension in the MMKP). The optimalities achieved by HEU, M-HEU, I-HEU and C-HEU are approximately 94%, 97%, 97% and 92% respectively.

### 9.1.3 Simulation of a set of Media Server Farms

A DMSS is used to route the content of multimedia streams over an Enterprise Network. A set of Media Server Farms is a DMSS providing videos to users with guaranteed QoS. We presented a simulation of admission control and QoS adaptation of requests for VoD services from users connected to a network. In this simulation we assumed that the Enterprise Network had sufficient bandwidth to carry multimedia streams for all users

from the servers, thereby suppressing the influence of the network on the problem and allowing concentration on the question of optimal server selection. Thus, the broker for the media server does admission control based solely on available resources in the media servers using heuristics for solving the MMKP. The performance of different heuristics for admission control and QoS adaptation were analyzed from the simulation results. This work is a contribution to the interesting area of optimal server selection in content routing.

### 9.1.4 Distributed Heuristics for Solving the MMMKP

In an MMMKP, we proposed the existence of one solver for each knapsack, picking items from the groups. Two distributed heuristics were developed to solve the MMMKP with this approach. D-HEU executes the distributed version of the steps of I-HEU to solve the MMKP. This heuristic requires $O(nlM)$ message passing ($M$ = number of knapsacks). A-HEU solves the MMMKP by arbitrating among the solvers with $O(M)$ complexity. The total value of the items picked by D-HEU is exactly the same as that obtained using I-HEU, assuming there are no floating-point precision errors. The time requirement for D-HEU is impractical for real time applications, as it requires large numbers of messages. A-HEU, however, requires less time than I-HEU, but must sacrifice approximately 10% of optimality on average. We presented pseudo code for A-HEU and D-HEU with an analysis of computational and message passing complexity.

### 9.1.5 Distributed SLA Controller for Interconnected Enterprise Networks

We introduce a new problem; the serving of SLAs in interconnected Enterprise Networks, owned and controlled by the autonomous subsidiaries of an organization. Each Enterprise Network has one SLA controller doing admission control and QoS adaptation of the SLAs submitted to it. Each SLA controller participates in a distributed computation to determine the path and QoS of those SLAs, which have source and destination in two

different Enterprise Networks. Distributed admission control and QoS adaptation of the SLAs using D-HEU or A-HEU was presented with a complexity analysis.

### 9.1.6 Java Simulation of Distributed SLA Controller

We developed D-SLAOpt, a Java simulation for Distributed SLA Controllers in interconnected ENs. This simulation demonstrates the admission and QoS adaptation of the SLAs in the ENs. We utilized this simulated system to obtain preliminary performance data by admitting batches of SLAs. A comparative discussion between centralized and distributed SLA controllers was presented. We found D-SLAOpt using A-HEU to be more scalable than SLAOpt using I-HEU.

## 9.2 Qualitative Comparison of Different Algorithms

*Table 9.1 Basic principle, application and performance of the newly developed algorithms presented in the dissertation*

| Algorithm | Basic Principle | Application | Performance |
|---|---|---|---|
| M-HEU | Modified version of HEU for solving the MMKP. It applies the same approach as HEU. | It can be applied to solve menu-planning, and admission controlling in ENs and in multimedia server systems. | It requires more time than HEU but achieves better optimality. |
| I-HEU | It is an incremental version of M-HEU. | Specially used in on line admission controllers for multimedia sessions, where sessions continually finish, while new requests arrive. | The worst-case computational complexity is the same as M-HEU but it scales better. |
| C-HEU | Finds the selected items based on the convex hull approach. This is an incremental heuristic. | The area of application is the same as I-HEU. We used C-HEU and I-HEU in the simulation for the set of Media Server Farms. | It has lower order of complexity than I-HEU, but worse optimality. |
| D-HEU | Distributed version of I-HEU for solving the MMMKP. | This heuristic can be applied to perform non - real time admission in distributed admission controllers. | It renders the same optimality as I-HEU but requires more time for message passing. |
| A-HEU | Distributed version of I-HEU; based on arbitration among autonomous solvers. | This heuristic can be applied to perform real time admission in distributed admission controllers such as D-SLAOpt and Distributed Utility Model Engines. | Scales better than I-HEU but achieves lower optimality. |

## 9.3 Future Research Work

We suggest the following research plans on heuristics for solving the variants of Knapsack Problem, the Utility Model-Distributed and its application to distributed multimedia server systems and Enterprise Networks as future research work.

- *Average Case Analysis:* We presented the worst-case complexities of the heuristics for solving the MMKP and MMMKP. The analysis of time requirement complexity and achieved optimality in the average case is a very interesting research topic in theoretical computer science.
- *Performance Analysis of the SLA Controllers:* Performance analysis of SLAOpt and D-SLAOpt is important before implementation. Different performance measures can be collected by running the simulation for a long period for different system parameters.
- *Implementation of SLA Controller Prototypes:* Development of a Centralized or Distributed SLA Controller prototype using switches and optical fibre links could be an important research project.
- *Provision of Reservations by the Admission Controllers:* As there is a chance of rejection for every request, some users may be interested to make reservations days or even weeks before the starting time of the session. This requires more computation as it increases the resource dimension by including the durations of sessions.
- *Design of Reliable SLA Controllers:* We can design a reliable SLA controller by using the group of reliable $k$ shortest paths [81]. Reliability can be improved if bandwidth on all the paths of a group is reserved by the SLA controller.
- *Statistical Prediction to Achieve Better Revenues*: Accepting a request may cause us to lose the opportunity of accepting a regular more profitable session. We can use

statistical techniques to predict regular future requests. Resources should be reserved for those future requests.

