A COMPARATIVE STUDY

OF THE PERFORMANCE OF

CONCURRENCY CONTROL ALGORITHMS

IN A CENTRALISED DATABASE

DONALD FRANCIS ROSS

B.Sc.

A Thesis submitted in fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

HERIOT-WATT UNIVERSITY

Department of Computer Science

March 1988

This copy of the thesis has been supplied on the condition that anyone who consults it is understood to recognise that the copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the prior written consent of the author or the University (as may be appropriate).

DEDICATION.

To my parents, J.M.R and D.G.R.,

To M.F.M., in memory,

and

To E.M.H. and S.D.L.S

DECLARATION.

I hereby declare that I have personally composed this thesis and the work herein was my own except where due acknowledgement is made. The material in this thesis has not been submitted to this or any other university for a degree.

Signed:

Dated:

CONTENTS

1. INTRODUCTION 1
   1.1. Concurrency Control 1
   1.2. Issues 2
   1.3. Algorithms 4
      1.3.1. Locking Algorithms 4
      1.3.2. Timestamping 6
      1.3.3. Serial Validation 7
      1.3.4. Hybrid Algorithms 8
   1.4. Performance 8
   1.5. This Study 9

2. PERFORMANCE MODELS 11
   2.1. Database System 11
   2.2. Concurrency Control Algorithms (CCAs) 13
      2.2.1. Preclaim Locking (PRE) 15
      2.2.2. Two-Phase Exclusive Locking (2PLE) 15
      2.2.3. Two-Phase Upgrade Locking (2PLU) 16
      2.2.4. Basic Timestamp Ordering (BTO) 18
      2.2.5. Serial Validation (SV) 18
   2.3. Logical Queueing Model (LQM) 19
   2.4. Performance Metrics and Assumptions 21
   2.5. Previous Results 23
   2.6. Performance Metrics and Workload Parameters 25
      2.6.1. Performance Metrics 25
      2.6.2. Workload Parameters 26
   2.7. Chapter Summary 27

3. TESTBED DATABASE SYSTEM 28
   3.1. The Testbed System: TDBS/C 28
      3.1.1. Overview of TDBS/C 29
      3.1.2. Data Manipulation Language (DML) 31
      3.1.3. TDBS/C Internal Organisation 32
   3.2. Experimental Test Database Design 33
      3.2.1. Experiment Design 34
      3.2.2. Test Data and Database Design 35
      3.2.3. Test Transactions 37
   3.3. Benchmark Tests 41
   3.4. Benchmark Results 42
      3.4.1. Results of Experiment 1 42
      3.4.2. Results of Experiment 2 47
   3.5. Conclusions 48
   3.6. Chapter Summary 48

4. PROTOTYPE EXPERIMENTS 50
4.1. Experimental Method 50
4.2. Test Data and Transactions 52
4.3. Preliminary Experiments 54
4.3.1. Critical Section Locking Technique 54
4.3.2. Blocking and Restarting Delay (B_DLY/R_DLY) 54
4.3.3. Fixed Length vs Dynamic CCA Information Tables 55
4.4. Experiments and Statistics. 55
4.5. Experiment 1 56
4.5.1. Experiment Description 56
4.5.2. Single User Overhead Costs 57
4.5.3. Experimental Results 58
4.5.4. Adaptive Restart 61
4.5.5. Overhead Costs and Efficiency Measures 62
4.5.6. Granularity Effects 63
4.6. Experiment 2 64
4.6.1. Experiment Description 65
4.6.2. Single User Overhead Costs 66
4.6.3. Multi User Results 66
4.6.4. Adaptive Restart 67
4.6.5. Overhead Costs and Efficiency Measures 67
4.7. Conclusions 68
4.8. Chapter Summary 70

5. SIMULATION EXPERIMENTS 71
5.1. Simulation 71
5.2. Experiment 1 74
5.2.1. Experimental Results 74
5.2.2. Adaptive Restart 76
5.2.3. Overhead and Efficiency Measures 77
5.2.4. Granularity Effects 78
5.3. Experiment 2 78
5.3.1. Experiment Results 78
5.3.2. Adaptive Restart 80
5.3.3. Overhead and Efficiency Measures 80
5.4. Adaptive Restart Revisited. 81
5.5. Simulation: Summary and Conclusions 83
5.6. Comparison with the Prototype. 84
5.6.1. Experiment 1 85
5.6.2. Experiment 2 86
5.6.3. Comments on the Comparison 86
5.7. General Conclusions 87
5.8. Chapter Summary 89

6. COMPARISON WITH OTHER STUDIES 90
6.1. Experimental Studies 90
6.1.1. Optimistic CC vs Locking 90
6.1.2. Other Experimental Studies 93
6.1.3. Conclusions from the Experimental Studies 95
6.2. Simulation Studies 96
6.2.1. Other Studies I 96
6.2.2. Other Studies II 97
6.3. An Analytic Study 104
6.4. Conclusions 107
6.5. Chapter Summary 108

7. SUMMARY AND CONCLUSIONS 109
7.1. A Wider Context 109
7.2. Review of This Study 110
7.2.1. TDBS/C Revisited 110
7.2.2. The CCAs Revisited 111
7.2.3. The Experiments Revisited 112
7.2.4. Limitations of This Study 113
7.2.5. Summary of Results 115
7.3. Future Research 115
7.4. Concluding Remarks 116

APPENDICES

1. TESTBED DATABASE SYSTEM (TDBS/C) DESCRIPTION 118
1.1. Historical Background 118
1.2. PRECI 119
1.3. TDBS/C Implementation 120
1.4. TDBS/C Overview 120
Schema Translator 120
Storage Schema Translator 121
Database Initialiser 121
View Schema Translator 121
Precompiler 121
listdb 121
Natural Language Interface 122
ICI 122
QBE 122
DBCS 122
1.5. Internal Organisation 123
1.5.1. Relation File Organisation 123
RIN (Relation Information) 124
PINDEX AREA (primary key index) 125
DATA AREA (tuple storage) 126
Page Header 127
1.5.2. Identifier List (IDLIST) 127
1.5.3. Buffer Management 130
1.6. Schema Examples 132
1.7. Syntax Definitions 135

2. SINGLE-USER BENCHMARK RESULTS 138

3. PROTOTYPE RESULTS 152
Experiment 1 Results: (GRAN = 1, T_TYP = get-replace) 153
Experiment 1 Results: (GRAN = 10, T_TYP = get-replace) 158
Experiment 2 Results: (T_TYP = get-replace) 163
Experiment 1 Results: (GRAN = 1, T_TYP = select-replace) 169
Experiment 1 Results: (GRAN = 10, T_TYP = select-replace) 174
Experiment 2 Results: (T_TYP = select-replace) 179

4. SIMULATION RESULTS 185
Experiment 1: Results 186
Experiment 2: Results 191

5. EXPERIMENTAL TECHNIQUES 197
5.1. Operating System Environment 197
5.2. Chapter 3: Benchmark Tests 198
5.3. Chapter 4: Prototype Experiments 199
5.4. Chapter 5: Simulation Experiments 201

REFERENCES 203

List of Illustrations.

Figure	Title	Page
1.1	Example Transactions	1
1.2	Lost Update Example	2
1.3	Timestamp Example	6
1.4	Serial Validation Example	7
2.1	Database System	11
2.2	CCA Tables	14
2.3	2PLE Request Procedure	16
2.4	2PLU Request Procedure	17
2.5	BTO Request Procedure	18
2.6	SV Validation Procedure	19
2.7	Logical Queueing Model (LQM)	20
3.1	Database Initialisation	29
3.2	Transaction Creation	30
3.3	Relation File Layout	32
3.4	Tuple Allocation Procedure	33
5.1	Physical Queueing Model (PQM)	72
A1.1	ANSI/SPARC Three Level Architecture	119
A1.2	TDBS/C - SYSTEM OVERVIEW	123
A1.3	Relation File Layout	124
A1.4	RIN Layout - PINDEX AREA	124
A1.5	RIN Layout - DATA AREA	125
A1.6	PINDEX AREA page layout	125
A1.7	PINDEX AREA slot layout	126
A1.8	DATA AREA page layout	126
A1.9	Page Header Layout	127
A1.10	IDLIST	127
A1.11	TDBS/C Organisation Levels	130
A1.12	Buffer Manager Structures	131
A1.13	Schema Definition	132
A1.14	Storage Schema Definition	133
A1.15	View Schema Definition	133
A1.16	Relation Information (RIN) Listing	134
A1.17	Schema Definition Syntax	135
A1.18	Storage Schema Definition Syntax	135
A1.19	View Schema Definition Syntax	136
A1.20	Relation Expression Syntax	136
A1.21	Database Definition Syntax	137
A5.1	Transaction Timing for Prototype Experiments	201

List of Illustrations.

Graph	Title	Page
3.1	get-replace (grp)/BF_SZ: ET vs DB_ACC	43
3.2	select-replace (srp)/BF_SZ: ET vs DB_ACC	44
3.3	BF_SZ 100 pages: ET vs T_TYP	44

List of Tables.

Table	Title	Page
2.1	Modelling Factors	12
2.2	General Performance Model	20
2.3	Summary of Granularity Results	24
2.4	Summary of TMPL and Conflict Results	25
2.5	Performance Metrics	26
2.6	Workload Parameters	27
3.1	DML Operations	31
3.2	Tuple Attribute Values	36
3.3	Storage Efficiency	37
3.4	Transaction Types	38
3.5	Transaction Descriptions	39
3.6	Factors and Factor Levels	41
3.7	Crossover Points	45
3.8	Experiment 1, Results Summary	46
3.9	Experiment 2 Factor Levels	47
4.1	Performance Metrics	51
4.2	Tuple Attribute Values	53
4.3	CCA Table Sizes	55
4.4	Experiment 1: Factor Levels	56
4.5	SU Ovehead Costs (%)	57
4.6	GRAN = 1, T_TYP = get-replace: Throughput	59
4.7	GRAN = 1, T_TYP = get-replace: Restart Counts	59
4.8	GRAN = 10, T_TYP = get-replace: Throughput	59
4.9	GRAN = 10, T_TYP = get-replace: Restart Counts	59
4.10	GRAN = 1, T_TYP = select-replace: Throughput	61
4.11	GRAN = 1, T_TYP = select-replace: Restart Counts	61
4.12	GRAN = 10, T_TYP = select-replace: Throughput	61
4.13	GRAN = 10, T_TYP = select-replace: Restart Counts	61
4.14	Overhead Costs, T_TYP = get_replace (%)	63
4.15	Efficiency Measure, T_TYP = get_replace (%)	63
4.16	Experiment 2: Factor Levels	65
4.17	Experiment 2: LC and EMPL	65
4.18	Experiment 2: Throughput	66
4.19	Experiment 2: Restart Counts	66
4.20	Overhead Costs: T_TYP = get-replace	68
4.21	Efficiency Measures: T_TYP = get-replace	68

List of Tables.

Table	Title	Page
5.1	PQM Parameters	72
5.2	PQM Parameter Values	74
5.3	GRAN = 1: Throughput	75
5.4	GRAN = 1: Restart Counts	75
5.5	GRAN = 10: Throughput	75
5.6	GRAN = 10: Restart Counts	75
5.7	GRAN = 1: Overhead Values (%)	77
5.8	GRAN = 10: Overhead Values (%)	77
5.9	GRAN = 1: Efficiency Measure (%)	78
5.10	GRAN = 10: Efficiency Measure (%)	78
5.11	Experiment 2: Throughput	79
5.12	Experiment 2: Restart Counts	79
5.13	Experiment 2: Overhead Values (%)	80
5.14	Experiment 2: Efficiency Measure (%)	80
5.15	CCAs with R_DLY Values	81
5.16	Simulation Results: Ranked CCAs	82
5.17	Summary of Prototype and Simulation Results: Ranked CCAs	84
6.1	Kersten and Tebra: SV vs 2PLU, ET (seconds)	91
6.2	This Study: SV vs Locking, ET (seconds)	92
6.3	Factor Level Comparison: This Study/(Carey)	99
6.4	Parameter Value Comparison: This Study/(Carey)	99
6.5	Throughput Comparison: This Study/(Carey)	100
6.6	Restart Count Comparison: This Study/(Carey)	100
6.7	CCA Rank Order in Descending T: This Study/(Carey)	101
6.8	Experiments 1 and 2: DC-workload	105
A1.1	TDBS/C Historical Development	118
A2.1	Experiments 1 & 2 Results Tables	138
A2.2	Throughput Results - Summary	139
A2.3	DB_SZ = 100, T_TYP = get-replace	140
A2.4	DB_SZ = 100, T_TYP = select-replace	141
A2.5	DB_SZ = 100, T_TYP = select	142
A2.6	DB_SZ = 1000, T_TYP = get-replace	143
A2.7	DB_SZ = 1000, T_TYP = select-replace	144
A2.8	DB_SZ = 1000, T_TYP = select	145
A2.9	DB_SZ = 10000, T_TYP = get-replace	146
A2.10	DB_SZ = 10000, T_TYP = select-replace	147
A2.11	DB_SZ = 10000, T_TYP = select	148
A2.12	DB_SZ = 100, T_TYP = get-replace (sequential/uniform)	149
A2.13	DB_SZ = 1000, T_TYP = get-replace (sequential/uniform)	150
A2.14	DB_SZ = 10000, T_TYP = get-replace (sequential/uniform)	151

List of Tables.

Table	Title	Page
A3.1	T_REQ = 1, Throughput, ET and run counts	153
A3.2	T_REQ = 1, Useful Work Costs	153
A3.3	T_REQ = 1, Wasted Work Costs	153
A3.4	T_REQ = 2, Throughput, ET and run counts	154
A3.5	T_REQ = 2, Useful Work Costs	154
A3.6	T_REQ = 2, Wasted Work Costs	154
A3.7	T_REQ = 5, Throughput, ET and run counts	155
A3.8	T_REQ = 5, Useful Work Costs	155
A3.9	T_REQ = 5, Wasted Work Costs	155
A3.10	T_REQ = 10, Throughput, ET and run counts	156
A3.11	T_REQ = 10, Useful Work Costs	156
A3.12	T_REQ = 10, Wasted Work Costs	156
A3.13	Overhead Summary (%)	157
A3.14	Efficiency Summary (%)	157
A3.15	T_REQ = 1, Throughput, ET and run counts	158
A3.16	T_REQ = 1, Useful Work Costs	158
A3.17	T_REQ = 1, Wasted Work Costs	158
A3.18	T_REQ = 2, Throughput, ET and run counts	159
A3.19	T_REQ = 2, Useful Work Costs	159
A3.20	T_REQ = 2, Wasted Work Costs	159
A3.21	T_REQ = 5, Throughput, ET and run counts	160
A3.22	T_REQ = 5, Useful Work Costs	160
A3.23	T_REQ = 5, Wasted Work Costs	160
A3.24	T_REQ = 10, Throughput, ET and run counts	161
A3.25	T_REQ = 10, Useful Work Costs	161
A3.26	T_REQ = 10, Wasted Work Costs	161
A3.27	Overhead Summary (%)	162
A3.28	Efficiency Summary (%)	162
A3.29	LC = 0%, Throughput, ET and run counts	162
A3.30	LC = 0%, Useful Work Costs	163
A3.31	LC = 0%, Wasted Work Costs	163
A3.32	LC = 20%, Throughput, ET and run counts	164
A3.33	LC = 20%, Useful Work Costs	164
A3.34	LC = 20%, Wasted Work Costs	164
A3.35	LC = 50%, Throughput, ET and run counts	165
A3.36	LC = 50%, Useful Work Costs	165
A3.37	LC = 50%, Wasted Work Costs	165
A3.38	LC = 80%, Throughput, ET and run counts	166
A3.39	LC = 80%, Useful Work Costs	166

List of Tables.

Table	Title	Page
A3.40	LC = 80%, Wasted Work Costs	166
A3.41	LC = 100%, Throughput, ET and run counts	167
A3.42	LC = 100%, Useful Work Costs	167
A3.43	LC = 100%, Wasted Work Costs	167
A3.44	Overhead Summary (%)	168
A3.45	Efficiency Summary (%)	168
A3.46	T_REQ = 1, Throughput, ET and run counts	169
A3.47	T_REQ = 1, Useful Work Costs	169
A3.48	T_REQ = 1, Wasted Work Costs	169
A3.49	T_REQ = 2, Throughput, ET and run counts	170
A3.50	T_REQ = 2, Useful Work Costs	170
A3.51	T_REQ = 2, Wasted Work Costs	170
A3.52	T_REQ = 5, Throughput, ET and run counts	171
A3.53	T_REQ = 5, Useful Work Costs	171
A3.54	T_REQ = 5, Wasted Work Costs	171
A3.55	T_REQ = 10, Throughput, ET and run counts	172
A3.56	T_REQ = 10, Useful Work Costs	172
A3.57	T_REQ = 10, Wasted Work Costs	172
A3.58	Overhead Summary (%)	173
A3.59	Efficiency Summary (%)	173
A3.60	T_REQ = 1, Throughput, ET and run counts	174
A3.61	T_REQ = 1, Useful Work Costs	174
A3.62	T_REQ = 1, Wasted Work Costs	174
A3.63	T_REQ = 2, Throughput, ET and run counts	175
A3.64	T_REQ = 2, Useful Work Costs	175
A3.65	T_REQ = 2, Wasted Work Costs	175
A3.66	T_REQ = 5, Throughput, ET and run counts	176
A3.67	T_REQ = 5, Useful Work Costs	176
A3.68	T_REQ = 5, Wasted Work Costs	176
A3.69	T_REQ = 10, Throughput, ET and run counts	177
A3.70	T_REQ = 10, Useful Work Costs	177
A3.71	T_REQ = 10, Wasted Work Costs	177
A3.72	Overhead Summary (%)	178
A3.73	Efficiency Summary (%)	178
A3.74	LC = 0%, Throughput, ET and run counts	179
A3.75	LC = 0%, Useful Work Costs	179
A3.76	LC = 0%, Wasted Work Costs	179
A3.77	LC = 20%, Throughput, ET and run counts	180
A3.78	LC = 20%, Useful Work Costs	180
A3.79	LC = 20%, Wasted Work Costs	180

List of Tables.

Table	Title	Page
A3.80	LC = 50%, Throughput, ET and run counts	181
A3.81	LC = 50%, Useful Work Costs	181
A3.82	LC = 50%, Wasted Work Costs	181
A3.83	LC = 80%, Throughput, ET and run counts	182
A3.84	LC = 80%, Useful Work Costs	182
A3.85	LC = 80%, Wasted Work Costs	182
A3.86	LC = 100%, Throughput, ET and run counts	183
A3.87	LC = 100%, Useful Work Costs	183
A3.88	LC = 100%, Wasted Work Costs	183
A3.89	Overhead Summary (%)	184
A3.90	Efficiency Summary (%)	184
A4.1	GRAN = 1, T_REQ = 1, Throughput, ET and run counts	186
A4.2	GRAN = 1, T_REQ = 2, Throughput, ET and run counts	186
A4.3	GRAN = 1, T_REQ = 5, Throughput, ET and run counts	186
A4.4	GRAN = 1, T_REQ = 10, Throughput, ET and run counts	187
A4.5	GRAN = 10, T_REQ = 1, Throughput, ET and run counts	187
A4.6	GRAN = 10, T_REQ = 2, Throughput, ET and run counts	187
A4.7	GRAN = 10, T_REQ = 5, Throughput, ET and run counts	188
A4.8	GRAN = 10, T_REQ = 10, Throughput, ET and run counts	188
A4.9	Useful Work Costs ET	189
A4.10	Wasted Work Costs ET	189
A4.11	Useful Work Costs ET	190
A4.12	Wasted Work Costs ET	190
A4.13	LC = 0%, Throughput, ET and run counts	191
A4.14	LC = 20%, Throughput, ET and run counts	191
A4.15	LC = 50%, Throughput, ET and run counts	191
A4.16	LC = 80%, Throughput, ET and run counts	192
A4.17	LC = 100%, Throughput, ET and run counts	192
A4.18	Useful Work Costs ET	193
A4.19	Wasted Work Costs ET	193
A4.20	Adaptive Restart 2PLU: T	194
A4.21	Adaptive Restart 2PLU: RC (BC)	194
A4.22	Adaptive Restart BTO: T	195
A4.23	Adaptive Restart BTO: RC	195
A4.24	Adaptive Restart 2PLU: ET	196
A4.25	Adaptive Restart BTO: ET	196
A5.1	Benchmark Error Estimations (90% CI)	199
A5.2	(GRAN = 1, T_REQ = 5): Example Experimental Values (T)	202
A5.3	(GRAN = 1, T_REQ = 5): Example Experimental Values (ET)	202

List of Abbreviations.

2PL	Two Phase Locking
2PLE	Two Phase Exclusive Locking
2PLU	Two Phase Upgrade Locking
ABTO	Adaptive Restart Basic Timestamp Ordering
A2PLU	Adaptive Restart Two Phase Upgrade Locking
BC	Block Count
BT	Batched Means: Simulation Run Length
BTO	Basic Timestamp Ordering
BF_SZ	Buffer Size in 1Kbyte Pages
B_DLY	Blocking Delay
CC	Concurrency Control
CCA	Concurrency Control Algorithm
CPU	Central Processing Unit (time used)
CWA	C Work Area
C-TS(X)	Commit Timestamp of Data Object X
CC_CPU	CPU used by Concurrency Control
CC_IO	I/O used by Concurrency Control
DBCS	Data Base Control System
DBMS	Data Base Management System
DBI	Database Initialiser
DML	Data Manipulation Language
DB_ACC	Database Access Pattern
DB_PG	Database Size in pages
DB_SZ	Database Size in tuples
EMPL	Effective Multi-Processing Level
ET	Elapsed Time
FLL	File Level Locking
grp	get-replace transaction
grs	get-replace transaction, sequential access
gru	get-replace transaction, uniform access
GRAN	Granularity of the Database
I2PLU	Immediate Restart 2PLU
iET	Idealised Elapsed Time
ICI	Interactive Command Interface
IDM	Intelligent Database Machine (Bitton-Lee)
IO	I/O time used
IPRE	Incremental Static Locking
I/O	Input/Output
LC	Level of Conflict
LQM	Logical Queueing Model
LRU	Least Recently Used

List of Abbreviations.

MLL	Mixed File/Page Level Locking
MMPL	Maximum Multi-Processing Level
MPL	Multi-Processing Level
MU	Multi-User
N2PL	Non-Two Phase Locking
NB	Number of Batches (Batched Means Method)
NC	Commit Count
NO CC	No Concurrency Control
NR	Number of Relations
NT	Number of Transactions
OCC	Optimistic Concurrency Control
O/S	Operating System
O_O/S	Operating System Overhead
PC	Probability of Conflict
PLL	Page Level Locking
PRE	Preclaim Locking (Atomic)
PRECI	Prototype of a Relational Canonical Interface
QBE	Query-by-Example
R	Read Only
RIN	Relation Information
RC	Restart Count
RW	Read/Write
RS(T)	Read set of Transaction T
R-TS(X)	Read Timestamp of Data Object X
R_DLY	Restart Delay
sel	select transaction
srp	select-replace transaction
SU	Single-User
SV	Serial Validation
T	Throughput (Transactions/second)
TDBS/C	Testbed Database System in C
TMPL	Transaction Multi-Processing Level
TWW	Thomas Write Rule
T_ACC	Transaction Access Pattern
T_CPU	CPU used by Transaction
T_IO	I/O used by Transaction
T_NGR	Number of Granules Accessed
T_REQ	Number of Requests/Transaction
T_TYP	Transaction Type
T_UPD	Transaction Update Propability
W	Write Only
WS(T)	Write set of Transaction T
W-TS(X)	Write Timestamp of Data Object X

ACKNOWLEDGEMENT.

I would like to express my thanks to my supervisor Dr R.H. Davis for his support during the course of this work.

My thanks also to the Department of Computer Science at Heriot-Watt University and members of staff for their support during the preparation of this thesis and their patience in allowing the many hours of single-user computer time required for the prototype experiments. In particular, to Dr P.J.B. King and Mr A.D. Ramage for help in accessing the SIMULA implementations used for the simulation runs and the Universities of Dundee and Edinburgh for the computer time required. For help with the text preparation system, Dr G. Chen, whose preview program was indispensable, Mr I. Crorie, who translated my sketches into real diagrams, and Mr C .D .F. Miller for help in sorting out the (many) bugs.

Thanks are also due to those who proofread the draft versions of this thesis, Dr R.H. Davis, Dr P.J.B. King, and Ms R. Michaelson; to Dr L. Paterson of the Department of Actuarial Mathematics and Statistics for initial advice on the statistical approach used in the thesis; and last but not least to the many people and institutions who gave time to discuss their work and who provided reports and documentation free of charge.

Finally, my thanks to my family, all my friends, and others for the many forms of support that kept me going during the course of this work.

ABSTRACT

The performance of Concurrency Control Algorithms (CCAs) in databases has been widely studied over the last ten years, mainly using analytic and simulation techniques. Results have been contradictory. The main CCA categories to emerge are locking, timestamping and serial validation. Most studies have considered one or two CCAs within the same framework and only one simulation study has included all the categories.

Experimental studies are rare and none has covered all three categories of CCA. In this study, the performance of atomic static locking (PRE), two-phase exclusive (2PLE) and upgrade (2PLU) locking, basic timestamping (BTO) and serial validation (SV) is compared using a prototype database system. Previous studies suggest that there is little difference in CCA performance at low levels of conflict. This study explores a "worst-case" scenario and uses update only transactions while recognising that such an approach may be biased against CCAs which allow share access such as 2PLU. Benchmark results are presented for single-user transactions. Two transaction types are selected for the multi-user experiments and are run under each CCA and without concurrency control to provide a baseline for comparison. An adaptive restart technique is introduced for 2PLU and BTO and shown to considerably improve performance.

The experiments are repeated using a simulation model and the adaptive restart technique is further investigated. Comparing results with the prototype and other simulation studies indicates that the effect of both blocking and restarting delays are important in determining relative CCA performance.

Chapter 1

INTRODUCTION

1.1. Concurrency Control

In any database system providing shared access to data, the database must remain in a "consistent" state [42] before and after the actions of each user. To enforce this, each user's actions on the database (reads and writes) are grouped into a transaction. These actions are considered to be atomic, that is, they cannot be further subdivided. The effect of the transaction on the database must also be atomic, either all the actions have been applied to the database or none at all. In a multi-user system where several transactions may run concurrently, the actions of a set of transactions may be interleaved. This sequence of actions is called a schedule. A consistent schedule [42] is one which gives each transaction a consistent view of the database. Algorithms which produce such schedules are called concurrency control algorithms (CCA). This thesis is concerned with such algorithms and performance issues associated with them.

An example is given in Figure 1.1 where T1, T2, T3 are transactions and S, P are data objects. Consider the transactions T1 and T2 and data object S in Figure 1.1.

Assume initially S = 100.

T1 {read S, add 100, write S}
T2 {read S, sub 50, write S}
T3 {read P, add 40, write P}

Figure 1.1: Example Transactions

Such transactions may occur for example in an accounting system. Two trivial schedules would be (T1, T2) where the value of S goes from 100->200->150 and (T2, T1) where the value of S goes from

100->50->150. The transactions are executed consecutively (serially) and the schedules are consistent. A possible non-consistent schedule, illustrating the "lost update" problem, is given in Figure 1.2. Again, the initial value of S is 100.

Time	Schedule	Trans. T1	Trans. T2
t	T1-read S	S = 100	
t+1	T2-read S		S = 100
t+2	T1-add 100	S = 200	
t+3	T1-write S	S = 200	
t+4	T2-sub 50		S = 50
t+5	T2-write S		S = 50

Figure 1.2: Lost Update Example

The final value of S is 50 instead of 150 since the final write of T2 has overwritten the result from T1.

Another result of inconsistent schedules arises when one transaction, T_a, reads several items, some of which have been updated and some of which will subsequently be updated by another transaction, T_b. The first transaction's view of the database is inconsistent.

Each transaction has a readset, RS(T), the set of all database items read by that transaction and a writeset, WS(T), the set of all items written by that transaction [12]. Two transactions, T1 and T2, are said to "conflict" [75] if

$$\{RS(T1) \cap WS(T2)\} \text{ or } \{WS(T1) \cap RS(T2)\} \text{ or } \{WS(T1) \cap WS(T2)\}$$

is non-empty. There is no conflict if $\{RS(T1) \cap RS(T2)\}$ is non-empty. Bernstein [12, 13] distinguishes further between types of conflict and classes CCAs according to the method of resolving read-write (RW) and write-write (WW) conflicts.

1.2. Issues

The function of the CCA is to produce a consistent schedule which is equivalent to the serial execution of the set of concurrently executing transactions {T1, T2, ..., Tn} [81, 10, 12]. The effect of the interleaved execution is the same as executing T1 followed by T2 ... followed by Tn. Such an execution is said to be a serialisable execution. Rosenkrantz [90] calls this linearisability. Since every

serialisable execution is equivalent to a serial execution, every serialisable execution must also be correct, that is, it is a consistent schedule. It follows that the aim of CCAs is to generate a serialisable execution which also aims at allowing maximum concurrency. In doing so such algorithms must resolve conflicts either by detection or avoidance.

In Figure 1.1, S is in RS(T1), RS(T2), WS(T1) and WS(T2) giving rise to both RW and WW conflict. T3 conflicts with neither T1 nor T2 hence the actions of T3 may be interleaved with those of T1 and T2.

In summary, the CCA manages read/write access to objects in a database. These objects may vary in size from the entire database to a field of a record. In a relational database this might correspond to the database, a relation, a logical page, a tuple or an attribute. These objects are referred to as granules. Hence for concurrency purposes the database may be considered as a set of granules $\{g_1, g_2, ..., g_n\}$ where n is the granularity of the database. The transaction requests may be either granted or rejected and if rejected the transaction may either block or restart. Blocking may lead to deadlock which in turn can cause restarts.

To check for deadlock a graph of all transactions waiting for resources owned by other transactions is constructed; the waits-for-graph [55]. Deadlock occurs iff a cycle is found in the waits-for-graph. In deadlock avoidance schemes each request may be tested to see if it results in a cycle in the waits-for-graph. For example, if T_a is waiting for a resource owned by T_b, and T_b requests a resource owned by T_a, a cycle occurs in the waits-for-graph (deadlock), the request is denied and one of the two transactions is restarted. If T_b is restarted the CCA is said to be "nonpreemptive" otherwise T_a is restarted and the CCA is said to be "preemptive" [12]. When a transaction successfully completes its actions, the results of the transaction writes (updates) are written back to the database and cannot be undone without running a second transaction. The writing of the results of the transaction is called "committing". A restarted transaction releases all its resources and does not commit its results to the database.

1.3. Algorithms

The most common classes of CCAs are locking, timestamping, and serial-validation, which is also known as "optimistic concurrency control" (OCC). Several variations exist within the locking and timestamping classes. Surveys of methods for centralised and distributed databases are given by Bernstein [12], Kohler [65] (annotated bibliography) and Gardarin [48], the first listing 47 CCAs based on either locking or timestamping or a combination of both. Milenkovic [75] notes the following properties of CCAs as desirable:

1) Parallelism among disjoint transactions should not be constrained

2) The solution should be efficient i.e., impose low storage and computational overhead on the database management system

3) Concurrency control and consequently consistency of the database should be the responsibility of the system and not of its users

4) The solution should be easy to implement and to comprehend

1.3.1. Locking Algorithms

These may be either static or dynamic. In static locking, the objects that the transaction wishes to lock are known at transaction startup time, implying some form of pre-execution analysis. This is also known as preclaim locking or simply "preclaim" for short. It is suggested that transactions with predictable resource requirements are a majority in most cases which has led to CCAs based on classes of transactions [9, 11]. If the transaction cannot acquire all the required locks, it is blocked and must wait until the required locks are free before continuing.

Two forms of preclaim algorithms are:

1) atomic static locking, where the transaction requests all resources and if any are unavailable the requesting transaction is blocked and acquires no resources [86, 23, 107].

2) incremental static locking, where requests are processed by a deadlock avoidance CCA and the transaction blocks if a request is denied but retains resources allocated so far [86, 107].

One potential problem of atomic static locking is that of resource starvation, where for example a transaction requesting two objects (X, Y) is blocked by a succession of transactions requesting either (X) or (Y) .

In dynamic locking, the transaction requests locks as required and if the request is rejected the transaction is again blocked and must wait. Locks may be either exclusive or upgradeable, for example, share to exclusive (read to write) and the CCA may use either deadlock avoidance or detection. Alternatives for dealing with deadlocks are discussed in [4] which provides a good set of references on this area.

One aspect of locking cost is the granule size in the database. From simulation studies [85,87,86] coarse granularity is suggested as the best general solution. This, however, leads to transactions which access few data items, blocking relatively large parts of the database and reducing concurrency. A solution to this is hierarchical locking [51] where the transaction initially acquires locks at a finer granularity and converts these (at a given threshold) to locks at a coarser granularity and higher level in the hierarchy. Deadlock detection is then required at each node in the hierarchy [73].

One important notion is that of two-phase locking (2PL) [42, 51] where the transaction may not release any locks until all locks have been acquired. The two phases are called the growing and shrinking phases where the shrinking phase occurs after transaction commit (i.e., the results of the transaction are written back to the database) and locks can be released.

Other locking methods include non-two-phase locking (N2PL) [57, 77, 58, 78, 79] and predicate locking [42]. In N2PL the database is partitioned into a tree with each node representing a disjoint part of the database. Shared or exclusive locks can then be set on the database at different levels of the tree to control access and conflicts. In predicate locking, a logical lock is set on the database corresponding to some predicate, for example "(all suppliers based in Edinburgh)". However, with this technique, it is not possible to detect overlapping sets of items locked by different predicates and hence resolve conflicts. All locking algorithms considered here will be 2PL.

1.3.2. Timestamping

In this method, the CCA maintains a logical clock from which each transaction, T, is assigned a clock value or timestamp, TS(T), on starting execution [12]. Each data object, X, in the database is assigned a read timestamp, R-TS(X), and a write timestamp, W-TS(X), stating the time of the last read and write respectively. The value assigned to R-TS(X)/W-TS(X) is the start time, TS(T), of the transaction which has read/written the object X. The timestamps are essentially sequencers which allow the actions of the transactions to be applied to the database in startup order. This is known as timestamp ordering.

Consider the transactions T1, T2 and data object S, from Figure 1.1, starting at times (t), (t+1) respectively and the schedule of Figure 1.3. Note the read and write timestamps for S.

Time	Schedule	R-TS(S)	W-TS(S)
t	T1-begin		
t+1	T2-begin		
t+2	T1-read S	t	0
t+3	T1-add 100	t	0
t+4	T2-read S	t+1	0
t+5	T1-write S	t+1	0

Figure 1.3: Timestamp Example

The action at (t+5) is the "write S" from T1 which creates RW conflict since R-TS(S) > TS(T1). The write will be rejected and T1 restarted at time (t+k) (k>5). Note that it is possible to set up a cycle of restarts if T1 (TS(T1) = (t+k)) is restarted before T2 has written back the value of S [99, 118, 33]. Timestamping with restarts is called basic timestamp ordering (BTO). Instead of restarting transactions the CCA may delay the transaction. This technique is called conservative timestamp ordering (CTO) and requires the reads and writes to arrive in timestamp order [13]. The Thomas write rule (TWW) [112] presents an improved algorithm for resolving WW conflicts. Under the assumption that transactions read an object before writing it, this technique is equivalent to BTO [23].

1.3.3. Serial Validation

This method, also known as certification [12] and OCC [67], allows transactions to continue unchecked until a commit (or validation) point when transactions are tested to prevent non-serialisable results. In OCC, it is argued that conflicts are in fact rare (the "optimistic" assumption) and each transaction keeps local copies of database objects before the validation point (shadow values [7, 59, 60]). In Serial Validation, each transaction is assigned a sequence number at startup time and the readsets and writesets are examined for potential conflict prior to commit. In the worst case where validation repeatedly fails, locking may be used. A more efficient implementation of this algorithm [23] is used in this study where each transaction is assigned a timestamp on startup, S-TS(T), and a commit timestamp, C-TS(T), on commit. A write timestamp, W-TS(X), is maintained for each object X, which is the commit timestamp of the most recent writer of that object. A transaction is only allowed to commit if S-TS(T) > W-TS(X) for all X in the readset of T.

Consider T1 and T2 from Figure 1.1 and the schedule in Figure 1.4.

Time	Schedule	W-TS(S)
t	T1-begin	0
t+1	T2-begin	0
t+2	T1-read S	0
t+3	T1-write S	0
t+4	T2-read S	0
t+5	T2-write S	0
t+6	T1-commit	t+6
t+7	T2-commit	t+6

Figure 1.4: Serial Validation Example

At (t+6) T1 enters the validation phase and is validated, setting W-TS(S) to (t+6), the value of C-TS(T1). At (t+7) T2 enters the validation phase hence C-TS(T2) = (t+7). Since S-TS(T2) = (t+1) and W-TS(S) = (t+6), S-TS(T2) < W-TS(S) so T2 is rejected. Essentially [23] uses timestamping to avoid the readset/writeset maintainence overhead in [67].

1.3.4. Hybrid Algorithms

Algorithms also exist using combinations of the above techniques, most commonly 2PL and timestamp ordering [12]. The "Wait-Die" and "Wound-Wait" protocols [90,33] are deadlock avoidance techniques which use locking with timestamps and prevent cycles from occurring in the waits-for-graph. A combined preclaim and timestamp algorithm is proposed in [75]. The optimistic approach, originally designed for systems with low levels of conflict, could be combined with 2PL if the conflict level rises beyond a given threshold.

1.4. Performance

Clearly if a CCA is to be efficient it must execute the set of transactions in no more time than the equivalent serial execution. Gray [53] indicates that conflict between transactions in a database system is rare. In such a case, concurrent execution is indeed required to make best use of system resources such as CPU and I/O parallelism. Studies on granularity [85,87,86] indicate that suitable choice of granularity can also reduce overhead. Other factors affecting performance include the ratio of large to small transactions (transaction mix), read only to read/write transactions, level of conflict, restart delays and the concurrency control overhead.

Performance studies can be divided into three categories; analytical, simulation and experimental. Many have used a combination of techniques, usually analytical and simulation. The most studied technique has been locking:

1) analytical studies [84,71,46,113,107],

2) simulation studies [85,87,86,68,23,24,110,25,4]

and more recently

3) experimental studies [66,82,63]

Fewer studies have been carried out on timestamping [46,23,111] using simulation techniques and the optimistic approach [82,63] using experimental testing.

Some general testbed benchmark studies on databases [54, 17, 18, 93] have also begun to appear and one study [56] has used a Prolog system to evaluate concurrent system design.

1.5. This Study

This thesis is an empirical study of the performance of five basic CCAs: preclaim (PRE), two-phase exclusive locking (2PLE), two-phase upgrade (share to exclusive) locking (2PLU), basic timestamp ordering (BTO) and serial validation (SV). Performance of those five CCAs plus that of the no concurrency control case (NO CC) is examined under different workloads at different conflict levels using update transactions (read/write) in a multi-user centralised testbed database system. The transactions are of two types

1) those which access the database on a record level

2) those which operate on relations.

These we feel cover most types of current database systems. Two problems are addressed; (1) would the results of previous studies predict the outcome from a prototype experimental system; (2) could a suitable simulation model reflect the prototype system and to what degree of accuracy.

Chapter 2 presents the components of a database system and factors involved in modelling. The CCAs used in this study are presented as informal procedures and described in the context of a logical queueing model. Metrics and assumptions from previous studies are summarised. Previous results are reviewed and used as a basis to establish performance metrics and workload parameters for this study.

The prototype system, TDBS/C, is introduced in Chapter 3. An overview is presented of the data manipulation language and the internal organisation and mechanisms. The design of the experiments, test data and transactions are described, followed by the results of the single-user benchmark tests. These are used to generalise conclusions from the results of the multi-user CCA tests. Finally, the main conclusions from the benchmarks are presented.

In Chapter 4, performance metrics, test data and transactions for the multi-user experiments are

described. A brief description of some preliminary experiments is given followed by the two sets of CCA experiments and a detailed discussion of the results. An adaptive restart technique is presented and tested under 2PLU and BTO and the results contrasted with the fixed length restart case.

In Chapter 5, a physical queueing model, derived from the logical model of Chapter 2, is used as a basis for a simulation model of the prototype. The simulation results are presented and discussed. The adaptive restart technique is further investigated using the simulator. The simulation results are compared with those from the prototype, differences noted and conclusions drawn.

The scope of the study is broadened in Chapter 6 where the prototype results are compared with results from some earlier experimental studies. The simulation results are firstly compared in general with some previous results and then in detail with a set of simulation studies based on the same model. A comparison with an analytical model concludes the chapter.

Chapter 7 places this study in a wider context, reviews the results and notes some limitations. A brief summary of results is followed by suggestions for future research.

Appendix 1 presents a detailed description of TDBS/C. The results of the single-user benchmarks, prototype and simulation experiments are listed in Appendices 2, 3, 4 respectively. Appendix 5 describes the operating system context, experimental and statistical techniques used in this study.

Chapter 2

PERFORMANCE MODELS

In this chapter we describe a centralised multi-user database system, factors involved in performance measurement, the concurrency control algorithms (CCAs) used in this study, a logical queueing model for the database system and review previous studies. Finally, performance metrics and workload parameters for this study are presented.

2.1. Database System

The components in a multi-user database system are reviewed. The system is presented in Figure 2.1.

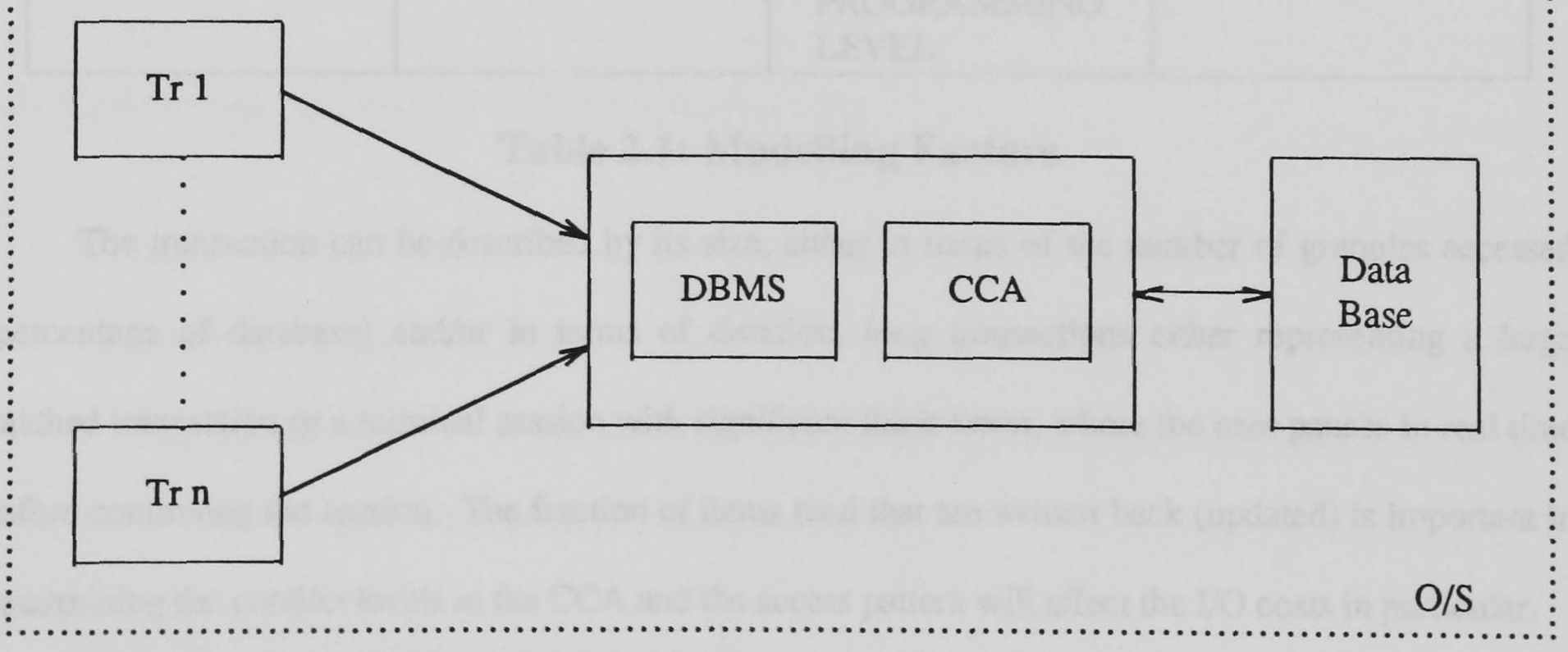


Figure 2.1: Database System

The system comprises a number of transactions (either batch or terminal transactions) which access the database via the database management system (DBMS). The DBMS includes a CCA component which schedules the requests issued by the transactions and is responsible for resolving conflicts between transactions. The majority of database systems are run as part of the standard

Operating System (O/S) on the given machine.

Table 2.1 summarises the main factors involved in modelling such a system. Clearly many of these are interdependent.

Transaction	Database	CCA	O/S
SIZE small/large (no. granules accessed)	SIZE (no. of objects)	ALGORITHM TYPE (blocking, restarting)	No. CPUs
DURATION short/long	GRANULARITY (no. of objects/granule)	DEADLOCK (prevention, detection)	No. DISKS
READ/WRITE ratio	ACCESS COSTS (I/O, CPU for DBMS)	LOCATION of CCA INFORMATION (primary, secondary memory)	MEMORY SIZE (primary, secondary)
ACCESS PATTERN (uniform, random, sequential)		ACCESS COSTS (I/O, CPU)	MULTI-PROGRAMMING LEVEL
THINK TIME		MULTI-PROGRAMMING LEVEL	

Table 2.1: Modelling Factors

The transaction can be described by its size, either in terms of the number of granules accessed (percentage of database) and/or in terms of duration, long transactions either representing a large batched transaction or a terminal session with significant think times, where the user pauses in real time before continuing the session. The fraction of items read that are written back (updated) is important in determining the conflict levels in the CCA and the access pattern will affect the I/O costs in particular.

A transaction may be modelled as

$$\textit{Begin transaction } \{a_1, a_2, ..., a_n\} \textit{ End transaction}$$

where the "*Begin transaction*" initiates startup functions in the DBMS and notifies the system of the existence of the transaction, the $\{a_i\}$ a set of actions (read/write) and the "*End transaction*" signals the DBMS to commit the transaction if possible.

The database is essentially modelled as a set of n granules $\{g_1, g_2, ..., g_n\}$. Each granule may contain one or more data objects. The access costs depend on the O/S disk overheads (I/O cost), the DBMS implementation (CPU cost) and the type of access, for example by record or by relation and whether sequential, random or uniform.

The CCAs incur both CPU and I/O costs. If the CCA information is stored in primary memory then the latter will be zero. The CCA may also add a delay cost for blocking and/or restarting the transaction.

The O/S overheads will depend on the number of processors, disks, and memory size. These overheads will heavily influence the other costs in the system (DBMS and CCA CPU and I/O costs) as well as adding their own costs to the transaction.

Both the CCA and O/S will be affected by limits on their respective multiprogramming levels, the O/S multiprogramming level being dominant. Within the system transactions will be subject to both data contention (access to the database objects) and resource contention (access to the processor(s) and disk(s)) [107].

2.2. Concurrency Control Algorithms (CCAs)

In chapter 1 the basic issues involved in concurrency control were presented. The algorithms used in this study are now presented in greater detail. They are preclaim (PRE), two-phase exclusive locking (2PLE), two-phase upgrade (share to exclusive) locking (2PLU), basic timestamp ordering (BTO) and serial validation (SV). Two assumptions are introduced here:

1) that all objects are read before being written

("no-blind-writes" assumption)

2) that updates to the database are deferred until commit time

Three tables, which are used for the CCA information, are presented in Figure 2.2.

TRTAB (Transaction Table)		
TRID	STATUS/C-TS	S-TS

RESTAB (Resource Table)		
OID	OWNER/R-TS	WAITS/W-TS

OWTAB (Owner/Waits Table)		
TRID	MODE	NEXT

Figure 2.2: CCA Tables

The TRTAB contains information on transactions; identification (TRID), status (STATUS: executing, blocked, commit, restart) and startup timestamp for BTO, SV (S-TS). For SV a commit timestamp (C-TS) is also kept. The RESTAB contains information on database granules; the granule identification (OID), either a pointer to the owner list for PRE, 2PLE, 2PLU (OWNER), or the R-TS, timestamp for the last read of this granule for BTO and either a pointer to the waits list for 2PLE, 2PLU (WAITS) or the W-TS, timestamp for the last write to this granule for BTO, SV. The OWTAB contains information on the owner transaction(s) (TRID) for PRE, 2PLE, 2PLU, transaction(s) waiting for a granule (NEXT) for 2PLE, 2PLU and the type of lock held (MODE), share (read) or exclusive (write).

An abstract model of CCAs which treats the CC information tables as a database such that each table is a relation, was proposed by Carey [23, 24] to facilitate implementation independent descriptions of CCAs. This model considered both the CPU and storage overheads. The results of the overhead analysis may be summarised as

1) for storage overhead, 2PLU was best under low conflicts, BTO best under high conflicts and SV the worst of the three

2) for CPU overhead (no conflict case), 2PLU was the best of the three, BTO second best under infrequent writing and SV second best under frequent writing.

From this model, it is clear that different CCAs will be "best" in different regions of the parameter space and that there is no one overall "best" algorithm. While examining the storage overhead, the model does not cover I/O costs which in a real system would probably be independent of the storage overhead and constant if the CC database remains reasonably small and page blocking is used.

2.2.1. Preclaim Locking (PRE)

This technique, also called atomic static locking, requires the transaction to exclusively lock all granules accessed (read or write) at transaction startup time. If not all granules can be claimed the transaction blocks without holding any locks. This algorithm is conservative, does not allow deadlocks and guarantees that the transaction will commit once started.

2.2.2. Two-Phase Exclusive Locking (2PLE)

On transaction startup (begin) the transaction id is entered in the TRTAB and the status set to execute. The transaction then requests each granule dynamically, "request(TRID, OID, EXCLUSIVE)", and the 2PLE CCA checks the request against the tables. The 2PLE request procedure is given informally in Figure 2.3.

```
request(FTRID, FOID, FMODE)
{
    lookup(FOID);
    if (not found) {
        enter FOID in RESTAB;
        allocate FOID to FTRID;
        request_status = granted;
        }
    else {
        add FTRID to waits list of FOID;
        FTRID.STATUS = blocked;
        request_status = rejected;
        }

    if (FTRID.STATUS is blocked) {
        construct(waits-for-graph);
        check_for_cycle(waits-for-graph);
        if (cycle found) restart(FTRID);
        }

    return(request_status);
    }
```

Figure 2.3: 2PLE Request Procedure

On commit all updated granules are written back to the database and either released or reallocated to the next transaction in the waiting list which may then resume execution; the STATUS is changed from blocked to execute. On restart the transaction's resources are released and possibly reallocated.

The deadlock detection strategy used here is similar to that in [1,23] in that a check for cycles in the waits-for-graph is carried out each time a transaction blocks and that transaction is selected as victim for restart. This method is easy to implement and should have a relatively low cost given that deadlocks are rare and when they do occur, the cycle length is short, usually only involving two transactions [51].

2.2.3. Two-Phase Upgrade Locking (2PLU)

On startup the transaction id is added to TRTAB and status set to execute. The transaction then requests each granule in share mode and then if the granule is updated, in exclusive mode. The 2PLU request procedure is given informally in Figure 2.4.

```
request(FTRID, FOID, FMODE)
{
   lookup(FOID);
   if (not found) {
      enter FOID in RESTAB;
      if (FMODE == share) {
         allocate FOID to FTRID in FMODE;
         request_status = granted;
         }
      else error( "objects should be read before update" );
      }
   else if (FOID is exclusive locked) {
      add FTRID to FOID waits list;
      FTRID.STATUS = blocked;
      request_status = rejected;
      }
   else if (FMODE is share) {
      if (FOID waits list is empty) {
         allocate FOID to FTRID in share mode;
         request_status = granted;
         }
      else {
         add FTRID to FOID waits list;
         FTRID.STATUS = blocked;
         request_status = rejected;
         }
      }
   comment FMODE is exclusive;
   else if ((FOID waits list is empty) and
            (this FTRID is unique owner)) {
      upgrade FTRID lock on FOID;
      request_status = granted;
      }
   else {
      add FTRID to FOID waits list;
      FTRID.STATUS = blocked;
      request_status = rejected;
      }

   if (FTRID.STATUS is blocked) {
      construct(waits-for-graph);
      check_for_cycle(waits-for-graph);
      if (cycle found) restart(FTRID);
      }

   return(request_status);
   }
```

Figure 2.4: 2PLU Request Procedure

Similarly to 2PLE, if blocked, the transaction re-requests the resource and if restarted, its resources are released and reallocated if the waits list is non-empty.

2.2.4. Basic Timestamp Ordering (BTO)

On startup the transaction is added to TRTAB together with a startup timestamp, S-TS. Each granule in RESTAB has two timestamps associated with it, R-TS, the time last read and W-TS, the time last written. The BTO request procedure is given informally in Figure 2.5.

```
request(FTRID, FOID, FMODE)
{
   lookup(FOID);
   if (not found) {
      add FOID to RESTAB;
      if (FMODE is read) set FOID.R-TS = FTRID.S-TS;
      else error("objects should be read before update");
      }
   else if (FMODE is read) {
      if (FTRID.S-TS < FOID.W-TS) FTRID.STATUS = restart;
      else FOID.R-TS = max(FOID.R-TS, FTRID.S-TS);
      }
   comment FMODE is write;
   else if ( FTRID.S-TS < FOID.R-TS or FTRID.S-TS < FOID.W-TS)
               FTRID.STATUS = restart;
   else FOID.W-TS = FTRID.S-TS;

   if (FTRID.STATUS is restart) restart(FTRID);

   return(granted);
   }
```

Figure 2.5: BTO Request Procedure

Note that restarted transactions are assigned a new timestamp greater than their previous timestamp. The timestamp of the oldest active transaction (in TRTAB) can also be used to flush RESTAB entries with R-TS and W-TS older than the oldest transaction.

2.2.5. Serial Validation (SV)

This algorithm is due to Carey [23], who presents a formal proof of its equivalence to that proposed by Kung [67] and uses timestamps for the validation phase. On startup the transaction is added to TRTAB together with a startup timestamp. The transaction then proceeds to perform reads and writes on local copies and also keeps a record of its read and write sets. On commit the transaction enters the validation phase and receives a commit timestamp C-TS. Each granule in the database has a write timestamp, W-TS, which is set to the commit timestamp of the last validated transaction which

wrote that granule. Basically each granule in the transaction's read set must have been committed before the transaction started. The SV validate procedure is given informally in Figure 2.6.

```
validate(FTRID)
{
   valid = true;

   foreach (Xr in readset(FTRID))
      if (FTRID.S-TS < Xr.W-TS) valid = false;

if (valid) {
   foreach (Xw in writeset(FTRID))
      Xw.W-TS = FTRID.S-TS;
   commit writeset to database;
   }
else restart(FTRID);
}
```

Figure 2.6: SV Validation Procedure

2.3. Logical Queueing Model (LQM)

The model adopted here, Figure 2.7, is one presented in [3,4] which is an extension to that presented in [23,24,25] which in turn was an extension to the model presented in [85,87,86] It is used for two reasons, firstly it provides a common framework for discussion of issues and assumptions in concurrency control and secondly, it adequately describes the algorithms used in this study. Other studies either use queueing models [94,71,110] or other techniques; probabilistic analysis and queueing model [84], Markov chain model [26], random graph model [44] and flowgraphs plus equations (analytic technique using steady state mean values) [107].

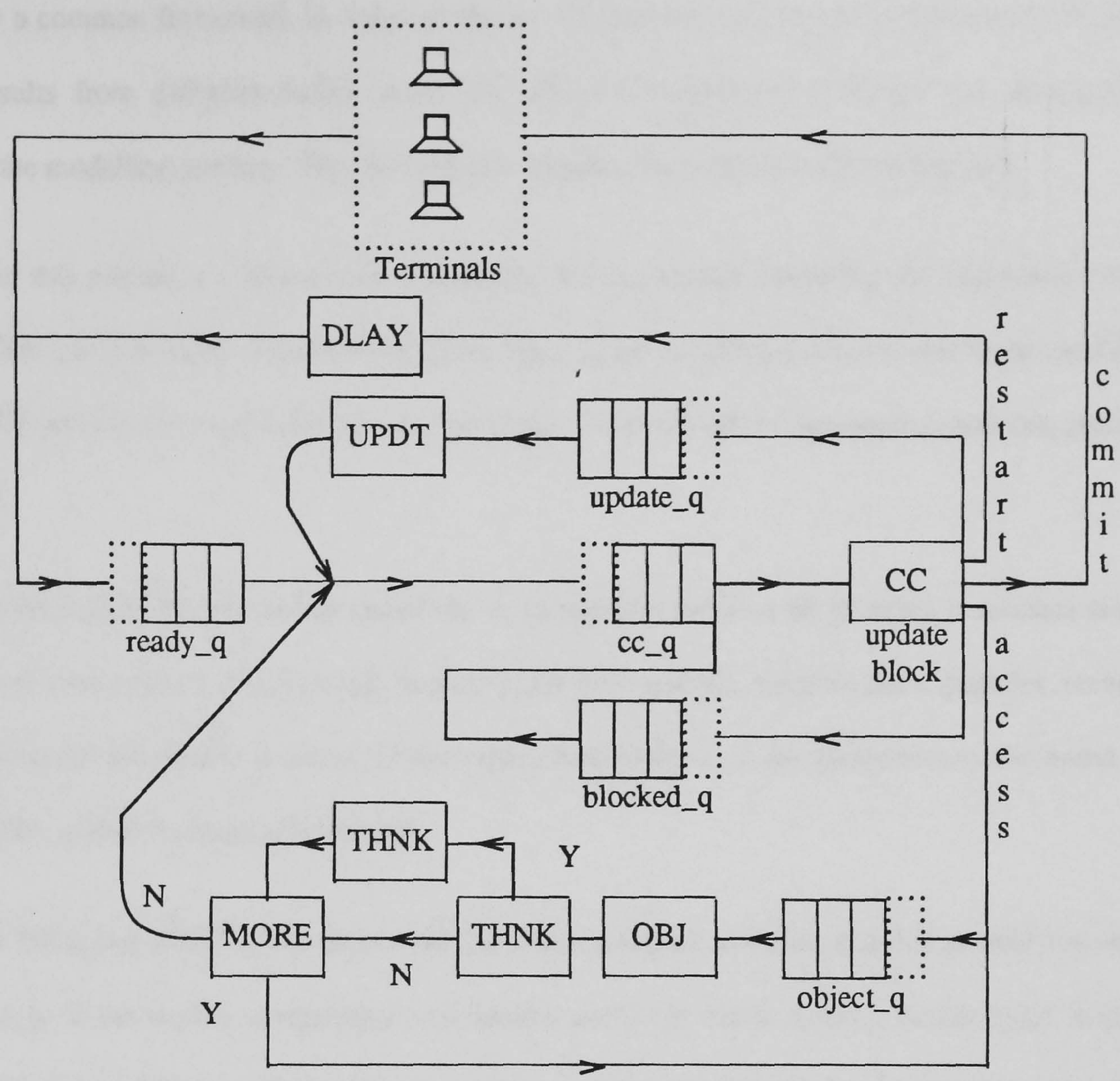


Figure 2.7: Logical Queueing Model (LQM)

(1)	Database System Model
	Hardware and software characteristics: disks and CPUs Database size and granularity maximum multiprogramming level CCA
(2)	User Model
	arrival process for transactions open or closed system interactive or non-interactive
(3)	Transaction Model
	reference behaviour (CC requests) processing requirements (database requests) types of transactions

Table 2.2: General Performance Model

As well as the LQM, [3] presents a general performance model for concurrency control (CC) which has three parts. This model is summarised in Table 2.2. These correspond to the factors listed in Table 2.1.

Given a common framework in which to discuss CC performance it is still extremely difficult to compare results from different studies given the different measurement metrics and assumptions involved in the modelling process. The next section examines these metrics and assumptions.

To end this section, we give a brief description of a transaction traversing the LQM under each CCA considered in this study. Transactions in the ready_q are considered inactive and those receiving or waiting for service are considered active. Recall the "no-blind-writes" assumption and that updates are deferred.

In the PRE CCA the transaction enters the cc_q, requests locks on all granules it accesses and if granted moves to the object_q (which will be empty) for each granule, accesses those granules, reenters the cc_q to commit and finally commits. If the request fails in the cc_q the transaction cycles round the block_q and cc_q until the request is granted.

In the 2PLE and 2PLU CCAs the transaction enters a request in the cc_q and if granted continues to the object_q. If the request is rejected the transaction enters the block_q and a further check is made for deadlock; the waits-for-graph is created and checked for cycles. If deadlock has occurred the transaction is restarted after releasing all currently held locks, possibly unblocking other transactions. A restart delay is introduced to lessen the chance of repeated deadlock and restart.

In the BTO CCA transactions are not blocked if the request is refused but are restarted again after a delay as in the 2PLE and 2PLU case.

In the SV CCA the first request is always granted; essentially it announces arrival of the transaction in the system. The granules are accessed (reads and writes to local copies) and only on commit does the transaction return to the cc_q. There, it enters the validation phase and if successful, the update_q. Otherwise the transaction is restarted.

2.4. Performance Metrics and Assumptions

Most studies present results in terms of throughput (number of transactions per second) and average elapsed time per transaction [84, 71, 74, 23, 24, 26, 66, 110, 111, 25, 107].

The granularity of the database was studied in [85, 87, 86]. Machine utilisation (CPU and I/O time) and useful I/O time plotted against the number of granules. Average response time and useful I/O were plotted against the number of locks for the distributed case. Granularity in locking CCAs was studied in [84] and plotted against mean number of blocked granules. Carey provides both throughput vs granularity results for PRE, 2PLE, 2PLU, BTO, SV and the "Wait-Die" protocol [90] and restart counts vs granularity [23].

The multiprogramming level of database systems (MPL) may either be expressed as the number of transactions in the system (TMPL), the maximum allowable level (MMPL) or the effective level (EMPL). The EMPL is a measure of transactions in the system which perform useful work. MMPL and EMPL are plotted for different probabilities of conflict in [44]. Throughput, conflict rates, delays and disk utilisation are plotted against MMPL in [3]. Throughput vs MMPL is presented in [107].

The probability of conflict between transactions vs response times is presented in [71, 110, 44] while other metrics such as restart counts [23, 82, 107], arrival rates and interarrival times vs response time [113], transactions waiting vs MMPL [44] and EMPL/q, where q is a measure of increased work caused by backup and reexecution of transactions, in [82].

Clearly there exist a large number of possibilities for measuring CC performance. Comparison is made more difficult by different interpretations of some metrics, for example definitions of conflict and multiprogramming level as MMPL or EMPL. Restrictions on the choice of metrics may also be imposed by the system used for such tests. Finally the assumptions behind the tests, either implicit or explicit, can strongly influence the outcome Since there are now sufficient studies available with apparently conflicting results, these assumptions are beginning to be investigated [3, 107, 4]. Empirical results for real systems are rare and provide the motivation for this study as a contribution to this area.

Contradictory results listed in [3] include some studies which suggest locking is better than restarting and SV outperforming locking (see also [63]) and some studies which suggest the opposite. Other assumptions listed are:

- infinite resources (multiple CPUs and disks)

- writelocks immediately set on items to be updated

- "fake restart" (that the restarted transaction is replaced by an independent transaction)

Given such assumptions and others which may be a matter of implementation choice, for example whether CC information is stored in primary memory, translating the results of such studies to real database systems is even more difficult. Even then there are a large number of relevant factors both in the DBMS and O/S and their interaction. Any results must be presented in a well described context, both in terms of workload parameters and assumptions.

In this study, using a real database, we will also need to make assumptions. With an explicit awareness of those assumptions and the results from other studies, we hope to gain a clearer understanding of influences on CC performance, particularly from the point of view of the database implementor.

2.5. Previous Results

In the above section, the main issues to emerge were granularity, multiprogramming and conflict levels with other indicators such as block and restart counts, waiting times and resource utilisation. The principle resources are CPU and disk, and main performance metrics throughput and average response time.

The first study on granularity [85] concluded that coarse granularity, in the order of 10..100 granules, was optimal. The tradeoff is between fine granularity which maximises parallelism but has a potentially high management overhead and coarse granularity with less parallelism but lower overhead. The granule may be anything from the field of a record (fine) to the entire database (coarse). This model used PRE CCA and did not consider rollback (recovery). Further, the result holds for both I/O bound, say a system with 1 disk, and CPU bound transactions and assumes that maintaining locks (CC information) in primary memory makes little difference. In a second paper [87], certain assumptions were reconsidered and a lock hierarchy and dynamic locking explored. It was noted that if the access pattern was random rather than sequential, finer granularity performed better and that dynamic locking

was generally more expensive that preclaim. Coarse granularity was still the preferred solution in most cases. Further factors supporting finer granularity are given in [86].

While Ries suggests a coarse granularity near the lower end of the range (10..100), Carey [23] suggests the upper end of the range is optimal and goes on to propose a granularity hierarchy for mixed (large/small) transaction loads. A granularity curve of throughput vs granularity (coarse to fine) is presented in [107] which first decreases then increases and finally decreases again, with excessive data contention taking place at coarse granularity levels and excessive resource contention at fine granularity levels. It is noted that previous results depend on system parameters which effectively define a window on this curve; throughput decreases with increasing number of granules for long transactions and increases for short transactions. Similar conclusions are reached in [66]. A summary is give in Table 2.3.

coarse granularity	fine granularity
most transactions	small transactions
mixed (large/small) loads	short transactions
locks held in secondary memory	locks held in primary memory
sequential access patterns	random access patterns

Table 2.3: Summary of Granularity Results

Both levels of granularity and multiprogramming affect the degree of conflict in a database system and the essential point of CCAs is to resolve conflicts. Increasing the MMPL can lead to decreasing EMPL and increase the number of restarts [107]. A saturation point was reached at TMPL of 4 in [23] with I/O being identified as the bottleneck resource. In [3], TMPL levels of 5, 10, 25, 50, 75, 100, and 200 were studied and saturation, in the limited resources case, noted at TMPL = 25 although conflict was low in this experiment. A further experiment yielded a throughput peak for 2PLE at TMPL = 25 (although the actual peak may be anywhere between 10 and 50) and for 2PLU and SV at 10. Results of EMPL vs TMPL at given probability of conflict levels [44] also indicate as TMPL increases (0 to 100), EMPL does not increase beyond 20 at probability value 0.1 (the highest value given). Results of average response time against probability of conflict [71] show almost asymptotic curves beyond a probability value of 0.75 for both locking and timestamping. Finally [94], a study on transaction

systems, also supports these conclusions.

Derivations of probability of conflict differ between studies and many use granularity and/or TMPL to increase/decrease the probability of conflict without stating actual values. Further the "infinite resources" assumption applies to many results and may tend to make them optimistic, given additional constraints imposed by say the EMPL of the O/S. Table 2.4 tentatively summarises these results.

TMPL limits			probability of conflict limits	
2PLE	25	(10-50?)	"Die-Wait"	0.75
2PLU	10			
SV	10		BTO	0.75

Table 2.4: Summary of TMPL and Conflict Results

All these studies highlight some aspects of the CC performance problem, indicate trends under given conditions and possible limits on parameter values for actual systems. The results are used here to determine parameter values in the experiments described in chapters 3, 4, and 5.

2.6. Performance Metrics and Workload Parameters

We conclude this chapter and overview of previous work by summarising and discussing the requirements of performance metrics and workload parameters for this study.

2.6.1. Performance Metrics

From the above overview both throughput and average elapsed time emerge as important. In addition, measures of both CPU and I/O usage are necessary in determining tendencies of transactions to become CPU or I/O bound. In some simulation studies these measures are further divided into T_IO and T_CPU, the resources used by the transaction and CC_IO and CC_CPU, the resources used by the CCA. Further, in a real system with limited resources there will be extra overhead from resource contention in the O/S, O_OS. We hypothesise also that the results in most previous studies correspond to an idealised system corresponding to (T_IO + T_CPU + CC_IO + CC_CPU) and do not take into account O_OS. Some of these metrics are used in workload parameters in models of systems but here we require to measure the values in our real system. In PRE, 2PLE and 2PLU, the elapsed time will be

affected by blocking overheads while in 2PLU, BTO and SV restart counts (RC) will indicate the effectiveness of the CCAs. The importance of a delay for restarted transactions, R_DLY, is noted in [94,3] with the latter suggesting an adaptive delay based on the average transaction length. The performance metrics are summarised in Table 2.5.

metric	meaning
ET	average elapsed time
T	throughput (trans/second)
iET	idealised ET
T_IO	I/O overhead for trans.
T_CPU	CPU overhead for trans.
CC_IO	I/O overhead for CCA
CC_CPU	CPU overhead for CCA
O_OS	overhead for O/S
BC	block count
RC	restart count

Table 2.5: Performance Metrics

The ET is a measure of (T_IO + T_CPU + CC_IO + CC_CPU + O_OS). The iET is (ET – O_OS).

2.6.2. Workload Parameters

The database is specified by both number of tuples and of pages, the granularity and number of relations. The number of transactions in the system is specified and for each, the number of granules accessed, access requests, probability of update request and the access pattern; sequential, random or uniform. Finally the type of transaction is specified; access by record (tuple) or relation. Concurrency control is specified by the algorithm (NO CC, PRE, 2PLE, 2PLU, BTO, SV), the multiprogramming level (MMPL) and finally restart delay (fixed, adaptive).

The workload parameters are summarised in Table 2.6.

Parameters	Meaning
DATABASE:	
DB_SZ	database size: tuples
DB_PG	database size: pages
GRAN	no. pages/granule
NR	no. relations
DB_ACC	database accessed
TRANS:	
NT	no. transactions
T_REQ	no. requests/trans.
T_ACC	access pattern
T_TYP	type: record/relation
T_UPD	update probability
BF_SZ	buffer size
CC:	
CCA	type of algorithm
MMPL	max. multiprog. level
B_DLY	blocking delay
R_DLY	restart delay

Table 2.6: Workload Parameters

2.7. Chapter Summary

We began by presenting a model of a multi-user database system and the main factors involved in performance studies. Models of components of this system, transaction, database and CC, were then presented and discussed before describing the five CCAs studied here (PRE, 2PLE, 2PLU, BTO, SV) as informal procedures. Next, a logical queueing model (LQM), first presented by Ries and later extended by Carey and Agrawal, was introduced to give an overview of a multi-user transactional database system. Each CCA was then described in terms of the LQM. Performance metrics and assumptions were reviewed from previous studies, with granularity and the multiprogramming level emerging as major influences on conflict between transactions. From this, performance metrics and workload parameters were established for this study.

Chapter 3

TESTBED DATABASE SYSTEM

In this chapter we describe the testbed database system, TDBS/C (Testbed DataBase System in C), the data and transactions used in and the results of the single-user benchmark tests. These results are important in that they define the system behaviour for a given set of parameters and support assumptions used in the multi-user tests presented in the next chapter. [17] notes that benchmarking is an important first step towards a methodology of performance evaluation and lists the motivations behind performance evaluation of real systems. We feel that in the last few years there has been a growing trend towards testing aspects of database system design in testbed systems [17, 66, 18, 59, 60] and that such results are important both from the implementation viewpoint and for comparison with other studies. The test data, test database and test transactions are presented and finally the benchmark results and conclusions.

3.1. The Testbed System: TDBS/C

TDBS/C [92] evolved from an implementation of PRECI (Prototype of a RElational Canonical Interface) [36, 37, 119] in C under UNIX† [89]. PRECI was designed to provide a common interface to different database models, for example, relational, network and hierarchical [115]. It was designed as a batch system running Fortran programs with embedded data manipulation language (DML) commands [119]. The terminology used here is that of relational databases [29, 30, 118, 33, 31, 34, 35].

† UNIX is a trademark of Bell Laboratories.

3.1.1. Overview of TDBS/C

TDBS/C is described in Appendix 1. Here we give an overview of the relevant parts for this study. As in PRECI, TDBS/C is based on the ANSI/SPARC three level architecture [114] which models the database as comprising:

1) user views which are defined on

2) the conceptual view, or schema which combined with

3) the storage, or physical specification, defines the database

See Appendix 1, Figure A1.1 for details. The corresponding elements of TDBS/C are the schema (logical definition) and storage schema (physical definition) which when processed by their respective translators are used by the database initialiser (DBI) to generate the empty database. The physical database consists of a database definition file and one file for each relation defined in the schema. The process is summarised in Figure 3.1. Example schemas are given in Appendix 1, Figures A1.13, A1.14, A1.15.

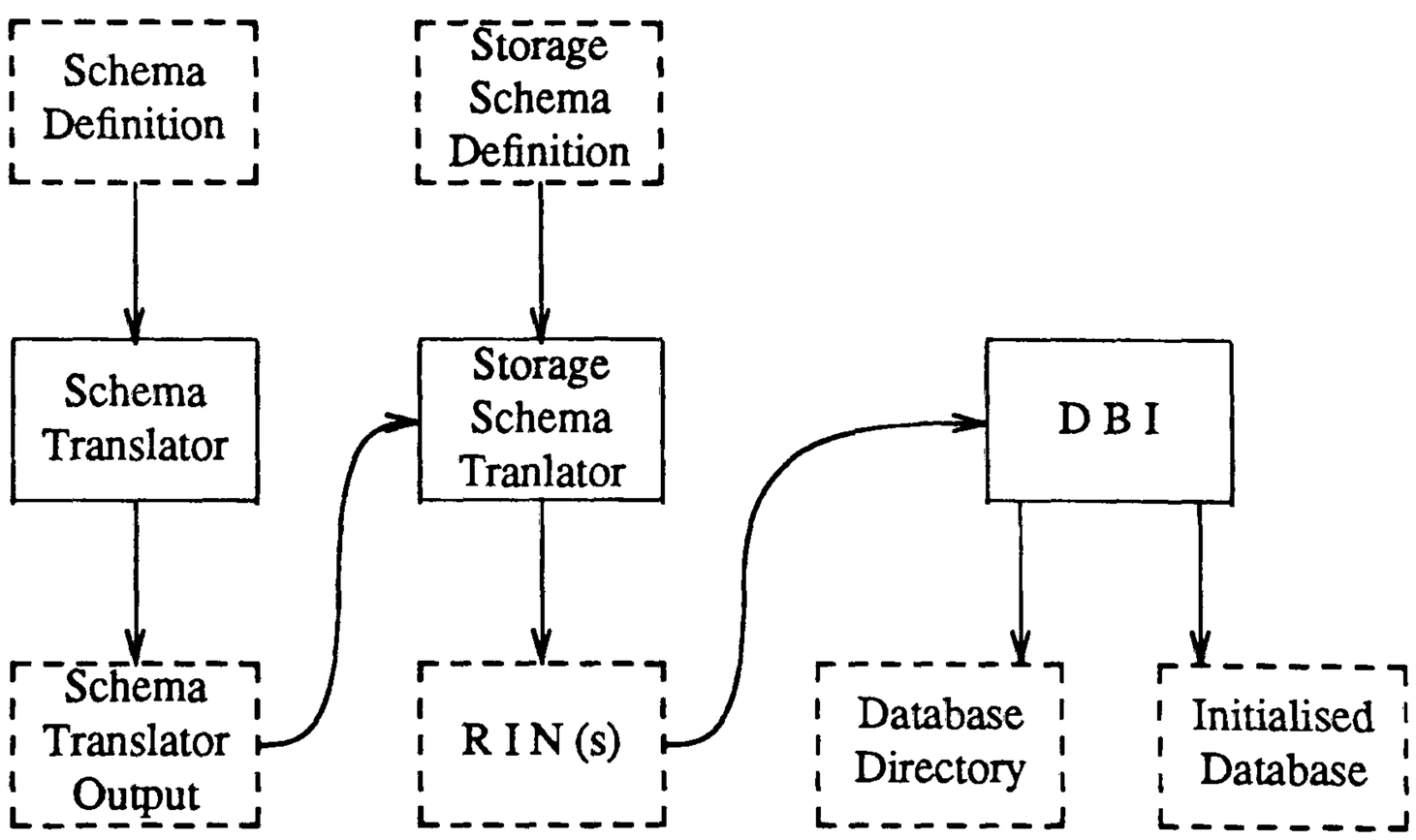


Figure 3.1: Database Initialisation

The transaction is specified as a C program with embedded DML which uses a view definition specified on the schema. The source program is precompiled and appropriate "C Work Areas" (CWAs) for each view are generated and "included" [61] in the C program which is then compiled by the C compiler and loaded with the Database Control System (DBCS) to create a transaction run-unit to access the database. This process is described in Figure 3.2.

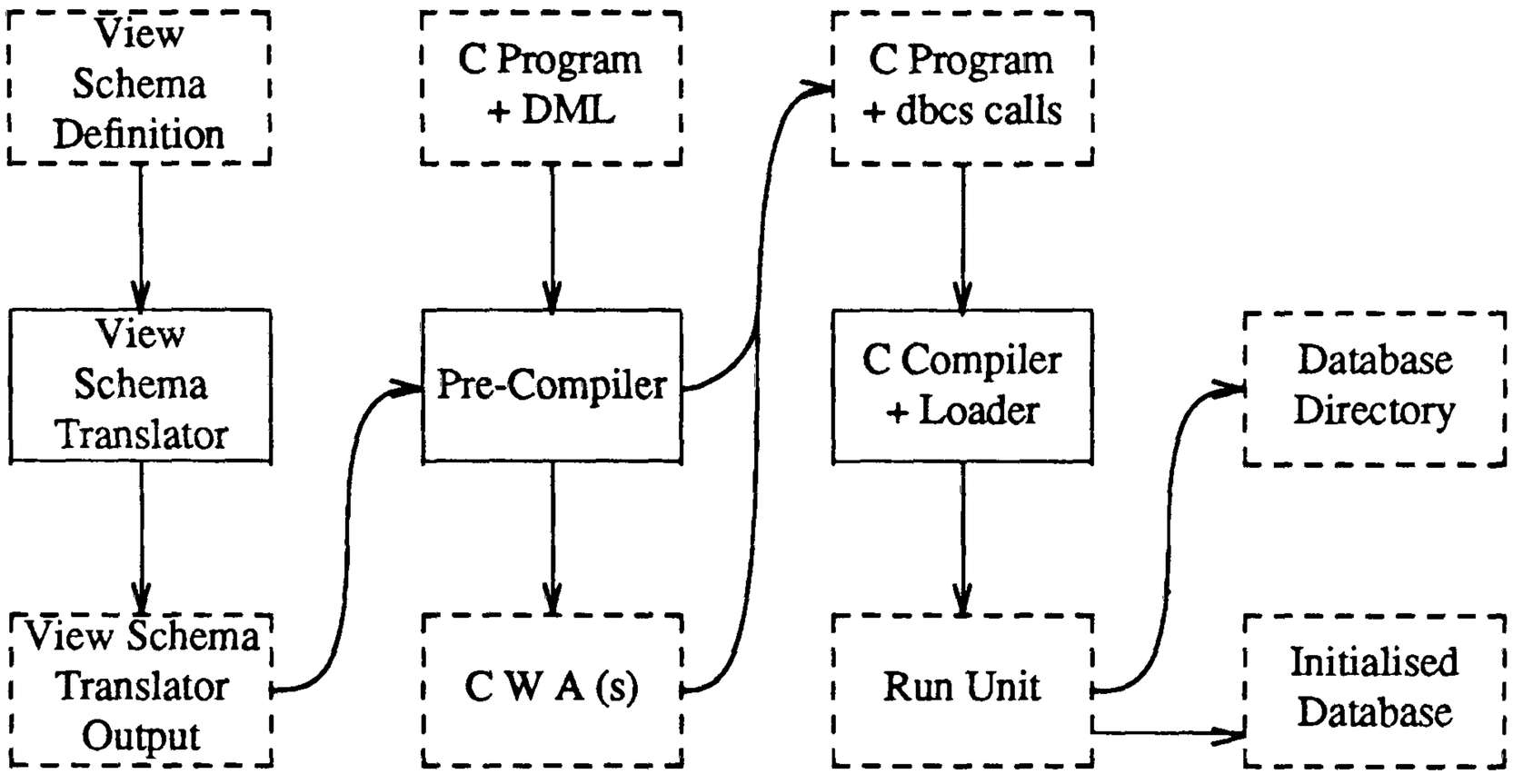


Figure 3.2: Transaction Creation

Views are defined using relational algebra. The CWA defines a C structure for each accessed view which acts as a tuple-buffer and provides the interface between the C environment and that of the database. Access using the CWA is thus by tuple. Relational access is provided via relational algebra commands.

Note that since TDBS/C is derived from the canonical model of PRECI, information is stored as relations. Access to the relations may be either by tuple or by relation. In this study, such access is called "tuple-access" and "relation-access" respectively.

The syntax for the schemas (conceptual, storage and view), the database definition and the relational expressions is given in Appendix 1, Figures A1.17 to A1.21 inclusive.

3.1.2. Data Manipulation Language (DML)

The DML is embedded in C programs and comprises:

1) general operations,

2) tuple operations

3) relational operations for relational algebra expressions [33]

The relational operations are selection, projection, division, join, union, intersection, difference and relational assignment. The general and tuple operations are summarised in Table 3.1.

DML Operation	Meaning	
GENERAL:		
OPEN	Open database	(Trans. begin)
CLOSE	Close database	(Trans. end)
PRINT	Print relation	(fixed format)
TUPLE:		
INSE	Insert tuple in relation	
REPL	Replace tuple in relation	
DELE	Delete tuple in relation	
GETFS	Get first tuple: storage order	
GETNS	Get next tuple: storage order	
GETPS	Get prior tuple: storage order	
GETFP	Get first tuple: indexed order	
GETNP	Get next tuple: indexed order	
GETPP	Get prior tuple: indexed order	
GETP	Get tuple by key (associative)	

Table 3.1: DML Operations

In TDBS/C the tuples are stored in a DATA AREA and the primary key index stored in a PINDEX AREA. Tuple access is supported to allow other data models to be interfaced to TDBS/C and may either be by stored order, primary key order or associative. The remaining tuple operations are insert, replace and delete (tuple is marked deleted). OPEN and CLOSE correspond to the transaction begin and end respectively and PRINT allows the relation to be displayed.

3.1.3. TDBS/C Internal Organisation

The internal organisation for TDBS/C is described in Appendix 1. A brief description is presented here in order to explain choices made in constructing test data for these benchmarks.

Relations in TDBS/C are stored one per file with the three main components illustrated in Figure 3.3.

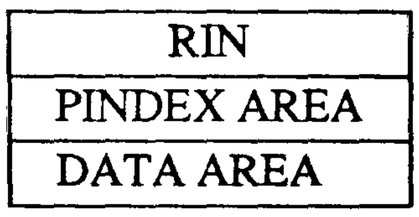


Figure 3.3: Relation File Layout

The RIN (Relation INformation) holds the domain and attribute information and layout information for the DATA and PINDEX areas. When the relation is accessed, this information is held in primary memory.

Both the DATA and PINDEX areas may be logically viewed as a set of hash slots (home + overflow). Each area has a hash function, Dhash, Phash respectively, which maps the primary key (pkey) value to the appropriate hash slot. In the DATA AREA, tuples are stored sequentially within the slot, with possible overflow, and the slot number and slot position are used to generate an effective key (ekey), essentially the tuple position within the DATA AREA. Each PINDEX AREA slot comprises a slot header giving the slot kind, current number of entries and pointers to next and prior slots and an entry for each pkey value, in ascending order, of the form {pkey, ekey}.

The tuple allocation procedure is summarised in Figure 3.4.

```
Dhash: pkey --> DATA AREA slot number (dslot)
Find free position in dslot (dpos)
Store tuple at (dslot, dpos)
Map (dslot, dpos) --> ekey

Phash: pkey --> PINDEX AREA slot number (pslot)
insert {pkey, ekey} in ascending order in (pslot, ppos)
```

Figure 3.4: Tuple Allocation Procedure

The DATA and PINDEX slots are stored on logical pages which in turn are mapped on to physical pages (in this study the mapping is one-to-one). Pages are called home or overflow in both areas, corresponding to the type of slot they contain. Further, the PINDEX AREA home pages have local overflow slots to minimise page faulting [36]. A bit map for PINDEX AREA overflow pages and a DATA AREA directory are held in primary memory (part of the RIN) to improve access and again minimise page faulting.

The "name space", the list of named objects in the database, is held in primary memory as a linked list called the IDLIST (Appendix 1, Figure A1.10). This corresponds to a data dictionary for the system [33].