- *Application of the Utility Model to a Multimedia Server System with an Underlying Enterprise Network:* In this case, the controller ensures both the delivery of multimedia streams from the servers and transmission of multimedia traffic through a path to the user.
- *MMKP as a Resource Manager in Multiple Layers of Service Providers:* We can run the heuristics of the MMKP to do admission control of resources in the layers of servers, Enterprise Networks, Lambda Vendors etc. where the requests for resources are placed from one layer to the next lower layer.
- *Maximization of Revenue with Guaranteed QoS for Mobile Network Provider:* The Utility Model is appropriate for admission control QoS adaptation of the calls from the cell phones to allocate bandwidth from the cell user to the base station.
- *Parallel Algorithms to Solve the MMKP:* We can design more scalable admission controllers by applying parallel versions of the heuristics of the MMKP.

# 10. Appendix

## 10.1 Pseudo code of D-HEU

// Following Notations are used in the pseudo code

$x \rightarrow y$: Send the result from the process $x$ to the set of machines or processes denoted by $y$

$p \leftarrow q$: Assign the variable $p$ to the value $q$

//Global variables

*no_of_solvers*: Total number of knapsacks or solvers in the MMMKP

*global_total_value*: Total value of the items picked by all the solvers.

//Following values of variable *purpose* defines different purposes of finding PCAR & TF, local and global candidate groups and items.

*FEASIBILITY_BY_DOWN* : To find a feasible solution by downgrade from a infeasible solution.

*FEASIBILITY_BY_UP* : To find a feasible solution by upgrade from a infeasible solution.

*FEASIBLE_UPGRADE* : To make a feasible upgrade from a feasible solution.

*INFEASIBLE_UPGRADE*: To make an infeasible upgrade from a feasible solution.

*FEASIBLE_DOWNGRADE*: To make a feasible downgrade from an infeasible solution.

// Following are the subroutines used by the D-HEU algorithm

*calculate_local_most_inf_res* ( ) : returns the most infeasible resource among the resources of the local knapsack with its infeasibility factor, the ratio of consumed resource to total resource.

*calculate_global_most_inf_res* ( ) : returns the most infeasible resource (among all the resources of the knapsacks) with its infeasibility factor. It is calculated by comparing the infeasibility factor of the local most infeasible resources.

*calculate_pcars&pfs* ( *purpose* ): Determines and returns the PCARs & PFs of the possible items of all the groups of all the solvers for *purpose.*

*calculate_local_candidate*( *purpose* ): Determines and returns the local candidate group and item of the groups and items of the local solver for *purpose.* It also returns the feasibility of the local candidate.

*calculate_global_candidate*( *purpose* ): Determines and returns the global candidate group and item of the solvers for *purpose.* It also updates the selection of global candidate group and returns status of the

candidate and solution denoted by *FEASIBLE* (solution is feasible), *INFEASIBLE* (solution is infeasible), *NO_CANDIDATE* (no candidate found).

*calculate_local_total_value*( ): Determines the total value of the picked items of the local groups of a solver.

*read_groups_from_other_solver*(*message*): This procedure reads the groups from another solver and stores in a two dimensional array of groups indexed by solver number and group number. These are *non-local groups* of a solver. The items of the non-local groups for a solver are specified by the resources of the solvers only. A solver does not send all the resource requirements of an item to every solver.

*read_pcars&pfs_other_solver*(*message*): This procedure reads PCARs and PFs of the items of the local groups for another solver and saves in the array of PCARs and PFs associated with each group.

*read_local_candidate*(*message*) : This procedure reads a local candidate from another solver and stores in a local candidate array at the specified location for the solver.

*calculate_tcars&tfs*( ) : This procedure calculates the TCARs & TFs by adding the PCARs and multiplying PFs.

*read_message_from_solver* (*s*): Reads message sent by Solver *s*.

*determine_message_type*(*message*) : Returns the type of message in *message*.

//The following routines must be synchronized because any of these routines may be called concurrently by more than one receiver thread.

*synchronized read_local_total_value*(*message*): This procedure reads a local total value from another solver and updates the *global_total_value*.

*synchronized read_local_most_inf_res*(*message*): This procedure reads a local most infeasible resource from another solver and updates the global most infeasible resource.

## *Procedure Begin D_HEU( )*

// This procedure is run by the sender thread of the solver. This is the main procedure running the heuristic of the MMMKP.

local groups → other solvers

*wait_for* (*no_of_groups_msgs*)

*//Step 1: Finding a feasible solution*

Initialize selected items to the lowest valued items of each new group

*if find_dist_feasible_solution_by_purpose*(*FEASIBILITY_BY_DOWN*)=*false then*
//trying by downgrading selected items
*if find_feasible_solution_by_upgrade*(*FEASIBILITY_BY_UP* ) = *false then*
//trying by upgrading selected items
*No feasible solution found for the MMMKP*
*return;*

```
        endif
    endif     //end of Step 1
    repeat_loop←true //loop control variable
    do
        do  //Step 2: Finding all feasible upgrades
            status←find_candidate_for_purpose(FEASIBLE_UPGRADE)
        while status= FEASIBLE
        //Step 3: Improving total value by an infeasible upgrade followed by downgrades
        save_solution( )
        status←find_candidate_for_purpose(INFEASIBLE_UPGRADE)
        if status ≠ NO_CANDIDATE then
            do
                status←find_candidate_for_purpose(FEASIBLE_DOWNGRADE)
            while status= INFEASIBLE
            if status= NO_CANDIDATE then   // Step 3 could not upgrade
                revive_solution( )      // No more upgrading
                Solution found
                repeat_loop←false
            end if
        end if
    while repeat_loop =true    //Step 3 upgrades so try Step 2 again
    calculate_local_total_value( ) → other solvers
    wait_for(no_of_total_value_msgs)
    return global_total_value.
    End Procedure
```

### *Procedure Begin Receiver_thread( Solver s )*

//This procedures reads messages sent from Solver *i*, analyze them and takes action. There are (*no_of_solvers* – 1) threads like this, one for each solver of the knapsack.

```
    while true
        message=read_message_from_solver (s)     //reading message from Solver s
        message_type=determine_message_type(message)   //decoding the message
        begin case message_type
        case "Groups":  // partial resources of the items of another solvers
            read_groups_from_other_solver(message)
            increment(no_of_group_msgs)
```

```
        case "PCARs & PFs":     //PCARs and PFs from other solvers
            read_groups_from_other_solver(message)      //saving PCARs and PFs
            increment(no_of_pcar_msgs)
        case "Local Candidate":                 //local candidate from other solvers
            read_local_candidate(message)       //Storing local candidate
            increment(no_of_local_candidate_msgs)
        case "Local Total Value":               // total value of another solver
            read_local_total_value(message)     //Must be synchronized
            increment(no_of_total_value_msgs)
        case "Local Most Infeasible Resource":  // reading k_ms and f_{k_ms} for another knapsack
            read_local_most_inf_res(message)    //Must be synchronized
            increment(no_of_local_most_inf_res_msgs)
        end case
    end while
end Procedure
```

*Procedure Begin wait_for (counter)*

```
//This procedure waits if counter does not get the value no_of_solvers
    begin synchronized
        if (counter < no_of_solvers) wait on monitor counter
    end synchronized
end Procedure
```

*Procedure Begin increment (counter)*

```
//This synchronized procedure increases the value of counter by 1. If the counter reaches no_of_solvers
then it notifies the monitor specified by counter
    begin synchronized
        counter← counter + 1
        if (counter =no_of_solvers) then
            notify monitor counter
            counter←0
        endif
    end synchronized
```

*end Procedure*

### *Procedure Begin find_candidate_for_purpose( purpose )*

//This procedure determines the PCARs & PFs, TCARs & TFs and local candidate for *purpose*, of the items of the local groups. Finally, it returns the global candidate by comparing this local candidate with the local candidates sent by the other solvers.

*calculate_pcars&pfs* ( *purpose* )→ other solvers //Calculating PCARs & PFs

*wait_for*(*no_of_pcars&pfs_msgs*) //waiting for PCARs & PFs from other solvers. TCARs & TFs are calculated by this time

*calculate_tcars&tfs*( )

*calculate_local_candidate*( *purpose* )→ other solvers //calculating local candidate

*wait_for* (*no_of_local_candidate_msgs*) //waiting for local candidate from other solvers

*return calculate_global_candidate*( *purpose* ) //calculating global candidate

*end Procedure*

### *Begin Procedure find_dist_feasible_solution_by_purpose( purpose )*

This procedure finds a feasible solution of the MMMKP by *purpose* (upgrade or downgrade). If it fails to find any feasible solution it returns with *false* otherwise it returns *true.*

*calculate_local_most_inf_res* ( )→ other solvers

*wait_for*(*no_of_local_most_inf_res_msgs*)

$K_m \leftarrow$ *calculate_global_most_inf_res*( )

*if* $f_{K_m} \leq 1$ *then*
    *return true;*
*endif*
*do*
    *status* ←*find_candidate_for_purpose* (*purpose* )
    *if status = NO_CANDIDATE then* //No suitable item available
        *return false*
    *endif*
*while status=INFEASIBLE* //still infeasible, try again
*return true*
*end procedure*

## 10.2 Pseudo code of A-HEU

This algorithm also uses the concept of sender and receiver threads like D-HEU. Routines and variables used by D-HEU have also been used in A-HEU to implement synchronized counters.

*global_proposed_list*: The list of the proposed selected items from all solvers.

*sort_by_value_per_PCAR* (*proposed_list*) : Sorts the items in *proposed_list* according to the change of value per PCAR.

*find_local_feasibility_index*(*proposed_list*): Finds how many items from the beginning of *proposed_list* can be picked by a solver subject to its resource constraints.

*calculate_global_feasibility_index*( ) : This routine compares the local feasibility indexes and returns the lowest one.

*read_proposed_list*(*message*): This procedure reads and saves the proposed items in *message*.

*read_local_feasibility_index*(*message*): This routine stores the local feasibility index of a solver in *message* at a specified location of the arrays for local feasibility indexes.