Finally, a buffer and buffer manager at the relation file/"database primitives" interface (Appendix 1, Figure A1.11) provide local buffer space for the transaction. In the single-user benchmark tests presented in this chapter, the buffer size is varied (10, 100, 1000 1K pages) to test the effect on the system. In the following chapter, the buffer space is used to emulate shadow paging.

3.2. Experimental Test Database Design

In designing a test database for the benchmark tests, there are several considerations:

1) efficiency with regard to storage and organisation

2) extra page faulting should be minimised

3) scaling the database (up/down) should be easy

4) the effects of other factors should be minimised (or should be explainable)

Secondly, attention must be given to the design of the test data for the database. To allow precise selection of n tuples or k% of the database, the test data must be both flexible and controllable [17]. In addition, different distributions of test data values (on the attribute level) can be used to provide sequential (here taken as the best case access) and uniform (here taken as the worst case access) distributions. Preliminary experiments have shown that random access to the database tends to be similar to uniform access. Being able to generate particular distributions of data allows other distributions, for example, Zipf [123], to be tested.

These tests, as well as providing benchmark results for the system, are required to select a reasonably sized database and a small set of test transactions for the multi-user performance tests. Further, an idea of upper and lower bounds for access and an estimate of degree of experimental error are required. The benchmark design comprises three components; experiment design, test data design and test database design.

3.2.1. Experiment Design

The two main requirements are (a) to test the limits of the system and (b) to select suitable transactions and test data for the multi-user performance tests. Two experiments are proposed:

1) Transactions accessing a portion of the database

   (uniform access)

   (1%, 5%, 10%, 20%, 25%, 50% and 100%)

2) Transactions accessing a small number of tuples

   (sequential and uniform access)

   (1, 2, 5, 10 and 100 tuples)

Experiment 1 allows upper and lower limits to be determined for database access while

experiment 2 allows the costs of accessing a fixed number of tuples under different conditions and of sequential vs uniform access. It is highly probable in the multi-user tests that the transaction size (in terms of accesses) will have to be restricted. Experiment 2 allows lower limits to be determined for the significance of the results with regard to experimental error. Finally, we want to test the assumption that access costs are in fact linear.

The two main components of the system being tested are (a) the DBMS and (b) the Buffer Manager. For each transaction we measure the following:-

1) ET - elapsed time

2) CPU - cpu usage

3) IO - i/o usage

The result of any experiment may then be expressed as:-

DBMS(ET, CPU, IO) + Buffer(ET, CPU, IO),

i.e., a combination of overheads for the DBMS and the Buffer Manager. Hence by varying the size of the buffer we can test if the DBMS costs remain constant. A description of the experimental techniques, experimental environment and statistical techniques and assumptions are given in Appendix 5.

3.2.2. Test Data and Database Design

To satisfy the access requirements for both experiments, the tuples must provide both sequentially and uniformly distributed attribute values. The tuple layout is described in Table 3.2.

In contrast to [17] which used random number generation to provide uniformly distributed attribute values, the values here are simply repeating sequences. This allows precise selection of page sets (granule sets) and is used as described in the next chapter to control conflict between transactions.

Attribute	Type and Range	
tkey	integer (0..n-1)	n: no. of tuples
tp50	integer (0..1)	
tp25	integer (0..3)	
tp20	integer (0..4)	
tp10	integer (0..9)	
tp05	integer (0..19)	
tp01	integer (0..99)	
tfill	character	(used as filler)
tpad1	character	(used as filler)
tpad2	character	(used as filler)
tpad3	character	(used as filler)

Table 3.2: Tuple Attribute Values

The primary key (tkey) allows sequential access and the remaining integer attributes allow uniform access. The physical tuple size is 99 bytes (98 + active/deleted marker). Given a page size of 1024 bytes, 10 tuples can be stored per page (DATA AREA) and by suitable choice of slot sizes, 100 pkey entries per page (PINDEX AREA).

For example, to select 10 tuples sequentially distributed, the select predicate might be "(tkey < 10)" and the tuples would be on the first page of the DATA AREA. To select 10 tuples uniformly distributed (1 per page) from a database of 1000 tuples, the select predicate might be "(tp01 = 0)". Given that tuples are stored 10 per page, a second (non-conflicting) page set may be obtained using the select predicate "(tp01 = 10)".

The test databases have 100, 1000 and 10000 tuples respectively in one relation. The filler attributes are chosen to maximise the tuples/page storage efficiency. The page requirements and storage efficiency for the three test databases are given in Table 3.3 where

$$storage\ efficiency = \frac{(tuple\ size * \ number\ of\ tuples)}{(page\ size * \ number\ of\ pages)} * 100$$

No. Pages/No. Tuples	100	1000	10000
RIN	1	2	5
Data home	10	100	1000
Data overflow	1	1	1
Pindex home	1	10	100
Pindex overflow	1	1	1
TOTAL pages	14	114	1107
Storage Efficiency (%)	68	83	85

Table 3.3: Storage Efficiency

3.2.3. Test Transactions

There are two methods of accessing the information in the database:-

1) tuple access: each tuple is accessed separately

2) relation-access: all tuples are accessed as one unit

This reflects the methods of the three main models in common use; network and hierarchical (tuple access) and relational (relation access).

In addition, tuple access operations may be further subdivided:

1) access by stored order (get first/next/prior): 1 disk access/tuple

2) access by indexed order (get first/next/prior): 2 disk accesses/tuple

3) access by primary key (associative access): 2 disk accesses/tuple

Using these distinctions, the transaction types are presented in Table 3.4.

Type	Operation
overhead	open/close database (no access)
relation-access	assign, selection, projection, select-replace (all RW)
tuple access stored order	saccess (R)
tuple access index order	paccess (R), insert (W), replace (W), delete (W), get-replace (RW), get-delete (RW)
tuple access associative	get (R), get-relpace (RW), get-delete (RW)

Table 3.4: Transaction Types

In terms of the transaction model of section 2.1, the transactions are also classified as read only (R), write only (W) and read-write (RW). Certain transactions (relation-access, insert, delete, get-delete) also update the PINDEX AREA.

An idea of the relative cost of particular mechanisms within the DBMS can be obtained by comparing sets of results from the tests. The primary key indexing overhead can be judged by comparing "saccess" with "paccess" (read only case) and "get-replace" with "get_delete" (read/write), the latter also updating the PINDEX AREA. Associative access may be compared with index order access by comparing "get-replace" and "get_delete" transactions in each category.

The transactions are sketched in terms of the DML commands, in Table 3.5.

Transaction	DML Commands		
overhead	OPEN		CLOSE
saccess	OPEN	GETFS {GETNS}*	CLOSE
paccess	OPEN	GETFP {GETNP}*	CLOSE
insert	OPEN	{CWA = tuple; INSE R}*	CLOSE
replace	OPEN	{CWA = tuple; REPL R}*	CLOSE
delete	OPEN	{CWA = tuple; DELE R}*	CLOSE
get-replace	OPEN	GETFP {REPL; GETNP}*	CLOSE
get-delete (index order)	OPEN	GETFP {DELE; GETNP}*	CLOSE
getp	OPEN	{CWA = pkey; GETP}*	CLOSE
get-replace	OPEN	{CWA = pkey; GETP; REPL}*	CLOSE
get-delete (associative)	OPEN	{CWA = pkey; GETP; DELE}*	CLOSE

Table 3.5: Transaction Descriptions

Where "{ --- }*" means loop while there are still tuples or over a given tuple/pkey set.

The relation-access transactions are:-

assign: R1 := R2
selection: R1 := R2 select (attribute = value)
projection: R1 := R2 project (all attributes)

where R1 and R2 are relations. These follow the standard relational algebra. In all cases, a second relation (R1) is created as a result of the relational expression. In terms of reads and writes, in the projection for example, if R2 has k pages then k reads (from R2) and k writes (to R1) will be performed. In general, access by relational algebra has a higher overhead in terms of page manipulations than tuple access. Relational algebra is more suited to read only queries.

Consider the transaction "increase all programmer salaries by 15%". Using tuple access and assuming the information is held in relation EMP, this might be written

```
still_tuples = GETFP EMP;
while (still_tuples) {

   if (EMP.job == "programmer")
      EMP.salary = EMP.salary * 1.15;

      REPL EMP;

   still_tuples = GETNP EMP;
   }
```

Similar examples [33] can be found in QBE [124] and SQL [95] for example.

In relation access, this is more difficult to achieve [33]. An extension to selection is proposed here to cover this case. The modified selection is

R1 := R1 select (attr1 = value1) update (attr2 = value2)

If a tuple satisfies the selection predicate it is updated according to the update clause. This is similar to a table variable with a simple predicate used in an assignment, as in Astral [5], with the addition that the relation name on both sides of the assignment is the same and an update clause introduced.

In the case where k tuples are to be updated, if the primary keys are known, the process can be efficiently achieved via the PINDEX otherwise the entire relation must be searched. In tuple access systems this implies that for a relation of cardinality n, n calls are made to the DBMS whereas in a relational system only one call is made. The disadvantage of relational systems is the creation of a temporary relation for the result of the relational expression if the left hand side of the assignment does not exist. In the "select-replace", given that the assignment is "reflexive", the implementation can be optimised and the update done "in-place" thus avoiding the generation of extra pages for the result of the relational expression.

The associative "get-replace" and the "select-replace" allow comparison of the two approaches. Here, the best-case tuple-access (pkeys are known) is compared with the worst-case relation-access (pkeys are not known). The corresponding worst-case tuple-access is given by the index order "get-replace".

The overhead costs include the O/S costs of scheduling the transaction. The costs of actual work done by the transaction may be expressed as

$$(T(ET) - O(ET), T(CPU) - O(CPU), T(IO) - O(IO))$$

given the measured costs of the overhead transaction O(ET, CPU, IO), that is, initialising the buffer and the O/S overheads, and the measured costs of a given transaction T(ET, CPU, IO).

The experiments using the transactions listed in Tables 3.4 and 3.5 also allow tuple and relational access to be compared.

3.3. Benchmark Tests

The previous sections have described the testbed system, test data, test database organisation and transactions used in the benchmark tests. We are now in a position to state the factors and factor levels [98] used for these experiments, in Table 3.5.

FACTOR	FACTOR LEVELS
DB_SZ	100, 1000, 10000 tuples
BF_SZ	10, 100, 1000 x 1K pages
T_TYP	See Table 3.4
T_ACC	sequential, uniform
DB_ACC:	
Exp 1	1%, 5%, 10%, 20%, 25%, 50%, 100%
Exp 2	1, 2, 5, 10, 100 tuples

Table 3.6: Factors and Factor Levels

DB_SZ is the database size, BF_SZ the buffer size, T_TYP the transaction type (which determines the read/write requests), T_ACC the access method used by the transaction and DB_ACC the number of tuples accessed.

The experiments were run as single-factor experiments. Each test was run 110 times on a VAX/750 in single-user mode, with one user disk and one system disk, and the first 10 results discarded to eliminate startup transient effects. The remainder were used to calculate the average values for ET, CPU and IO. Further details are given in Appendix 5.

3.4. Benchmark Results

The results for the single-user benchmark tests are presented in Appendix 2. The performance measures are ET (elapsed time), CPU (CPU time used) and IO (IO time used). In addition the transaction time - overhead transaction time are presented as a measure of the "work" involved in database access.

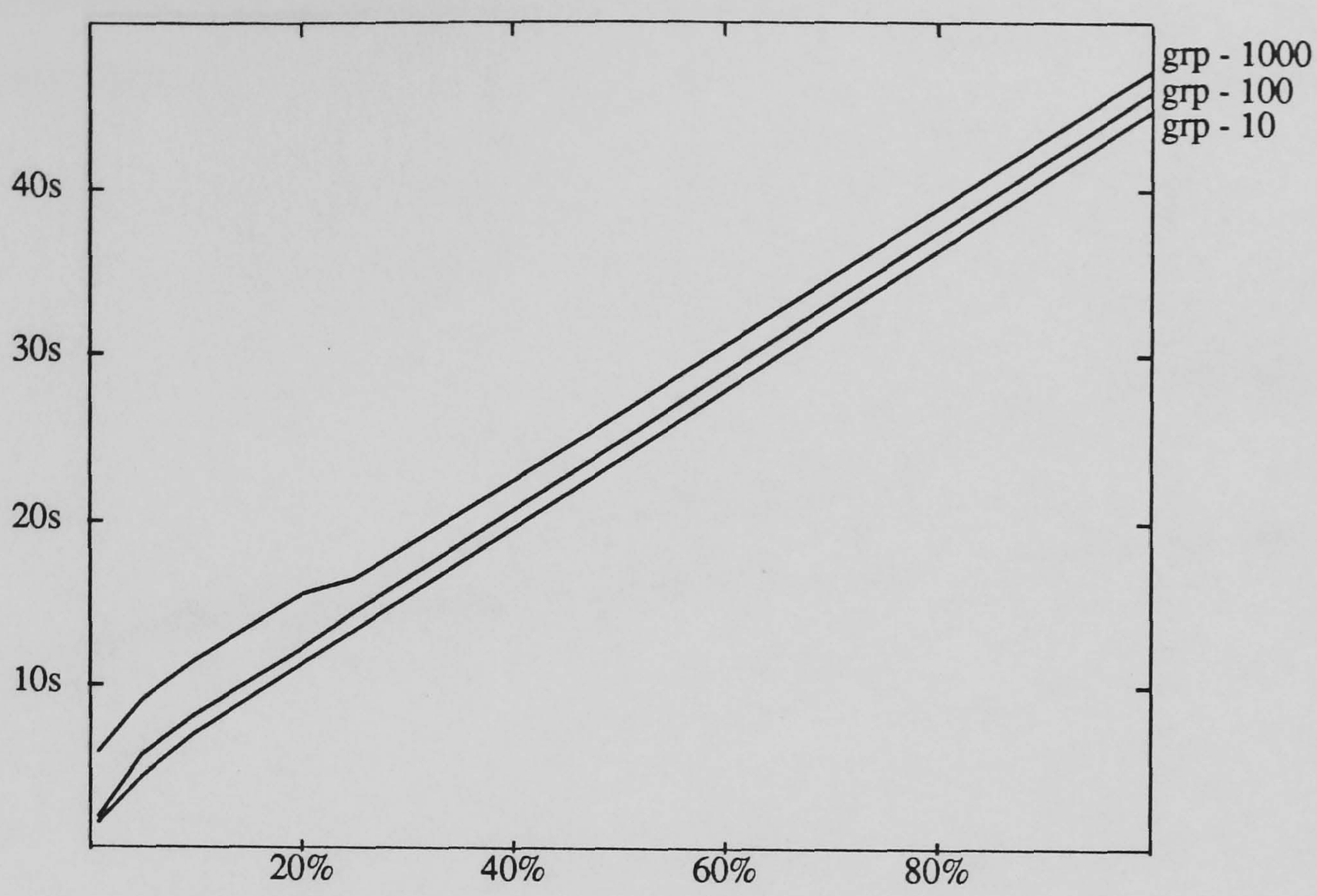
3.4.1. Results of Experiment 1

These results are presented in Tables A2.2 to A2.11 inclusive in Appendix 2. Here we give a graphical presentation of some of these results for the case DB_SZ = 1000 tuples. The graphs plot elapsed time in seconds (ET) against DB_ACC.

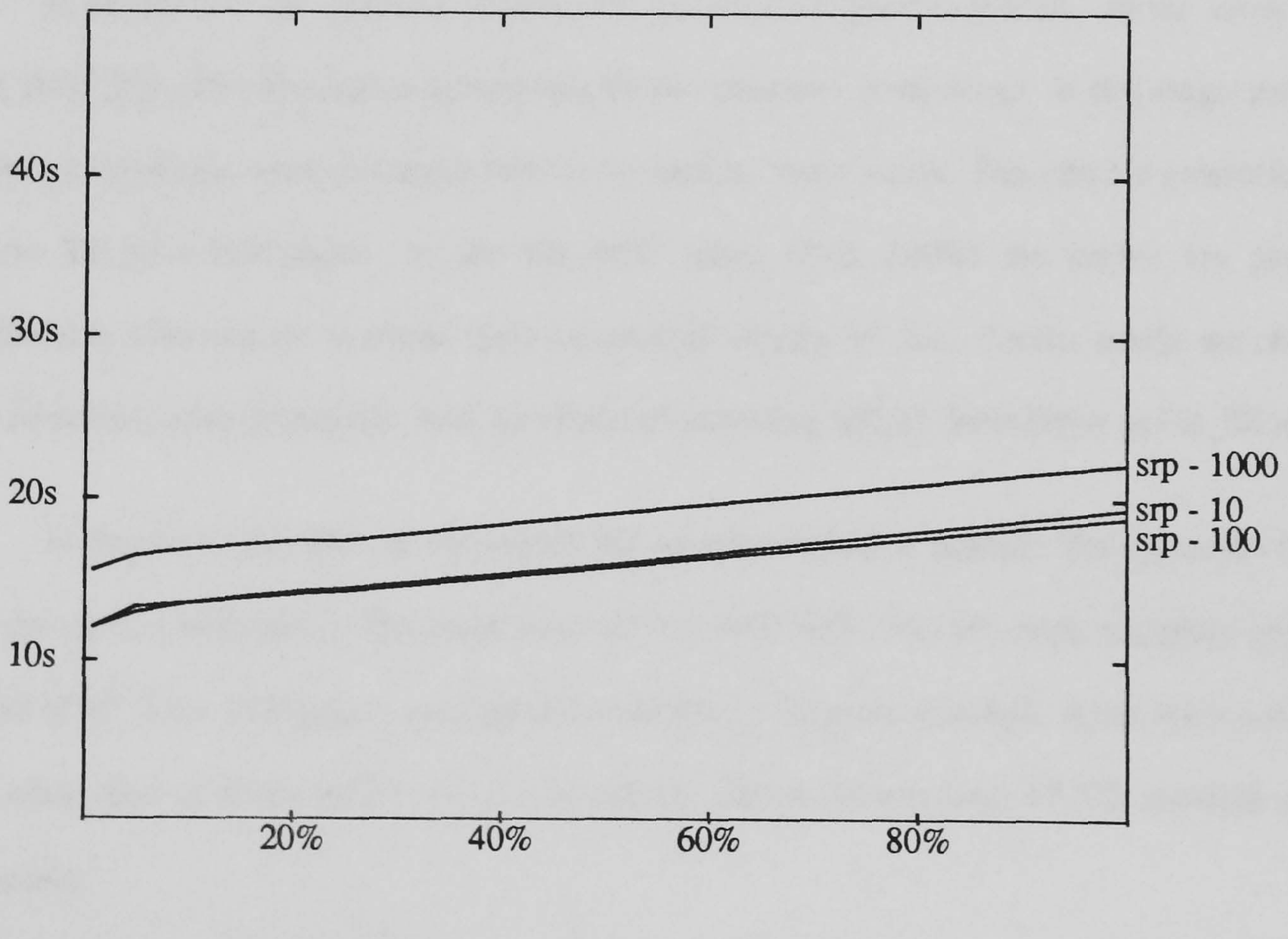
GRAPH 3.1: get-replace (grp), BF_SZ = 10, 100, 1000 pages

GRAPH 3.2: select-replace (srp), BF_SZ = 10, 100, 1000 pages

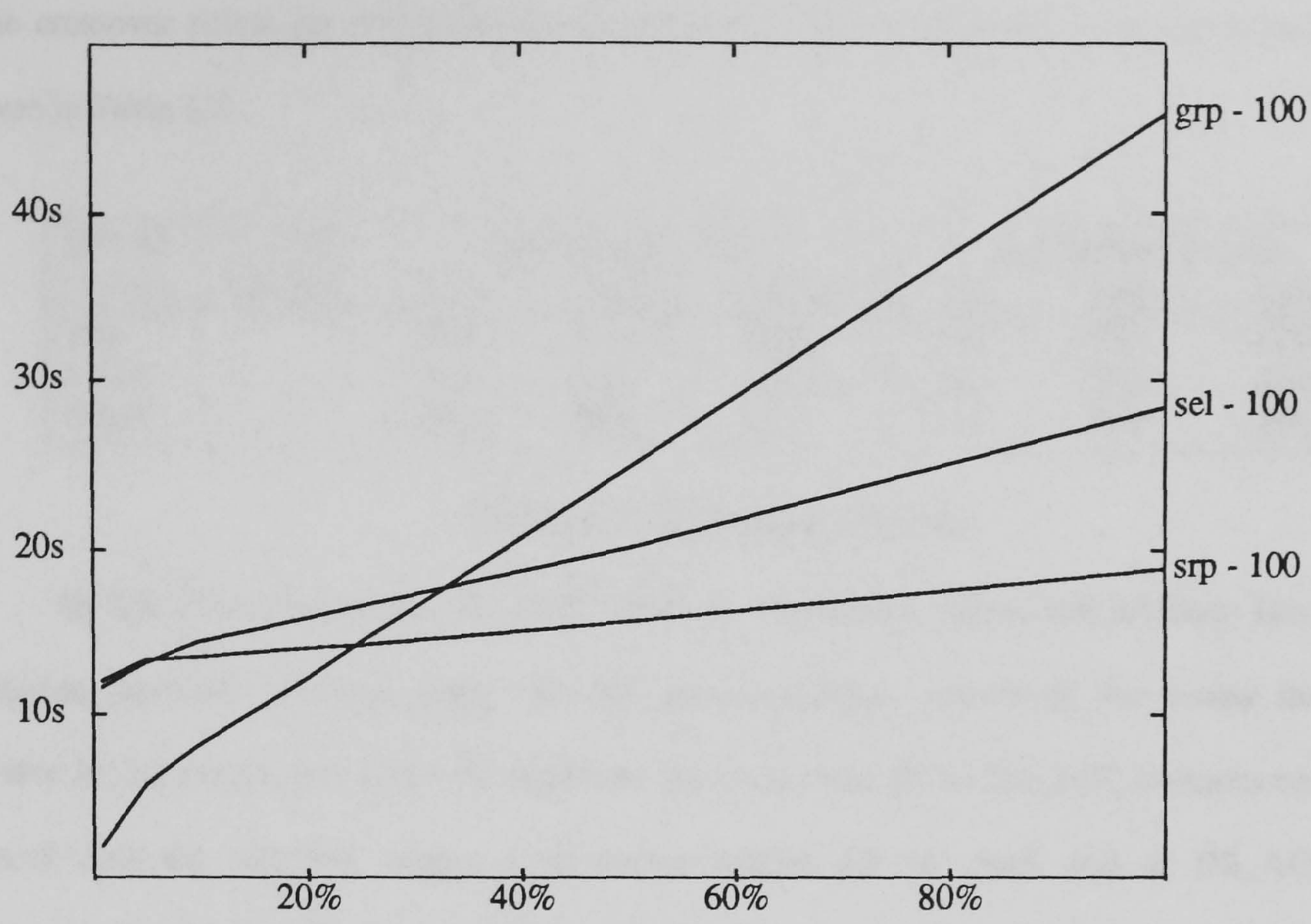
GRAPH 3.3: BF_SZ = 100 pages, get-replace (grp), select-replace (srp), selection (sel)



Graph 3.1: get-replace (grp)/BF_SZ: ET vs DB_ACC



Graph 3.2: select-replace (srp)/BF_SZ: ET vs DB_ACC



Graph 3.3: BF_SZ 100 pages: ET vs T_TYP

In Graph 3.1, the effect of varying BF_SZ on get-replace is plotted. In the lower range of DB_ACC (0%...5%) the slope is greater than for the remainder of the range. In this range, the overhead costs are significant when compared with the transaction "work" costs. This effect is greater for the case where BF_SZ = 1000 pages. In the DB_ACC range (25%...100%) the curves are parallel, the differences reflecting the overhead costs incurred by varying BF_SZ. Similar results are obtained for the other runs using get-replace, with the effects of increasing BF_SZ diminishing as DB_SZ increases.

In Graph 3.2, the effect of varying BF_SZ on select-replace is plotted. The curves are flatter than for get-replace and again in the lower range of DB_ACC (0%...5%) the slope is slightly greater. The effect of BF_SZ = 1000 pages is also similar to Graph 3.1 but more flattened. Again the results obtained for other runs of select-replace are similar and the effects of increasing BF_SZ diminish as DB_SZ increases.

Finally in Graph 3.3 the results for get-replace, select-replace and selection with BF_SZ = 100 pages are plotted. Recall that get-replace uses a best case assumption (pkeys are known) and the select-replace and selection use a worst case assumption (pkeys are not known). The results show that as DB_ACC increases, select-replace and selection become more efficient than get-replace.

The crossover points for get-replace/select-replace and get-replace/selection, for all experiments, are given in Table 3.7.

DB_SZ	T_TYP:	get-/select-replace			get-replace/selection		
(tuples)	BF_SZ:	10K	100K	1000K	10K	100K	1000K
100		80%	76%	75%	50%	45%	47%
1000		27%	25%	27%	40%	33%	33%
10000		20%	20%	20%	85%	37%	32%

Table 3.7: Crossover Points

As DB_SZ increases, the DB_ACC value at which select-replace and selection becomes more efficient, decreases in most cases. For the two exceptions, examining the results for selection (Table A2.10) reveals that as BF_SZ decreases, the slope of the ET vs DB_ACC increases considerably. Recall that the selection creates a temporary relation for its result and at DB_ACC = 100%, DB_SZ = 10000 tuples, is reading 1100 pages and writing 1100 pages (see Table 3.3) using a buffer size of 10/100 pages. The result is probably due to thrashing in the database buffer. Remarkably, get-replace and select-replace are less affected by thrashing at these factor levels.

One assumption being tested is that costs in the system are linear, i.e., that the system can be (ideally) modelled as

$$ET = a * (CPU) + b * (IO)$$

where a and b are constants. This is supported from the results as plotted in Graphs 3.1, 3.2 and 3.3.

We now examine the results for DB_SZ = 100, 1000, 10000, BF_SZ = 10, 100, 1000, T_TYP = get-replace, select-replace, selection, T_ACC = uniform and DB_ACC = 100%, which are given in Table 3.8.

Program	Experimental Times Elapsed	CPU	IO	Thru'put	Time - Overhead Elapsed	CPU	IO
BF_SZ = 10 pages; DB_SZ = 100, 1000, 10000 tuples							
get-repl	4.85	4.08	0.58	0.206	4.20	3.78	0.38
get-repl	40.74	37.63	2.45	0.025	40.11	37.33	2.25
get-repl	392.05	368.15	20.70	0.003	391.45	367.85	20.50
select	3.98	2.48	1.13	0.251	3.33	2.18	0.93
select	31.39	22.15	8.06	0.032	30.76	21.85	7.86
select	417.92	259.61	155.08	0.002	417.32	259.31	154.88
sel-repl	4.56	1.84	1.95	0.219	3.91	1.54	1.75
sel-repl	19.45	12.86	5.27	0.051	18.82	12.56	5.07
sel-repl	163.20	122.90	36.08	0.006	162.60	122.60	35.88
BF_SZ = 100 pages; DB_SZ = 100, 1000, 10000 tuples							
get-repl	5.24	4.14	0.82	0.191	4.19	3.73	0.39
get-repl	41.46	37.42	2.74	0.024	40.44	37.02	2.29
get-repl	401.49	374.05	22.84	0.003	400.49	373.64	22.41
select	4.45	2.57	1.39	0.225	3.40	2.16	0.96
select	28.45	20.79	6.07	0.035	27.43	20.39	5.62
select	299.26	222.06	72.83	0.003	298.26	221.65	72.40
sel-repl	4.91	1.96	2.19	0.204	3.86	1.55	1.76
sel-repl	18.87	11.70	5.47	0.053	17.85	11.30	5.02
sel-repl	161.80	118.24	36.76	0.006	160.80	117.83	36.33
BF_SZ = 1000 pages; DB_SZ = 100, 1000, 10000 tuples							
get-repl	8.86	5.37	3.17	0.113	4.31	3.77	0.44
get-repl	44.66	38.41	5.05	0.022	40.09	36.79	2.28
get-repl	401.30	369.78	23.71	0.003	396.78	368.19	20.94
select	7.87	3.81	3.67	0.127	3.32	2.21	0.94
select	29.97	20.92	7.80	0.033	25.40	19.30	5.03
select	265.26	206.14	54.07	0.004	261.74	204.55	51.30
sel-repl	8.41	3.18	4.46	0.119	3.86	1.58	1.73
sel-repl	22.17	12.94	7.70	0.045	17.60	11.32	4.93
sel-repl	155.52	110.18	37.36	0.006	151.00	108.59	34.59

Table 3.8: Experiment 1, Result Summary

Since the linear model $ET = a^*(CPU) + b^*(IO)$ is a simplification of the database system, the results are not expected to correspond exactly to this model. Examining the results, especially the (time - overhead), seems however to support this assumption.

Several points are noted:

- The tuple-access operations in general show values of $a = 10$ and $b = 10$ when the DB_SZ is increased by 10 (the IO values at DB_SZ = 100 are probably too low to be significant).

- The selection transaction is more sensitive than get-replace or select-replace to the buffer thrashing effect which becomes more pronounced at DB_SZ = 10000 as BF_SZ decreases. The CPU costs increase by a factor 10 as the DB_SZ is increased by a factor 10.

- The select-replace transaction shows values of $a = 10$ and $b = 7$. Determining exact values for a and b is not important here, rather confirmation that the system costs behave in a linear fashion.

3.4.2. Results of Experiment 2

These results are presented in Appendix 2, Tables A2.12 to A2.14 inclusive. The factor levels tested are given in Table 3.9.

Factor	Factor Level
DB_SZ:	100, 1000, 1000 tuples
BF_SZ:	10, 100 1000 pages
T_TYP:	get-replace
T_ACC:	sequential/uniform
DB_ACC:	1, 2, 5, 10, 100 tuples

Table 3.9: Experiment 2 Factor Levels

In this experiment we want to compare sequential vs uniform acess and determine the smallest significant values for DB_ACC, taking experimental error into account.

Up to and including DB_ACC = 10, the effects of T_ACC are not significant but become so at DB_ACC = 100. Results for DB_ACC = 1 give a first estimation of experimental error between batches.

Similar conclusions were reached from the results of other experiments with different tuple-access transactions.

3.5. Conclusions

The following conclusions are made from the results of the single-user benchmarks:

- the assumption that costs are linear is reasonable

- the overhead for indexed access is low, indicating that the pindex implementation is efficient

- beyond a certain DB_ACC value, the relation-access transactions become more efficient than the tuple-access transactions

  (this is probably due to the call overhead on the DBMS incurred by the tuple-access technique since both techniques use the same primitive functions and the get-replace and select-replace are functionally identical)

- the measurement error was the 90% confidence interval, expressed as a percentage of the point estimator, and was below 2%;

  similar measures for the standard deviation and variance were 10% and <1% respectively;

  a more detailed example is given in Appendix 5

As a result of these tests, the choice of test database size is 1000 tuples with buffer size 100 pages for the multi-user CC performance tests. The test transactions chosen are get-replace and select-replace. The timing mechanism was refined to improve accuracy (see Appendix 5).

3.6. Chapter Summary

A brief introduction to TDBS/C, the testbed system, was presented as well as requirements for synthetic test data. The DML was introduced and the transactions used in the single-user benchmarks described in terms of the DML. A modified form of selection was proposed, and the two experiments, together with the factor levels were described. Results for the transactions used in the following chapter were presented and other results discussed more generally. As well as describing the database system behaviour, the single-user results are important in providing a baseline for the multi-user results [17].

The method presented here, i.e., use of synthetic data, best and worst case access patterns for different database sizes and transactions may also be extended to the multi-user experiments. Finally, from the tests, two transaction types (one tuple-access, one relation-access) have been chosen for the multi-user tests.

Chapter 4

PROTOTYPE EXPERIMENTS

The first section presents the experimental method and the background to the experimental results presented in this chapter. The test data is described and the transactions are reviewed. Important points arising from preliminary tests are presented. Finally the experiments are summarised and the conclusions presented. The statistical techniques are summarised in Appendix 5.

4.1. Experimental Method

The experimental method for this study was developed from ideas presented by Bitton [17, 18] who makes the following points:

1) synthetic data allows a wide range of controlled queries to be specified

2) single-user (SU) benchmarks constitute a necessary baseline measure for the interpretation of multi-user (MU) benchmarks

3) database evaluation should be compared using "quantifiable performance indices" from both the DBMS and the O/S

4) a representative set of queries should be specified

In [17], commercial INGRES was compared with the Bitton-Lee IDM 500 database machine, using SU benchmarks. These were further developed in [18], to MU benchmarks which have become known as the "Wisconsin Benchmark". These MU benchmarks tested

1) the effects of multiprogramming level using simple selection queries

2) mixes of DBMS and non-DBMS programs at the same multiprogramming level to obtain measurements of throughput and resource utilisation.

No update queries were used in these studies however and concurrency control performance was not considered.

In Chapter 3, the importance of synthetic data was discussed and SU benchmarks analysed. Performance metrics for the MU experiments are presented in Table 4.1.

metric	meaning
T	throughput (trans/second)
ET	average elapsed time
CPU	transaction CPU usage
IO	transaction I/O usage
iET	idealised ET
RC	restart count

Table 4.1: Performance Metrics

These are similar to Table 2.4 except for CPU and IO, which are measures of (T_CPU+CC_CPU) and (T_IO+CC_IO) respectively. One idealised measure, iET, is calculated as (CPU+IO). The O/S overhead, O_OS, may be inferred from (ET-TMPL*iET).

For each CCA, (NO CC, PRE, 2PLE, 2PLU, BTO, SV), both SU and MU tests were run and the above performance metric values were measured. These provide:

1) SU-NO CC vs SU-CCA: a measure of the cost of the CCA compared with that of the transaction

2) SU vs MU: a measure of the O/S overheads for each CCA

3) MU-NO CC vs MU-CCA: a measure of the cost of the CCA in a multi-user environment

In CCA performance evaluation, one important factor is probability of conflict (PC). This has been controlled in other studies by using a probabilistic approach to define access patterns and by increasing/decreasing MPL and/or granularity to increase/decrease PC. A second technique is proposed, where transactions access controlled portions of the database and PC is pre-determined. This case will be called level of conflict (LC) to distinguish it from the case where conflict is probablistic.

To submit transactions and gather values for the performance metrics, a scheduler was written. This reads a script containing a list of transactions to be run concurrently, which defines the TMPL. Up to a given MMPL, each transaction is submitted and on completion, the performance measures are noted and the transaction is resubmitted. The scheduler thus maintains a "steady-state" of TMPL transactions in the system. The scheduler also delays restarted transactions. At the end of a batch of a fixed number of transactions, the results for the batch are calculated. Further details of the scheduler and measurement techniques are given in Appendix 5.

In summary, the experiments presented here are run for all CCAs, in SU and MU tests, using both a probabilistic approach and pre-determined access to control conflict. The performance measures obtained are listed in Table 4.1 and full results presented in Appendix 3. These include T, ET and RC for the transactions. The average ET, CPU and IO measures for successful and restarted transactions are presented as Useful and Wasted Work Costs respectively. The Wasted Work Costs give a quantative indication of work lost by restarting and at which point during the transaction restarts occur. The effects of blocking are indicated by comparing the NO CC and CCA ET values in the tables of Useful and Wasted Work Costs. The cost of restarts is given by the difference between transaction ET and the Useful Work ET values.

A similar method was also proposed in [20] but the method of pre-determined access to control LC is different. This is discussed in Section 4.6.

4.2. Test Data and Transactions

The test data is synthetically generated as in Chapter 3. The attributes and attribute values are presented in Table 4.2.

Attribute	Type and Range	
tkey	integer (0..n-1)	n: no. of tuples
tsq1	integer (0..99)	sequentially distributed
tsq2	integer (0..9)	sequentially distributed
tsq3	integer (0..1)	sequentially distributed
tun1	integer (0..99)	uniformly distributed
tun2	integer (0..9)	uniformly distributed
tun3	integer (0..1)	uniformly distributed
tfill	character	(used as filler)
tpad1	character	(used as filler)
tpad2	character	(used as filler)
tpad3	character	(used as filler)

Table 4.2: Tuple Attribute Values

Again the attribute values are chosen to provide controlled sequential and uniform access to the database.

The test transactions are the two selected from the benchmark tests of Chapter 3. The "get-replace" provides tuple access to the database and the "select-replace" provides relation access. This implies that for a database of 100 data pages and a transaction which accesses the database uniformly and performs 5 updates that the transaction will read and write 5 pages. As the DB_ACC increases however, this increased I/O is offset by the increasing CPU overhead of the get-replace as reported in Section 3.5.

Note that since all transactions are update transactions, the results for 2PLU under the assumption that the transaction sets an exclusive lock immediately on any data object to be updated (see Section 2.4), would be the same as those for 2PLE. Comparing the results for 2PLE and 2PLU measures the effect of this write lock acquisition assumption at all factor levels.

Each transaction has its own copy of the DBCS and its own buffer space (100 1K pages), which is used here to emulate shadow paging by providing a local copy of data objects.

4.3. Preliminary Experiments

A number of preliminary experiments were run to select parameter values and techniques for the CCA performance experiments. The results of these experiments are summarised here.

4.3.1. Critical Section Locking Technique

Since the UNIX O/S used here provided no semaphores, two techniques to emulate semaphores were investigated here.

1) The transaction attempts to create a lock file in UNIX using "link"/"unlink" [117] If the creation fails, the transaction must wait and try again. This is the technique is used in [63]. It was found to have an unacceptably high overhead in terms of file system activity.

2) The transaction attempts to place an exclusive advisory lock using UNIX "flock" [117]. This was the technique adopted since the overheads were considerably lower than the "link"/"unlink" method.

4.3.2. Blocking and Restarting Delay (B_DLY/R_DLY)

Each time a transaction blocks it must wait for a given delay, B_DLY, before re-requesting the resource. With B_DLY = 0, severe thrashing took place due to re-request contention. The corresponding condition under restart based CCAs is cyclic restarts as noted in section 1.3.2. The value used in these experiments is 2 seconds and was selected by experimentation. R_DLY is also set to this value in order to compare the CCAs within these experiments. A previous simulation study [2] suggests that R_DLY should be in the order of the ET of the transaction. In this study, an adaptive restart technique is introduced where the R_DLY is calculated from a running average of the iET times MMPL. This is used as an approximation of ET and avoids including the B_DLY in the adaptive R_DLY calculation for the 2PLU case. The adaptive restart results for 2PLU and BTO are labelled A2PLU and ABTO respectively.

4.3.3. Fixed Length vs Dynamic CCA Information Tables

Initially, dynamic length tables were implemented in order to gain advantage for those CCAs which require smaller storage overhead for the information tables. See the discussion of Carey's model in Section 2.2. While decreasing the I/O overhead, dynamic tables increased the CPU overhead compared with fixed length tables and gave no overall improvement. Recall that the transactions in this study tend to become CPU bound. Fixed length tables were therefore chosen for these experiments. The approximate table sizes for a maximum of 10 transactions and 120 database objects accessed, are given in Table 4.3.

CCA	Table Size
PRE, 2PLE, 2PLU	4K
BTO	3K
SV	2.5K

Table 4.3: CCA Table Sizes

In practice, the main factor here is probably the size of the tables with respect to the O/S blocking size for I/O. Further, these tables are referred to frequently and hence are more likely to remain in the O/S file buffer.

4.4. Experiments and Statistics.

Two sets of experiments are proposed:

1) using random access to the database

2) using pre-determined access to the database

These experiments and the results are described in the following sections. The statistical techniques used are standard. Brief descriptions are given with the experiments and further detail in Appendix 5.

4.5. Experiment 1

4.5.1. Experiment Description

This experiment uses associative random access to a database of 1000 tuples in 1 relation. A random set of n key values is generated and the select-replace transaction in this experiment makes use of primary key access. PC is increased by two methods; (a) increasing the number of objects accessed by the transaction and (b) changing the granularity. The main aim of the experiment is to measure the performance of each CCA as PC increases. One other aim is to check the behaviour of the CCAs under different granularities against that predicted by earlier studies. The factors and factor levels are summarised in Table 4.4.

FACTORS	FACTOR LEVELS
DATABASE:	
DB_SZ	1000 tuples
DB_PG	100 pages
GRAN	1, 10 pages/granule
NR	1
TRANS:	
NT	10
T_REQ	1, 2, 5, 10
T_ACC	random
T_TYP	get-replace, select-replace
T_UPD	1.0
CC:	
CCA	NO CC, PRE, 2PLE, 2PLU, BTO, SV
MMPL	10
B_DLY	2 seconds
R_DLY	2 seconds/adaptive

Table 4.4: Experiment 1: Factor Levels

The factor levels were chosen from consideration of the results of other studies and preliminary experiments within this study. The experiments were run as single factor experiments. Each batch consisted of 500 transaction runs and 10 batches were run for each factor level. The batches are assumed to be independent. Statistics are derived using 400 transaction runs/batch to eliminate startup and end of batch anomaly effects. Average values for each batch were calculated assuming a normal distribution within the batch. The batch averages were then used to derive a point estimator for the experiment using a Student-T distribution [122] with 9 degrees of freedom. The full results, with 90%

confidence intervals expressed as a percentage of the point estimator are presented in Appendix 3, Tables A3.1 to A3.28 inclusive. The throughput values (T) and restart counts (RC) are presented below.

4.5.2. Single User Overhead Costs

To determine the basic overheads for each CCA, single user tests were run as one batch of 50 runs for each level of T_REQ and T_TYP. The average values for ET, CPU and IO for T_TYP = get-replace are calculated assuming a normal distribution and compared against the NO CC case and the results presented in Table 4.5 as percentage overheads.

Cost Overheads: (CCAs/NO CC) * 100					
ET:T_REQ.	PRE	2PLE	2PLU	BTO	SV
1	108	112	112	107	107
2	109	114	117	114	109
5	115	120	127	121	106
10	126	129	146	141	110
CPU:T_REQ.	PRE	2PLE	2PLU	BTO	SV
1	103	106	106	103	101
2	103	101	102	103	103
5	106	104	110	108	101
10	107	110	110	114	107
IO:T_REQ.	PRE	2PLE	2PLU	BTO	SV
1	109	114	114	114	112
2	110	120	124	119	107
5	120	130	141	133	110
10	144	150	187	165	110

Table 4.5: SU Overhead Costs (%)

From these results, the CPU overheads for CC are low with the IO overheads emerging as the more significant. At the lowest PC (T_REQ = 1), there is no real difference between CCAs. As PC increases, differences emerge. In order of increasing overhead, the CCAs are SV, PRE, 2PLE, BTO and 2PLU. The differences are explained by the techniques used. With SV only one call is made on the CCA to validate the transaction. Under both PRE and 2PLE, T_REQ calls are made to the CCA. PRE sets exclusive locks at the beginning of the transaction. 2PLE makes the calls dynamically and therefore risks greater overhead from I/O swapping than PRE. Under BTO and 2PLU, 2 * T_REQ calls are made to the CCA, one when the data object is read and a second when it is written. The CCA tables for BTO are smaller than for 2PLU and the algorithm slightly simpler.

The results for the select-replace were similar.

4.5.3. Experimental Results

In this section, the results for the five basic techniques, PRE, 2PLE, 2PLU, BTO and SV, are discussed. A discussion on A2PLU and ABTO is presented in the following section. The results for get-replace are discussed first. Tables 4.6 and 4.7 give the T results and RC respectively for GRAN = 1 (100 granules). Tables 4.8 and 4.9 give results for GRAN = 10 (10 granules). Since MMPL and TMPL are fixed, PC is controlled by the granularity and the number of access requests. Each transaction randomly accesses T_REQ tuples, selected from a uniform distribution (0..DB_SZ-1) and then mapped on to the corresponding granule. The lowest PC occurs at (GRAN = 1, T_REQ = 1), and the highest at (GRAN = 10, T_REQ = 10). The results of SU-NO CC are also presented which when compared with MU-NO CC give an indication of the throughput gain inherent in the O/S at PC = 0. Note that a "---" value indicates that the result was not considered statistically significant (i.e., fewer than 50 commits).

At the lowest PC, SV performs better than the other CCAs. As PC rises, the performance of SV drops below that of PRE and 2PLE but generally remains above that of 2PLU and BTO. Finally at the highest PC (GRAN = 10, T_REQ = 10), SV outperforms 2PLE. Recall that in any conflict situation between k transactions, SV guarantees that at least one conflicting transaction will commit. This is not the case for either 2PLU, which uses blocking to resolve conflicts and restarts to resolve deadlocks, or BTO which uses restarts to resolve conflicts. At this highest PC, 2PLU and BTO have virtually no commits. The better performance of SV over 2PLE, at the highest PC, is reflected in the restart counts in Table 4.9.

BTO, the other CCA using restarts for conflict resolution, generally has a lower T than SV despite a lower RC on two occasions. As PC increases, the overall performance of BTO degrades much faster than SV. This is expressed in Tables 4.14 and 4.15 below.

T_REQ	NOCC SU	NO CC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	0.789	0.824	0.752	0.758	0.708	0.700	0.757	0.753	0.781
2	0.745	0.787	0.685	0.689	0.591	0.600	0.670	0.678	0.684
5	0.625	0.695	0.493	0.517	0.254	0.399	0.291	0.389	0.398
10	0.556	0.577	0.277	0.235	0.048	0.218	---	0.301	0.166

Table 4.6: GRAN = 1, T_TYP = get-replace: Throughput

T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	0	0	0	0	0	0	0
2	0	0	23	21	22	21	31
5	0	11	182	125	197	137	163
10	0	164	345	195	380	180	284

Table 4.7: GRAN = 1, T_TYP = get-replace: Restart Counts

T_REQ	NOCC SU	NO CC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	0.789	0.824	0.727	0.734	0.529	0.640	0.583	0.711	0.513
2	0.745	0.787	0.604	0.594	0.227	0.416	0.267	0.529	0.339
5	0.652	0.695	0.419	0.181	---	0.247	---	0.326	0.159
10	0.556	0.577	0.341	---	---	---	---	0.309	0.082

Table 4.8: GRAN = 10, T_TYP = get-replace: Throughput

T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	0	0	99	59	97	54	137
2	0	17	247	150	251	143	222
5	0	256	383	214	388	193	308
10	0	370	399	384	399	181	344

Table 4.9: GRAN = 10, T_TYP = get-replace: Restart Counts

Of the CCAs using locking, 2PLE outperforms PRE at lower PC with the reverse being true as PC increases where the T for 2PLE falls more dramatically than that of PRE. Throughout this experiment, 2PLE performs better than 2PLU. In this study, T_UPD is 1.0, that is, all selected data items are updated. Hence the difference between the 2PLE and 2PLU results reflect the effect of setting an exclusive lock at read request time (2PLE) and upgrading a share lock to exclusive lock at update request time (2PLU). An alternative, less pessimistic interpretation of write lock acquisition under 2PLU is to immediately set an exclusive lock at read request time on data items to be updated, and a share lock on all other data items. This technique is used for example in [68, 107]. This implies some kind of pre-execution analysis to determine the read and write sets for the transaction. As PC increases, the difference in T can be as much as a factor of 2 to 6 times between 2PLE and 2PLU. Comparing results for PRE, 2PLE and 2PLU, suggest that lock upgrading is more expensive than the difference

between static and dynamic locks in terms of cost. An indication of the effects of blocking is obtained by comparing the Useful Work ET for the transaction to that for the NO CC case in Appendix 3. In the case of PRE and transactions accessing one object under 2PLE there are no restarts hence these results give a more accurate indication of blocking effects. The worst case occurs for PRE in Appendix 3, Table A3.11.

From this experiment, SV emerges as a viable CCA for low conflict situations. PRE and 2PLE are reasonable choices for higher conflict situations. If a low cost pre-execution analysis is possible then PRE is the better and more robust of the two otherwise 2PLE would be the choice. One disadvantage of PRE is the high overhead costs as shown in Table 4.14. BTO and 2PLU emerge as the CCAs with the lowest performance.

The select-replace results for Experiment 1 are presented below in Tables 4.10 to 4.13. The relative performance is similar to that for get-replace, except at GRAN = 10 where 2PLE outperformed PRE and SV performed best at T_REQ = 1,2. In this transaction, the entire relation is read and a small portion updated, which favours SV. Since 2PLE holds locks for shorter times and restarts occur due to deadlocks, performance is better than under PRE. At GRAN = 1, these effects are less apparent.

From Chapter 3 it would be expected that the select-replace will perform better than the get-replace at some T_REQ value. Comparing the get-replace and the select-replace transactions in the MU experiments, the crossover point where relation access becomes more efficient than tuple access in the multi-user environment occurs at T_REQ = 10. At T_REQ > 10, the select-replace should perform better as measured by T.

In general the relative performance in this experiment was similar to that for the get-replace transaction.

T_REQ	NOCC SU	NO CC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	0.697	0.734	0.668	0.664	0.633	0.634	0.670	0.670	0.689
2	0.675	0.721	0.625	0.624	0.553	0.564	0.622	0.625	0.627
5	0.633	0.680	0.476	0.521	0.276	0.386	0.334	0.396	0.384
10	0.584	0.630	0.296	0.262	0.059	0.178	---	0.315	0.203

Table 4.10: GRAN = 1, T_TYP = select-replace: Throughput

T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	0	0	0	0	0	0	0
2	0	0	20	21	15	16	33
5	0	0	167	119	160	124	166
10	0	149	337	215	359	180	267

Table 4.11: GRAN = 1, T_TYP = select-replace: Restart Counts

T_REQ	NOCC SU	NO CC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	0.697	0.734	0.627	0.656	0.517	0.570	0.555	0.619	0.669
2	0.675	0.721	0.538	0.562	0.232	0.375	0.318	0.485	0.610
5	0.633	0.680	0.398	0.487	---	0.252	---	0.303	0.368
10	0.584	0.630	0.355	---	---	0.106	---	0.322	0.099

Table 4.12: GRAN = 10, T_TYP = select-replace: Throughput

T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	0	0	71	50	70	45	0
2	0	15	233	149	205	128	35
5	0	0	374	204	362	197	172
10	0	364	399	293	397	181	338

Table 4.13: GRAN = 10, T_TYP = select-replace: Restart Counts

4.5.4. Adaptive Restart

The poor performance of BTO above may be ascribed to cyclic restarts where, in a high conflict situation, no conflicting transaction will commit. Under SV, at least one transaction will commit, which explains why SV generally performed better than BTO. In this experiment, restarts are fake since a restarted transaction selects a random, hence potentially different, set of data objects to access. This suggest the results would be worse if restarts were real. Breaking cyclic restarts depends on allowing at least one transaction to commit. In this situation, an adaptive delay, R_DLY, might lead to improved performance. 2PLU would also benefit from an adaptive delay. Under SV, the transactions have already completed and increasing R_DLY would have little effect on PC.

The adaptive restart was implemented taking a worst case approach with R_DLY = MMPL * iET. This was used in preference to ET to avoid including blocking effects under 2PLU. Since O_OS is reasonably low in these experiments this seems a good approximation to ET. The results for A2PLU and ABTO are included in Tables 4.6 to 4.13.

The effect of the adaptive restart at the lowest PC is neglible, which is to be expected. As PC rises, the effect becomes more dramatic, especially for BTO in terms of T. 2PLU is also subject to blocking effects which tend to reduce T. The reduction in RC as PC rises, compared with the corresponding results for the non-adaptive restart version, is also significant, indicating the effectiveness of the technique. In fact, as PC rises, A2PLU outperforms SV at (GRAN = 1, T_REQ = 5,10) and ABTO outperforms the other CCAs at (GRAN = 1, T_REQ = 10).

The main result of the adaptive restart experiment is in underlining the importance of R_DLY with respect to ET and its effect on T. Under 2PLU and BTO, R_DLY was less than ET (see Appendix 3). For A2PLU and ABTO, R_DLY is approximately equal to or greater than ET, yielding a marked improvement in performance. Further research is required in this area to determine an optimal value.

4.5.5. Overhead Costs and Efficiency Measures

Tables 4.14 and 4.15 present the overhead costs and efficiency measures for each CCA. The overhead is the extra cost incurred by the transaction running under a given CCA compared with the NO CC case, expressed as a percentage. The efficiency is a measure of resources used by committed transactions over total resources used for committed and restarted transactions expressed as a percentage. This is presented for iET to reflect CPU and IO resource usage and remove O_OS. An idea of the overhead costs of blocking can be obtained by comparing the NO CC and CCA values for the Useful Work Costs in Appendix 3. The overhead costs of restarting are indicated by the differences between transaction ET and the Useful Work Cost ET. Efficiency measures for ET, for resource usage, are given in Appendix 3.

	GRAN = 1 (100 granules)							GRAN = 10 (10 granules)						
CPU: T_REQ.	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	104	103	105	105	105	104	104	106	105	142	125	136	120	158
2	106	104	115	114	112	112	114	113	114	296	175	281	163	233
5	116	114	214	166	217	167	178	127	295	2749	240	-	202	446
10	142	189	902	229	-	204	366	126	-	-	-	-	190	733
IO: T_REQ.	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	117	115	127	130	113	114	106	120	119	166	146	148	132	162
2	124	122	149	149	125	127	114	144	144	366	219	307	185	225
5	169	153	329	234	260	207	166	203	424	3835	351	-	262	412
10	305	301	1558	382	-	282	314	219	-	-	-	-	280	653

Table 4.14: Overhead Costs, T_TYP = get-replace (%)

	GRAN = 1 (100 granules)							GRAN = 10 (10 granules)						
iET: T_REQ.	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	100	100	100	100	100	100	100	100	100	76	86	76	87	68
2	100	100	94	95	95	95	93	100	96	39	63	38	65	47
5	100	98	57	70	52	67	62	100	41	5	49	3	54	25
10	100	64	16	54	5	57	31	100	11	0	1	0	58	15

Table 4.15: Efficiency Measure, T_TYP = get-replace (%)

In general, the IO overhead is greater as suggested by Table 4.5. At low PC, the differences between the CCAs are small. As PC increases, comparing PRE (pure blocking) with BTO and SV (pure restarts) indicates that blocking has lower costs. The overheads for A2PLU and ABTO are noticeably lower than for 2PLU and BTO respectively as PC increases.

The efficiency measure reflects the response of each CCA to increasing PC. The effect of granularity is also shown in the tables. Values for select-replace show similar trends and are presented in Appendix 3. In general the finer granularity has lower overheads and higher efficiency.

4.5.6. Granularity Effects

In Chapter 2, other studies were cited as suggesting an optimal granularity of between 10 and 100 granules, which lead to the choice of granularity for this study.

In general, the effect of the coarser granularity was to decrease T and increase RC. Some exceptions were noted. At T_REQ = 10, PRE performed better at GRAN = 10 than at GRAN = 1. Since PC is very high, EMPL is probably reduced to 1. This implies that when the transaction

terminates and releases its resources, there is no competition for these resources and the next transaction to wake or start has a higher chance of acquiring all its resources and executing. This seems supported by the ET values in Appendix 3, Tables A3.11 and A3.25 which suggest that more blocking takes place at GRAN = 10. Under 2PLE, T is reduced at GRAN = 10. Again from the RCs in Table 4.9 and ETs in Appendix 3 we can infer that more blocking has taken place.

For T_REQ = 10 ABTO also performs better at GRAN = 10 than at GRAN = 1. Again the argument used for PRE gives the explanation. EMPL is probably greatly reduced and the ET is small compared to R_DLY, allowing the transaction to complete quickly. The other CCA using pure restarts, SV, shows relative performance decreasing at the finer granularity but increasing at the coarser, which also suggests a much lower EMPL at GRAN = 10. The efficiency of 2PLE, 2PLU and A2PLU falls more rapidly at the coarser granularity. That of ABTO was less affected and in one case was higher, reflecting the higher T noted above.

In summary, the finer granularity gave better performance figures with the exception of PRE and ABTO which had higher T at the highest PC.

4.6. Experiment 2

In this experiment, the effect of increased access to one object is measured in contrast to conflict on access to several objects as in Experiment 1. This reflects a slightly more extreme interpretation of the so called "80-20 rule" where 80% of the transactions access 20% of the database. Here, an increasing percentage of transactions, 20% - 100%, accesses 10% of the database. Access to one object allows a comparison of static and dynamic locking, PRE and 2PLE, and dynamic exclusive versus upgrade locking, 2PLE and 2PLU, by removing restarts caused by deadlocks arising from access to more than one object. All restarts under 2PLU are caused by lock upgrade conflicts. Since the object accessed by each transaction is fixed, restarts are non fake and the expected EMPL will be closer to the actual EMPL.

4.6.1. Experiment Description

This experiment uses controlled access to a database of 10 identical relations, each containing 100 tuples. Each relation is a granule and each transaction accesses one relation. The factors and factor levels are presented in Table 4.16.

FACTORS	FACTOR LEVELS
DATABASE:	
DB_SZ	1000 tuples
DB_PG	100 pages
GRAN	10 pages/granule
NR	10
TRANS:	
NT	10
T_REQ	10
T_ACC	uniform
T_TYP	get-replace, select-replace
T_UPD	1.0
CC:	
CCA	NO CC, PRE, 2PLE, 2PLU, BTO, SV
MMPL	10
B_DLY	2 seconds
R_DLY	2 seconds/adaptive

Table 4.16: MU Exeriment 2: Factor Levels

Note that for this experiment, T_REQ and T_ACC refer to the access pattern within the relation.

The batches and number of runs are as in Experiment 1. The LC is controlled by the number of transactions, N, that access the same relation and is denoted by a percentage value derived from N/NR. Table 4.17 gives those values and the corresponding expected EMPLs.

N	1	2	5	8	10
LC	0%	20%	50%	80%	100%
EMPL	10	9	6	3	1

Table 4.17: Experiment 2: LC and EMPL

This differs from the technique proposed by Boral [20] where the LC is expressed as a percentage "degree of data sharing" of the number of partitions (granules) in the database which is equal to MMPL. For example, with MMPL = 10 and degree of data sharing = 50%, there would be 5 active partitions with two transactions accessing each active partition. The EMPL would be 5. The results presented by Boral are however based on the random access technique presented in Experiment 1 of this study and

the controlled access technique was not explored in the paper.

The full results for Experiment 2 are presented in Appendix 3, Tables A3.29 to A3.45 inclusive.

4.6.2. Single User Overhead Costs

The overhead costs of the CCAs were lower than in Experiment 1. Two factors influence this; the transactions only request one granule and access more data objects thus increasing the transaction costs with respect to the CCA costs. The relative costs were similar to Experiment 1.

4.6.3. Multi User Results

Tables 4.18 and 4.19 present T and RC for T_REQ = 10, T_ACC = uniform and T_TYP = get-replace.

T: LC	NOCC SU	NO CC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
0%	0.490	0.505	0.468	0.467	0.461	0.459	0.475	0.475	0.478
20%			0.467	0.466	0.417	0.447	0.419	0.466	0.445
50%			0.476	0.461	0.289	0.448	0.248	0.471	0.319
80%			0.446	0.432	0.151	0.409	0.099	0.428	0.175
100%			0.390	0.362	---	0.401	---	0.387	0.073

Table 4.18: MU Experiment 2: Throughput

LC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
0%	0	0	0	0	0	0	0
20%	0	0	39	18	53	19	36
50%	0	0	157	78	202	82	152
80%	0	0	276	139	323	146	271
100%	0	0	360	141	391	167	348

Table 4.19: MU Experiment 2: Restart Counts

At 0% LC there is little difference between the CCAs with SV performing best as would be expected. At 20% LC, PRE performs best, followed by 2PLE. SV performs next best followed by BTO and 2PLU. In fact the ordering PRE, 2PLE, SV remains as LC increases to 100%. 2PLU and BTO have the lowest T with BTO having almost no commits at 100% LC. These results are also reflected in the RC figures.

Comparison of the results for 2PLE and 2PLU again illustrates the effect of different write acquisition policies. However, since each transaction accesses only one granule, the result is biased in favour of 2PLE which will have no deadlocks and hence no restarts. All restarts under 2PLU in this experiment are due to share to exclusive lock upgrades and illustrate the overhead that this upgrading imposes. Static and dynamic locking can be compared in the results for PRE and 2PLE respectively. Again the results show lock upgrading to be quite costly. The difference in T between dynamic and static locking is low, indicating smaller differences in overheads. This is also reflected in Table 4.20.

As LC increases, BTO performs worse than 2PLU. In Experiment 1, the differences between the two were less clear. In this experiment, the restarts under BTO and 2PLU are non-fake which may explain the relative decrease in performance of BTO given the increased probability of cyclic restarts occurring. The fake versus real restart assumption has been found to have less impact on the relative performance in a previous simulation study [2]. The results here support this conclusion.

4.6.4. Adaptive Restart

A similar adaptive restart to that used in Experiment 1 was introduced in this experiment. The results are included in Tables 4.18 and 4.19 above. The effect was more dramatic than in Experiment 1 with RC being reduced in general by 50%. Since there were no fake restarts in this experiment, RC would be expected to be higher for 2PLU and BTO, and hence the greater improvement in RC reduction.

Both A2PLU and ABTO have significantly improved T as LC increases. The relative performance of A2PLU increased with increasing LC while that of ABTO remained fairly static. A2PLU outperformed all the other CCAs at LC = 100%.

4.6.5. Overhead Costs and Efficiency Measures

Tables 4.20 and 4.21 present the overhead costs for CPU and IO, and efficiency measures for iET for Experiment 2.

CPU: LC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
0%	102	102	103	103	101	101	101
20%	102	103	114	108	116	106	111
50%	105	104	169	127	203	127	163
80%	109	106	332	156	522	157	310
100%	112	108	-	158	-	170	764
IO: LC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
0%	116	116	118	119	113	113	112
20%	117	116	130	124	126	116	117
50%	124	126	178	139	201	132	150
80%	136	140	323	163	476	158	253
100%	141	152	-	165	-	164	577

Table 4.20: MU Overhead Costs get-replace (%)

iET: LC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
0%	100	100	100	100	100	100	100
20%	100	100	91	96	88	95	92
50%	100	100	63	82	52	81	65
80%	100	100	33	67	21	65	35
100%	100	100	10	66	2	60	14

Table 4.21: MU Efficiency Measures get-replace (%)

IO overheads are greater than CPU overheads for the pure locking techniques, PRE and 2PLE. Comparing 2PLE and 2PLU reflects the costs of lock upgrades. 2PLE has marginally greater IO overhead costs than PRE. For SV, the CPU overhead costs are greater than those for IO indicating the increasing cost of validation as LC increases. The efficiency measures for A2PLU and ABTO fall less sharply than those of the fixed delay counterparts, 2PLU and BTO, indicating the usefulness of the adaptive restart technique.

The results for select-replace were similar (see Appendix 3 Tables 3.46 to 3.90).

4.7. Conclusions

The SU overhead costs in Table 4.5 provide a guide to the inherent cost of each CCA but are of little use in predicting performance overheads in the MU environment. The MU Overhead Costs in Tables 4.14 and 4.20 provide an indicator of the resource usage for committed transactions under each CCA as PC/LC increases. In interpreting the Overhead Costs, the following should be noted. In Experiment 1, where access is probabilistic, EMPL cannot be determined precisely. Further, increasing

T_REQ will influence overhead costs and lower T by definition hence not all differences are the sole result of conflict. Additionally, CCAs using dynamic blocking retain locks on resources thus increasing PC whereas CCAs using restarts do not share this effect. Under 2PLE and 2PLU, blocking effects are also counterbalanced by the use of restarting to resolve deadlocks. The finer granularity gave better performance for most CCAs. In Experiment 2, T_REQ is fixed and EMPL is controlled. Differences in performance are directly related to conflict in this experiment.

The Efficiency Measures in Tables 4.15 and 4.21 provide an indicator which includes resources "wasted" by restarts. As such they provide indicators at the same level as T, that is, on the concurrent system level. Both Overhead and Efficiency measures provide useful additions to performance measurement by T.

Table 5.17 in Chapter 5 gives the relative ordering by throughput of the CCAs in Experiment 1, GRAN = 1, GRAN = 10 and Experiment 2. The table shows the relative movement of each CCA as LC increases. The effect of the adaptive restart is also illustrated.

At low LC, there is little difference in performance of the CCAs. SV, with its low overhead, performed best. As LC increases, PRE and 2PLE emerge as having best performance. If the transaction requirements are known in advance, PRE is the best overall choice otherwise 2PLE would be recommended. The performance of 2PLU and BTO became unacceptable at higher LC with high RC. Adaptive restarts improved the performance of both 2PLU and BTO. However in only a few cases was there improvement relative to the other CCAs.

One significant result in this chapter is the improvement using the adaptive restart technique. The relative performance of ABTO was better in higher conflict situations but was less efficient at GRAN = 10. This illustrates the importance of R_DLY, the delay time in restarting an aborted transaction, with respect to ET in CCA performance experiments. Contradictory results from previous simulation experiments, where timestamp CCAs have outperformed locking CCAs in some studies [46, 69] and the opposite has been found in other studies [23], may be explained by this result.

In addition to throughput results, this study has presented results for ET, CPU and IO usage for both useful and wasted (restarted) work for transactions. These have:

1) allowed the overhead costs of the CCA to be calculated using the NO CC case as the baseline

2) shown the effects of blocking under PRE, 2PLE and 2PLU

3) provided an efficiency measure of each CCA at all factor levels

4) given an idea of how much work is lost by restarts for each CCA

The different transaction types, get-replace and select-replace, were similar in terms of relative performance showing only a few differences at GRAN = 10. At the levels tested in the prototype experiments there was little difference in the quantitive results. The larger differences found in Chapter 3 occurred at higher levels of T_REQ.

4.8. Chapter Summary

Two experiments to compare the performance of CCAs in a multi user database were presented and tested. Adaptive restart versions of 2PLU and BTO were found to greatly improve performance for those CCAs. Several measures of performance were introduced. The results were compared using SU amd MU tests as baselines. The conclusions reached from the experimental results generally agreed and highlighted different aspects of CC performance.

Chapter 5

SIMULATION EXPERIMENTS

The two experiments of Chapter 4 are repeated using a simulator. The simulator results are presented and then compared with the results presented in Chapter 4. The differences are discussed. The simulator is written in Simula.

5.1. Simulation

As noted in Chapter 1, simulation has been widely used to model CCA performance and has been the major alternative to analytic studies. The simulator is based on a physical queueing model (PQM) presented in [23]. This PQM is presented in Figure 5.1 and is an abstraction of the LQM given in Figure 2.7, for a system with one CPU and one disk. A version for several CPUs and several disks is presented in [2].

The PQM models a system with one disk, one CPU and a number of terminals or batch transactions. Each server, I/O and CPU, has two queues, one for the CCA and one for the transaction processing. In contrast to [23], the CCA queues are not given priority over the transaction queues. This reflects the prototytpe system. Both are served using a first-come, first-served (FCFS) model as in the testbed system. The parameters for the system are given in Table 5.1.

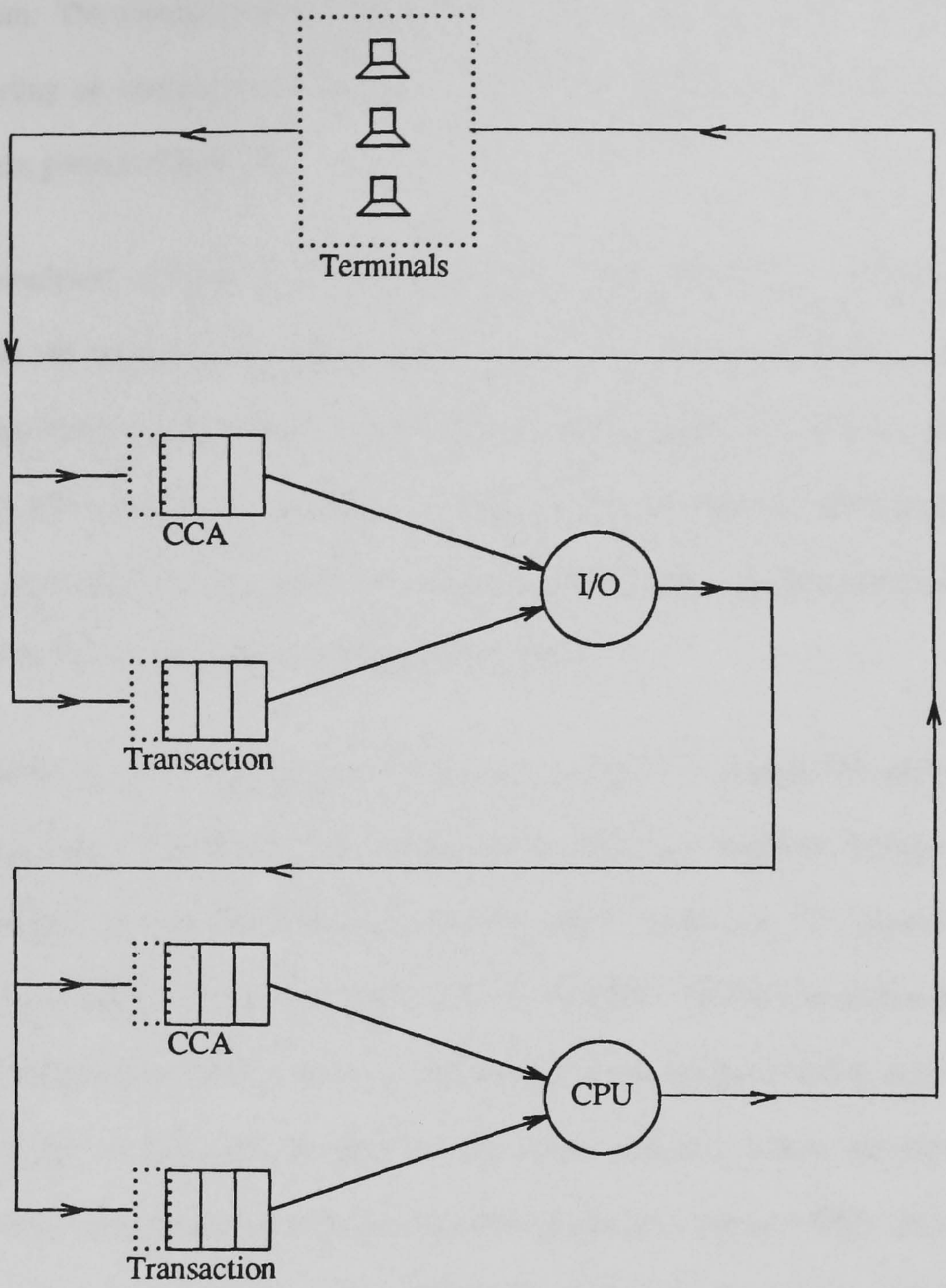


Figure 5.1: Physical Queueing Model (PQM)

parameter	meaning
CC_IO	CCA I/O cost
CC_CPU	CCA CPU cost
TR_IO	Transaction I/O cost
TR_CPU	Transaction CPU cost
STGM	Exponential Randomising Delay

Table 5.1: PQM Parameters

When a transaction reads an object, it makes a request to the CCA (except in the case of SV) which incurs a cost, CC_IO + CC_CPU, and then accesses the object at cost TR_IO + TR_CPU. On writing the object a further cost of TR_IO + TR_CPU is incurred and extra CCA cost in the cases of 2PLU and BTO. Recall that SV incurs little CCA cost; once at transaction startup and once at transaction validation. STGM is used to stagger transaction startup, that is, when a new transaction

enters the system. The simulation is run as a closed system with a fixed number of transactions, a new transaction starting on completion of a currently executing transaction. The CCA and transaction descriptions were given in Chapter 2.

Three simulation techniques are possible for the experiments; batched means, independent replications and the regenerative method [32,96,43,76]. The technique chosen is that of batched means. The regenerative method was rejected since the regeneration state does not occur sufficiently frequently [23]. The independent replications method was rejected partially on the grounds of cost and partly because the initial transient results may produce biased estimators of average values [96]. The batched means method is "generally accepted practice" [76].

Using batched means, each simulation comprises a number of batches, NB, each of which is BT simulation units long. The first batch is discarded to eliminate transient startup effects and the remaining runs used to provide observations for ET and T which are then used to provide point estimators for these values. In this simulation, NB = 21 and BT = 100000 simulation time units. Each simulation time unit is interpreted as a millisecond and the results are presented in units of seconds. BT is chosen to be large, in fact twice that used in [23]. Firstly, this is to support the assumption that the sample observations are normally distributed, from the central limit theorem [96]. Secondly, to be able to treat the batches as effectively independent by assuming dependencies between adjacent batches to be neglible, [32, 76]. This was verified by assuming the correlation between non-adjacent batches to be neglible and calculating the covariance for adjacent batches. These values were found to be extremely small which supported the assumption that the batches are effectively independent. This is further discussed in Appendix 5.

The parameter values for the simulation experiments are given in Table 5.2.

parameter	values (milliseconds)
CC_IO	2
CC_CPU	3
TR_IO	20
TR_CPU	25
STGM	20

Table 5.2: PQM Parameter Values

The values were chosen from the average disk access time of the testbed system which was 18ms [45], and the results of the testbed system, noting that TR_CPU tends to be greater than TR_IO. The CC_IO cost is assumed to be low since the tables will effectively reside in the O/S buffer when called frequently.

The two experiments described below are simulation equivalents of the two experiments described in Chapter 4.

5.2. Experiment 1

5.2.1. Experimental Results

The factors and factor levels are similar to Table 4.4. The database is now modelled as a set of granules corresponding to DB_PG and GRAN. There is no distinction in T_TYP. The full results are presented in Appendix 4. Results for T and RC are presented below in Tables 5.3, 5.4 respectively for GRAN = 1 and Tables 5.5, 5.6 respectively for GRAN = 10. An additional run of BTO with R_DLY set to 1 second is introduced and labelled XBTO.

T_REQ	NO CC	PRE	2PLE	2PLU	A2PLU	BTO	XBTO	ABTO	SV
1	20.000	17.800	17.760	16.270	16.420	16.650	16.650	16.660	18.460
2	10.000	9.069	8.769	7.786	7.971	8.390	8.365	8.327	9.052
5	4.000	3.425	1.976	1.952	1.911	3.077	3.017	2.903	2.974
10	2.000	1.397	0.481	0.442	0.397	1.074	0.860	0.521	0.980

Table 5.3: GRAN = 1: Throughput

T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	XBTO	ABTO	SV
1	0	0	52	62	55	55	54	76
2	0	0	63	74	67	73	81	127
5	0	12	63	73	114	131	161	193
10	0	40	62	67	193	256	388	200

Table 5.4: GRAN = 1: Restart Counts

T_REQ	NO CC	PRE	2PLE	2PLU	A2PLU	BTO	XBTO	ABTO	SV
1	20.000	15.770	8.456	6.613	6.787	14.560	14.530	13.400	16.570
2	10.000	6.775	1.685	1.521	1.303	6.690	6.354	4.307	7.239
5	4.000	2.173	0.234	0.169	0.138	1.687	1.218	0.116	2.005
10	2.000	1.045	---	---	---	---	---	---	0.649

Table 5.5: GRAN = 10: Throughput

T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	XBTO	ABTO	SV
1	0	0	156	239	268	389	675	311
2	0	13	131	170	299	472	1065	348
5	0	65	117	137	359	624	1364	346
10	0	81	118	131	383	675	843	263

Table 5.6: GRAN = 10: Restart Counts

For both levels of granularity, PRE emerges with best T as PC increases. SV has higher T at lower PC ((GRAN = 1, T_REQ = 1) and (GRAN = 10, T_REQ = 1,2)) but higher RC. At T_REQ = 5, 10, BTO outperforms SV at GRAN = 1 but is outperformed by SV at GRAN = 10. SV performs better at the higher PC since at least one of the conflicting transactions is guaranteed to commit.

At the coarser granularity, GRAN = 10, and at (GRAN = 1, T_REQ = 5,10), both BTO and SV perform better than 2PLE and 2PLU. This suggests that restart based techniques perform better at higher PC and dynamic locking based techniques perform worse. Since blocked transactions hold locks on objects, the potential for conflict is increased for the remaining transactions. Restarted transactions are removed from the system leaving fewer transactions competing for the same number of objects and hence reducing the potential conflict. This would explain the reduction in performance of 2PLE and 2PLU as PC increases. PRE does not hold any resources when blocked, hence the better performance.

Throughout the simulation, 2PLE performs better than 2PLU. 2PLU gave the worst performance overall.

In a simulation study, the system is an idealised model. In this study, the differences in results are due to the effects of blocking and restarts since all costs for the CCAs and transaction CPU and IO are fixed. The blocking effect is controlled by B_DLY and the restart delay by R_DLY. Both reduce EMPL and hence improve execution times for those transactions still active.

The overall result of this experiment was that PRE performed best and in general, SV and BTO outperformed 2PLE and 2PLU. The effects of the adaptive restart are discussed below.

5.2.2. Adaptive Restart

Adaptive restart versions of 2PLU and BTO, A2PLU and ABTO respectively, were run with R_DLY set equal to the average ET. An adaptive restart for BTO seems especially useful to avoid the occurrence of cyclic restarts. The results are included in Tables 5.3 to 5.6 above. In general the performance of ABTO was worse than that of BTO but that of A2PLU better than 2PLU.

The explanation can be found by comparing R_DLY to the ET values in Appendix 4. For BTO, with fixed R_DLY = 2 seconds, R_DLY >> ET in all instances but one at (GRAN = 1, T_REQ = 10). At this point ET for BTO is 3.029 seconds (Appendix 4, Table A4.9). This is also confirmed for XBTO where R_DLY = 1 second. Where R_DLY > ET and ET for ABTO is less than 1 second, XBTO outperforms ABTO. Comparative tables of ET and T are given in Appendix 4, Tables A4.20 to A4.25.

The above explanation does not however hold for 2PLU and A2PLU. As well as being subject to R_DLY effects, 2PLU and A2PLU are also affected by B_DLY. Examination of the ET values for 2PLU and A2PLU in Appendix 4, Tables A4.9 and A4.11, shows greater increases in ET than for the corresponding cases of BTO and ABTO. This also demonstrates that for 2PLU, the relative difference between the (fixed value) R_DLY and ET is much greater than for BTO. The chances of a conflict reoccurring between a restarted and executing transaction are thus correspondingly increased.

Two points therefore emerge in this part of the experiment. Firstly, that R_DLY should be in the order of magnitude of ET still seems a reasonable heuristic. Secondly, the relative difference between a fixed R_DLY and ET also plays an important role in determining performance. These points are further explored below in Sections 5.3.2 and 5.4. A summary of the CCAs ranked in order of decreasing T is presented in Table 5.17.

5.2.3. Overhead and Efficiency Measures

In the simulation, CPU and IO costs are fixed values for simplicity therefore the effects of blocking and restarting are reflected in the ET and T values in the results. In this experiment, the locking based CCAs have higher overheads, particularly at GRAN = 10. PRE has the lowest overheads of the locking based CCAs. Comparing PRE, 2PLE and 2PLU shows that the cost of dynamically setting locks is high while the cost of lock upgrades is relatively low. Comparing BTO and XBTO indicates differences due to different fixed value R_DLYs. Tables 5.7 to 5.10 inclusive show the overhead and efficiency measures for Experiment 1.

ET: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	XBTO	ABTO	SV
1	112	113	123	122	120	120	120	108
2	110	114	128	127	119	120	119	110
5	117	202	205	199	130	133	130	134
10	143	416	452	373	186	232	175	204

Table 5.7: GRAN = 1: Overhead Values (%)

ET: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	XBTO	ABTO	SV
1	127	237	302	299	137	138	142	121
2	148	593	657	669	149	157	166	138
5	184	1713	2360	1566	237	328	336	199
10	191	---	---	---	---	---	---	308

Table 5.8: GRAN = 10: Overhead Values (%)

ET: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	XBTO	ABTO	SV
1	100	100	95	95	100	100	98	98
2	100	100	88	87	100	100	95	93
5	100	99	70	68	95	93	77	74
10	100	58	44	40	64	48	23	48

Table 5.9: GRAN = 1: Efficiency Measure (%)

ET: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	XBTO	ABTO	SV
1	100	100	61	56	98	96	77	90
2	100	99	47	39	92	84	43	77
5	100	14	15	12	58	33	2	50
10	100	0	3	2	26	2	1	30

Table 5.10: GRAN = 10: Efficiency Measure (%)

The efficiency measure gives the "useful work" achieved by each CCA. Note that in some cases, for example comparing 2PLE and SV, one CCA may have lower efficiency but higher T reflecting the costs of restarts and blocking. In Tables 5.9 and 5.10, by measuring efficiency using ET, blocking delays are included as "useful work".

5.2.4. Granularity Effects

In this experiment, for all CCAs, T was reduced and RC increased at GRAN = 10 compared with the corresponding results at GRAN = 1. Finer granularity therefore in all cases gave better performance.

5.3. Experiment 2

5.3.1. Experiment Results

The factors and factor levels are similar to Table 4.16. T_REQ is set to 1 and T_ACC and T_TYP are not used. Recall that this experiment represents a situation where an increasing percentage of transactions access a small percentage of the database. The throughput results are given in Table 5.11 and restart counts in Table 5.12.

LC	NO CC	PRE	2PLE	2PLU	A2PLU	BTO	XBTO	ABTO	SV
0%	20.000	17.860	17.860	16.950	16.950	16.950	16.950	16.950	18.870
20%		17.830	17.820	16.770	16.710	16.730	16.950	16.950	18.640
50%		17.720	17.710	15.950	15.580	15.870	15.130	10.150	17.920
80%		17.670	14.060	12.370	11.910	14.240	12.660	4.373	17.110
100%		9.961	0.820	0.434	0.480	7.835	7.378	---	9.995

Table 5.11: Experiment 2, Throughput

LC	PRE	2PLE	2PLU	A2PLU	BTO	XBTO	ABTO	SV
0%	0	0	0	0	0	0	0	0
20%	0	0	29	37	42	0	0	42
50%	0	0	98	164	176	328	1292	178
80%	0	0	172	313	301	598	2369	332
100%	0	0	390	412	436	831	3042	428

Table 5.12: Experiment 2, Restart Counts

As in Experiment 1 at GRAN = 10, PRE and SV give the best performance. Under PRE, the conflicting transactions are serialised by blocking for B_DLY seconds. Examination of ET values for PRE in Appendix 4, Table A4.18, reveals that ET is smaller than B_DLY and decreasing as LC increases. Hence after initial blocking, it is highly probable that a blocked transaction is delayed for a sufficient length of time to allow other potentially conflicting transactions to execute several times. Similarly under SV, the ET is small compared to R_DLY and decreasing as LC increases. Now restarted transactions are removed from the system for a sufficient length of time to allow other potentially conflicting transactions to execute several times.

Under 2PLE and 2PLU, blocked transactions are queued in order of blocking. When one transaction completes, a blocked transaction is allocated the resource but may not become active immediately. Compared to PRE, the length of time the resource remains locked is greater since there is an extra length of time when the resource is allocated to a (still) blocked transaction. For 2PLE, the locking overhead is considerable as shown by the ET in Table A4.18. 2PLU also suffers from the overhead of lock upgrading. is mollified by the effects of restarting. At LC = 100%, the cost of lock upgrades almost doubles the overhead for 2PLU. outweighed by the advantages of restarting as shown by the better performance of BTO.

Both 2PLU and BTO, and to a lesser extent, PRE and 2PLE, also suffer queueing delays in accessing CC information. SV, which makes fewer requests, is less affected.

In this experiment, PRE and SV give the best performance, that of BTO increases as LC increases and that of 2PLE and 2PLU decreases. 2PLE always outperforms 2PLU. The relationship of B_DLY and R_DLY to ET also emerges as an important factor in determining performance. The results, with CCAs ranked in order of decreasing T, are presented in Table 5.17.

5.3.2. Adaptive Restart

Again ABTO performed less well than BTO at all levels of LC. A2PLU performed less well than 2PLU in all cases but one, at LC = 100%. In this case the difference in T is small and can be attributed to the increased R_DLY for A2PLU where the ET value is 2.188 (Appendix 4, Table A4.18). The explanations are similar to those presented in Section 5.2.2.

5.3.3. Overhead and Efficiency Measures

At LC = 0%, the overhead represents the additional cost of each CCA. The differences in overhead are lower except at LC = 100% where dynamic locking has very high overhead and PRE and SV have the lowest. The following are the overhead and efficiency measures for Experiment 2.

T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	XBTO	ABTO	SV
0%	112	112	118	118	118	118	118	106
20%	112	112	119	120	120	118	118	107
50%	113	113	125	128	126	132	148	112
80%	113	142	162	167	140	158	226	117
100%	201	2439	4608	4211	255	271	---	200

Table 5.13: Experiment 2, Overhead Values (%)

T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	XBTO	ABTO	SV
0%	100	100	100	100	100	100	100	100
20%	100	100	96	91	100	100	100	99
50%	100	100	76	62	99	98	57	94
80%	100	100	47	33	95	90	22	90
100%	100	100	46	11	88	70	0	71

Table 5.14: Experiment 2, Efficiency Measures (%)

Comparing PRE and 2PLE measures the cost of static vs dynamic locking, that is, the cost of keeping a lock queue on an object. AT LC = 100% the overhead for 2PLE is extremely high. Under

2PLU, this cost is lessened by the effects of restarting, although there is no improvement in T. Comparing the overhead of PRE to SV at LC = 100%, shows the high cost of locking in an extreme situation. At this LC, the only solution is serial execution of transactions. Both CCAs will remove 9 transactions from the system for B_DLY/R_DLY seconds leaving the remaining transaction free to execute.