*merge_proposed_lists*( ): The procedure merges the saved proposed lists into a global list. The merging order (Solver 1 to Solver *M*) is kept the same in all the solvers.

*Procedure Begin A_HEU*( )

//This is the main procedure of A-HEU. This is run by the sender thread.

*wait_for* (*no_of_group_msgs*)

//Finding whether lowest valued items gives a feasible solution or not.

Initialize selected items to the lowest valued item of each new group

*Run Step 1 of I-HEU* to determine local proposed selection list→ other solvers

*if no feasible solution found then*

Soluition Not Found → other solvers

*exit*

*end if*

*wait_for* (*no_of_local_proposed_list_msgs*)

//proposed selected items for a feasible solution from other solvers are coming

*find_local_feasibility_index* (*global_proposed_list*) → other solvers

*wait_for* (*no_of_local_feasibility_msgs*) //feasibility index from the other solvers are coming

*index* ← *calculate_global_feasibility_index* ( )

*if index* ≠ *size*(*global_proposed_list*) *then* // The selected items are not feasible.

Soluition Not Found → other solvers

*exit*

*else*

update the selection of all the groups of *global proposed list.*

*end if*

*do*

*save_solution*( )

*saved_total_value* ← *global_total_value* // saving global total value

*Run Step 2 & 3 of I-HEU* to determine local proposed selection list→ other solvers

*wait_for* (*no_of_local_proposed_list_msgs*)

//proposed selected items for an upgraded solution from other solvers are coming

*global_proposed_list* ← *merge_proposed_lists*( )

*sort_by_value_per_PCAR* (*global_proposed_list*) //sorting proposed items by the solvers

*find_local_feasibility_index* (*global_proposed_list*) → other solvers

*wait_for* (*no_of_local_feasibility_msgs*) //feasibility index from the other solvers are coming

*index* ← *calculate_global_feasibility_index* ( )

update the selection of the groups of *global proposed list* from the beginning up to *index*

*calculate_local_total_value*( ) → other solvers // local total value

*wait_for*(*no_of_local_total_value_msgs*) //calculation of global total value in the meantime

*while global_total_value* > *saved_total_value* // looking for another arbitration

*revive_solution*( ) //No more arbitration

*return global_total_value.*

*End Procedure*

### *Procedure Begin Receiver_thread( Solver s )*

// This procedure reads messages from Solver *s*, decodes the message and takes action according to the message.

*while true*

*message*←*read_message_from_solver*(*s*) //receiving message

*message_type*←*determine_message_type*(*message*) //decoding message

*begin case message_type*

*case "Groups":* //New groups from other solvers

*read_groups_from_other_solver*(*message*)

```
            increment(no_of_group_msgs)
        case "Proposed Selected Item List":
            read_proposed_list(message)
            increment(no_of_local_proposed _list_msgs)
        case "Local Feasibility Index":
            read_local_feasibility_index(message)
            increment(no_of_local_feasibility_msgs)
        case "Local Total Value":          // Local total value of Solver i
            read_local_total_value(message)        //Must be synchronized
            increment(no_of_total_value_msgs)
        case "solution Not Found":
            Solver s could not find a feasible solution.
            exit
        end case
    end while
    end Procedure
```

## 10.3 Algorithm for finding $K$ Candidate paths by a DSC in interconnected ENs

// The following are necessary variables and functions to describe the heuristic.

$SLA_i$ : The $i$th SLA in the DSC

$n_i$: Number of SLAs submitted to the $i$th DSC

$S_i$,$D_i$ : The source and destination node of $SLA_i$

$d_{max}(SLA_i)$: The maximum allowable delay for the QoS levels of $SLA_i$

$d(L)$ : The delay on a link $L$

$d(p)$ : The delay on path $p$

$GN$: The global network with external links and the gateways.

$e(i)$ : The Enterprise Network containing node $i$.

$L_{ikt}$: The $t$th external link on the $k$th path for $SLA_i$ in $GN$

$\bar{p}_{ikt}$ : The $t$th (source, destination) pair that indicates a sub path inside an EN on the $k$th path for $SLA_i$ in $GN$.

$S(\bar{p}_{ikt})$ : Source of the sub path denoted by $\bar{p}_{ikt}$

$D(\bar{p}_{ikt})$ : Destination of the sub path denoted by $\bar{p}_{ikt}$

$q_{ikr}$ : The $r$th shortest sub path from the source $S(\vec{p}_{ikt})$ to the destination $D(\vec{p}_{ikt})$ over the internal links of $e(S(\vec{p}_{ikt}))$.

$\vec{l}_{iktr}$ : The vector of links representing the sub path $q_{iktr}$.

//Following pseudo code describes the heuristic

*for* $i \leftarrow 1$ *to* $n_i$ *do*

$X_{Si}.Y_{Si} \leftarrow$ A node in $GN$ which also represents a gateway in $e(S_i)$

$X_{Di}.Y_{Di} \leftarrow$ A node in $GN$ which also represents a gateway in $e(D_i)$

$\vec{P}_i \leftarrow$ Paths from $X_{Si}.Y_{Si}$ to $X_{Di}.Y_{Di}$ in $GN$

Where, the $k$th shortest path $P_{ik} = \{L_{ik1}, L_{ik2}, \cdots, L_{ikl_{ik}}, \vec{p}_{ik1}, \vec{p}_{ik2}, \cdots, \vec{p}_{ikm_{ik}}\}$ satisfying the following constraints $\sum_{t=1}^{l_{ik}} d(L_{ikt}) \le d_{\max}(SLA_i)$

*for* $t \leftarrow 1$ *to* $m_{ik}$ *do*

*push "Determine the K' shortest paths from* $S(\vec{p}_{ikt})$ *to* $D(\vec{p}_{ikt})$ *by the DSC of* $e(S(\vec{p}_{ikt}))$*" in the message queue.*

*end for*

*push "Determine the K' shortest paths from* $S_i$ *to* $X_{Si}.Y_{Si}$ *by the DSC of* $e(S_i)$*" in the message queue.*

*push "Determine the K' shortest paths from* $X_{Di}.Y_{Di}$ *to* $D_i$ *by the DSC of* $e(D_i)$*" in the message queue.*

*end for*

*for i=1 to M do*

*Find the requests to be processed by EN i and send them to EN i.*

*end for*

/* There must be another thread that will wait for the requests from the DSCs to determine $K'$ shortest sub paths. This thread determines the $K'$ shortest sub paths and sends them to the corresponding DSC. */

*Wait for the messages with the K' shortest paths determined by the other DSCs.*

*for* $i \leftarrow 1$ *to* $n_i$ *do*

$\vec{l}_{iSr} \leftarrow$ Vector of the links representing the $r$th sub path from $S_i$ to $X_{Si}.Y_{Si}$

$\vec{l}_{iDr} \leftarrow$ Vector of the links representing the $r$th sub path from $D_i$ to $X_{Di}.Y_{Di}$

The combination of all $K'^{M+1}$ alternative paths can be found by joining the $K'$ sub paths determined by the $M$ DSCs. We can express a path as follows by joining the $r$th sub path from each EN and the $k$th path from $GN$:

$$\left[\{\vec{l}_{iSr}\}, L_{ik1}, L_{ik2}, \ldots\ldots, L_{ikl_{ik}}, \{\vec{l}_{ik1r}\}, \{\vec{l}_{ik2r}\}, \ldots\ldots, \{\vec{l}_{ikm_{ik}r}\}, \{\vec{l}_{iDr}\}\right]$$

Where, Delay on this path $= d(\vec{l}_{iSr}) + \sum_{t=1}^{l_{ik}} d(L_{ikt}) + \sum_{t=1}^{m_{ik}} d(\vec{l}_{iktr}) + d(\vec{l}_{iDr}) \le d_{\max}(SLA_i)$

The first $K$ paths of this combination of alternative paths sorted according to the total delay are considered for routing $SLA_i$.

*end for*

# 11. Refererences


[1] A. Borodin & R. El-Yaniv. *Online Computation & Competitive Analysis.* Cambridge University Press, 1998.

[2] A. Vina et al. Real-time Multimedia Systems. *19th IEEE Symposium on Mass Storage System*, pp. 77-83, 1994.

[3] S. Plotkin, B. Awerbuch, Y. Azar. Throughput Competitive On-line Routing. *34th Annual Symposium on Foundations Computer Science*, Los Alamitos, CA, Nov 1993.

[4] B. Awerbuch, Y. Azar, S. Plotkin, O. Waarts. Competitive Routing of Virtual Circuits with Unknown Duration. *Fifth Annual SIAM Symposium on Discrete Algorithms*, Arlington, VA, pp.321-327.

[5] C. W. Mercer, S. Savage, & H. Tokuda. Processor Capacity Reserves. *IEEE Conf. Multimedia Computing & Systems*, Boston, pp. 90-99.

[6] G. Campbell, D. Coulson. A Quality of Service Architecture. *ACM Operating Systems Review*, Volume 24. April 1994.

[7] C. Lee. *On QoS Management.* PhD Dissertation. School of Computer Science, Carnegie Mellon University, August 1999.

[8] D. Eppstein. Finding the *k* Shortest Paths, *35th IEEE Symposium on Foundations of Computer Science*, Santa Fe, pp. 154-165.

[9] E. Chang & A. Zakhor. Cost Analysis for VBR video servers. *IEEE Multimedia*, pp. 56-71, Winter 1996.

[10] E. Dijkstra. A Note on two Problems in Connexion with Graphs. *Numerische Mathematik*, pp. 269-271, 1959.

[11] M. M. Akbar, E. G. Manning, R. K. Watson, G. C. Shoja, S. Khan, K. F. Li. Optimal Admission Controllers for Service Level Agreements in Enterprise Networks. *SCI 2002*, Orlando, FL July 14-18, 2002.

[12] E. Q. V. Martins, J. L. E. Santos. A New Shortest Paths Ranking Algorithm. *Departamento de Matematica, Universidade de Coimbra*, http://www.mat.uc.pt/~eqvm, 1996.

[13] E. Q. V. Martins, M. Pascoal, J.L.E. Santos. A New Algorithm for Ranking Loopless Paths. *Departamento de Matematica, Universidade de Coimbra*, http://www.mat.uc.pt/~eqvm, May 1997.