The efficiency measure again reflects the "useful work" achieved in this scenario. The measure for 2PLE is somewhat unrepresentative since deadlock never occurs under 2PLE on access to one object. A measure of the effect of lock upgrades is obtained by comparing 2PLE to 2PLU since all restarts here are caused by deadlocks due to lock upgrades. Again the differences in T between static and dynamic locking are smaller than those due to lock upgrades.

5.4. Adaptive Restart Revisited.

In the simulation, the performance of A2PLU and ABTO was worse than their non-adaptive restart versions. The explanation appears to be in the relationship between the restart delay, R_DLY, and the transaction elapsed time ET. Under 2PLU and BTO, the R_DLY was in most cases greater than under A2PLU and ABTO respectively. To investigate this further, two more sets of tests were made with an adaptive restart of 0.001 * ET to reflect a situation with almost no restart delay and 10.0 * ET to reflect one with a long restart delay. Table 5.15 summarises the CCAs and corresponding restart delays and introduces the notation for the new CCAs used in Table 5.16. Table 5.16 presents the results of this experiment combined with previous results. The CCAs are ranked in decreasing order of T.

CCA	R_DLY	CCA	R_DLY
2PLU	2 seconds	BTO	2 seconds
		XBTO	1 second
<A2PLU	0.001 * ET	<ABTO	0.001 * ET
=A2PLU	1.000 * ET	=ABTO	1.000 * ET
>A2PLU	10.000 * ET	>ABTO	10.000 * ET

Table 5.15: CCAs with R_DLY Values

An asterisk (*) beside 2PLU, BTO or XBTO indicates that the R_DLY was greater than or equal to the average transaction ET.

T_REQ	GRAN = 1											
1	SV	PRE	2PLE	<ABTO	BTO*	XBTO*	=ABTO	>ABTO	<A2PLU	=A2PLU	2PLU*	>A2PLU
2	PRE	SV	2PLE	BTO*	XBTO*	=ABTO	>ABTO	<ABTO	<A2PLU	=A2PLU	2PLU*	>A2PLU
5	PRE	>ABTO	BTO	=ABTO	XBTO	<ABTO	SV	>A2PLU	=A2PLU	2PLE	2PLU	<A2PLU
10	PRE	>ABTO	=ABTO	BTO	SV	XBTO	>A2PLU	=A2PLU	<ABTO	2PLE	2PLU	<A2PLU
T_REQ	GRAN = 10											
1	SV	PRE	BTO*	XBTO*	>ABTO	=ABTO	<ABTO	2PLE	<A2PLU	=A2PLU	>A2PLU	2PLU*
2	SV	PRE	BTO*	>ABTO	XBTO*	=ABTO	<ABTO	>A2PLU	2PLE	2PLU*	=A2PLU	<A2PLU
5	PRE	SV	>ABTO	BTO*	XBTO*	=ABTO	>A2PLU	=A2PLU	2PLE	2PLU	<A2PLU	<ABTO
10	PRE	>ABTO	SV	BTO*	=ABTO	>A2PLU	XBTO	=A2PLU	2PLE	2PLU	<ABTO	<A2PLU
LC												
0%	SV	PRE	2PLE	>A2PLU	2PLU*	=A2PLU	<A2PLU	BTO*	>ABTO	XBTO*	=ABTO	<ABTO
20%	SV	PRE	2PLE	>ABTO	XBTO*	=ABTO	<ABTO	>A2PLU	2PLU*	BTO*	=A2PLU	<A2PLU
50%	SV	PRE	2PLE	>ABTO	>A2PLU	2PLU*	BTO*	=A2PLU	<A2PLU	XBTO*	=ABTO	<ABTO
80%	PRE	SV	BTO*	>ABTO	2PLE	XBTO*	>A2PLU	2PLU*	=A2PLU	<A2PLU	=ABTO	<ABTO
100%	SV	PRE	BTO*	>ABTO	XBTO*	>A2PLU	2PLE	=ABTO	<A2PLU	=A2PLU	2PLU	<ABTO

Table 5.16: Simulation Results: Ranked CCAs

Tables of T, RC, BC and ET for (2PLU, >A2PLU, =A2PLU, <A2PLU) and (BTO, XBTO, >ABTO, =ABTO, <ABTO) are given in Appendix 4, Tables A4.20 to A4.25. The statistical results of the tests are not reported but are of the same level of accuracy as the main experiments.

At lower levels of conflict, T_REQ = 1,2, increasing R_DLY has little effect. In fact, for GRAN = 1, <A2PLU and <ABTO perform best. Examination of Appendix 4, Tables A4.20 to A4.25 shows little difference in T and RC at this level. As PC increases, increasing R_DLY improves performance. The relationship of R_DLY to ET is also important. The two cases where R_DLY is much less than ET, <A2PLU and <ABTO, tend to have lowest T as PC increases.

Of more interest in Table 5.16 is the occurrence of groupings of 2PLU, =A2PLU, >A2PLU and BTO, XBTO, =ABTO, >ABTO, especially at lower levels of conflict. In low conflict situations, increasing R_DLY improves performance slightly but not relative to the other CCAs. For BTO, there is a relative improvement at (GRAN = 10, T_REQ = 10) and LC = 20%, 50%. Adaptive restart for 2PLU does improve performance over 2PLE in some cases where conflict is higher. ((GRAN = 1, T_REQ = 5,10); (GRAN = 10, T_REQ = 1,2,5,10); LC = 100%).

To sum up, R_DLY greater than ET generally gives better performance. Setting R_DLY much larger than ET does not give a proportionally greater increase in performance as shown by >A2PLU and

>ABTO. Further, there is little performance increase relative to the other CCAs. From this simulation study we conclude that in fact the adaptive restart technique does not have a strong influence on CCA performance as long as R_DLY is greater than ET.

5.5. Simulation: Summary and Conclusions

The simulation was set up to model the prototype testbed and examine the behaviour of the CCAs tested using the prototype. A reasonably simple model was adopted using fixed cost overheads for CPU and IO for both the transaction processing and the concurrency control. As such, the information yielded by the model was limited to throughput values, elapsed times and restart and block counts. No priority was given to the queues for CC IO and CPU in order to reflect the situation of the prototype. A fixed blocking and restarting time was used, corresponding to the times in the prototype. Results were presented with a 90% confidence interval expressed as a percentage of the mean value. Additional measures for overhead and efficiency were introduced. Since this is a comparitive study, the results were also summarised in order of decreasing T.

From Experiment 1, it was found that PRE emerged at the "best" CCA, the relative performance of 2PLE and 2PLU decreased with increasing PC and that of BTO increased. The adaptive restart, setting R_DLY equal to the average ET, performed less well for BTO and better for 2PLU. The importance of the relationship between R_DLY and ET emerged. These results were also found in Experiment 2 where SV and PRE emerge as "best" CCAs.

A second set of experiments investigating the adaptive restart technique and values for R_DLY, showed that with a very small R_DLY relative to ET, performance fell and RC increased. A reasonable value seems to be one greater than ET although a large increase of R_DLY relative to ET did not yield a proportional increase in performance. The effect of varying R_DLY with respect to ET was found not to produce an increase in performance relative to other CCAs at most factor levels.

The results also allow the performance of static and dynamic locking to be compared, the former giving better performance at all factor levels. The effect of writelock placement was measure by comparing 2PLE with 2PLU and to a lesser extent the effect of the fake restart assumption by comparing

Experiment 1 results with Experiment 2. The relative order of locking based policies remained (PRE, 2PLE, 2PLU) when fixed B_DLY and R_DLY were used. The cost of upgrading locks reduces performance although less so than the cost of setting dynamic locks with respect to static locks. The effects of granularity were measured in Experiment 1. Finer granularity gave better performance at all factor levels. In general the locking policies were more affected by coarser granularity.

5.6. Comparison with the Prototype.

The results of the simulation experiments are now compared with those from the prototype. Firstly a summary of the results is presented in Table 5.17 with the CCAs ranked in descending order of T.

T_REQ	PROTOTYPE GRAN = 1							SIMULATION GRAN = 1						
1	SV	2PLE	BTO	ABTO	PRE	2PLU	A2PLU	SV	PRE	2PLE	BTO	ABTO	A2PLU	2PLU
2	2PLE	PRE	SV	ABTO	BTO	A2PLU	2PLU	PRE	SV	2PLE	BTO	ABTO	A2PLU	2PLU
5	2PLE	PRE	A2PLU	SV	ABTO	BTO	2PLU	PRE	BTO	ABTO	SV	A2PLU	2PLE	2PLU
10	ABTO	PRE	2PLE	A2PLU	SV	2PLU	BTO	PRE	ABTO	BTO	SV	A2PLU	2PLE	2PLU
T_REQ	PROTOTYPE GRAN = 10							SIMULATION GRAN = 10						
1	2PLE	PRE	ABTO	A2PLU	BTO	2PLU	SV	SV	PRE	BTO	ABTO	2PLE	A2PLU	2PLU
2	PRE	2PLE	ABTO	A2PLU	SV	BTO	2PLU	SV	PRE	BTO	ABTO	2PLE	2PLU	A2PLU
5	PRE	ABTO	A2PLU	2PLE	SV	2PLU	BTO	PRE	SV	BTO	ABTO	A2PLU	2PLE	2PLU
10	PRE	ABTO	SV	2PLE	A2PLU	2PLU	BTO	PRE	SV	BTO	ABTO	A2PLU	2PLE	2PLU
LC	PROTOTYPE							SIMULATION						
0%	SV	BTO	ABTO	PRE	2PLE	2PLU	A2PLU	SV	PRE	2PLE	2PLU	A2PLU	BTO	ABTO
20%	PRE	2PLE	ABTO	A2PLU	SV	BTO	2PLU	SV	PRE	2PLE	ABTO	2PLU	BTO	A2PLU
50%	PRE	ABTO	2PLE	A2PLU	SV	2PLU	BTO	SV	PRE	2PLE	2PLU	BTO	A2PLU	ABTO
80%	PRE	2PLE	ABTO	A2PLU	SV	2PLU	BTO	PRE	SV	BTO	2PLE	2PLU	A2PLU	ABTO
100%	A2PLU	PRE	ABTO	2PLE	SV	2PLU	BTO	SV	PRE	BTO	2PLE	ABTO	A2PLU	2PLU

Table 5.17: Summary of Prototype and Simulation Results: Ranked CCAs

Clearly the simulation model is a greatly simplified model of the prototype system. The aim here is to show whether the trends in the simulation results coincide with those of the prototype. The two sets of experiments also provide a framework in which to explore the similarities, differences and assumptions underlying the model.

5.6.1. Experiment 1

One obvious difference in the performance of the adaptive restart versions. In the prototype, performance improved dramatically and in a number of cases, the adaptive restart version ranked first or second and the non adaptive restart version ranked second last or last. Explanation of this behaviour has been given above in Section 5.4.

One approach is therefore to compare the fixed length blocking and restarting CCA versions and the relative movement in Table 5.17. In both the prototype and simulation, the performance of PRE increases as PC increases. That of SV decreases at GRAN = 1 but increases again in the prototype at GRAN = 10. At GRAN = 1, the relative performance of 2PLE decreases in the simulation but less conclusively so in the prototype results. At GRAN = 10, the relative performance of 2PLE decreases in the prototype but remains static in the simulation. For BTO there is a performance decrease in the prototype results but increase in the simulation results. In all cases 2PLE performs better than 2PLU. In the prototype, 2PLE sometimes performs better than PRE but inspecting the T values shows that the difference is small. PRE always performs better than 2PLE in the simulation.

The differences in R_DLY with respect to ET suggest that a similar effect may be caused by B_DLY. In the prototype experiment, B_DLY is less than ET whereas in the simulation it is mainly greater. B_DLY > ET implies a potentially greater number of blocks and since blocked transactions hold existing locks for longer periods, more wasted time since checking for released locks is performed less frequently. This affects 2PLE and 2PLU in the simulation experiments. PRE is unaffected since no resources are held while blocked. R_DLY > ET on the other hand would benefit pure restart based CCAs such as BTO and SV. This is the case in the simulation. In the prototype where R_DLY < ET, restart based CCAs are disadvantaged since cyclic restart situations can now arise. This would explain the difference in results for BTO and the relative position of SV. Given the optimistic assumption underlying SV, that conflicts are rare, it is not surprising that performance degrades as PC increases. The relative difference between SV and 2PLE, the latter tending to perform better in the prototype, may be explained by the effect of B_DLY with respect to ET. Similarly the differences between BTO and 2PLU could arise from the effect of B_DLY with respect to ET. In both the prototype and the

simulation, 2PLU generally seems to have the worst performance. Both B_DLY and R_DLY differences with respect to ET affect 2PLU so that it is difficult to determine which effect is dominant. The trend can be measured by the overhead values which tend to increase under blocking effects and decrease under restarting effects.

5.6.2. Experiment 2

In this experiment SV performed less well in the prototype. Again this may be explained by R_DLY < ET, allowing restarted transactions to reenter the system earlier and increase the risk of conflict, since restarts here are non fake. Differences in 2PLE, 2PLU and BTO can also be explained as in Experiment 1, in terms of the relationship of B_DLY and R_DLY to ET. Further research aimed at establishing a more precise relationship between ET and the blocking and restarting delay times would be useful in this area.

5.6.3. Comments on the Comparison

One point highlighted by the comparison, is the difficulty of calibrating a simulation model and determining the relationship between paramaters. In the SU tests of Chapter 3, CPU was found to be the main overhead of a transaction (Table 3.8). Comparing overheads for CCAs, IO emerges as the main overhead (Table 4.5). In the simulation tests, the CC costs were set to be about 10% of the transaction costs. Such choices are made to model particular situations and it is difficult to predict the sensitivity of the model to these choices. This point is further discussed in Chapter 6.

Comparison with any prototype system can at best be only approximate because of the large number of parameters involved. In Chapter 3, the behaviour of the prototype was examined to determine potential bottlenecks, the synthetic data placed optimally with respect to the logical data pages, and access costs shown to be linear as an increasing proportion of the database is accessed. Index accessing was shown to be efficient and have little extra cost. Interpretation of the results however still involves treating the prototype as an idealised system.

As a result of these comparisons, a major performance factor to emerge was the difference between the delay times, B_DLY and R_DLY, and ET. In Section 5.4 it was shown that when R_DLY was set to a value of 10.0 * ET, the improvement in performance was small and on only a few occasions did performance relative to the other CCAs improve.

It seems reasonable to conclude that within the simulation, the fixed value of R_DLY, which remained below 10 * ET in all but a few cases, had little effect on the relative performance of the CCAs. What has been demonstrated is that the relationship of R_DLY to ET is a significant determining factor with regard to performance.

In both the prototype and the simulation, static locking performed better than dynamic locking in Experiment 2. In Experiment 1 the same result was found in the simulation but in the prototype, dynamic locking performed better at low PC.

5.7. General Conclusions

The simulation model was chosen to be reasonably simple, with a limited number of parameters. Since the transaction and CC costs were fixed, the information yielded from the model was less detailed than that from the prototype. The performance measures from the simulation model were T, ET, RC and BC. The simulation model also excluded other costs such as O/S overhead and deadlock detection for dynamic locking. This seems justifiable since examination of the prototype results shows these costs to be relatively small.

Using the simulation model, the experiments run in the prototype system were repeated and the results compared, revealing several differences. At low conflict levels, performance differences between the CCAs were small in both the prototype and simulation, confirming a similar conclusion reached in [23]. One major difference between the studies is in the performance of locking vs restart techniques. In the prototype, PRE and 2PLE showed better performance than BTO. However the adaptive restart in ABTO improved performance over 2PLE at higher PCs. In the simulation, 2PLE performed better at lower levels of conflict and BTO better at higher levels. The restart based technique of BTO was shown to work better in higher conflict situations in these experiments.

For both sets of tests, the CC information resided on disk. In the simulation model, the CC costs were set low relative to the transaction costs on the assumption that the CC tables would be located in the system buffer under high access frequencies. IO was the greater overhead as predicted from the SU tests even though the transactions themselves were CPU bound. In the prototype, IO and CPU costs were measured separately allowing overheads to be easily determined. In the simulation, the model was much simplified by assuming fixed costs. Measurement of overheads using ET produced less clear results since two opposing factors were at work; blocking, which increases the overhead measure, and restarting, which decreases it. Other studies have assumed the CC information to be resident in primary memory, hence giving a CC_IO cost of zero. However in systems without memory sharing such as the UNIX system used for the prototype, extra costs will be incurred from process to process communication as noted in [100, 66, 63]. In this case communication costs replace IO costs and a value of zero for CC_IO is perhaps optimistic.

Granularity has been previously studied elsewhere and the results here confirm conclusions that generally a finer granularity in the region of 100 granules is to be preferred. It was noted however in the prototype that in two cases, PRE and ABTO, T improved at the coarser granularity of 10 granules. This also supports the belief that a restart based CCA (with suitable R_DLY) performs better at higher conflict levels.

By comparing results from both the prototype and simulation experiments, it was shown that an adaptive restart delay of greater than or equal to ET improved performance. Further simulation experiments showed that a greater increase in the delay value did not improve the relative performance of the CCA involved.

The comparison suggest that a closer tuning of the simulation to the prototype is required for more conclusive results. As a first step, choosing different values for B_DLY and R_DLY should improve the simulation model. B_DLY should have a value less than ET and R_DLY one greater than or equal to ET. Further research is required to determine optimal values. For B_DLY, the balancing factors are the locking overhead costs, which if small suggests a smaller value for B_DLY, and the number of locks and length of time held which increase PC. Investigation into the sensitivity of the model to changes in parameter values would also be useful.

5.8. Chapter Summary

The simulation model was presented and simulation techniques briefly discussed. Results, using the simulation model, for Experiments 1 and 2 of Chapter 4 were presented and discussed. Several issues in CCA performance were covered including blocking vs restarts, dynamic vs static locking, lock upgrade costs, CC overhead costs, the effects of granularity and the adaptive restart technique. A second, more detailed discussion of adaptive restart was presented, followed by a summary of and conclusions from the simulation experiments. These results were then compared with those of the prototype. Finally conclusions and some improvements to the simulation model were presented.

Chapter 6

COMPARISON WITH OTHER STUDIES

In this chapter, the results from Chapters 4 and 5 are compared with other studies in the literature. In the majority of cases a detailed comparison is not possible, for reasons discussed in Chapter 2 but where possible, such a comparison is attempted.

6.1. Experimental Studies

Three experimental studies were cited in Chapter 2 [66, 82, 63]. These studies are discussed further in the following sub-section.

6.1.1. Optimistic CC vs Locking

Kersten and Tebra presented the results of a study comparing serial validation and locking [63] using the PLAIN database handler [62, 120, 109] and the Troll interface [121, 64]. Transactions are written using the PLAIN programming language in a Pascal-like syntax with extensions to handle operations on relations. Tuples in the relation were stored within a tuple area and could be accessed by key via a keyset organised as a B-Tree [8, 35]. Granularity within this system was at the page level and updates were written to shadow pages before commit. The prototype system was implemented under UNIX, Version 7.

From this brief overview it can be seen that this prototype system resembles TDBS/C in that the approach is relational, the DML is embedded in a procedural programming language, each transaction is assigned a DBCS and runs as a separate process and the implementation runs under UNIX. The main internal difference is in the use of B-trees in place of hash tables for the key index organisation.

The prototype system of Kersten and Tebra was used to compare the performance of SV and locking, where the locking protocol was based on share and exclusive locks, corresponding to 2PLU in this study. Two architectures were used to implement a semaphore mechanism (which was not available under UNIX). The first, call "decentralized", used a shared file to hold CC information and the creation and release of a lock file to simulate a semaphore. The disadvantage of this technique is the high overhead in terms of file system activity which was noted in Section 4.3.1 of this study. The second technique, called "centralized", was based on the UNIX fork and lseek primitives, the atomicity of the lseek primitive and a shared file descriptor. The CC information was held in primary memory by the monitor process (corresponding to the scheduler in this study) and communication between a transaction process and the monitor process was by means of a shared pipe and interrupts (signals). TDBS/C uses a different technique based on the UNIX flock primitive and a shared file for the CC information.

The database used by Kersten and Tebra was based on one relation of 32,000 tuples. Four disk accesses were required to obtain the tuple value. The transaction, using uniform access, consisted of three tuple searches and one non key attribute update. The number of transactions was varied from one to ten. At all levels, SV outperformed 2PLU using both the "centralized" and "decentralized" techniques described above. The results for average transaction length (in seconds) at number of transactions = 10 are presented in Table 6.1. The large difference in the "decentralized" 2PLU results from the high file system overhead in semaphore simulation.

Decentralized		Centralized	
SV	2PLU	SV	2PLU
6.13	72.26	5.79	6.50

Table 6.1: Kersten and Tebra: SV vs 2PLU, ET (seconds)

The nearest corresponding results from this study are presented in Table 6.2.

	GRAN = 1					GRAN = 10				
T_REQ	SV	2PLU	A2PLU	2PLE	PRE	SV	2PLU	A2PLU	2PLE	PRE
1	12.812	14.123	14.278	13.191	13.299	19.506	18.910	15.625	13.630	13.757
2	14.624	16.924	16.667	14.508	14.590	29.562	44.187	24.050	16.832	16.545
5	25.198	39.371	25.102	19.336	20.298	62.919	499.632	40.739	55.444	23.841
10	60.161	211.973	46.262	42.696	36.099	122.326	---	---	---	29.290

Table 6.2: This Study: SV vs Locking, ET (seconds)

Inspection of Table 6.2 shows that SV does outperform 2PLU. A2PLU outperforms SV at higher probabilities of conflict, (GRAN = 1, T_REQ = 10) and GRAN = 10. In most cases however, both PRE and 2PLE outperform SV and where SV performs better than PRE and 2PLE, the differences are small.

The basic assumption of SV is that conflicts are rare, the optimistic assumption. In Kersten and Tebra's study, each transaction accesses 4 tuples (three retrievals and one update) out of 32,000 so that conflict would be expected to be low. The actual granularity is unfortunately not stated. The authors state that SV is investigated as a suitable CCA for query dominated environments, justifying the choice of transaction. If PC is low then few or no restarts might be assumed and the result expressed in average ET would hold for T otherwise restart effects must be taken into consideration.

In this study, SV was also found to perform better than 2PLU at most factor levels but the performance of the latter could be improved by using an adaptive restart technique. As PC increased, the other locking techniques, 2PLE and PRE, performed better that SV. It should be noted that the environment in this study was less favourable to SV with T_UPD = 1.0.

In conclusion, the findings here tend to support those of Kersten and Tebra, that is, that SV generally outperforms 2PLU. This was not the case for 2PLE and PRE so that agreement with Kersten and Tebra's results is qualified. It was also noted that with a suitable R_DLY value as in A2PLU, the performance of 2PLU could be increased over that of SV

6.1.2. Other Experimental Studies

The study by Peinl and Reuter [82] compared SV, 2PLU and two further locking protocols not relevant to this study. The database investigated used the network data model. Quantative measures were made of (a) the number and length of blocking situations and (b) the overhead incurred by restarts due to deadlocks or validation conflicts. In place of implementing each CCA in the database, logical page reference strings for six applications were recorded over a period of time, giving an internal audit trail of DBMS activities for a real database. Between 10,000 and 100,000 page references for six transaction mixes were obtained. The reference strings were then input to a scheduler and performance statistics recorded for each CCA under consideration. Granularity was at the page level.

Two parameters were defined for performance comparison, (a) $\bar{n}$, the average degree of effective multiprogramming, which corresponds to EMPL in this study, and (b) q, a measure of the overhead incurred by restarts in terms of re-referencing objects. Direct comparison with this study is still difficult since EMPL, where stated in this study, is an expected value and not an average measure at run time. The value measured by q is expressed in this study by a combination of RC and the efficiency measure. An overview of the main results from Pienl and Reuter's study and important points are presented here.

The main conclusion was:

"Optimistic schemes achieve excellent performance figures in all cases, but this evaluation does heavily depend on the (idealized) processor model... This is especially remarkable since the applications have a comparatively high amount of update transactions... A more realistic comparison must, of course, be based on response time and throughput."

The "(idealized) processor model" assumes that an executing transaction has no effect on any other executing transaction. That is, a high degree of parallelism rather than interleaved processing. Mixed workloads of long and short transactions were examined and were found to lead to problems in the SV implementation with long transactions never validating. A technique called forward oriented OCC was implemented where there are three possible policies on detection of conflict, (1) abort and restart the validating transaction, (2) kill all conflicting transactions and (3) defer validation until

conflicting transactions have finished. Best results were found for policy (2). Policy (1) is the same as backward oriented OCC which was used in this study.

Despite these differences, the results of this experimental study also show that SV outperforms 2PLU. Peinl and Reuter found that SV had a higher number of restarts than 2PLU. This was found in this simulation but the opposite was found in the prototype study. The importance of the effects and costs of lock upgrades was noted as a major cause of deadlocks and is confirmed in this study. By comparing results in Chapter 4 for 2PLE and 2PLU, the effects of lock upgrades can be measured both in terms of T and RC. The effect on ET is shown in Appendix 3. These results are important since similar conclusions are reached using radically different experimental frameworks.

The study by Kohler, Wilner and Stankovic [66] compared 2PLE, 2PLU and hierarchical locking policies at file level (FLL), page level (PLL) and mixed file/page level (MLL). The testbed system, CARAT [47] was based on the database system of Figure 2.1 and consisted of a number of communicating processes. Transactions were written in Fortran with embedded DML and accessed a network database system call WAND [49]. Each transaction process communicated with a database server process which forwarded requests to a file server process. The latter managed the database and journal files and allocated WAND server processes to the transaction request. In this organisation, it was found that a very large number of process to process messages were required, the system became CPU bound and the file server proved to be the main system bottleneck which in turn limited the potential concurrency of the system. The system ran on a DEC VAX 11/780 under VMS with 4MB of primary memory and two RM03 disks.

The two granularity levels investigated were (1) FLL, where the entire database is a granule, and (2) PLL. Transactions were classed as small, accessing 5 records, and large, accessing 102 records on average. Accesses were read only and records were selected uniformly with replacement. Performance comparisons were based on T and ET for small, large and combined small/large transaction mixes. Percentage values for the number of aborted transactions were also given. The first experiment investigated the effects of file size (number of pages) on T. The ranked order of the policies was FLL, MLL and PLL. Effectively this is saying that serialised execution performed better than concurrent execution. Overall, T decreased by 9% for FLL and increased by 12% for MLL and by 31% for PLL as

the number of pages increased. These results were ascribed to the design of CARAT, the fact that it was CPU bound, and the bottleneck caused by the file server process. The experiment does show that as the number of pages increases, conflict decreases, hence the improvement in T under MLL and PLL. A second experiment investigated the addition of lock wait times and think times to the transaction while a third experiment examined concurrency and locking. The conclusion was that the "best" locking policy depended on system and transaction characteristics. FLL was found to be best in the basic system and when the lock wait time was added, PLL best when the internal think time was added and MLL best when both the think time and lock wait time were added. It was also noted that small transactions ran so fast that they would be best running sequentially since the overhead in setting locks in this system was expensive.

In comparison, TDBS/C had higher IO overhead and no explicitly controlled lock wait time or think time. Lock overheads were found to increase costs by a factor 2 for ET and a factor 3 for IO. Kohler et al. compared T with a baseline, the no concurrency control case, for a read only situation (no conflict) and the reduction in T estimated at about 12%. In Experiment 2 of this study, the no conflict situation occurs at LC = 0%. Reduction in T was about 6%, 7% and 9% for PRE, 2PLE and 2PLU respectively. CPU overheads were neglible and IO overheads showed an increase of around 20%. The results for Experiment 2, LC = 100% might be compared with FLL in Kohler et al. The reduction in T is about 22%, 27% and 89% for PRE, 2PLE and 2PLU respectively in this study compared with a best case of 11% and worst case of 54% reduction in Kohler et al. Since such values are system dependent, comparison remains difficult. More important is the development of a set of methodologies arising from these studies. One important point raised in Kohler et al. is the question of scaling up prototype results to larger real-life systems.

6.1.3. Conclusions from the Experimental Studies

Of the three experimental studies examined, that of Kersten and Tebra was closest to TDBS/C in architecture and environment. In general, the results of this study agreed with those of Kersten and Tebra in that SV was found to outperform 2PLU. However, in this study, other locking policies, namely PRE and 2PLE performed better that SV. The result is of interest since in this study, update only

transactions were used. The second study considered here, that of Peinl and Reuter, also agrees with the result that SV outperforms 2PLU, even in an environment containing a large number of update transactions. Peinl and Reuter also examined mixed small/large transaction loads and noted some problems caused by large transactions failing to validate under SV. The problem of lock upgrades and the effect on CCA performance was also noted. Mixed loads were also studied by Kohler et al. Such a study under TDBS/C must be left for future research.

The results discussed in this section are interesting in that some similar conclusions are reached despite the radical differences in experimental frameworks and performance metrics. In terms of comparative CCA performance, this study has examined a wider range of CCAs in greater detail.

6.2. Simulation Studies

In this section, the results and conclusions of other simulation studies are compared with the findings in this study. Some studies of general interest are presented first, followed by a more detailed comparison with a group of related studies.

6.2.1. Other Studies I

Madelaine [71] compared BTO to 2PLE in a model of a database machine (SABRE). The model assumed infinite resources for some components in the system. The deadlock resolution for the 2PLE was the "WAIT-DIE" scheme [90]. The general findings were that locking performed better than BTO in high conflict situations between small transactions. BTO performed better if the average number of requests per transaction is large or if the number of small transactions increases. The average response time was plotted against the probability of conflict (PC) and several curves presented for different values of number of transactions in the system. The curves showed similar shapes with an asymptote at around PC = 0.75, for transactions making one request. This result can be compared with Experiment 2 in Chapter 5. The ET for 2PLE in Table A4.18 does increase sharply between LC = 80% and LC = 100%, but that for BTO is in fact decreasing. From Table 5.17, BTO outperformed 2PLE in Experiment 1 at GRAN = 1 and in Experiment 2 as T_REQ increased. In the higher conflict situations of Experiment 1

where GRAN = 10, BTO outperformed 2PLE at all T_REQ values. The differences may lie in the 2PLE deadlock resolution techniques, with greater blocking occurring in this study or in other modelling assumptions. In the prototype experiments, 2PLE outperformed BTO at all factor levels.

Lin and Nolte in one study [69] compared BTO, 2PL and a multi-version timestamp CCA (MVT). They concluded that when transaction size was small (4 accesses), BTO outperformed 2PL and when transaction size was large (>16 accesses), 2PL outperformed BTO. BTO was found to favour read only transactions. When PC is high, waiting is better than restarting. Costs in the system were expressed as one overhead, the communications delay, and waiting for CPU and IO resources was not simulated. Further, the technique used for 2PL used share locks for read transactions but set exclusive locks at read request time for items to be updated. Lock upgrade costs were therefore not taken into account. Both the prototype and simulation results here have shown that lock upgrades decrease performance. In our simulation results, the performance of BTO is better than 2PLU throughout Experiment 1 and improves over that of 2PLE as PC increases (by increasing T_REQ). In Experiment 2, both 2PLE and 2PLU outperform BTO up to LC = 50%. These results seem to contradict in part those of Lin and Nolte. In this simulation, the blocking for 2PLE may reduce EMPL more than the restarts do for BTO but as PC/LC increases the effect of blocking is to further increase PC since the number of unblocked granules is decreasing.

Other studies have investigated distributed algorithms [46, 110, 111].

6.2.2. Other Studies II

A set of studies based on a similar logical queueing model is now discussed. This model (Figure 2.7) has also been adopted in this study. The model is first presented by Ries [85, 87, 86] who used it to investigate the effects of granularity in both centralised and distributed DB systems. In the centralised case, the CCAs investigated were PRE (as defined in this study) and incremental static locking which is labelled IPRE to distinguish it from PRE.

For PRE, balancing the lower lock overheads at coarser granularity against higher lock overheads at finer granularity, an optimum value of 10 granules resulted. If almost all transactions are small and

exhibit random access patterns then a finer granularity is recommended. For a large number of transactions accessing a large proportion of the database, coarser granularity is again preferred. For sequential access patterns, coarser granularity was also found to be the most effective. Quantitive values suggested for coarse and fine are 10 and 100 granules respectively.

The results for IPRE were similar to those for PRE. The advantage of IPRE of only re-requesting denied locks was more than offset by the cost of restarting deadlocked transactions. Finer granularity benifited IPRE but overall, PRE performed better. Ries suggested that CCA information could be stored in primary memory for greater efficiency. Even in our study where the CCA information is held in secondary memory, our results aggree with those of Ries. Since locks are exclusive under PRE and IPRE, the proportion of update transactions has no effect. The results of Ries are also confirmed in the prototype experiment, where nearest equivalent to IPRE is 2PLE. At lower T_REQ and finer granularity, 2PLE outperforms PRE. As PC rises, 2PLE is less efficient in terms of resource usage due to restarts.

Carey extended Ries's queueing model and performed a comparative study of seven CCAs [23]. In addition to the five in this study (PRE, 2PLE, 2PLU, BTO, SV), two additional CCAs were studied: WD (2PLU with "WAIT-DIE" deadlock prevention [90] and TWW (BTO with the Thomas Write Rule [112])). Under the no blind writes assumption, TWW performed identically to BTO.

In this study, both the logical and physical queueing models used by Carey were adopted as the basis for the simulation implementation. Since there are a number of papers based on this model this provides a reasonable framework for comparison. The simulation here was used to model the prototype testbed and these results compared in Chapter 5.

The relevant simulation results from Experiment 1 are now compared with the corresponding results of Carey. Firstly the differences in parameter values and factor levels are presented in Tables 6.3 and 6.4, followed by a comparison of quantitive results for T and RC in Tables 6.5 and 6.6. The comparison is made for PRE, 2PLE, 2PLU, BTO and SV for fixed blocking and restarting delays. Carey's results are presented in brackets. Finally, Table 6.7 summarises the comparison by ranking the CCAs in order of decreasing T.

Factors	Factor Levels	
DATABASE:		
DB_SZ	1000 tuples	(10000 objects)
DB_PG	100 pages	(-)
GRAN	1,10 pages/granule	(100, 10 granules)
NR	1	(1)
TRANS:		
NT	10	(10)
T_REQ	1,2,5,10	(1,2,5,10 (mean values))
T_ACC	random	(random)
T_UPD	1.0	(0.5)
CC:		
CCA	PRE,2PLE,2PLU,BTO,SV	(PRE,2PLE,2PLU,BTO,SV)
MMPL	10	(10)
B_DLY	2 seconds	(1 second)
R_DLY	2 seconds	(1 second)

Table 6.3: Factor Level Comparison: This Study/(Carey)

Parameter	Value (Milliseconds)	
CC_IO	2	(0)
CC_CPU	3	(1)
TR_IO	20	(35)
TR_CPU	25	(10)
STGM	20	(20)

Table 6.4: Parameter Value Comparison: This Study/(Carey)

The differences in factor levels occur in T_REQ where Carey used stochastic values, in T_UPD and in the delay values B_DLY and R_DLY. The parameter value differences are mainly in the CC overheads where Carey assumes the CCA tables to be resident in primary memory. The transaction costs, TR_IO + TR_CPU, are the same. The allocation of values to TR_IO and TR_CPU was based on the SU results of Chapter 3.

Table 6.5 below gives a comparison of T values and Table 6.6, RCs for this study and the relevant figures from Carey's study which are given in brackets.

GRAN = 1 (100 Granules)										
T_REQ	PRE		2PLE		2PLU		BTO		SV	
1	17.800	(11.420)	17.760	(11.419)	16.270	(11.373)	16.650	(11.328)	18.640	(11.262)
2	9.069	(7.163)	8.769	(7.161)	7.786	(7.039)	8.390	(6.906)	9.052	(6.714)
5	3.425	(3.360)	1.976	(3.335)	1.952	(2.823)	3.077	(2.231)	2.974	(2.408)
10	1.397	(1.759)	0.481	(1.414)	0.442	(0.827)	1.074	(0.235)	0.980	(0.784)
GRAN = 10 (10 Granules)										
T_REQ	PRE		2PLE		2PLU		BTO		SV	
1	15.770	(11.421)	8.456	(11.420)	6.613	(10.971)	14.560	(10.648)	16.572	(10.314)
2	6.775	(7.161)	1.685	(7.096)	1.521	(5.974)	6.690	(5.119)	7.239	(5.231)
5	2.173	(3.028)	0.234	(2.097)	0.169	(0.946)	1.687	(0.405)	2.005	(1.200)
10	1.045	(1.415)	0.032	(0.432)	0.024	(0.074)	0.458	(0.004)	0.649	(0.355)

Table 6.5: Throughput Comparison, This Study/(Carey)

GRAN = 1 (100 Granules)										
T_REQ	PRE		2PLE		2PLU		BTO		SV	
1	0	(0)	0	(0)	52	(109)	55	(224)	76	(378)
2	0	(0)	0	(3)	63	(247)	67	(508)	127	(879)
5	0	(0)	12	(66)	63	(907)	114	(1910)	193	(1617)
10	0	(0)	40	(769)	62	(1543)	193	(2482)	200	(1612)
GRAN = 10 (10 Granules)										
T_REQ	PRE		2PLE		2PLU		BTO		SV	
1	0	(0)	0	(0)	15	(1096)	268	(1909)	311	(2753)
2	0	(0)	13	(242)	131	(2366)	299	(4071)	348	(3844)
5	0	(0)	65	(3123)	117	(4061)	359	(4940)	346	(3651)
10	0	(0)	81	(4223)	118	(2816)	383	(2848)	263	(2293)

Table 6.6: Restart Count Comparison, This Study/(Carey)

Comparing results for BTO and SV, which use restarting, T is higher and RC lower in this study. For the locking based methods, as PC increases, T tends to be lower despite the lower RC. Carey's study did not provide figures for blocking counts (BC) so it is not possible to compare these directly. Carey concluded that locking based CCAs outperformed resrart base CCAs. In this study, with the exception of PRE, the opposite is true.

In both studies 2PLE peformed better than 2PLU, and PRE generally performed better than 2PLE. In the majority of cases, SV performed better than BTO. Carey found that 2PLE outperformed both BTO and SV whereas the results of this study are less definite. In this study, the restart based CCAs perform better probably due to the increased R_DLY whereas the locking based CCAs perform worse with an increased B_DLY. The result of the adaptive restart from Chapters 4 and 5 support the argument that increased R_DLY favours restart based transactions. Comparison of the two studies also

indicates that the smaller B_DLY improves T for the locking based CCAs. Recall that PRE and 2PLE will be unaffected by the T_UPD value although the transactions in this study will have extra overhead from TR_IO caused by the write.

GRAN = 1 (100 Granules)										
T_REQ	This Study					Carey				
1	SV	PRE	2PLE	BTO	2PLU	PRE	2PLE	2PLU	BTO	SV
2	PRE	SV	2PLE	BTO	2PLU	PRE	2PLE	2PLU	SV	BTO
5	PRE	BTO	SV	2PLE	2PLU	PRE	2PLE	2PLU	SV	BTO
10	PRE	BTO	SV	2PLE	2PLU	PRE	2PLE	2PLU	SV	BTO
GRAN = 10 (10 Granules)										
T_REQ	This Study					Carey				
1	SV	PRE	BTO	2PLE	2PLU	PRE	2PLE	2PLU	BTO	SV
2	SV	PRE	BTO	2PLE	2PLU	PRE	2PLE	2PLU	SV	BTO
5	PRE	SV	BTO	2PLE	2PLU	PRE	2PLE	SV	2PLU	BTO
10	PRE	SV	BTO	2PLE	2PLU	PRE	2PLE	SV	2PLU	BTO

Table 6.7: CCA Rank Order by Descending T, This Study/Carey

The CCAs from both studies are given in Table 6.7, ranked in order of decreasing T. In both studies, the relative rank order of the three locking CCAs is the same: PRE, 2PLE, 2PLU. For BTO and SV, the differences are greater. Table 6.6 shows that in Carey's study the RC for BTO and SV is higher, in some cases considerably. In this study, both B_DLY and R_DLY are twice the corresponding value in Carey, thus removing blocked and restarted transactions from the system for longer periods and implying a lower EMPL. CCAs based solely on restarts, BTO and SV, will then have fewer conflicts. This seems to be confirmed by the higher T under BTO and SV in this study. Examining ET values in Appendix 4, Tables A4.9 and A4.11 also shows that in most cases, ET < R_DLY for transactions under BTO and SV.

With an increased R_DLY, CCAs based on locking will incur higher cost per block, increasing ET and consequently reducing T. The values in Table 6.5 confirms this. Direct comparison on blocking patterns is not possible since BC was also not provided by Carey. Increasing B_DLY therfore seems to reduce the performance of locking based CCAs but increasing the value of R_DLY seems to improve the performance of restart based CCAs.

The difference in T_UPD does not affect PRE and 2PLE so that differences for PRE and 2PLE can be ascribed to the effects of B_DLY. The increase in update transactions will produce greater

conflict and a further reduction in EMPL under 2PLU, BTO and SV. The RC for Carey's results for 10 granules suggest thrashing and cyclic restarts were occurring. Carey notes this in the case of BTO at higher T_REQ values as an algorithm anomaly.

The main outcome of this comparison is in demonstrating the diffrence in results under different assumptions and parameter values using basically the same model. The major difference in this study being the longer delay times and setting all transactions to be update transactions. The CC overheads were set to higher values to reflect the use of secondary memory for the CC information in the prototype. In addition, this study investigated the effects of adaptive restarts for 2PLU and BTO. Again there is no "best" CCA although the very conservative approach under PRE gave good overall performance in both studies. The relative movement under different factors and assumptions is of more importance in understanding CCA performance.

The model used by Carey was further extended in a study by Agrawal [4,3]. The extensions consisted of the addition of think times in the object access and a delay time in the restart. See Figure 2.7. Agrawal et al. examined the performance of two versions of 2PLU and SV under different assumptions concerning resources, modelling of restarts (fake/non fake) and write acquisition policies. Resources were modelled from being finite (1 CPU, 2 disks) to infinite. Here we are mainly concerned with the results for the finite resource case. The database comprised 1000 objects, transactions had an 8 page readset (maximum 12, minimum 4) with T_UPD = 0.25 and transaction overheads (TR_CPU + CC_CPU + TR_IO + CC_IO) set at 50 milliseconds. A variation of 2PLU used immediate restarts to resolve conflicts [107], where instead of blocking the transaction is restarted after a delay of approximately ET (the delay is adaptive). This algorithm is referred to here as I2PLU. The other version corresponds to 2PLU as defined in this study. Both the 2PLU and SV algorithms did not have restart delays.

The results of Agrawal et al. showed that for the finite resource case, T peaked at MPL = 25 for 2PLU and at MPL = 10 for I2PLU and SV. Maximum T was obtained with 2PLU and I2PLU performed better or as well as SV, restarts being more expensive for the latter. The restart mechanism of I2PLU was noted as a crude mechanism for limiting the MPL when restarts become frequent. It was also suggested that a restart delay would benefit the performance of 2PLU and SV. The blocking CCA

gave the best performance and was considered best for situations with medium to high levels of resource utilisation. Restart policies were considered better for situations where a large amount of wasted resources could be tolerated and a large number of transactions were available to execute, in which case SV was the best choice. This situation could occur in a database machine with a large number of CPUs and disks or, in a primarily interactive environment where transactions have large think times. However doubt was voiced as to whether this reflected real systems in which case blocking was the solution. It was also noted that MPL should be carefully controlled to avoid thrashing behaviour and an adaptive algorithm to control MPL was proposed.

While concentrating on the effects of resource assumptions (finite/infinite), the study also examined assumptions about restarts and write lock acquisition, again using 2PLU, I2PLU and SV. Restart based CCAs were shown to be more sensitive to the fake restart assumption, non fake restarts giving lower T, although there was little difference at MPL < 25. The write lock acquisition policy had a significant effect on T. Again 2PLU emerged as the best CCA.

In this simulation study, there is no equivalent to I2PLU, ABTO being the nearest. The 2PLU version of Agrawal et al. corresponds to <A2PLU (small restart delay) and SV is the same CCA but used here with a fixed length restart. Examining Tables 5.16 and 5.17 reveals that overall, SV performed best. In Experiment 1, where restarts were fake, <A2PLU performed worst and in Experiment 2, where restarts were non fake, ABTO performed worst at LC = 50%, 80%. Although we should beware of making direct comparisons, the results suggest that the performance of SV is considerably enhanced by a restart delay and even when compared to BTO in Table 5.17, still performs better. In a comparison of SV, BTO and 2PLU, where each has the same fixed R_DLY, the conclusions from this study are still that overall SV performs best and 2PLU worst. One other difference between the two studies is in T_UPD. The higher update rate in this study has possibly led to a higher restart rate and greater reduction in EMPL, favouring the restart based CCAs. On the write acquisition assumption, the results here show that it can have considerable effect on CCA performance. Differences between Experiments 1 and 2 here also support the conclusion about restarts in Agrawal et al. Even with a similar model, different assumptions reflecting our simulation model of the prototype have led to different conclusions. By contrasting these differences, the behaviour of the CCAs under varying

modelling assumptions is explained.

6.3. An Analytic Study

In the previous sections, the results of this study have been compared with other related experimental and simulation studies. We conclude by comparing the results with those from an analytic study of locking performance [107]. This is a compilation of previous works [105, 106, 103] in turn derived from [104].

The model used, considered the database as a set of granules, a number of transactions making requests to a scheduler (the CCA) and was represented as a flow diagram. A set of equations was derived describing the behaviour of the system using steady state average values. It is also noted that within the system, EMPL cannot be known exactly. Two factors influence locking performance:

1) DC, data contention over access to data granules

2) RC*, resource contention over access to OS resources

RC is mainly factored out in Tay's study but is reintroduced from time to time to make statements about real systems.

When too many transactions access some shared data, DC-thrashing is said to occur. The DC-thrashing point is where the two forces affecting T balance out. These forces are (1) increasing the number of transactions which increases T and (2) the resultant increase in conflict which reduces T. The two main parameters of the model were "k", the number of lock requests per transaction and the load "λ", defined as "N/D", where "N" is the number of transactions and "D" the number of granules. Two techniques are used to resolve conflicts:

1) immediate restart: the "no-waiting" case

2) blocking: the "waiting" case

---

* RC in this section is used to refer to resource contention and NOT restart counts as elsewhere in this document.

The DC-workload was defined as $k^2\lambda$ and a rule of thumb gave $k^2\lambda < 6$ for the "no-waiting" case and $k^2\lambda < 1.5$ for the "waiting" case. Two classes of transaction were defined, query and update, which set readlocks and writelocks respectively. Uniform access was assumed. Different length transactions were studied as well as mixes of query and update transactions. Both static and dynamic locking were considered.

Before comparing results, the DC-workloads for this study are presented in Table 6.8. In Experiment 1, k is T_REQ. In Experiment 2, D is decreasing.

T_REQ	1	2	5	10
GRAN = 1	0.1	0.4	2.5	10.0
GRAN = 10	1.0	4.0	25.0	100.0
LC	20%	50%	80%	100%
	1.1	1.7	3.3	10.0

Table 6.8: Experiments 1 and 2: DC-workload

In both the prototype and simulation runs in this study, no distinction in DC/RC was made. This introduces a new factor into the performance evaluation discussion, which may prove useful in explaining CCA behaviour. RC and RC-thrashing are dependent on EMPL and/or the available resources. Since in this study the resources were fixed and limited, RC will be interpreted as depending on EMPL.

Comparison of the "waiting" and "no-waiting" cases, that is blocking and restarts, showed that as N increased, restarts performed better despite a high restart rate. If the cost of restarts is high then blocking performs better. At low levels of RC, restarts performed better. The "waiting" case in our study is 2PLE but there is no precise equivalent to the "no-waiting" case. The nearest equivalent is BTO.

For 2PLE, DC-thrashing occurs at (GRAN = 1, T_REQ = 5,10), (GRAN = 10, T_REQ = 2,5,10) and LC > 20%. For BTO the corresponding values are (GRAN = 1, T_REQ = 10), (GRAN = 10, T_REQ = 5,10) and LC = 100%. In the prototype, restarts were found to perform well under ABTO which gave a greater reduction in EMPL, hence RC, supporting Tay's result that restarts performed better at low RC. Further, the restarted transaction costs are lower under ABTO (Appendix 3). Beyond the DC-thrashing limits for 2PLE, the restart costs for ABTO were lower than those for 2PLE. The

result depends on R_DLY. Tay suggested an average conflict avoidance delay of ET/2. In the prototype 2PLE/BTO comparison, the restart costs for BTO, where R_DLY < ET/2, are more expensive than those for ABTO. The simulation results also agree with Tay's assertions (recall that BTO here has a large R_DLY compared to ET) with BTO performing better above the DC-thrashing point for 2PLE.

Comparing atomic static locking with dynamic locking, that is PRE with 2PLE in this study, Tay predicted PRE would perform better when 2PLE starts to thrash at some level of N. Again RC will determine the cross-over point between T for PRE and 2PLE. In our simulation study, PRE performed better that 2PLE throughout. For 2PLE in both the simulation and prototype, restarts occur beyond the DC-thrashing points predicted by Tay. In the prototype, 2PLE performed slightly better where PC is lower, that is under the light DC-workload condition of Tay's study.

The effects of parametric changes in the model for N, k, and D were examined. Increasing N led to more restarts which would be expected since PC would be expected to rise. Increasing k was shown to reduce T and increase the restart costs, presenting an argument in favour of short transactions. Restart costs in Appendices 3 and 4 agree with this result. As the granularity D increased, a point is reached where finer granularity makes little difference to T. A curve of T vs D was presented, firstly decreasing at coarse granularity due to excessive DC, then increasing and finally decreasing again, with excessive RC taking place. It was noted that the choice of system parameters would define a window on this granularity curve. In this study only two granularities were examined since the subject has been already investigated in detail [85, 87, 86, 23, 25]. The finer granularity gave better performance.

Tay concluded his study with a comparison with other analytic and simulation studies, including those of Carey and Agrawal. Carey concluded that blocking was preferable to restarts in most situations, even with heavy DC-workloads. The apparent contradiction with Tay's results was explained by possible high RC in Carey's study. A second point noted is the effect of R_DLY on restarts where Tay noted that in Carey's study the conflict avoidance delay was less than the response time. This is confirmed by results in this study for the cases R_DLY < ET and R_DLY = ET in the prototype experiments (BTO and ABTO respectively). Carey ascribed his result to the correlation between a high restart count and low T. In one of his experiments (10 granules, mean value of 10 granules accessed) 2PLE had a higher restart count than BTO but still better T. Carey explained this by suggesting that

under 2PLE, transactions restarted earlier in their lifetimes. Tay notes that in such high conflict situations, the number of blocks will also rise. This can be observed in the simulation results in Appendix 4. The high block count will reduce EMPL and hence RC, confirming Tay's explanation.

Despite the fact that, as in Carey's study, the results presented here lie partly in the thrashing region as defined by Tay, there is general agreement on a number of points. Tay also contributes a number of ideas which are useful in interpreting CCA performance results.

6.4. Conclusions

In comparing the results here with other studies, in some cases the results agree and in others, where the outcome apparently contradicts results of previous studies, examination of different assumptions and parameter settings has accounted for such differences. In the prototype and other experimental studies, SV was found to perform well under a wide range of assumptions, including high incidence of update transactions. This is perhaps surprising in view of the basic assumption for SV to perform well, namely that conflicts are rare. The results here agreed with those of Kersten and Tebra, that SV outperformed 2PLU but with the qualification that other locking policies, PRE and 2PLE, outperformed SV in some cases. That SV outperforms 2PLU also agrees with the result of Peinl and Reuter who used a radically different experimental framework. Results from this prototype were also in agreement with some conclusions of Kohler et al. which examined only factors affecting locking performance.

The results from the simulation studies under consideration were less clearly in agreement. Again, contrasting assumptions and parameter values led to the conclusion that the relationship between restart and blocking delay times, R_DLY and B_DLY, and ET is an important factor in determining CCA performance. Comparison of values from Carey's study show similar trends as T_REQ increased but gave different relative performance orders for the CCAs. From the adaptive restart delay results in Appendix 4, Tables A4.20 to A4.25, the effect of R_DLY on T, RC (restart count), BC and ET can be seen in quantitive terms. This comparison and that with the study by Agrawal et al. suggest that the model is sensitive to changes in assumptions and parameter values. Lock upgrades were shown to have

a high cost, which has not been considered in some previous studies, and the effects of restart assumptions were noted.

Finally, comparison with the analytic model of Tay introduced some new concepts to explain CCA performance: the separation of data and resource contention considerations. Results in this study confirmed some predictions by Tay, in particular the behavior of locking algorithms in DC-thrashing regions. The results suggest that further detailed comparison with the analytic model may well prove useful.

From most of the above studies, PRE remains a good "best" choice of CCA where it is possible to determine transaction requirements in advance. Otherwise, our results suggest a restart based technique with adaptive restart delay will give good performance. The comparison highlighted important points about the choice of parameter values for the model and explained apparently contradictory results. Each of the studies has by necessity had to restrtict the number of factor levels studied which in turn imposes a window on the performance space. Tay notes this as a disadvantage of simulation (and by extension, prototyping). The advantage of analytic models is the relative ease with which parametric changes can be studied. The great difficulty is in interpreting an analytic model in terms of a real life database.

6.5. Chapter Summary

The prototype results were compared with those of the few other experimental studies performed to date. The conclusions from these experimental studies agreed. Next, the simulation results were compared at a general level with two previous simulation studies. A more detailed comparison was then undertaken with a set of simulations studies based on the same model and differences and similarities explored. Lastly a comparison with an analytic study was made and points of agreement noted. From these comparisons, a better understanding of the mechanisms of CCAs in different contexts was gained which helped further clarify the results of previous chapters.

Chapter 7

SUMMARY AND CONCLUSIONS

In this final chapter, the results and conclusions of this study are summarised, reviewed and placed in the wider context of other studies concerning database performance. The limitations of this study are noted and points made for future research.

7.1. A Wider Context

In examining database technology, a wide range of techniques, both hardware and software, has been implemented in order to improve performance. Hardware solutions have included database machines, for example the Bitton Lee IDM/500*, or specialised hardware as in ICL CAFS [6, 19]. The software approach is the more usual and includes many systems currently operational. This approach includes optimisation of database sub-parts and [102, 70] and extension of existing programming language features [97, 88, 28]. Between these two approaches lies the "firmware" approach involving operating system extensions [101, 102] or bypassing the file system and providing direct access to the "raw" disk as in WiSS (Wisconsin Storage System) [27] implemented under UNIX. In this context, TDBC/C running in single-user mode represents a software implemented system running on a dedicated machine (software database machine).

Although optimal conditions within the database organisation were created (Chapter 3), no performance improvements have been carried out. Similarly no external load, representing other non-DBMS users of the O/S was added [17]. A realistic estimation of such loads is at best extremely difficult and system dependent. Again, in the wider context, experimental benchmarks of database systems are few. Performance measurements of the IDM 500 database machine and "university" and "commercial" INGRES appeared in [17] (the single user case) and [18] (the multi-user case). The two

---

* Bitton Lee Inc., Los Gatos, California, USA.

INGRES versions were compared with WiSS in [27]; the performance of INGRES and ORACLE compared in [80]. A more descriptive evaluation of INGRES, DREAM/CS5 (a Swedish product, written in C running under UNIX), PROGRESS (a relational based system) and UNIPLEX-II/DBMS (C/UNIX version) was given in [83]. Other evaluations have also been more descriptive than quantitive [22, 108]. Evaluations of commercial products are no doubt also rare because of commercial sensitivities. New technologies will change the order of magnitude of DB costs, but not the fundamental components of the database system, including CCAs, hence the methodology used here will still be valid.

7.2. Review of This Study

The component parts of this study, the prototype system TDBS/C, the CCAs, the prototype and simulation experiments are reviewed. Results and conclusions are summarised.

7.2.1. TDBS/C Revisited

One disadvantage of prototyping is that the results apply to a particular system running under a particular O/S. As far as possible, the prototype was benchmarked to minimise or identify bottlenecks or anomalies (see Chapter 3 and Appendix 5). For example, one feature, that of embedding procedure calls to a DBCS was identified as having a substantial effect on performance; using tuple access there is one dbcs call/access whereas only one dbcs call is made using relational access. Although only one tuple access method was used in the prototype experiments, the benchmark results in Chapter 3 indicate that the results can be generalised to include the other DML operations. Updates to the index (PINDEX) were not dealt with since the test transactions updated the data only. Recall that, in these experiments, each index page holds information on ten data pages. Two solutions are possible: (1) that the data granule includes the corresponding portion of the index or (2) hierarchichal granularities can be implemented.

The MPL of the O/S was also not considered in detail. A preliminary experiment indicated that costs were linear up to 25 concurrent transactions. This limit is important in determining any potential

thrashing limit for resource contention and also in answering Kohler's question about the effects of the scaling up of results. Again preliminary experiments indicated that this had little effect on the relative performance. Tay also argues that scaling has little effect in order to justify reducing N (the MPL) and D (the number of granules) to the load λ, but in a real system, some upper bound must exist. From Table 3.8, severe limits are imposed on practical experiments simply by the costs involved. Nonetheless, in setting up a prototype system, we have shown that multiuser benchmarks are possible using relatively simple techniques. We believe that TDBS/C is representative of modular DBMS systems, and the choice of UNIX provides an O/S which is in widespread use.

7.2.2. The CCAs Revisited

The main points affecting the behaviour and performance of the CCAs are summarised. CCAs included are those considered in this study and related CCAs from other studies.

PRE The transaction requirements must be known in advance or be determined by some pre-execution analysis. Two categories are atomic and incremental static locking. Here atomic exclusive locking was used but, if the transaction read/write requirements can be determined, share and exclusive locks would prove a greater advantage. No work done by the transaction is wasted and if the CCA costs are low, this method should be efficient. With atomic static locking deadlocks do not occur but may do so under incremental static locking. All transactions are guaranteed to commit once started.

2PLE has advantages over PRE: (1) requirements need not be known in advance and (2) locks are held for a shorter period. Restarts due to deadlock may occur. Instead of blocking, the immediate restart technique of Tay may be used. For read-only transactions, the technique may be too conservative. Performance may also depend on the transaction restart selection criteria.

2PLU reduces the disadvantage of 2PLE for read-only transactions while retaining the advantage of 2PLE over PRE. In frequent update situations, restarts can also arise as a result of lock upgrades causing deadlocks. Instead of blocking, the immediate restart technique of Tay may be used. The performance was found to increase dramatically using an adaptive restart

technique.

BTO is an alternative to the locking techniques. In high conflict situations, BTO may be subject to cyclic restarts. Restarts also represent wasted work which may be an unacceptable use of resources. The performance was found to increase dramatically using an adaptive restart technique.

SV has low overhead in terms of CC since transactions are allowed to execute and are then validated. Most suitable for read-only transaction mixes.

In this study, it was also found that R_DLY $\geq$ ET and B_DLY $\leq$ ET gave better performance. At low PC there was little difference in performance. The adaptive restart delay was found to improve restart based CCAs.

7.2.3. The Experiments Revisited

Given awareness of the factors, described above, affecting CCA performance, the purpose of any experiment is to determine how each CCA behaves in a given situation described by the experimental factors and parameters representing the DB system. By comparing CCA behaviour at different factor levels, and the results of independent experiments, factors affecting each CCA can be determined. Previous, "apparently contradictory", results have differed because each study has explored a different window on a very large problem space. No overall "best" CCA has emerged although PRE is the nearest contender. The experiments in this study have provided some answers and also some questions, and indicate that further refinement is required in setting up such experiments. This study has demonstrated that quantitive results can be obtained for a real database using relatively simple techniques. It also indicates that suitable scaling of parameter values and relative costs in a simulation model would realistically reflect the performance of real systems.

7.2.4. Limitations of This Study

One of the limiting factors is the cost of the prototype experiments. Galler notes the cost advantage of analysis over (one) simulation as being in the order of seconds of computing time compared to minutes [46]. For a more comprehensive set of simulation experiments, the cost is probably measured in hours and for prototype studies in tens of hours (or more). The disadvantage of analytical techniques is the difficulty in extending one model to encompass the range of CCAs studied here. Conclusions from simulation studies, while useful, are incomplete. The next stage in performance studies is either to test prototype systems or to increase the sophistication of the analytical and simulation models.

In previous studies, little difference between CCAs was noted at low PCs. To increase PC, T_UPD was set to 1.0 in this study. This has a particularly negative effect on 2PLU but does allow the effect of restarts due to lock upgrades to be directly measured. Mixed transaction loads (large/small) were studied elsewhere but not here. Both different levels of T_UPD and mixed loads should be subjects of future research.

In the prototype experiments, BC was not measured directly as it was in the simulation experiments. However, the effects of blocking were filtered out in the prototype by using iET as a measure. This, for example, gives a more realistic comparison of efficiency measures than using ET where B_DLY is included as useful work. A more precise measurement of BC and average length of blocking delay/ transaction would have been more useful.

In the simulation experiments, while useful for a first order approximation, fixed overhead costs are probably too simplistic. The resultant use of ET only, meant that B_DLY was included in the efficiency measure in these experiments.

In both the prototype and simulation experiments, we extended the performance metrics beyond T and ET to provide a greater level of detail than in previous studies, particularly for CPU and IO costs and overheads. One disadvantage in interpreting the overhead costs is in not being able to precisely determine EMPL. In some cases the overhead costs fall below 100%. Since the base against which this

is measured is ten concurrently executing transactions without CC, this implies that EMPL has fallen below MMPL. Blocking effects were observed to increase, and restart effects to decrease the overhead costs which thus provides a crude measure of which effect is dominant. Again this may be acceptable as a first order approximation and is used as such by Tay for atomic static locking [107]. Measurement of the average EMPL could be achieved by sampling the transaction table at regular intervals.

Finally, neither the prototype nor simulation experiments took crash recovery into account [14]. Both assumed a "perfect" system. Other CCA studies have made a similar assumption. Crash recovery costs have been studied experimentally by Kent et al. [59,60]. The two crash recovery techniques studied were shadowing and logging, using a testbed system (PREDATOR) to model the transaction processing system. PREDATOR ran on a Vax/750 under Berkeley 4.1 UNIX but the kernel was modified to bypass the file system. The implementation language was C. Two phase locking with per-item deadlock detection was used, together with page granules. The implementation was the equivalent of the System R RSS component [52] complete with crash recovery and concurrency control. (Interestingly WiSS was also modelled on RSS). The transactions and request were driven by a simulator. Kent et al. made the following observations:

1) logging was preferred when DB access was random and a dedicated recovery device was available

2) transactions aborts could strongly influence the performance of logging recovery mechanisms... This was especially important in environments where aborts were common, e.g., SV

3) the disk scheduler and IO device could affect performance in unexpected ways

4) concurrency control takes up little CPU resources

Observation (1) was qualified by a later statement that shadowing was recommended for small databases. Observation (3) is interesting since in this study the file system was assumed to be well behaved. Observation (4) was confirmed in this study by the prototype experiments. Kent et al.'s study also suggests extending the TDBS/C prototype to cope with crash recovery as future research.

7.2.5. Summary of Results

Of the CCAs examined in this study, (PRE, 2PLE, 2PLU, BTO, SV), PRE gave the best overall performance in both the prototype and simulation experiments. Under random access (Experiment 1), SV performed surprisingly well even in high conflict situations. The adaptive restart technique improved the performance of 2PLU and BTO in the prototype. Under high levels of contention for one granule (Experiment 2) in the prototype, the exclusive locking based techniques, PRE and 2PLE, performed best. The principle explanation for differences in results between the prototype and simulation experiments in this study and between this study and other studies, was shown to be in the relationship of B_DLY and R_DLY to ET.

7.3. Future Research

Firstly the range of this study should be extended to include different values for T_UPD, mixed workloads and the crash recovery mechanism. The prototype should also be extended to be a distributed system.

Secondly, and more importantly, the results suggest further refinements to both the prototype and simulation models are required. The basic performance metrics T and ET are essential for any performance study together with BC and RC. A division of CPU and IO costs into TR_CPU, TR_IO, CC_CPU and CC_IO is also required to pinpoint bottlenecks and/or anomalies in the model. Results presented here show that CC_CPU costs are low but CC_IO costs are significant. Expressing (CPU + IO) as iET and comparing values with ET can also give an idea of O/S overheads in a prototype system. A measure of the average time/transaction lost through blocking would also complement the information on resources wasted through restarts presented in Appendices 3 and 4.

For both systems, a measurement of average EMPL is required. In any discussion of blocking vs restarts, this measure will reveal if the better performance of either technique is due mainly to a relatively large reduction in EMPL. In turn, this would help determine overhead costs more accurately and might be used in place of MMPL to calculate the adaptive restart.

The values of B_DLY and R_DLY relative to ET were shown in this study to be crucial in determining relative CCA performance. These values should be determined by ET; B_DLY ≤ ET and R_DLY ≥ ET. An adaptive restart technique, such as the one user here, is recommended. Further study of the effect of these parameters on performance is required. From comparisons with other studies in Chapter 6, we feel that the values chosen for these parameters in the different studies (where present) have strongly influenced the outcome and explain the "apparently contradictory" results obtained.

In the simulation, the CPU and IO costs for both the transaction and CCA should be stochastic. Further work is also required to see if the simulation model, with changes in the values of B_DLY and R_DLY, would have more accurately predicted the outcome of the prototype experiments. In any prototype, a clearer statement of the O/S limits is required. Changing the O/S characteristics is probably easiest achieved by running the prototype on a different machine or under a different version of the operating system. Such an experiment is expensive and assumes the availability of different machine environments. A more open ended topic for research is the incorporation of Tay's ideas with the separation of resource and data contention measures, especially for a prototype system.

7.4. Concluding Remarks

At the outset of this project, the prototype experiments were designed to determine the performance of CCAs in a real database system, given the inconclusive results of previous analytic and simulation studies and the lack of comparative experimental studies. A second question, "could a suitably designed simulation experiment have predicted the prototype results?", was then posed. Given the large degrees of difference between other simulation studies and the prototype, a set of corresponding simulation experiments were designed and run. The answer to that question, borrowing from the Scottish legal system, is "not proven". Comparison and contrast with some previous studies has provided explanation for both sets of studies and pointed the way for future research. If, as Gray contends, conflict in database systems is rare [53], then it makes little difference which CCA is used. If, on the other hand, such a claim is made from average measurements of performance metrics, and a real system may have peaks of high conflict, i.e., is not in a "steady state", then the choice and performance of a CCA remains important.

The combined results of this and previous studies point the way to an adaptive form of CCA [21], possibly using an expert system, based on continous performance measurements of the running system. There is no "best" CCA for all situations. What the various studies have identified is areas in which, and conditions under which, some CCAs perform better than others.

Appendix 1

TESTBED DATABASE SYSTEM (TDBS/C) DESCRIPTION

This appendix provides an overview of the component parts of TDBS/C (Testbed Database System in C), the internal organisation and primitive functions, the identifier list, buffer manager, example schemas for the relation used in Chapter 3 and finally syntax diagrams for the schema, storage schema, view schema definitions, relational expressions and the database definition. A tutorial introduction can be found in [92].

1.1. Historical Background

TDBS/C was developed as a tool for database understanding and research, in particular to allow different aspects of database design to be studied within a testbed prototype system. TDBS/C follows the PRECI (Protype of a RElational Canonical Interface) proposal [36], for a testbed database system with a data manipulation language (DML) embedded in a standard programming language. This gave rise to three implementations, PRECI/H [37, 119], at Aberdeen University and PRECI/C [91] and TDBS/C at Heriot-Watt University. The details are summarised in Table A1.1.

Prototype	Implementation	(lang.-O/S)	DML+	date
PRECI/H	PL/1 Honeywell	(batch)	Fortran	1981
PRECI/C	C/UNIX Zilog Z-80	(inter.)	C	1983
TDBS/C	C/UNIX Vax-750	(inter.)	C	1985

Table A1.1: TDBS/C Historical Development.

The PRECI/C implementation highlighted some design and implementation problems which were removed in TDBS/C but in order to retain an outward functional similarity, TDBS/C has kept the separate schema/storage schema/view schema translators which were influenced by the batch environment of PRECI/H. Some alternatives such as relation definition (combined schema/storage schema definition with default values for storage) and an interactive command interpreter have been

added.

1.2. PRECI

The PRECI proposal was to implement a relational-like database capable of interfacing to user-views defined in network, hierarchical or relational terms. It followed the ANSI/SPARC 3-level architecture (Figure A1.1) [114] where the database is separated into a user (view), conceptual and storage level, each independent of the others.

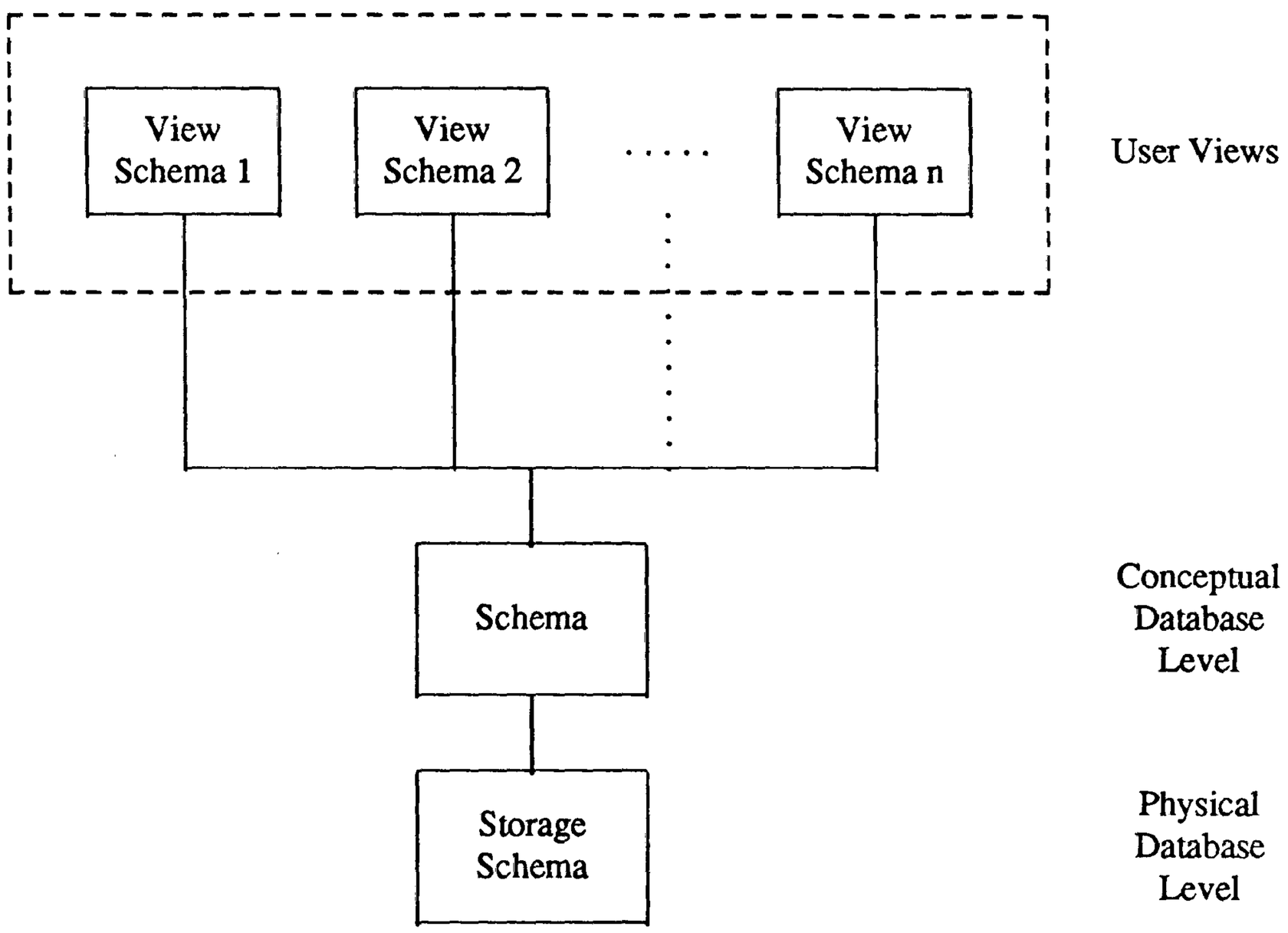


Figure A1.1: ANSI/SPARC Three Level Architecture.

The ideas being that the user could deal with his abstraction of the database (TDBS/C View Schema), the totality of the database is expressed by the conceptual level (TDBS/C Schema) and this in turn is separated from the physical implementation (TDBS/C Storage Schema) of the database. Proposed research included investigations of data allocation (on pages), indexing strategies, buffer management strategies, concurrency control, schema extensions for data consistency, storage schema enhancements, hash storage organisation versus B-trees and natural language interface. TDBS/C has tried to retain the philosophy of a modular system allowing diverse investigations into database design.

1.3. TDBS/C Implementation

TDBS/C is implemented following the relational model which may be described by the following abstract model. A database is a set of RELATIONS and each relation is a set of ATTRIBUTES. Each attribute is defined on a set called a DOMAIN.

$$DB = \{R_1, R_2, ..., R_n\}$$

$$R = \{A_1, A_2, ..., A_n\}$$

where each A_i belongs to a set D_i

where $\{D_1, D_2, ..., D_n\}$ are the DOMAINS of R

The basic implementation philosophy has been to use an abstract data type (data structure + operations on those structures) approach which follows the above abstract definition of the database.

A Data Dictionary (DD) [33], contains descriptions of all objects in a database system. In TDBS/C the DD is the set of all named objects in the database (name space) and is implemented as a linked list. Finally the UNIX philosophy on commands has been adopted where each sub-system is called as a simple command (with possible options) and may be used to build up more complex commands.

1.4. TDBS/C Overview

A diagrammatic overview of the system is given in Figure A1.2. The sub-systems are:

Schema Translator

Process the schema (database) definition (file), produce listing (optional) and generate output file to be read by (a) Storage Schema Translator and (b) View Schema Translator.

Storage Schema Translator

Process the storage schema definition and add this information to that read from the schema translator output to generate an output file to be read by the database initialiser (DBI).

Database Initialiser

Produce initialised (empty) database i.e. one file per relation and a database directory containing a list of relations in this database from the output of the storage schema translator.

View Schema Translator

Process view definitions given in the view schema definition (file). View definitions are relational expressions on the relations defined in the schema definition. The output of the schema translator is read, the relational expressions checked syntactically and an output file for the view definition (used by the precompiler) is generated.

Precompiler

Precompile DML+C programs and convert DML (Data Manipulation Language) statements into calls on the database control system (dbcs) thus producing a C program. At the same time generate C structures (tuple buffers) for relations used in tuple operations (see next section). These structures are included in the C program generated and are processed by the C compiler.

listdb

List internal structures (diagnostic). This is a utility program to examine the various subparts of the database.

Natural Language Interface

Experimental system for "Natural Language" queries. Semantic information is added to the schema and is used to disambiguate queries in a (very) restricted subset of English.

ICI

Interactive Command Interpreter: gives direct access to a database. Also useful as a teaching tool.

QBE

Query-by-Example interface: partial implementation of Zloof's [124] QBE.

DBCS

Database Control System (for compiled progs). Interface between C programs and the database primitives.

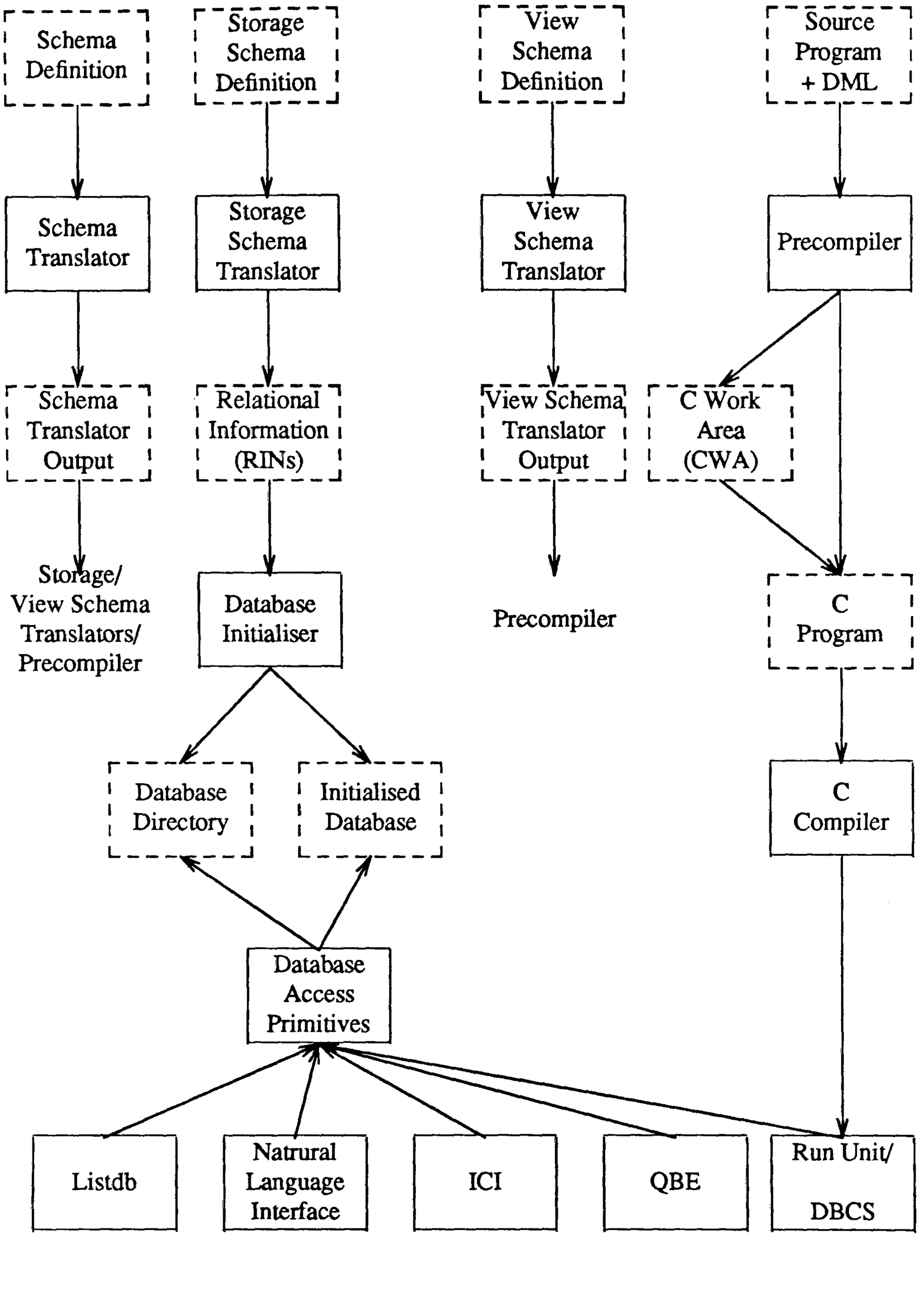


Figure A1.2: TDBS/C - System Overview.

1.5. Internal Organisation

This section deals with the logical and physical structures within TDBS/C. The most important layout is that of the relation object file (<relation name>.rob) followed by the IDLIST (identifier list which contains information about all named entities in the database, similar to a data dictionary) and the buffer organisation.

1.5.1. Relation File Organisation

Each relation is stored in a file with the layout shown in Figure A1.3.

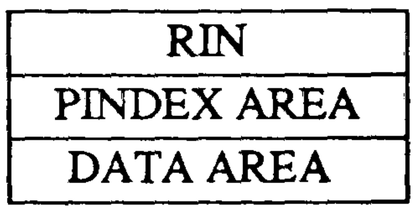


Figure A1.3: Relation File Layout.

The primary key index and tuple storage both use hashing techniques and may therefore be logically viewed as a set of home slots plus overflow slots. The physical implementation is described below.

RIN (Relation Information)

Domain and attribute information; layout information for the DATA and PINDEX areas; data area directory. The RIN is held in primary memory when the relation is open.

SEE Figures A1.4 and A1.5 for details.

PA_HP:	number of home pages
PA_HS:	number of home slots/page
PA_HW:	home slot width
PA_OP:	number of overflow pages
PA_OS:	number of overflow slots/page (global)
PA_OW:	overflow slot width (local/global)
PA_PO:	percentage global overflow
PA_LP:	percentage local overflow/home page
PA_LO:	number of local overflow slots/home page
PA_CK:	primary key size
PA_KR:	primary key range
PA_FS:	first slot
PA_FP:	first page of PINDEX
PA_BM:	bit map size - overflow pages
PA_HB:	bit map size - local overflow slots
PA_OB:	bit map size - global overflow slots
PINDEX	area bit map (global overflow pages)

Figure A1.4: RIN Layout - PINDEX AREA.

DA_HP:	number of home pages
DA_HS:	number of home slots
DA_HW:	home slot width
DA_HP:	number of overflow pages
DA_HS:	number of overflow slots
DA_HW:	overflow slot width
DA_TS:	tuple size
DA_CC:	current cardinality
Data area	
directory:	one entry/slot (home + overflow).

Figure A1.5: RIN Layout - DATA AREA.

PINDEX AREA (primary key index)

Pages are divided into home and overflow pages and comprise a number of slots each having a slot header. In addition home pages have a local overflow area to minimise page faults. The page layout is given in Figure A1.6 and the slot layout in Figure A1.7.

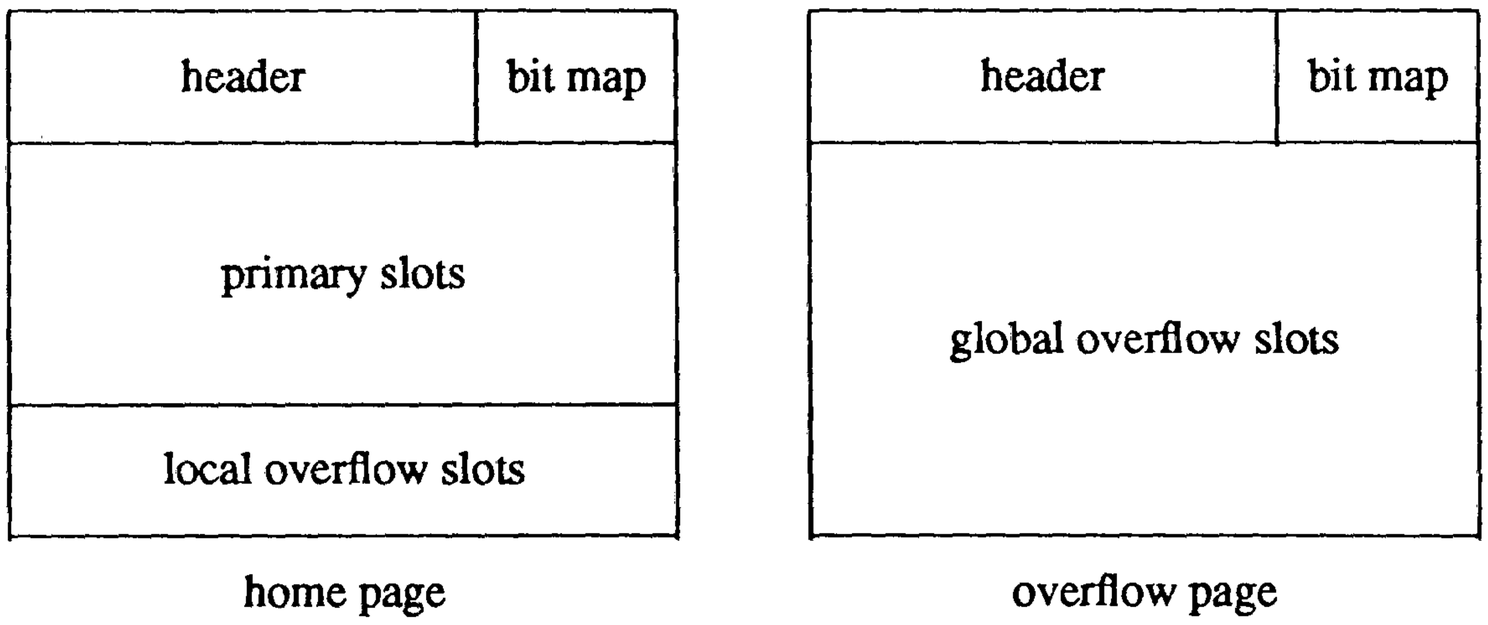


Figure A1.6: PINDEX AREA page layout.