[14] G. F. Luger and W. A. Stubblefield. *Artificial Intelligence, Structures and Strategies for Complex Problem Solving, Second edition.* The Benjamin/Cummings Publishing Company, Inc., 1993.

[15] M. R. Garey, and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[16] I. R. Chen & T. H. Tsui. Performance Analysis of Admission Control Algorithms based on Reward Optimization. *Performance Evaluation*, pp. 89-112, Volume 33(2).

[17] J. Kleinburg & E. Tardos. Approximations for the Disjoint Paths Problem in High-Diameter Planar Networks. *Journal of Computer and System Sciences*. pp. 61-73, Volume 57(1), Orlando, Aug 1998.

[18] J. Kleinburg, Y. Rabani, & E. Tardos. Allocating Bandwidth for Bursty Connections. *Annual ACM Symposium on Theory of Computing*, El. Paso, TX, pp. 664-673, May 1997.

[19] J. Y. Yen. Finding the K Shortest Loopless Paths in a Network, *Management Science* pp. 712-716, Volume 17, 1971.

[20] K. Dudziniski and W. Walukiewicz. A Fast Algorithm for the Linear Multiple Choice Knapsack Problem. *Operation Research Letters*, pp 205-209, Volume 3, 1984.

[21] K. Kawachiya and H. Tokuda. QOS-Ticket: A New Resource-Management Mechanism for Dynamic QOS Control of Multimedia. *Proceedings of Multimedia Japan 1996*, pp 14-21, April 1996.

[22] L. Chen, S. Khan, K. F. Li and E. Manning. Building an Adaptive Multimedia System using the Utility Model. *International Workshop on Parallel and Distributed Realtime Systems*, San Juan, Puerto Rico, April, 1999.

[23] L. Chen. *Utility Model Applied to Layered-coded Sources*, M.Sc. Thesis, Department of Computer Science, University of Victoria, October 1998.

[24] L. C. Schreier, B. Davis. System-Level Resource Management for Network-based Multimedia Applications. *Fifth International Workshop on Network and Operating System Support for Digital Audio and Video*, NOSSDAV 95, Durham, NH, 1995.

[25] M. M. Akbar, E. G. Manning, G. C. Shoja. Admission Control and QoS adaptation in Distributed Multimedia Server System, *ITCom 2001*, Denver, USA, August 2001.

[26] M. M. Akbar, E. G. Manning, G. C. Shoja, S. Khan. Heuristic Solutions for the Multiple-Choice Multi-Dimension Knapsack Problem. *International Conference on Computational Science*, San Francisco, USA, May 2001.

[27] M. Moser, D. P. Jokanovic and N. Shiratori. An Algorithm for the Multidimensional Multiple-Choice Knapsack Problem. *IEICE Transactions on Fundamentals of Electronics*, pp 582-589, Volume 80(3), 1997.

[28] M. Moser. Declarative Scheduling for Optimally Graceful QoS Degradation. *IEEE Multimedia Systems*, June 1996, Hiroshima, Japan.

[29] M. Magazine and O. Oguz. A Heuristic Algorithm for Multidimensional Zero-One Knapsack Problem. *European Journal of Operational Research*, pp 319-326, Volume 16(3), 1984.

[30] M. Magazine, G. Nemhauser and L. Trotter. When the Greedy Solution Solves a Class of Knapsack Problem. *Operations Research*, pp 207-217, Volume 23, 1975.

[31] M. D. Biddiscombe, J. E. Midwinter, & S. Sabean. Application of Free-market Principles to Telecom Resource Allocation. *IEEE Electronics Letters*, Stevenage, pp, 264-266, Volume 35(4), Feb. 1999.

[32] N. Nishio and H. Tokuda. QOS Translation and Session Coordination Techniques for Multimedia Systems. *Sixth International Workshop on Network and Operating System Support for Digital Audio and Video*, Jushi, Japan, April 23—26, 1996.

[33] N. Venkatasubramanian, K. Nahrstedt. An Integrated Metric for Video QoS. *Fifth ACM International Multimedia Conference*, Seattle, USA.

[34] N. Davies et. al. Supporting Adaptive Service in a Heterogeneous Mobile Environment, *Workshop on Mobile Computing*, Santa Cruz, 1994.

[35] P. Kirkby & R. Kadengal. Traffic Management & Control using a Single 'Congestion Price'. *IEEE Colloquium on Control of Next Generation Networks*, London, October 1999.

[36] P. Koleser. A Branch and Bound Algorithm for Knapsack Problem. *Management Science*, pp 723-735, Volume 13, 1967.

[37] R. Armstrong, D. Kung, P. Sinha and A. Zoltners. A Computational Study of Multiple Choice Knapsack Algorithm. *ACM Transaction on Mathematical Software*, pp 184-198, Volume 9, 1983.

[38] R. Gopalakrishnan and G. Parulkar. A Framework for QoS Guarantees for Multimedia Applications within an End System. *Swiss German Computer Society Conference*, September, 1995.

[39] R. Gopalakrishnan and G. Parulkar. *Efficient Quality of Service Support in Multimedia Computer Operating Systems*. Technical Report WUCS-94-26, Dept. of Computer Science, Washington University, St. Louis.

[40] R. K. Watson. *Applying the Utility Model to IP Networks: Optimal Admission & Upgrade of Service Level Agreements*. MASc Thesis, Dept of ECE, University of Victoria, April, 2001.