The bit map for the home page refers to the local overflow slots and that for the overflow page to the global overflow slots (free/used). In addition there is a page bit map in the RIN for the global overflow pages.

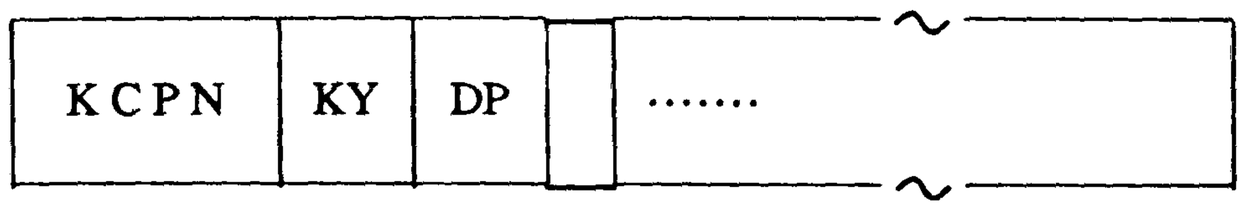


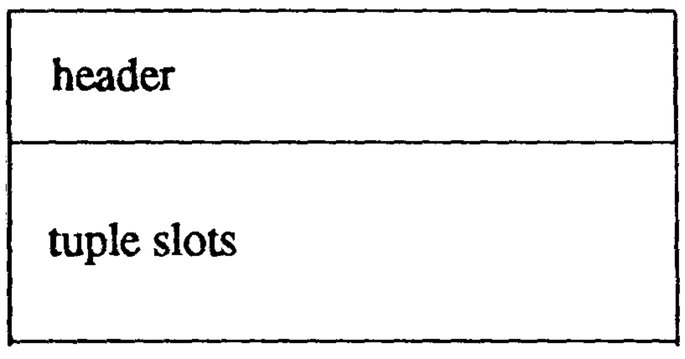
Figure A1.7: PINDEX AREA slot layout.

Slot Header	K:	slot kind (home, local overflow, global overflow)
	C:	number of entries currently in slot
	P:	pointer to prior slot (slot number)
	N:	pointer to next slot (slot number)
Slot (Index) Entries	KYi:	key value (ith entry)
	DPi:	position in DATA AREA (ekey)

DATA AREA (tuple storage)

Pages are divided into home and overflow pages.

Both PINDEX and DATA area pages have page headers giving the creation and update dates for the page, page number within area and version number. The DATA AREA page layout is given in Figure A1.8.



home and overflow slots

Figure A1.8: DATA AREA page layout.

DATA AREA Slot Layout: each slot contains n tuples; each tuple is prefixed with a marker A (active) or D (deleted).

Page Header

The page header for both PINDEX AREA and DATA AREA pages is given in Figure A1.9.

time created	time updated	page no.	version no.

Figure A1.9: Page Header Layout.

1.5.2. Identifier List (IDLIST)

This is a list of named objects, Figure A1.10, in the database system, for example relations, attributes, database definitions etc. The IDLIST is organised as a simple linked list in alphabetical order. Within this, to improve access efficiency, there are sub-lists for important objects such as relations and domains. The logical structure is:

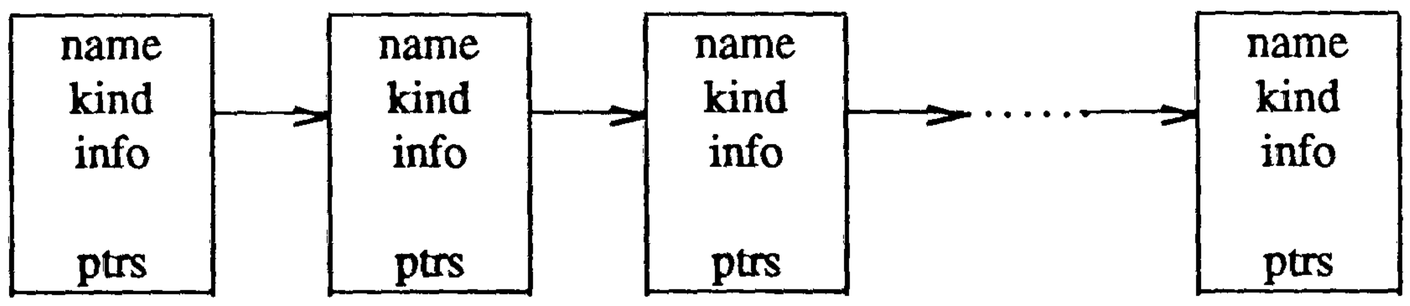


Figure A1.10: IDLIST.

The main kinds of objects (& info/definitions) are:

database:	set of relations
domain:	domain type & storage size
view:	set of base relations, relational expression defining view
relation:	number of attributes & attribute list, storage & access information, pkey & tuple buffers, currency pointers
attribute:	set of relations with this attribute name, set of synonyms for this attribute name
synonym:	set of attributes for which this is a synonym (used only for the Nat. Lang. interface)
user:	user names for access control

Most definitions are sets of objects e.g.

$$\text{database} = \{rel_1, rel_2, ..., rel_n\}$$
$$\text{relation} = \{dom_1, dom_2, ..., dom_n\} \text{ X } \{att_1, att_2, ..., att_n\}$$
$$\text{domain} = \{val_1, val_2, ..., val_n\}$$

and by extension, a distributed database

$$\text{ddb} = \{db_1, db_2, ..., db_n\}$$

The kinds ("types") these objects may assume are:

database:	local, remote
relation:	persistent, transient, intermediate, remote,
view:	renaming, sub- or super-view
attribute:	primary key/non (primary) key
domain:	character/integer (enumerated char/int)
access:	ALL; PERMIT/DENY + user list

Other objects such as operators are included in the IDLIST at startup time (all objects in the database system must be defined in the IDLIST).

This list is the central structure of TDBS/C. All sub-systems work on the IDLIST (creating/adding information). A number of functions are therefore "housekeeping" functions operating on this structure e.g. copying to/from files, lookup, search, retrieval, print sub-parts (diagnostics & list functions). An idea of the importance of the IDLIST (name space in primary memory) is given Figure A1.11.

The main buffer management function called from the system is "getpage". The main "primitive functions" which operate on the IDLIST and page/slot structures described in this section are:

```
phkey(*)        -- hash function: vkey->slotno p-space
dhkey(*)        -- hash function: vkey->slotno d-space

calcpkey(*)     -- calculate pkey value (int)
calcekey(*)     -- calculate effective key (position in DATA AREA - data space only)

getpslot(*)     -- get a PINDEX slot
getdslot(*)     -- get a data slot
getpelem(*)     -- get a PINDEX element from (slot,pos)
getdelem(*)     -- get a data element (tuple)  "
getPIN(*)       -- get PINDEX entry given pkey

pkeycopy(*)     -- copy pkey from tuple to pkey buffer
copytuple(*)    -- copy tuple TO/FROM database and a character buffer
pk_insert(*)    -- insert pkey in PINDEX

find_dpos(*)    -- set up tuple POSITION in descriptor (for NEW tuples)
setdpos(*)      -- set tuple position (dslot, dpos) (for EXISTING tuples - from ekey)
setp(*)         -- set tuple pos - pkey (dslot, dpos)

setdf(*)        -- set first   tuple pos   (dslot, dpos)
setdn(*)        -- set next    tuple pos   (dslot, dpos)
setdp(*)        -- set previous tuple pos   (dslot, dpos)

setpf(*)        -- set first   PINDEX pos  (pslot, ppos)
setpn(*)        -- set next    PINDEX pos  (pslot, ppos)
setpp(*)        -- set previous PINDEX pos  (pslot, ppos)

inserel(*)      -- insert  tuple in relation (pkey)
dputrel(*)      -- insert  tuple in relation (no pkey)
replrel(*)      -- replace tuple in relation
delerel(*)      -- delete  tuple in relation
```

The "vkey" is the numeric value of the primary key ("pkey"), the "ekey" (effective key) is the tuple position in the data space.

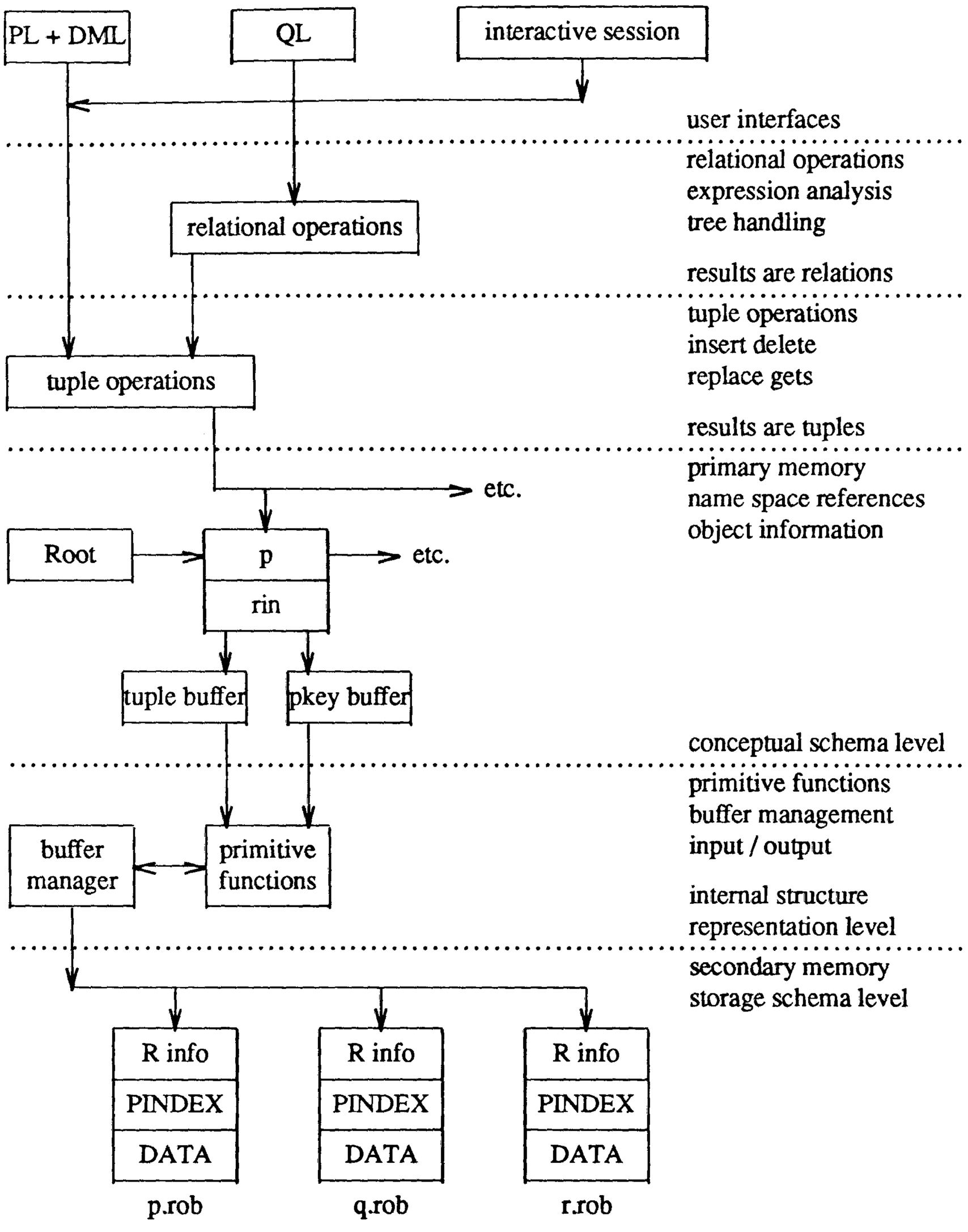


Figure A1.11: TDBS/C Organisation Levels.

1.5.3. Buffer Management

This is part of the database system and comprises a pool of (1Kbyte) pages and a buffer table with 1 entry/page. Requests for index entries/ tuples are translated into logical page requests which in turn are translated into physical page requests within the buffer manager and a page pointer returned. The implementation is a module with two structures as shown in Figure A1.12.

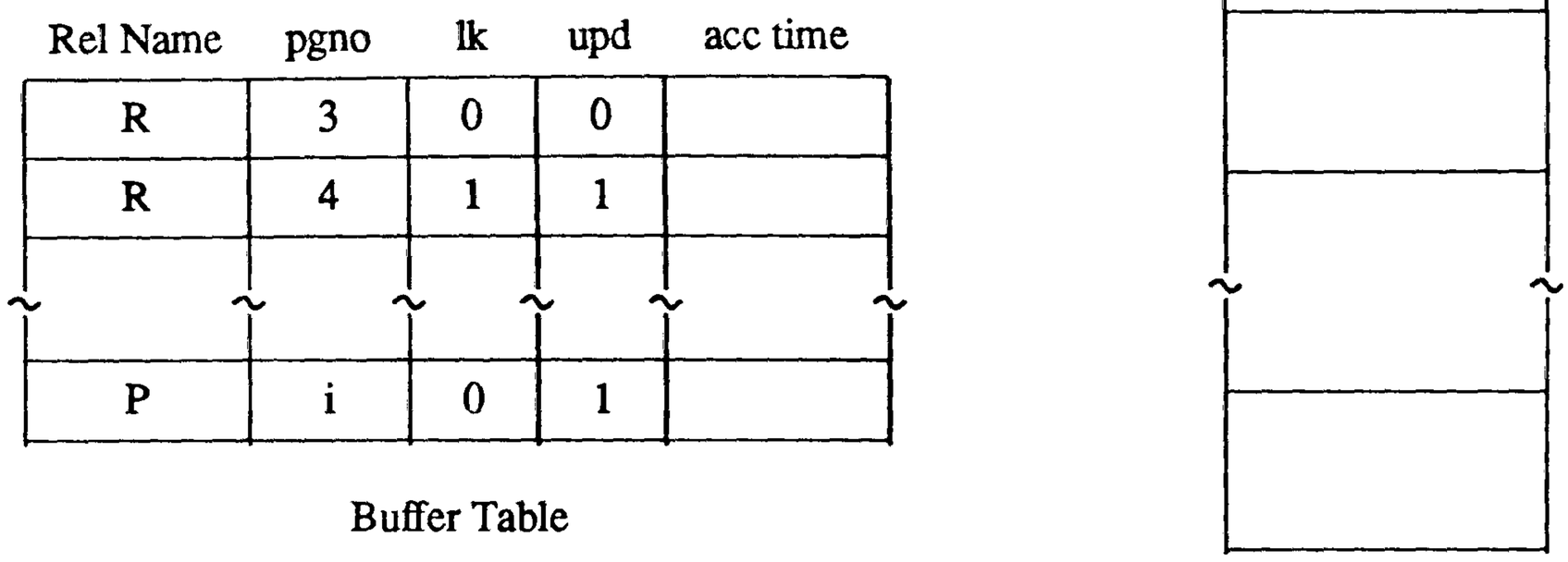


Figure A1.12: Buffer Manager Structures.

The buffer manager operations are:

create and initialise buffer pool & table
get page (from file)
put page (to file)
set update marker on page
set lock on page (lock within buffer)

The concurrency control mechanisms (at page granularity) also operate within the buffer manager by checking access before get/put page operations.

The buffer manager uses a hash function to select a page and if the page is not free, searches within a "virtual window" (the page pool is treated as a circular buffer) for a free page. If there is no free page then a least recently used (LRU) algorithm is used to select a free page. This algorithm will also try to choose a data page first rather than an index page on the assumption that index pages are more frequently used. LRU is not perhaps the most efficient algorithm [17] but it is commonly used in other systems [59, 60]. The virtual window with a hash algorithm is used to avoid increasing linear search costs as the size of the buffer pool (and hence buffer table) increases. For these tests the size of the virtual window is that of the smallest page pool used i.e., 10 pages.

1.6. Schema Examples

The schema, storage schema and view schema definitions used in the single-user benchmarks of Chapter 3 are presented in Figures A1.13, A1.14, A1.15, respectively. This is followed by an example listing of the corresponding RIN for the initialised database in Figure A1.16. The syntax definitions for the schemas are given in the next section.

```
TDBS/C V1.0 Schema Definition Listing    Thu Jul 16 13:34:44 1987

 1 :
 2 : SCHEMA tdb
 3 :
 4 : DOMAIN
 5 :
 6 : dtint2 INTE 2
 7 : dtint4 INTE 4
 8 : dtfill CHAR 12
 9 : dtch20 CHAR 20
10 :
11 : RELATION tr1
12 : PKEY tkey dtint2
13 : ATTR tp50 dtint4
14 : ATTR tp25 dtint4
15 : ATTR tp20 dtint4
16 : ATTR tp10 dtint4
17 : ATTR tp5 dtint4
18 : ATTR tp1 dtint4
19 : ATTR tfill dtfill
20 : ATTR tpad1 dtch20
21 : ATTR tpad2 dtch20
22 : ATTR tpad3 dtch20
23 :

*** End of Schema Definition ***
```

Figure A1.13: Schema Definition.

```
TDBS/C V1.0 Storage Schema Defn. Listing Thu Jul 16 13:35:11 1987

 1 :
 2 :     STORAGE SCHEMA tdb
 3 :
 4 :
 5 :     DATA SECTION
 6 :
 7 :        tr1    10    10    10   1
 8 :
 9 :     PINDEX SECTION
10 :
11 :        tr1    10   1   5   1   02

*** End of Storage Schema File ***
```

Figure A1.14: Storage Schema Definition.

The entries under the DATA SECTION are respectively, the relation name, home slot width (number of tuples), number of home slots, the overflow slot width (number of tuples) and the number of overflow slots.

The entries under the PINDEX SECTION are respectively, the relation name, home slot width (number of index entries), the percentage of home area to be allocated to global overflow slots, the local overflow slot width (number of index entries), the percentage of the home page area to be allocated to local overflow slot(s), and the primary key range specifier n (pkey is in range 0..(10**n -1)).

```
TDBS/C V1.0 View Schema Listing          Thu Jul 16 13:35:45 1987

1 : VIEW SCHEMA tdbv
2 :
3 : IMPORT tdb;
4 :
5 :
6 : tv1   := tr1;
7 :

*** End of View-Schema File ***
```

Figure A1.15: View Schema Definition.

The view (virtual relation) is defined on one (or more) base relations and the appropriate schemas are specified in the import list. Here the view corresponds to the base relation tr1.

```
TDBS/C V1.0 Relation Information Listing Thu Jul 16 13:36:50 1987

  Domains:        Type     Size

    dtint2     : INTE       2
    dtint4     : INTE       4
    dtfill     : CHAR      12
    dtch20     : CHAR      20


  Relation: tr1

  *  tkey          dtint2
     tp50          dtint4
     tp25          dtint4
     tp20          dtint4
     tp10          dtint4
     tp5           dtint4
     tp1           dtint4
     tfill         dtfill
     tpad1         dtch20
     tpad2         dtch20
     tpad3         dtch20


Data Area:

     DA_HP:    10      DA_HS:    10      DA_HW:    10
     DA_OP:     1      DA_OS:     1      DA_OW:    10
     DA_TS:    98      DA_CC:     0


Pindex Area:

     PA_HP:     1      PA_HS:    10      PA_HW:    10
     PA_OP:     1      PA_OS:    17      PA_OW:     5
     PA_PO:     1      PA_LP:     1      PA_LO:     0
     PA_CK:     2      PA_KR:   100      PA_FS:     0
     PA_FP:     2      PA_BM:     4      PA_HB:     0
     PA_OB:    20

DA Dir:    0  0  0  0  0  0  0  0  0  0  0


 Pindex Area Bit Map:  0000

 *** No entry for relation : tr1
```

Figure A1.16: Relation Information (RIN) Listing.

An explanation of the various fields is given in section 1.5 of this appendix.

1.7. Syntax Definitions

<schema>	::=	<schema_def> <domset> <rel_list>
<schema_def>	::=	SCHEMA <id>
<domset>	::=	DOMAIN <domain_list>
<domain_list>	::=	<domain_list> <domain> | <domain>
<domain>	::=	<id> <domain_type> <domain_size>
<domain_type>	::=	INTE | CHAR
<domain_size>	::=	<int>
<rel_list>	::=	<rel_list> <relation> | <relation>
<relation>	::=	RELATION <id> <attr_list>
<attr_list>	::=	<attr_list> <attr> | <attr>
<attr>	::=	<attr_kind> <id> <id>
<attr_kind>	::=	ATTR | PKEY
<id>	::=	<id> <letdig> | <letter>
<letdig>	::=	<letter> | <digit>
<int>	::=	<int> <digit> | <digit>
<letter>	::=	A..Z | a..z
<digit>	::=	0..9

Figure A1.17: Schema Definition Syntax.

<st_schema>	::=	STORAGE SCHEMA <id> <da_sect> <pi_sect>
<da_sect>	::=	DATA SECTION <da_list>
<pi_sect>	::=	PINDEX SECTION <pi_list>
<da_list>	::=	<da_list> <da_entry> | <da_entry>
<da_entry>	::=	<id> <int> <int> <int> <int>
<pi_list>	::=	<pi_list> <pi_entry> | <pi_entry>
<pi_entry>	::=	<id> <int> <int> <int> <int> <int>
<id>	::=	<id> <letdig> | <letter>
<letdig>	::=	<letter> | <digit>
<int>	::=	<int> <digit> | <digit>
<letter>	::=	A..Z | a..z
<digit>	::=	0..9

Figure A1.18: Storage Schema Definition Syntax.

```
<viewschema>      ::=   <viewname> <schimport> <viewdefs>
<viewname>        ::=   VIEW SCHEMA <id>
<schimport>       ::=   IMPORT <snamelist> ;
<snamelist>       ::=   <snamelist> <sname> | <sname>
<sname>           ::=   <id>
<viewdefs>        ::=   <viewdefs> <vdef> | <vdef>
<vdef>            ::=   <id> := <rel_expr> ;
<id>              ::=   <id> <letdig> | <letter>
<letdig>          ::=   <letter> | <digit>
<letter>          ::=   A..Z | a..z
<digit>           ::=   0..9
```

Figure A1.19: View Schema Definition Syntax.

```
<rel_expr>        ::=   <rel_term>
                        | <rel_term> <relop> <rel_term>
                        | <rel_term> SELECT <lexpr>
                        | <rel_term> PROJECT <Aname_list>
<rel_term>        ::=   <relation_name> | ( <rel_expr> )
<relop>           ::=     UNION   | DIFFERENCE | INTERSECTION
                        | EQUIJOIN | DIVISION
<relation_name>   ::=   <id>
<Aname_list>      ::=   <attr_name> | [ <attr_list> ]
<attr_list>       ::=   <attr_name> | <attr_name> , <attr_list>
<attr_name>       ::=   <id>
<lexpr>           ::=     ( <lexpr> ) | <lexpr> AND <lexpr>
                        | <lexpr> OR <lexpr> | NOT <lexpr> | <cexpr>
<cexpr>           ::=   <attr_name> <conop> <expr>
<conop>           ::=    != | = | < | > | <= | >=
<expr>            ::=   <attr_name> | <string> | <int>
<id>              ::=   <id> <letdig> | <letter>
<letdig>          ::=   <letter> | <digit>
<letter>          ::=   A..Z | a..z
<digit>           ::=   0..9
<int>             ::=   <int> <digit> | <digit>
<string>          ::=   " <id> "
```

Figure A1.20: Relation Expression Syntax.

<schema>	::=	<schema_def> <rel_list>
<schema_def>	::=	SCHEMA <id>
<rel_list>	::=	RELATIONS <idlist>
<idlist>	::=	<idlist> <id> | <id>
<id>	::=	<id> <letdig> | <letter>
<letdig>	::=	<letter> | <digit>
<int>	::=	<int> <digit> | <digit>
<letter>	::=	A..Z | a..z
<digit>	::=	0..9

Figure A1.21: Database Definition Syntax.

Appendix 2

SINGLE-USER BENCHMARK RESULTS

For experiment 1, BF_SZ is 10, 100, 100 pages, T_ACC is uniform, Table A2.2 gives a summary of the throughput values for this experiment. The remaining factor levels and corresponding table of results are given in Table A2.1.

Experiment 1		
Table	DB_SZ	TR_TYP
A2.3	100	get-replace
A2.4	100	select-replace
A2.5	100	select
A2.6	1000	get-replace
A2.7	1000	select-replace
A2.8	1000	select
A2.9	10000	get-replace
A2.10	10000	select-replace
A2.11	10000	select
Experiment 2		
A2.12	100	get-replace
A2.13	1000	get-replace
A2.14	10000	get-replace

Table A2.1: Experiments 1 & 2 Results Tables

For experiment 2, BF_SZ is 10, 100, 1000 pages, T_TYP is get-replace, T_ACC is sequential, uniform and DB_ACC is 1, 2, 5, 10, 100 tuples. The sequential get-replace and uniform get-replace are abbreviated to "grs" and "gru" respectively. DB_SZ for this experiment is given in Table A2.1.

Database		DB2 (100)			DB3 (1000)			DB4 (10000)	
Buffer(K)	10	100	1000	10	100	1000	10	100	1000
overhead	1.548	0.955	0.220	1.590	0.980	0.219	1.662	1.000	0.221
grp 1%	0.909	0.667	0.199	0.597	0.508	0.168	0.094	0.087	0.073
grp 5%	0.729	0.534	0.186	0.223	0.174	0.111	0.029	0.026	0.024
grp 10%	0.612	0.476	0.178	0.143	0.124	0.087	0.017	0.016	0.014
grp 20%	0.488	0.399	0.166	0.089	0.082	0.065	0.010	0.009	0.009
grp 25%	0.452	0.370	0.161	0.075	0.069	0.061	0.008	0.008	0.008
grp 50%	0.309	0.269	0.139	0.042	0.040	0.038	0.004	0.004	0.004
grp 100%	0.191	0.175	0.109	0.022	0.022	0.021	0.002	0.002	0.002
srp 1%	0.263	0.241	0.130	0.083	0.083	0.065	0.011	0.011	0.011
srp 5%	0.255	0.230	0.128	0.077	0.076	0.061	0.010	0.010	0.010
srp 10%	0.252	0.232	0.127	0.074	0.074	0.060	0.010	0.010	0.010
srp 20%	0.251	0.229	0.126	0.071	0.071	0.058	0.009	0.009	0.009
srp 25%	0.247	0.228	0.128	0.070	0.070	0.058	0.009	0.009	0.009
srp 50%	0.236	0.216	0.123	0.063	0.064	0.053	0.008	0.008	0.008
srp 100%	0.219	0.204	0.119	0.051	0.053	0.045	0.006	0.006	0.006
sel 1%	0.426	0.355	0.157	0.089	0.086	0.069	0.010	0.010	0.010
sel 5%	0.410	0.344	0.154	0.078	0.076	0.063	0.009	0.009	0.009
sel 10%	0.369	0.325	0.151	0.070	0.069	0.059	0.008	0.008	0.008
sel 20%	0.344	0.317	0.147	0.061	0.063	0.055	0.006	0.007	0.007
sel 25%	0.349	0.313	0.146	0.059	0.060	0.053	0.006	0.006	0.007
sel 50%	0.306	0.283	0.140	0.046	0.050	0.044	0.004	0.005	0.005
sel 100%	0.251	0.225	0.127	0.032	0.035	0.033	0.002	0.003	0.004

Table A2.2: Throughput Results - Summary

Program	Experimental Times Elapsed	CPU	I/O	Thru'put	Time - Overhead Elapsed	CPU	I/O
BUFFER = 10K							
overhead	0.65	0.30	0.20	1.538			
grp 1%	1.10	0.49	0.43	0.909	0.45	0.19	0.23
grp 5%	1.38	0.65	0.46	0.725	0.73	0.35	0.26
grp 10%	1.64	0.87	0.50	0.610	0.99	0.47	0.30
grp 20%	2.05	1.25	0.52	0.488	1.40	0.95	0.32
grp 25%	2.22	1.44	0.52	0.450	1.57	1.14	0.32
grp 50%	3.24	2.44	0.54	0.309	2.59	2.14	0.34
grp 100%	5.25	4.39	0.60	0.190	4.60	4.09	0.40
BUFFER = 100K							
overhead	1.05	0.41	0.43	0.952			
grp 1%	1.48	0.61	0.65	0.676	0.43	0.20	0.22
grp 5%	1.87	0.78	0.70	0.535	0.82	0.37	0.27
grp 10%	2.10	0.98	0.75	0.476	1.05	0.57	0.32
grp 20%	2.51	1.38	0.76	0.398	1.46	0.97	0.33
grp 25%	2.71	1.56	0.76	0.369	1.66	1.15	0.33
grp 50%	3.71	2.56	0.78	0.270	2.66	2.15	0.35
grp 100%	5.71	4.50	0.84	0.175	4.66	4.09	0.41
BUFFER = 1000K							
overhead	4.55	1.60	2.73	0.220			
grp 1%	5.04	1.84	2.93	0.198	0.49	0.24	0.20
grp 5%	5.37	1.98	3.00	0.186	0.82	0.38	0.27
grp 10%	5.61	2.19	3.05	0.178	1.06	0.59	0.32
grp 20%	6.01	2.59	3.06	0.166	1.46	0.99	0.33
grp 25%	6.22	2.80	3.06	0.161	1.67	1.20	0.33
grp 50%	7.21	3.76	3.08	0.139	2.66	2.16	0.35
grp 100%	9.22	5.71	3.13	0.108	4.67	4.11	0.40

Table A2.3: DB_SZ = 100, T_TYP = get-replace

Program	Experimental Times Elapsed	CPU	I/O	Thru'put	Time - Overhead Elapsed	CPU	I/O
BUFFER = 10K							
overhead	0.65	0.30	0.20	1.538			
srp 1%	3.81	1.32	1.79	0.262	3.16	1.02	1.59
srp 5%	3.92	1.35	1.81	0.255	3.27	1.05	1.61
srp 10%	3.97	1.40	1.82	0.252	3.32	1.10	1.62
srp 20%	3.98	1.44	1.82	0.251	3.33	1.14	1.62
srp 25%	4.05	1.41	1.89	0.247	3.40	1.11	1.69
srp 50%	4.24	1.55	1.89	0.236	3.59	1.25	1.69
srp 100%	4.56	1.84	1.95	0.219	3.91	1.54	1.75
BUFFER = 100K							
overhead	1.05	0.41	0.43	0.952			
srp 1%	4.16	1.46	2.03	0.240	3.11	1.05	1.60
srp 5%	4.34	1.48	2.06	0.230	3.29	1.07	1.63
srp 10%	4.32	1.49	2.08	0.231	3.27	1.08	1.65
srp 20%	4.38	1.53	2.12	0.229	3.33	1.12	1.69
srp 25%	4.37	1.51	2.15	0.228	3.32	1.10	1.72
srp 50%	4.63	1.65	2.15	0.216	3.58	1.24	1.72
srp 100%	4.91	1.96	2.19	0.204	3.86	1.55	1.76
BUFFER = 1000K							
overhead	4.55	1.60	2.73	0.220			
srp 1%	7.71	2.69	4.32	0.130	3.16	1.09	1.59
srp 5%	7.84	2.68	4.37	0.128	3.29	1.08	1.64
srp 10%	7.90	2.69	4.41	0.127	3.35	1.09	1.68
srp 20%	7.92	2.75	4.41	0.126	3.37	1.15	1.68
srp 25%	7.83	2.74	4.43	0.128	3.28	1.14	1.70
srp 50%	8.14	2.87	4.45	0.123	3.59	1.27	1.72
srp 100%	8.41	3.18	4.46	0.119	3.86	1.58	1.73

Table A2.4: DB_SZ = 100, T_TYP = select-replace

Program	Experimental Times Elapsed	CPU	I/O	Thru'put	Time - Overhead Elapsed	CPU	I/O
BUFFER = 10K							
overhead	0.65	0.30	0.20	1.538			
sel 1%	2.35	1.20	0.92	0.426	1.70	0.90	0.72
sel 5%	2.44	1.30	0.92	0.410	1.79	1.00	0.72
sel 10%	2.71	1.38	0.97	0.369	2.06	1.08	0.77
sel 20%	2.91	1.52	1.00	0.344	2.26	1.26	0.80
sel 25%	2.87	1.55	1.00	0.348	2.22	1.25	0.80
sel 50%	3.26	1.90	1.05	0.307	2.61	1.60	0.85
sel 100%	3.98	2.48	1.13	0.251	3.33	2.18	0.93
BUFFER = 100K							
overhead	1.05	0.41	0.43	0.952			
sel 1%	2.82	1.30	1.14	0.355	1.77	0.89	0.71
sel 5%	2.91	1.39	1.14	0.344	1.86	0.98	0.71
sel 10%	3.08	1.46	1.20	0.325	2.03	1.05	0.77
sel 20%	3.16	1.54	1.22	0.316	2.11	1.13	0.79
sel 25%	3.20	1.60	1.23	0.313	2.15	1.19	0.80
sel 50%	3.54	1.93	1.27	0.282	2.49	1.52	0.84
sel 100%	4.45	2.57	1.39	0.225	3.40	2.16	0.96
BUFFER = 1000K							
overhead	4.55	1.60	2.73	0.220			
sel 1%	6.38	2.52	3.44	0.157	1.83	0.92	0.71
sel 5%	6.49	2.59	3.47	0.154	1.94	0.99	0.74
sel 10%	6.62	2.65	3.50	0.151	2.07	1.05	0.77
sel 20%	6.78	2.77	3.53	0.147	2.23	1.17	0.80
sel 25%	6.83	2.81	3.55	0.146	2.28	1.21	0.82
sel 50%	7.16	3.16	3.57	0.140	2.61	1.56	0.84
sel 100%	7.87	3.81	3.67	0.127	3.32	2.21	0.94

Table A2.5: DB_SZ = 100, T_TYP = select

Program	Experimental Times Elapsed	CPU	I/O	Thru'put	Time - Overhead Elapsed	CPU	I/O
BUFFER = 10K							
overhead	0.63	0.30	0.20	1.587			
grp 1%	1.68	0.92	0.57	0.595	1.05	0.62	0.37
grp 5%	4.49	2.71	1.10	0.223	3.86	2.41	0.90
grp 10%	6.99	4.94	1.60	0.143	6.36	4.64	1.40
grp 20%	11.28	9.02	1.69	0.089	10.65	8.72	1.49
grp 25%	13.36	11.05	1.75	0.075	12.73	10.75	1.55
grp 50%	23.83	21.29	1.97	0.042	23.20	20.99	1.77
grp 100%	44.79	41.73	2.42	0.022	44.16	41.43	2.20
BUFFER = 100K							
overhead	1.02	0.40	0.45	0.980			
grp 1%	1.97	1.00	0.77	0.508	0.95	0.60	0.32
grp 5%	5.75	2.72	1.35	0.174	4.73	2.32	0.90
grp 10%	8.06	4.92	1.87	0.124	7.04	4.52	1.42
grp 20%	12.23	8.99	1.97	0.082	11.21	8.59	1.52
grp 25%	14.45	11.05	2.02	0.069	13.43	10.65	1.57
grp 50%	24.95	21.26	2.26	0.040	23.93	20.86	1.81
grp 100%	45.86	41.72	2.72	0.022	44.84	41.32	2.27
BUFFER = 1000K							
overhead	4.57	1.62	2.77	0.219			
grp 1%	5.95	2.28	3.11	0.168	1.38	0.66	0.34
grp 5%	8.98	4.00	3.66	0.111	4.41	2.38	0.89
grp 10%	11.50	6.20	4.14	0.087	6.93	4.58	1.37
grp 20%	15.45	10.04	4.21	0.065	10.88	8.42	1.44
grp 25%	16.35	12.01	4.15	0.061	11.78	10.39	1.38
grp 50%	26.63	22.04	4.39	0.038	22.06	20.42	1.72
grp 100%	47.14	42.05	4.89	0.021	42.57	40.43	2.12

Table A2.6: DB_SZ = 1000, T_TYP = get-replace

Program	Experimental Times Elapsed	CPU	I/O	Thru'put	Time - Overhead Elapsed	CPU	I/O
BUFFER = 10K							
overhead	0.63	0.30	0.20	1.587			
srp 1%	11.99	7.29	3.70	0.083	11.36	6.99	3.50
srp 5%	13.00	7.50	4.09	0.077	12.37	7.20	3.89
srp 10%	13.47	7.74	4.36	0.074	12.84	7.44	4.16
srp 20%	14.15	8.26	4.47	0.071	13.52	7.96	4.27
srp 25%	14.33	8.43	4.55	0.070	13.70	8.13	4.35
srp 50%	15.93	9.80	4.73	0.063	15.30	9.50	4.53
srp 100%	19.45	12.86	5.27	0.051	18.82	12.56	5.07
BUFFER = 100K							
overhead	1.02	0.40	0.45	0.980			
srp 1%	12.00	6.67	4.04	0.083	10.98	6.27	3.59
srp 5%	13.23	6.79	4.35	0.076	12.21	6.39	3.90
srp 10%	13.52	7.11	4.68	0.074	12.50	6.71	4.23
srp 20%	14.03	7.47	4.78	0.071	13.01	7.07	4.33
srp 25%	14.24	7.71	4.77	0.070	13.22	7.31	4.32
srp 50%	15.67	8.91	5.06	0.064	14.65	8.51	4.61
srp 100%	18.87	11.70	5.47	0.053	17.85	11.30	5.02
BUFFER = 1000K							
overhead	4.57	1.62	2.77	0.219			
srp 1%	15.45	7.96	6.33	0.065	10.88	6.34	3.56
srp 5%	16.36	8.03	6.65	0.061	11.79	6.41	3.88
srp 10%	16.74	8.27	6.95	0.060	12.17	6.65	4.18
srp 20%	17.36	8.69	7.08	0.058	12.79	7.07	4.31
srp 25%	17.37	8.84	6.99	0.058	12.80	7.22	4.22
srp 50%	19.02	10.22	7.21	0.053	14.45	8.60	4.44
srp 100%	22.17	12.94	7.70	0.045	17.60	11.32	4.93

Table A2.7: DB_SZ = 1000, T_TYP = select-replace

Program	Experimental Times Elapsed	CPU	I/O	Thru'put	Time - Overhead Elapsed	CPU	I/O
BUFFER = 10K							
overhead	0.63	0.30	0.20	1.587			
sel 1%	11.28	7.18	3.01	0.089	10.65	6.88	2.81
sel 5%	12.88	7.89	3.66	0.078	12.25	7.59	3.46
sel 10%	14.27	8.83	4.28	0.070	13.64	8.53	4.08
sel 20%	16.28	10.26	4.74	0.061	15.65	9.96	4.54
sel 25%	16.91	10.82	4.95	0.059	16.28	10.52	4.75
sel 50%	21.79	14.53	6.04	0.046	21.16	14.23	5.84
sel 100%	31.39	22.15	8.06	0.032	30.76	21.85	7.86
BUFFER = 100K							
overhead	1.02	0.40	0.45	0.980			
sel 1%	11.62	7.18	3.27	0.086	10.60	6.78	2.82
sel 5%	13.15	7.81	3.77	0.076	12.13	7.41	3.32
sel 10%	14.48	8.62	4.30	0.069	13.46	8.22	3.85
sel 20%	15.84	9.92	4.51	0.063	14.82	9.52	4.06
sel 25%	16.56	10.42	4.58	0.060	15.54	10.02	4.13
sel 50%	20.18	13.76	5.02	0.050	19.16	13.36	4.57
sel 100%	28.45	20.79	6.07	0.035	27.43	20.39	5.62
BUFFER = 1000K							
overhead	4.57	1.62	2.77	0.219			
sel 1%	14.56	7.98	5.54	0.069	9.99	6.36	2.77
sel 5%	15.94	8.65	5.93	0.063	11.37	7.03	3.16
sel 10%	16.84	9.42	6.25	0.059	12.27	7.80	3.48
sel 20%	18.10	10.58	6.42	0.055	13.53	8.96	3.65
sel 25%	18.83	11.10	6.48	0.053	14.26	9.48	3.71
sel 50%	22.48	14.42	6.93	0.044	17.91	12.80	4.16
sel 100%	29.97	20.92	7.80	0.033	25.40	19.30	5.03

Table A2.8: DB_SZ = 1000, T_TYP = select

Program	Experimental Times Elapsed	CPU	I/O	Thru'put	Time - Overhead Elapsed	CPU	I/O
BUFFER = 10K							
overhead	0.60	0.30	0.20	1.667			
grp 1%	10.70	5.11	2.31	0.094	10.10	4.81	2.11
grp 5%	34.10	22.66	7.21	0.029	33.50	22.36	7.01
grp 10%	60.41	44.54	12.08	0.017	59.81	44.24	11.88
grp 20%	102.34	84.99	13.06	0.010	101.74	84.69	12.86
grp 25%	123.18	105.21	14.02	0.008	122.58	104.91	13.82
grp 50%	226.63	207.24	16.20	0.004	226.03	206.94	16.00
grp 100%	430.49	407.00	20.72	0.002	429.89	406.70	20.52
BUFFER = 100K							
overhead	1.00	0.41	0.43	1.000			
grp 1%	11.48	5.24	2.69	0.087	10.48	4.83	2.26
grp 5%	38.98	23.34	8.28	0.026	37.98	22.93	7.85
grp 10%	64.31	46.00	13.41	0.016	63.31	45.59	12.98
grp 20%	106.60	86.81	14.47	0.009	105.60	86.40	14.04
grp 25%	127.79	107.40	14.95	0.008	126.79	106.99	14.52
grp 50%	232.14	209.29	17.89	0.004	231.14	208.88	17.46
grp 100%	439.70	412.25	22.74	0.002	438.70	411.84	22.31
BUFFER = 1000K							
overhead	4.52	1.59	2.77	0.221			
grp 1%	13.72	6.10	4.80	0.073	9.20	4.51	2.03
grp 5%	42.42	23.14	10.34	0.024	37.90	21.55	7.57
grp 10%	69.05	44.66	15.49	0.014	64.53	43.07	12.72
grp 20%	110.19	85.06	16.34	0.009	105.67	83.47	13.57
grp 25%	131.28	105.37	16.90	0.008	126.76	103.78	14.13
grp 50%	234.37	206.11	19.20	0.004	229.85	204.52	16.43
grp 100%	451.80	419.98	23.72	0.002	447.28	418.39	20.95