[41] R. Nauss. The 0-1 Knapsack Problem with Multiple Choice Constraints. *European Journal of Operation Research*, pp 125-131, Volume 2, 1978.

[42] S. Chatterjee, J. Sydir and B. Sabata and T Lawrence. Modeling Applications for Adaptive QoS – based Resource Management. *2nd IEEE High Assurance Systems Engineering Workshop*, August, 1997.

[43] S. Khan and K. F. Li and E. G. Manning. Padma: An Architecture for Adaptive Multimedia Systems. *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, Aug 20- 22, 1997, pp 105-108.

[44] S. Khan and K. F. Li and E. G. Manning. The Utility Model for Adaptive Multimedia Systems. *International Conference on Multimedia Modelling*, Singapore, pp 111-126, Nov 17-20, 1997.

[45] S. Khan, K. F. Li, E. G. Manning, M. M. Akbar. Solving the Knapsack Problem for Adaptive Multimedia System, *Studia Informatica*, 2002.

[46] S. Khan. *Quality Adaptation in a Multi-Session Adaptive Multimedia System: Model and Architecture*. PhD Dissertation, Department of Electrical and Computer Engineering, University of Victoria, 1998.

[47] S. Martello and P. Toth. Algorithms for Knapsack Problems. *Annals of Discrete Mathematics*, pp 70-79, Volume 31, 1987.

[48] V. P. Kumar, T.V. Lakshman, D. Stiliadis. *Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow's Internet*. Technical Report, Bell Laboratories (Lucent Technologies).

[49] V. E. Benes, *Mathematical Theory of Connecting Networks and Telephone Traffic*. New York, Academic Press, 1965.

[50] W. Shih. A branch and Bound Method for Multiconstraint Knapsack Problem. *Journal of the Operational Research Society*, pp 369-378, Volume 30, 1979.

[51] W. H. Press, S.A. Teukolsky, W. T. Vetterling and B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, UK, Second edition 1992.

[52] Y. Tobe. & H. Tokuda. Integrated QoS Management: Cooperation of Processor Capacity Reserves and Traffic Management. *IEICE Transactions on Communications*, Tokyo, pp. 1998-2006, Volume E81-B(11), Nov 1998.

[53] Y. Toyoda. A Simplified Algorithm for Obtaining Approximate Solution to Zero-one Programming Problems. *Management Science*, pp 1417-1427, Volume 21, 1975.

[54] H. Perros and K. Elsayed. Call Admission Control Schemes: A Review. *IEEE Communications Magazine*, November 1996, Volume 34(11).

[55] D. D. Clark and W. Fang. *Explicit Allocation of Best Effort Packet Delivery Service*, Technical Report, MIT Lab for Computer Science.

[56] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. *RFC 2475*, December 1998.

[57] P. Milgrom. The Economics of Competitive Bidding. *Social Goals and Social Organization*, University Press, Cambridge, 1985.

[58] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press / McGraw-Hill, 1990.

[59] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. Requirements for Traffic Engineering over MPLS. *Draft-ietf-mpls-traffic-eng-01.txt*, June 1999.

[60] R. Braden, L. Zhang , S. Berson , S. Herzog, S. Jamin. Resource ReSerVation Protocol (RSVP) Version 1 Functional Specification. *RFC 2205*, September 1997, Proposed Standard.

[61] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. *RFC 2475*, December 1998.

[62] J. W. Baek, Park and Hong. Management of Service Level Agreements for Multimedia Internet Service using a Utility Model. *IEEE Communications Magazine*, pp 2-8, May 2001.

[63] H. Perros and K. Elsayed. Call Admission Control Schemes: A Review. *IEEE Communications Magazine*, Volume 34(11), November 1996.

[64] E. Rosen, A. Viswanathan, R. Callon. Multiprotocol Label Switching Architecture. *RFC 3031*, IETF, January 2001.

[65] D. Hudson. *Proposal for a High-Performance Internet Optical Overlay*, Nortel Networks Inc., private communication.

[66] M. Gerla, C. Casetti, S. S. Lee, and G. Reali. Resource Allocation and Admission Control Styles in QoS DiffServ Networks. *QoS-IP 2001*, Rome, Italy, Jan 2001.

[67] R. J. Gibbens and F. P. Kelly. Distributed Connection Acceptance Control for a Connectionless Network. *ITC16*, Edinburgh, 1999.

[68] A. Greenberg, R. Srikant and W. Whitt. Resource Sharing for Book-ahead and Instantaneous-request Calls. *IEEE/ACM Transactions on Networking*, pp 10-22, Volume 7(1), Feb 1999.

[69] O. C. Imer, S. Compans, T. Basar and R. Srikant. ABR Congestion Control in ATM Networks. *IEEE Control Systems Magazine*, Feb 2001.

[70] F. Kelly, P. Key and S. Zachary. Distributed Admission Control. *IEEE Journal on Selected Areas in Communications*, pp. 2617-2628, Volume 18, 2000.

[71] T. Kelly. An ECN Probe-Based Connection Acceptance Control. *Computer Communication Review*, Volume 31(3), July 2001.

[72] S. Lu, V. Bharghavan and R. Srikant. Fair Scheduling in Wireless Packet Networks. *IEEE/ACM Transactions on Networking*, pp. 473-489, Volume 7(3) August 1999.

[73] S. Shakkottai and R. Srikant. Scheduling Real-time Traffic With Deadlines over a Wireless Channel. *Wireless Networks*, pp. 13-26, Volume 8, 2002.

[74] R. Srikant and W. Whitt. Resource Sharing with Book-Ahead and Instantaneous-Request Calls Using a CLT Approximation. *Telecommunication Systems*. pp. 235-255, Volume 16(3), March/April 2001.

[75] B. Teitelbaum. Future Priorities for Internet2 QoS. *Working Group: Papers, http://www.internet2.edu/qos/wg/documents.shtml*, October 2, 2001.

[76] L. Breslau, E. Knightly, S. Shenker, I. Stoica, H. Zhang. Endpoint Admission Control: Architectural Issues and Performance. *SIGCOMM 2000.*

[77] A. Dasylva and R. Srikant. Optimal WDM Schedules for Optical Star Networks. *IEEE/ACM Transactions on Networking*, pp 446-456, June 1999.

[78] Y. Rekhter, T. Li. A Border Gateway Protocol 4 (BGP-4). *Network Working Group, RFC 1771*, March 1995.

[79] J. Moy. *OSPF: Anatomy of an Internet Routing Protocol.* Addison-Wesley, February 1998.

[80] J. Moy, OSPF Version 2, *RFC 2328*, April 1998.

[81] J. Pu, E. Manning, G. C. Shoja, A. Srinivasan. A New Algorithm to Compute Alternate Paths in Reliable OSPF (ROSPF). *2001 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001)*, pp. 299-304, Volume 1, June 2001, Las Vegas, Nevada, USA.

[82] *Technology – Our Product – Ninth House Network*, http://www.ninthhouse.com/product/technology/server.htm.

[83] D. Johnson. *A Protocol for the Interoperability of Content Distribution Networks.* M.Sc. thesis, Department of CSC, University of Victoria, 2002.

[84] B. Cain, O. Spatscheck, M. May, A. Barbir. Request-Routing Requirements for Content Internetworking. *IETF Internet Draft draft-ietf-cain-request-routing-req-03.txt, http://www.ietf.org/internet-drafts/draft-cain-request-routing-req-03.txt.*

[85] M. Green, B.Cain. CDN Peering Architectural Overview. *IETF draft draft-green-cdnp-gen-arch-01.txt, http://www.content-peering.org/draft-green-cdnp-gen-arch-01.html.*

[86] A. Barbir, B. Cain. Known CN Request-Routing Mechanisms. *IETF draft draft-cain-cdnp-known-reqeust-routing-04.txt, http://www.ietf.org/internet-drafts/draft-cain-cdnp-known-request-routing-04.txt.*

[87] M. Day, B. Cain, et al. A Model for Content Internetworking. *IETF draft draft-day-cdnp-model-09.txt. http://www.ietf.org/internet-drafts/draft-day-cdnp-model-09.txt.*

[88] S. Shelford, M Akbar, E. G. Manning G. C. Shoja. *D-SLAOpt, Java Simulation of Distributed SLA Controller.* DCS 272-IR, Technical Report Department of CSC, UVic, Aug 2002.

[89] K. Sohraby, K. Ryan. An Architecture for Signaling and Control in Optical Networks. *SCI 2002.* Orlando, pp 319-326, July 2002

[90] R. Singh, M. Yuksel, S. Kalyanaraman and T. Ravichandran. A comparative evaluation of Internet Pricing models: Smart market and Dynamic Capacity Contracting. *Workshop on Information Technologies and Systems (WITS).* Queensland, Australia, 2000.

[91] J. K. MacKie-Mason, H. R. Varian. Pricing Congestible Network Resources. *IEEE J. Selected Areas Comm.* Vol 13, pp 1141-1149, 1995.

[92] S. Shenker, D. Clark, D. Estrin, S. Herzog. Pricing in Computer Networks: Reshaping the Research Agenda. *ACM Computer Communication Review.* Vol 26, pp 19-43, April 1996.

[93] C. Courcoubetis, G. Kesidis, A. Ridder, J. Walrand and R. Weber. Admission Control and Routing in ATM Networks using Inferences from Measured Buffer Occupancy. *IEEE Transaction on Communication.* Feb./March/April 1995.

[94] A. S. Tanenbaum. *Computer Networks.* Prentice Hall. 1996.

[95] M. Law and W. Kelton. *Simulation Modeling and Analysis.* McGraw-Hill Series in Industrial Engineering and Management Science.

[96] S. Khuri, T. Back and J. Heitkotter. The Zero/One Multiple Knapsack Problem and Genetic Algorithms. *ACM Simposium of Applied Computation.* 1994.

[97] F. Dammeyer and S. Voss. Dynamic Tabu List Management Using the Reverse Elimination Method. *Annals of Operations Research*. 1991.

[98] A. Drexel. A Simulated Annealing Approach to the Multiconstraint Zero-One Knapsack Problem. *Annals of Computing*, Vol 40, pp 1-8, 1988.

[99] G. Hadley. *Linear Algebra*. Narosa Publishing House. 1990

[100] M. Kearns and S. Singh. Near-optimal Reinforcement Learning in Polynomial Time. *International Conference on Machine Learning*. 1998.