Table A2.9: DB_SZ = 10000, T_TYP = get-replace

Program	Experimental Times Elapsed	CPU	I/O	Thru'put	Time - Overhead Elapsed	CPU	I/O
BUFFER = 10K							
overhead	0.60	0.30	0.20	1.667			
srp 1%	92.21	68.40	21.60	0.011	91.61	68.10	21.40
srp 5%	98.32	69.43	25.05	0.010	97.72	69.13	24.85
srp 10%	103.64	71.31	28.21	0.010	103.04	71.01	28.01
srp 20%	109.08	76.48	28.87	0.009	108.48	76.18	28.67
srp 25%	110.91	78.10	29.48	0.009	110.31	77.80	29.28
srp 50%	127.24	93.13	31.23	0.008	126.64	92.83	31.03
srp 100%	163.20	122.90	36.08	0.006	162.60	122.60	35.88
BUFFER = 100K							
overhead	1.00	0.41	0.43	1.000			
srp 1%	89.01	63.94	21.94	0.011	88.01	63.53	21.51
srp 5%	99.13	66.59	25.58	0.010	98.13	66.18	25.15
srp 10%	102.82	67.36	28.77	0.010	101.82	66.95	28.34
srp 20%	110.66	74.05	29.68	0.009	109.66	73.64	29.25
srp 25%	111.90	74.98	30.34	0.009	110.90	74.57	29.91
srp 50%	126.97	87.86	32.33	0.008	125.97	87.45	31.90
srp 100%	161.80	118.24	36.76	0.006	160.80	117.83	36.33
BUFFER = 1000K							
overhead	4.52	1.59	2.77	0.221			
srp 1%	88.95	60.36	24.29	0.011	84.43	58.77	21.52
srp 5%	96.98	62.64	27.29	0.010	92.46	61.05	24.52
srp 10%	102.04	63.61	30.36	0.010	97.54	62.02	27.59
srp 20%	106.84	68.29	31.25	0.009	102.32	66.70	28.48
srp 25%	109.69	70.14	31.81	0.009	105.17	68.55	29.04
srp 50%	125.18	83.24	34.11	0.008	120.66	81.65	31.34
srp 100%	155.52	110.18	37.36	0.006	151.00	108.59	34.59

Table A2.10: DB_SZ = 10000, T_TYP = select-replace

Program	Experimental Times Elapsed	CPU	I/O	Thru'put	Time - Overhead Elapsed	CPU	I/O
BUFFER = 10K							
overhead	0.60	0.30	0.20	1.667			
sel 1%	96.86	67.47	23.42	0.010	96.26	67.17	23.22
sel 5%	109.79	74.38	30.87	0.009	109.19	74.08	30.57
sel 10%	126.31	84.67	38.38	0.008	125.71	84.37	38.18
sel 20%	157.63	103.82	51.14	0.006	157.03	103.52	50.94
sel 25%	174.23	112.73	58.52	0.006	173.63	112.43	58.32
sel 50%	256.08	160.90	92.18	0.004	255.48	160.60	91.98
sel 100%	417.92	259.61	155.08	0.002	417.32	259.31	154.88
BUFFER = 100K							
overhead	1.00	0.41	0.43	1.000			
sel 1%	99.98	70.55	23.35	0.010	98.98	70.14	22.92
sel 5%	113.08	76.80	30.00	0.009	112.08	76.39	29.57
sel 10%	129.16	88.35	36.60	0.008	128.16	87.94	36.17
sel 20%	148.66	103.93	40.66	0.007	147.66	103.52	40.23
sel 25%	154.88	108.26	42.32	0.006	153.88	107.85	41.89
sel 50%	201.26	146.84	50.49	0.005	200.26	146.43	50.06
sel 100%	299.26	222.06	72.83	0.003	298.26	221.65	72.40
BUFFER = 1000K							
overhead	4.52	1.59	2.77	0.221			
sel 1%	97.57	68.38	24.15	0.010	93.05	66.79	21.38
sel 5%	109.70	72.58	29.60	0.009	105.18	70.99	26.83
sel 10%	126.11	85.97	35.62	0.008	121.59	84.38	32.85
sel 20%	141.03	97.54	38.11	0.007	136.51	95.95	35.34
sel 25%	147.92	103.07	39.21	0.007	143.40	101.48	36.44
sel 50%	186.17	136.35	44.35	0.005	181.65	134.76	41.58
sel 100%	265.26	206.14	54.07	0.004	261.74	204.55	51.30

Table A2.11: DB_SZ = 10000, T_TYP = select

Program		Experimental Times Elapsed	CPU	I/O	Thru'put	Time - Overhead Elapsed	CPU	I/O
BUFFER = 10K								
overhead		0.65	0.30	0.20	1.538			
grs	1	1.11	0.52	0.41	0.901	0.46	0.22	0.21
gru	1	1.10	0.51	0.41	0.909	0.45	0.21	0.21
grs	2	1.20	0.56	0.41	0.833	0.55	0.26	0.21
gru	2	1.15	0.55	0.42	0.870	0.50	0.25	0.22
grs	5	1.30	0.69	0.41	0.769	0.65	0.39	0.21
gru	5	1.34	0.68	0.45	0.746	0.69	0.38	0.25
grs	10	1.51	0.90	0.42	0.632	0.86	0.60	0.22
gru	10	1.60	0.91	0.50	0.625	0.95	0.61	0.30
grs	100	5.61	4.82	0.57	0.178	4.96	4.52	0.37
gru	100	--	--	--	---	--	--	--
BUFFER = 100K								
overhead		1.05	0.41	0.43	0.952			
grs	1	1.42	0.61	0.64	0.704	0.37	0.20	0.21
gru	1	1.41	0.59	0.66	0.709	0.36	0.18	0.23
grs	2	1.50	0.64	0.66	0.667	0.45	0.23	0.23
gru	2	1.49	0.64	0.66	0.671	0.44	0.23	0.23
grs	5	1.60	0.77	0.65	0.625	0.55	0.36	0.22
gru	5	1.62	0.79	0.68	0.617	0.57	0.38	0.25
grs	10	1.80	0.98	0.65	0.556	0.75	0.57	0.22
gru	10	1.90	0.99	0.75	0.526	0.85	0.58	0.32
grs	100	5.74	4.74	0.81	0.174	4.69	4.33	0.38
gru	100	--	--	--	---	--	--	--
BUFFER = 1000K								
overhead		4.55	1.60	2.73	0.220			
grs	1	5.00	1.84	2.96	0.200	0.45	0.24	0.23
gru	1	4.99	1.81	2.98	0.200	0.44	0.21	0.25
grs	2	5.04	1.88	2.97	0.198	0.49	0.28	0.24
gru	2	5.01	1.88	2.97	0.200	0.46	0.28	0.24
grs	5	5.19	2.04	2.94	0.193	0.64	0.44	0.21
gru	5	5.20	2.01	2.98	0.192	0.65	0.41	0.25
grs	10	5.40	2.23	2.97	0.185	0.85	0.63	0.24
gru	10	5.44	2.19	3.07	0.184	0.89	0.59	0.34
grs	100	9.59	5.98	3.17	0.104	5.04	4.38	0.44
gru	100	--	--	--	---	--	--	--

Table A2.12: DB_SZ = 100, T_TYP = get-replace (sequential/uniform)

Program		Experimental Times Elapsed	CPU	I/O	Thru'put	Time - Overhead Elapsed	CPU	I/O
BUFFER = 10K								
overhead		0.63	0.30	0.20	1.587			
grs	1	1.10	0.52	0.41	0.910	0.37	0.22	0.21
gru	1	1.09	0.50	0.41	0.917	0.46	0.20	0.21
grs	2	1.10	0.56	0.41	0.906	0.37	0.26	0.21
gru	2	1.11	0.53	0.41	0.901	0.48	0.23	0.21
grs	5	1.26	0.68	0.42	0.794	0.53	0.38	0.22
gru	5	1.30	0.66	0.46	0.769	0.67	0.36	0.26
grs	10	1.50	0.90	0.43	0.667	0.87	0.60	0.23
gru	10	1.60	0.88	0.50	0.625	0.97	0.58	0.30
grs	100	5.59	4.78	0.59	0.179	4.96	4.48	0.39
gru	100	7.47	5.22	1.59	0.134	6.84	4.92	1.39
BUFFER = 100K								
overhead		1.02	0.40	0.45	0.980			
grs	1	1.40	0.61	0.65	0.714	0.38	0.21	0.20
gru	1	1.50	0.62	0.66	0.667	0.48	0.22	0.21
grs	2	1.46	0.64	0.66	0.685	0.44	0.24	0.21
gru	2	1.50	0.67	0.66	0.667	0.48	0.27	0.21
grs	5	1.60	0.77	0.66	0.625	0.58	0.37	0.21
gru	5	1.70	0.81	0.70	0.588	0.68	0.41	0.25
grs	10	1.80	0.99	0.66	0.556	0.78	0.59	0.21
gru	10	2.00	1.04	0.74	0.500	0.98	0.64	0.29
grs	100	5.70	4.69	0.84	0.175	4.68	4.29	0.39
gru	100	8.31	5.07	1.84	0.120	7.29	4.67	1.39
BUFFER = 1000K								
overhead		4.57	1.62	2.77	0.219			
grs	1	4.98	1.82	2.97	0.201	0.41	0.20	0.20
gru	1	5.00	1.85	2.97	0.200	0.43	0.23	0.20
grs	2	5.00	1.88	2.96	0.200	0.43	0.26	0.19
gru	2	5.04	1.89	2.99	0.198	0.47	0.27	0.22
grs	5	5.12	2.00	2.96	0.195	0.55	0.38	0.19
gru	5	5.21	2.03	3.02	0.192	0.64	0.41	0.25
grs	10	5.35	2.20	2.98	0.187	0.78	0.58	0.21
gru	10	5.50	2.27	3.05	0.182	0.93	0.65	0.28
grs	100	9.31	6.00	3.14	0.107	4.74	4.38	0.37
gru	100	11.70	6.37	4.20	0.085	7.13	4.75	1.43

Table A2.13: DB_SZ = 1000, T_TYP = get-replace (sequential/uniform)

Program		Experimental Times Elapsed	CPU	I/O	Thru'put	Time - Overhead Elapsed	CPU	I/O
BUFFER = 10K								
overhead		0.60	0.30	0.20	1.667			
grs	1	1.10	0.48	0.48	0.909	0.50	0.18	0.28
gru	1	1.10	0.48	0.44	0.909	0.50	0.18	0.24
grs	2	1.19	0.52	0.48	0.840	0.59	0.22	0.28
gru	2	1.13	0.52	0.45	0.885	0.53	0.22	0.25
grs	5	1.30	0.64	0.48	0.769	0.70	0.34	0.28
gru	5	1.33	0.67	0.46	0.752	0.73	0.37	0.26
grs	10	1.50	0.85	0.49	0.667	0.90	0.55	0.29
gru	10	1.60	0.87	0.54	0.625	1.00	0.57	0.34
grs	100	5.46	4.61	0.65	0.183	4.86	4.31	0.45
gru	100	6.71	5.03	1.50	0.149	6.11	4.73	1.30
BUFFER = 100K								
overhead		1.00	0.41	0.43	1.000			
grs	1	1.50	0.62	0.70	0.667	0.50	0.21	0.27
gru	1	1.47	0.61	0.67	0.679	0.47	0.20	0.24
grs	2	1.56	0.67	0.70	0.641	0.56	0.26	0.27
gru	2	1.50	0.66	0.68	0.667	0.50	0.25	0.25
grs	5	1.70	0.79	0.71	0.588	0.70	0.38	0.28
gru	5	1.70	0.78	0.72	0.588	0.70	0.37	0.29
grs	10	1.90	1.01	0.71	0.526	0.90	0.60	0.28
gru	10	1.93	1.01	0.76	0.518	0.93	0.60	0.33
grs	100	5.92	4.79	0.91	0.169	4.92	4.38	0.48
gru	100	7.71	5.09	1.87	0.130	6.71	4.64	1.44
BUFFER = 1000K								
overhead		4.52	1.59	2.77	0.221			
grs	1	5.22	1.87	3.01	0.192	0.70	0.28	0.24
gru	1	5.00	1.80	3.01	0.200	0.48	0.21	0.24
grs	2	5.09	1.88	3.00	0.196	0.57	0.29	0.24
gru	2	5.03	1.87	3.00	0.199	0.51	0.28	0.23
grs	5	5.20	2.02	3.00	0.192	0.68	0.43	0.23
gru	5	5.20	2.00	3.01	0.192	0.68	0.41	0.24
grs	10	5.41	2.24	3.00	0.185	0.89	0.64	0.23
gru	10	5.49	2.20	3.09	0.182	0.97	0.61	0.32
grs	100	9.43	6.08	3.16	0.106	4.91	4.49	0.39
gru	100	11.45	6.15	4.24	0.087	6.93	4.56	1.47

Table A2.14: DB_SZ = 10000, T_TYP = get-replace (sequential/uniform)

Appendix 3

PROTOTYPE RESULTS

Experiment 1 Results: (GRAN = 1, T_TYP = get-replace)

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.824 (+- 0.307%)	12.132 (+- 0.307%)	400	400	0
PRE	0.752 (+- 0.323%)	13.299 (+- 0.323%)	400	400	0
2PLE	0.758 (+- 0.216%)	13.191 (+- 0.216%)	400	400	0
2PLU	0.708 (+- 0.563%)	14.123 (+- 0.565%)	400	400	0
A2PLU	0.700 (+- 0.269%)	14.278 (+- 0.268%)	400	400	0
BTO	0.757 (+- 0.557%)	13.209 (+- 0.557%)	400	400	0
ABTO	0.753 (+- 0.390%)	13.280 (+- 0.393%)	400	400	0
SV	0.781 (+- 0.406%)	12.812 (+- 0.407%)	400	400	0

Table A3.1: T_REQ = 1, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	12.132 (+- 0.307%)	0.650 (+- 0.328%)	0.551 (+- 0.481%)
PRE	13.299 (+- 0.323%)	0.674 (+- 0.345%)	0.643 (+- 0.354%)
2PLE	13.191 (+- 0.216%)	0.670 (+- 0.174%)	0.635 (+- 0.259%)
2PLU	14.123 (+- 0.565%)	0.684 (+- 0.682%)	0.702 (+- 0.532%)
A2PLU	14.278 (+- 0.268%)	0.684 (+- 0.480%)	0.716 (+- 0.218%)
BTO	13.209 (+- 0.557%)	0.682 (+- 0.779%)	0.620 (+- 0.309%)
ABTO	13.280 (+- 0.393%)	0.678 (+- 0.696%)	0.630 (+- 0.308%)
SV	12.812 (+- 0.407%)	0.679 (+- 0.780%)	0.583 (+- 0.259%)

Table A3.2: T_REQ = 1, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
A2PLU	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
BTO	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
ABTO	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
SV	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)

Table A3.3: T_REQ = 1, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.787 (+- 0.303%)	12.705 (+- 0.302%)	400	400	0
PRE	0.685 (+- 0.223%)	14.590 (+- 0.224%)	400	400	0
2PLE	0.689 (+- 0.232%)	14.508 (+- 0.232%)	400	399	0
2PLU	0.591 (+- 0.751%)	16.924 (+- 0.762%)	400	376	23
A2PLU	0.600 (+- 0.685%)	16.667 (+- 0.687%)	400	378	21
BTO	0.670 (+- 0.767%)	14.933 (+- 0.765%)	400	378	22
ABTO	0.678 (+- 0.735%)	14.742 (+- 0.734%)	400	378	21
SV	0.684 (+- 1.167%)	14.624 (+- 1.147%)	400	368	31

Table A3.4: T_REQ = 2, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	12.705 (+- 0.302%)	0.694 (+- 0.410%)	0.572 (+- 0.444%)
PRE	14.590 (+- 0.224%)	0.733 (+- 0.262%)	0.707 (+- 0.498%)
2PLE	14.503 (+- 0.224%)	0.722 (+- 0.444%)	0.698 (+- 0.301%)
2PLU	15.916 (+- 0.256%)	0.752 (+- 0.519%)	0.805 (+- 0.416%)
A2PLU	15.009 (+- 0.640%)	0.750 (+- 0.616%)	0.808 (+- 1.105%)
BTO	14.014 (+- 0.508%)	0.734 (+- 0.498%)	0.677 (+- 0.370%)
ABTO	13.187 (+- 1.163%)	0.738 (+- 0.689%)	0.688 (+- 0.391%)
SV	13.302 (+- 0.635%)	0.731 (+- 0.710%)	0.609 (+- 1.074%)

Table A3.5: T_REQ = 2, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	1.576 (+- ---)	0.140 (+- ---)	0.110 (+- ---)
2PLU	14.000 (+- 3.632%)	0.749 (+- 1.432%)	0.748 (+- 1.029%)
A2PLU	13.449 (+- 2.786%)	0.746 (+- 1.558%)	0.753 (+- 1.536%)
BTO	13.872 (+- 2.733%)	0.731 (+- 1.088%)	0.639 (+- 1.847%)
ABTO	12.955 (+- 2.614%)	0.723 (+- 1.150%)	0.659 (+- 1.648%)
SV	13.448 (+- 2.731%)	0.715 (+- 0.796%)	0.513 (+- 1.088%)

Table A3.6: T_REQ = 2, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.695 (+- 0.471%)	14.392 (+- 0.470%)	400	400	0
PRE	0.493 (+- 0.359%)	20.298 (+- 0.357%)	400	400	0
2PLE	0.517 (+- 0.806%)	19.336 (+- 0.804%)	400	388	11
2PLU	0.254 (+- 2.525%)	39.371 (+- 2.506%)	400	218	182
A2PLU	0.399 (+- 1.115%)	25.102 (+- 1.121%)	400	274	125
BTO	0.291 (+- 2.650%)	34.452 (+- 2.549%)	400	202	197
ABTO	0.389 (+- 1.923%)	25.744 (+- 1.955%)	400	262	137
SV	0.398 (+- 1.399%)	25.138 (+- 1.409%)	400	237	163

Table A3.7: T_REQ = 5, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	14.392 (+- 0.470%)	0.825 (+- 0.283%)	0.611 (+- 0.655%)
PRE	20.298 (+- 0.357%)	0.961 (+- 0.291%)	1.031 (+- 0.475%)
2PLE	18.792 (+- 0.302%)	0.913 (+- 0.371%)	0.911 (+- 0.457%)
2PLU	22.342 (+- 0.898%)	0.963 (+- 0.444%)	1.166 (+- 0.504%)
A2PLU	11.421 (+- 2.458%)	0.942 (+- 0.490%)	1.020 (+- 0.388%)
BTO	16.888 (+- 0.473%)	0.913 (+- 0.432%)	0.842 (+- 0.391%)
ABTO	10.677 (+- 1.819%)	0.907 (+- 0.614%)	0.856 (+- 0.276%)
SV	14.434 (+- 0.438%)	0.887 (+- 0.558%)	0.643 (+- 0.505%)

Table A3.8: T_REQ = 5, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	16.085 (+- 8.348%)	0.818 (+- 2.244%)	0.739 (+- 2.114%)
2PLU	18.312 (+- 1.039%)	0.954 (+- 0.549%)	1.009 (+- 0.600%)
A2PLU	10.282 (+- 2.138%)	0.939 (+- 0.563%)	0.897 (+- 0.457%)
BTO	15.942 (+- 0.709%)	0.897 (+- 0.659%)	0.763 (+- 0.755%)
ABTO	11.192 (+- 1.559%)	0.893 (+- 0.920%)	0.777 (+- 0.816%)
SV	13.564 (+- 1.055%)	0.844 (+- 0.713%)	0.541 (+- 0.724%)

Table A3.9: T_REQ = 5, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.577 (+- 0.276%)	17.337 (+- 0.275%)	400	400	0
PRE	0.277 (+- 0.809%)	36.099 (+- 0.814%)	400	400	0
2PLE	0.235 (+- 2.196%)	42.696 (+- 2.284%)	400	236	164
2PLU	0.048 (+- 8.153%)	211.973 (+- 8.281%)	400	55	345
A2PLU	0.218 (+- 5.735%)	46.262 (+- 6.368%)	400	204	195
BTO	0.023 (+-16.919%)	479.024 (+-19.606%)	400	19	380
ABTO	0.301 (+- 5.032%)	33.457 (+- 4.744%)	400	219	180
SV	0.166 (+- 1.723%)	60.161 (+- 1.723%)	400	115	284

Table A3.10: T_REQ = 10, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	17.337 (+- 0.275%)	1.049 (+- 0.291%)	0.670 (+- 0.439%)
PRE	36.099 (+- 0.814%)	1.494 (+- 0.350%)	2.043 (+- 1.082%)
2PLE	26.228 (+- 0.539%)	1.245 (+- 0.240%)	1.324 (+- 0.495%)
2PLU	36.094 (+- 2.917%)	1.341 (+- 0.719%)	1.781 (+- 1.005%)
A2PLU	9.819 (+- 4.681%)	1.233 (+- 0.362%)	1.446 (+- 0.507%)
BTO	21.291 (+- 1.487%)	1.200 (+- 0.580%)	1.067 (+- 0.736%)
ABTO	5.159 (+- 4.924%)	1.182 (+- 0.532%)	1.130 (+- 0.973%)
SV	17.390 (+- 0.801%)	1.151 (+- 0.625%)	0.706 (+- 0.891%)

Table A3.11: T_REQ = 10, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	21.650 (+- 1.728%)	1.053 (+- 0.483%)	1.000 (+- 0.998%)
2PLU	25.506 (+- 0.703%)	1.271 (+- 0.353%)	1.355 (+- 0.402%)
A2PLU	11.118 (+- 4.522%)	1.218 (+- 0.351%)	1.158 (+- 0.540%)
BTO	19.035 (+- 0.318%)	1.153 (+- 0.375%)	0.900 (+- 0.323%)
ABTO	11.328 (+- 6.536%)	1.166 (+- 0.605%)	0.924 (+- 0.466%)
SV	15.342 (+- 0.393%)	1.089 (+- 0.478%)	0.568 (+- 0.513%)

Table A3.12: T_REQ = 10, Wasted Work Costs

Summary Tables for GRAN = 1, T_REQ = 1, 2, 5, 10, T_TYP = get-replace

ET: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	110	109	116	118	109	109	106
2	115	114	133	131	118	116	115
5	141	134	274	174	239	179	175
10	208	246	1223	267	2763	193	347
CPU: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	104	103	105	105	105	104	104
2	106	104	115	114	112	112	114
5	116	114	214	166	217	167	178
10	142	189	902	229	2508	204	366
IO: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	117	115	127	130	113	114	106
2	124	122	149	149	125	127	114
5	169	153	329	234	260	207	166
10	305	301	1558	382	3081	282	314

Table A3.13: Overhead Summary (%)

ET: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	100	100	100	100	100	100	100
2	100	100	94	90	94	90	91
5	100	97	57	46	49	41	57
10	100	61	17	21	5	15	29
CPU: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	100	100	100	100	100	100	100
2	100	100	94	95	95	95	92
5	100	98	55	69	51	66	60
10	100	63	14	51	5	55	30
IO: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	100	100	100	100	100	100	100
2	100	100	95	95	95	95	93
5	100	98	58	71	53	68	63
10	100	66	17	57	6	60	33

Table A3.14: Efficiency Summary (%)

Experiment 1 Results: (GRAN = 10, T_TYP = get-replace)

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.824 (+- 0.307%)	12.132 (+- 0.307%)	400	400	0
PRE	0.727 (+- 0.290%)	13.757 (+- 0.290%)	400	400	0
2PLE	0.734 (+- 0.519%)	13.630 (+- 0.520%)	400	400	0
2PLU	0.529 (+- 1.248%)	18.910 (+- 1.250%)	400	300	99
A2PLU	0.640 (+- 0.484%)	15.625 (+- 0.482%)	400	340	59
BTO	0.583 (+- 1.073%)	17.170 (+- 1.072%)	400	302	97
ABTO	0.711 (+- 0.849%)	14.068 (+- 0.850%)	400	345	54
SV	0.513 (+- 1.220%)	19.506 (+- 1.223%)	400	262	137

Table A3.15: T_REQ = 1, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	12.132 (+- 0.307%)	0.650 (+- 0.328%)	0.551 (+- 0.481%)
PRE	13.757 (+- 0.290%)	0.690 (+- 0.278%)	0.664 (+- 0.492%)
2PLE	13.630 (+- 0.520%)	0.680 (+- 0.640%)	0.653 (+- 0.523%)
2PLU	14.304 (+- 0.655%)	0.693 (+- 0.607%)	0.702 (+- 0.305%)
A2PLU	11.488 (+- 0.839%)	0.690 (+- 0.537%)	0.694 (+- 0.377%)
BTO	12.493 (+- 0.637%)	0.668 (+- 0.838%)	0.623 (+- 0.293%)
ABTO	10.327 (+- 1.283%)	0.676 (+- 0.612%)	0.629 (+- 0.461%)
SV	12.048 (+- 0.890%)	0.680 (+- 0.871%)	0.619 (+- 0.361%)

Table A3.16: T_REQ = 1, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	11.935 (+- 1.415%)	0.696 (+- 1.004%)	0.644 (+- 0.616%)
A2PLU	9.970 (+- 1.936%)	0.691 (+- 0.594%)	0.645 (+- 0.873%)
BTO	12.551 (+- 0.878%)	0.668 (+- 0.796%)	0.601 (+- 0.395%)
ABTO	10.810 (+- 2.442%)	0.676 (+- 0.930%)	0.610 (+- 0.805%)
SV	12.236 (+- 1.107%)	0.660 (+- 0.909%)	0.520 (+- 0.498%)

Table A3.17: T_REQ = 1, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.787 (+- 0.303%)	12.705 (+- 0.302%)	400	400	0
PRE	0.604 (+- 0.364%)	16.545 (+- 0.361%)	400	400	0
2PLE	0.594 (+- 0.713%)	16.832 (+- 0.714%)	400	382	17
2PLU	0.227 (+- 2.750%)	44.187 (+- 2.756%)	400	152	247
A2PLU	0.416 (+- 1.903%)	24.050 (+- 1.923%)	400	250	150
BTO	0.267 (+- 3.719%)	37.549 (+- 3.574%)	400	149	251
ABTO	0.529 (+- 1.778%)	18.937 (+- 1.842%)	400	256	143
SV	0.339 (+- 2.200%)	29.562 (+- 2.243%)	400	177	222

Table A3.18: T_REQ = 2, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	12.705 (+- 0.302%)	0.694 (+- 0.410%)	0.572 (+- 0.444%)
PRE	16.545 (+- 0.361%)	0.784 (+- 0.363%)	0.824 (+- 0.253%)
2PLE	16.149 (+- 0.613%)	0.758 (+- 0.683%)	0.796 (+- 0.660%)
2PLU	17.529 (+- 1.288%)	0.784 (+- 0.576%)	0.852 (+- 0.520%)
A2PLU	9.231 (+- 1.641%)	0.758 (+- 0.513%)	0.806 (+- 0.364%)
BTO	12.947 (+- 0.877%)	0.729 (+- 0.971%)	0.677 (+- 0.430%)
ABTO	6.529 (+- 3.946%)	0.729 (+- 0.876%)	0.696 (+- 0.521%)
SV	12.506 (+- 0.705%)	0.728 (+- 0.640%)	0.629 (+- 0.315%)

Table A3.19: T_REQ = 2, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	13.040 (+- 4.265%)	0.722 (+- 1.062%)	0.663 (+- 2.782%)
2PLU	14.379 (+- 0.952%)	0.779 (+- 0.539%)	0.765 (+- 0.640%)
A2PLU	9.008 (+- 2.464%)	0.759 (+- 0.727%)	0.740 (+- 0.866%)
BTO	12.531 (+- 0.685%)	0.720 (+- 0.574%)	0.638 (+- 0.498%)
ABTO	7.985 (+- 1.945%)	0.726 (+- 0.720%)	0.648 (+- 0.585%)
SV	11.551 (+- 0.804%)	0.706 (+- 0.792%)	0.524 (+- 0.439%)

Table A3.20: T_REQ = 2, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.695 (+- 0.471%)	14.392 (+- 0.470%)	400	400	0
PRE	0.419 (+- 0.511%)	23.841 (+- 0.510%)	400	400	0
2PLE	0.181 (+- 1.965%)	55.444 (+- 1.989%)	400	143	256
2PLU	0.020 (+- 8.344%)	499.632 (+- 8.310%)	400	17	383
A2PLU	0.247 (+- 5.385%)	40.739 (+- 4.861%)	400	185	214
BTO	0.018 (+-23.181%)	654.739 (+-24.686%)	400	11	388
ABTO	0.326 (+- 4.015%)	30.824 (+- 3.956%)	400	206	193
SV	0.159 (+- 1.574%)	62.919 (+- 1.545%)	400	92	308

Table A3.21: T_REQ = 5, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	14.392 (+- 0.470%)	0.825 (+- 0.283%)	0.611 (+- 0.655%)
PRE	23.841 (+- 0.510%)	1.045 (+- 0.272%)	1.239 (+- 0.486%)
2PLE	21.476 (+- 1.194%)	0.959 (+- 0.529%)	1.101 (+- 0.441%)
2PLU	27.550 (+- 2.313%)	0.989 (+- 1.070%)	1.246 (+- 1.804%)
A2PLU	5.790 (+- 5.446%)	0.915 (+- 0.553%)	1.106 (+- 1.189%)
BTO	16.307 (+- 3.435%)	0.877 (+- 0.736%)	0.837 (+- 1.520%)
ABTO	3.341 (+- 1.544%)	0.864 (+- 0.706%)	0.913 (+- 1.164%)
SV	14.154 (+- 0.745%)	0.860 (+- 0.594%)	0.678 (+- 0.820%)

Table A3.22: T_REQ = 5, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	16.933 (+- 1.283%)	0.823 (+- 0.801%)	0.830 (+- 0.658%)
2PLU	18.546 (+- 0.588%)	0.945 (+- 0.696%)	0.966 (+- 0.386%)
A2PLU	9.892 (+- 8.558%)	0.913 (+- 0.642%)	0.893 (+- 1.048%)
BTO	14.222 (+- 0.338%)	0.859 (+- 0.653%)	0.723 (+- 0.213%)
ABTO	11.526 (+- 4.161%)	0.856 (+- 0.682%)	0.730 (+- 0.939%)
SV	12.555 (+- 0.462%)	0.840 (+- 0.481%)	0.549 (+- 0.485%)

Table A3.23: T_REQ = 5, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.577 (+- 0.276%)	17.337 (+- 0.275%)	400	400	0
PRE	0.341 (+- 0.278%)	29.290 (+- 0.278%)	400	400	0
2PLE	0.036 (+- 7.588%)	278.911 (+- 7.948%)	400	30	370
2PLU	0.001 (+- ---)	2965.004 (+- ---)	400	1	399
A2PLU	0.011 (+- ---)	105.790 (+- ---)	400	15	384
BTO	0.001 (+- ---)	2623.789 (+- ---)	400	1	399
ABTO	0.309 (+- 2.918%)	32.431 (+- 2.984%)	400	218	181
SV	0.082 (+- 2.370%)	122.326 (+- 2.402%)	400	55	344

Table A3.24: T_REQ = 10, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	17.337 (+- 0.275%)	1.049 (+- 0.291%)	0.670 (+- 0.439%)
PRE	29.290 (+- 0.278%)	1.327 (+- 0.295%)	1.466 (+- 0.609%)
2PLE	27.026 (+- 3.607%)	1.226 (+- 1.330%)	1.404 (+- 1.394%)
2PLU	14.305 (+- ---)	0.501 (+- ---)	0.688 (+- ---)
A2PLU	1.116 (+- ---)	0.234 (+- ---)	0.294 (+- ---)
BTO	10.569 (+- ---)	0.471 (+- ---)	0.417 (+- ---)
ABTO	3.446 (+- 1.127%)	1.094 (+- 0.608%)	1.159 (+- 0.457%)
SV	17.128 (+- 1.025%)	1.078 (+- 0.649%)	0.745 (+- 0.636%)

Table A3.25: T_REQ = 10, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	17.993 (+- 0.941%)	0.881 (+- 0.716%)	0.886 (+- 0.399%)
2PLU	22.771 (+- 0.787%)	1.176 (+- 0.492%)	1.153 (+- 0.752%)
A2PLU	23.825 (+- 4.146%)	1.177 (+- 0.518%)	1.164 (+- 1.111%)
BTO	17.660 (+- 0.173%)	1.087 (+- 0.222%)	0.840 (+- 0.231%)
ABTO	12.296 (+- 5.008%)	1.084 (+- 0.603%)	0.855 (+- 0.585%)
SV	14.810 (+- 0.471%)	1.057 (+- 0.503%)	0.580 (+- 0.661%)

Table A3.26: T_REQ = 10, Wasted Work Costs

Summary Tables for GRAN = 10, T_REQ = 1, 2, 5, 10, T_TYP = get-replace

ET: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	113	112	156	129	142	116	161
2	130	132	348	189	296	149	233
5	166	385	3472	283	4549	214	437
10	169	1609	17102	610	15134	187	706
CPU: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	106	105	142	125	136	120	158
2	113	114	296	175	281	163	233
5	127	295	2749	240	4213	202	446
10	126	1175	13457	256	13791	190	733
IO: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	120	119	166	146	148	132	162
2	144	144	366	219	307	185	225
5	203	424	3835	351	4794	262	412
10	219	1876	20602	410	16717	280	653

Table A3.27: Overhead Summary (%)

ET: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	100	100	76	74	73	73	62
2	100	96	40	38	35	34	42
5	100	39	6	14	3	11	22
10	100	10	0	0	0	11	14
CPU: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	100	100	75	85	76	86	66
2	100	96	38	62	38	64	45
5	100	39	4	46	3	52	23
10	100	10	0	1	0	55	14
IO: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	100	100	77	86	76	87	69
2	100	96	41	64	39	66	49
5	100	43	5	52	3	57	27
10	100	11	0	1	0	62	17

Table A3.28: Efficiency Summary (%)

Experiment 2 Results: (T_TYP = get-replace)

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.505 (+- 0.133%)	19.816 (+- 0.132%)	400	400	0
PRE	0.468 (+- 0.192%)	21.354 (+- 0.192%)	400	400	0
2PLE	0.467 (+- 0.203%)	21.418 (+- 0.203%)	400	400	0
2PLU	0.461 (+- 0.282%)	21.679 (+- 0.283%)	400	400	0
A2PLU	0.459 (+- 0.200%)	21.781 (+- 0.200%)	400	400	0
BTO	0.475 (+- 0.100%)	21.048 (+- 0.099%)	400	400	0
ABTO	0.475 (+- 0.145%)	21.072 (+- 0.145%)	400	400	0
SV	0.478 (+- 0.342%)	20.899 (+- 0.342%)	400	400	0

Table A3.29: LC = 0%, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	19.816 (+- 0.132%)	1.178 (+- 0.190%)	0.787 (+- 0.362%)
PRE	21.354 (+- 0.192%)	1.196 (+- 0.200%)	0.914 (+- 0.288%)
2PLE	21.418 (+- 0.203%)	1.205 (+- 0.171%)	0.910 (+- 0.283%)
2PLU	21.679 (+- 0.283%)	1.211 (+- 0.153%)	0.926 (+- 0.503%)
A2PLU	21.781 (+- 0.200%)	1.209 (+- 0.143%)	0.935 (+- 0.203%)
BTO	21.048 (+- 0.099%)	1.191 (+- 0.157%)	0.888 (+- 0.267%)
ABTO	21.072 (+- 0.145%)	1.189 (+- 0.233%)	0.887 (+- 0.254%)
SV	20.899 (+- 0.342%)	1.195 (+- 0.193%)	0.878 (+- 0.284%)

Table A3.30: LC = 0%, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
A2PLU	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
BTO	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
ABTO	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
SV	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)

Table A3.31: LC = 0%, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.505 (+- 0.133%)	19.816 (+- 0.132%)	400	400	0
PRE	0.467 (+- 0.215%)	21.436 (+- 0.215%)	400	400	0
2PLE	0.466 (+- 0.305%)	21.472 (+- 0.305%)	400	400	0
2PLU	0.417 (+- 0.236%)	23.981 (+- 0.236%)	400	360	39
A2PLU	0.447 (+- 0.554%)	22.369 (+- 0.549%)	400	381	18
BTO	0.419 (+- 1.084%)	23.890 (+- 1.088%)	400	346	53
ABTO	0.466 (+- 0.315%)	21.475 (+- 0.315%)	400	381	19
SV	0.445 (+- 0.296%)	22.497 (+- 0.297%)	400	364	36

Table A3.32: LC = 20%, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	19.816 (+- 0.132%)	1.178 (+- 0.190%)	0.787 (+- 0.362%)
PRE	21.436 (+- 0.215%)	1.197 (+- 0.046%)	0.919 (+- 0.293%)
2PLE	21.472 (+- 0.305%)	1.210 (+- 0.275%)	0.912 (+- 0.203%)
2PLU	21.593 (+- 0.125%)	1.207 (+- 0.147%)	0.934 (+- 0.329%)
A2PLU	20.459 (+- 0.229%)	1.208 (+- 0.319%)	0.937 (+- 0.508%)
BTO	20.799 (+- 0.137%)	1.187 (+- 0.193%)	0.878 (+- 0.270%)
ABTO	19.498 (+- 0.172%)	1.189 (+- 0.302%)	0.876 (+- 0.291%)
SV	20.391 (+- 0.238%)	1.191 (+- 0.105%)	0.858 (+- 0.777%)

Table A3.33: LC = 20%, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	19.790 (+- 1.141%)	1.221 (+- 0.627%)	0.788 (+- 0.832%)
A2PLU	17.277 (+- 2.001%)	1.203 (+- 0.817%)	0.750 (+- 1.213%)
BTO	18.130 (+- 1.294%)	1.179 (+- 0.445%)	0.723 (+- 1.092%)
ABTO	18.785 (+- 1.358%)	1.178 (+- 1.101%)	0.797 (+- 1.263%)
SV	19.309 (+- 3.104%)	1.173 (+- 0.448%)	0.600 (+- 1.489%)

Table A3.34: LC = 20%, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.505 (+- 0.133%)	19.816 (+- 0.132%)	400	400	0
PRE	0.476 (+- 0.944%)	21.035 (+- 0.939%)	400	400	0
2PLE	0.461 (+- 0.346%)	21.673 (+- 0.346%)	400	400	0
2PLU	0.289 (+- 0.512%)	34.549 (+- 0.509%)	400	243	157
A2PLU	0.448 (+- 0.678%)	22.348 (+- 0.678%)	400	321	78
BTO	0.248 (+- 0.851%)	40.328 (+- 0.848%)	400	197	202
ABTO	0.471 (+- 0.609%)	21.255 (+- 0.612%)	400	317	82
SV	0.319 (+- 0.443%)	31.324 (+- 0.447%)	400	247	152

Table A3.35: LC = 50%, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	19.816 (+- 0.132%)	1.178 (+- 0.190%)	0.787 (+- 0.362%)
PRE	21.035 (+- 0.939%)	1.232 (+- 0.151%)	0.979 (+- 1.018%)
2PLE	21.673 (+- 0.346%)	1.231 (+- 0.294%)	0.990 (+- 0.349%)
2PLU	21.405 (+- 0.325%)	1.206 (+- 0.259%)	0.921 (+- 0.240%)
A2PLU	13.986 (+- 1.241%)	1.206 (+- 0.276%)	0.913 (+- 0.423%)
BTO	20.448 (+- 0.350%)	1.189 (+- 0.189%)	0.875 (+- 0.357%)
ABTO	12.545 (+- 1.059%)	1.185 (+- 0.266%)	0.854 (+- 0.259%)
SV	19.785 (+- 0.231%)	1.196 (+- 0.141%)	0.824 (+- 0.317%)

Table A3.36: LC = 50%, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	18.361 (+- 0.515%)	1.208 (+- 0.273%)	0.747 (+- 0.675%)
A2PLU	13.079 (+- 6.041%)	1.208 (+- 0.390%)	0.745 (+- 0.822%)
BTO	17.421 (+- 0.364%)	1.172 (+- 0.254%)	0.693 (+- 0.326%)
ABTO	13.084 (+- 1.950%)	1.175 (+- 0.545%)	0.718 (+- 1.317%)
SV	16.767 (+- 0.727%)	1.170 (+- 0.389%)	0.576 (+- 0.434%)

Table A3.37: LC = 50%, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.505 (+- 0.133%)	19.816 (+- 0.132%)	400	400	0
PRE	0.446 (+- 0.470%)	22.416 (+- 0.466%)	400	400	0
2PLE	0.432 (+- 0.399%)	23.131 (+- 0.400%)	400	400	0
2PLU	0.151 (+- 1.002%)	66.455 (+- 0.999%)	400	123	276
A2PLU	0.409 (+- 3.258%)	24.506 (+- 3.244%)	400	260	139
BTO	0.099 (+- 2.001%)	100.784 (+- 2.024%)	400	76	323
ABTO	0.428 (+- 1.999%)	23.383 (+- 2.022%)	400	254	146
SV	0.175 (+- 1.691%)	57.274 (+- 1.672%)	400	128	271

Table A3.38: LC = 80%, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	19.816 (+- 0.132%)	1.178 (+- 0.190%)	0.787 (+- 0.362%)
PRE	22.416 (+- 0.466%)	1.284 (+- 0.166%)	1.067 (+- 0.573%)
2PLE	23.131 (+- 0.400%)	1.247 (+- 0.260%)	1.099 (+- 0.303%)
2PLU	21.716 (+- 0.769%)	1.208 (+- 0.360%)	0.905 (+- 0.670%)
A2PLU	6.865 (+- 2.699%)	1.190 (+- 0.214%)	0.889 (+- 0.627%)
BTO	20.097 (+- 0.757%)	1.186 (+- 0.252%)	0.869 (+- 0.466%)
ABTO	5.763 (+- 1.160%)	1.177 (+- 0.231%)	0.838 (+- 0.348%)
SV	19.092 (+- 0.451%)	1.180 (+- 0.292%)	0.792 (+- 0.482%)

Table A3.39: LC = 80%, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	18.022 (+- 0.402%)	1.211 (+- 0.238%)	0.734 (+- 0.520%)
A2PLU	12.216 (+- 5.718%)	1.198 (+- 0.284%)	0.739 (+- 0.638%)
BTO	17.026 (+- 0.295%)	1.171 (+- 0.206%)	0.679 (+- 0.303%)
ABTO	10.499 (+- 3.464%)	1.168 (+- 0.395%)	0.699 (+- 0.369%)
SV	16.009 (+- 0.417%)	1.165 (+- 0.381%)	0.566 (+- 0.622%)

Table A3.40: LC = 80%, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.505 (+- 0.133%)	19.816 (+- 0.132%)	400	400	0
PRE	0.390 (+- 0.329%)	25.668 (+- 0.330%)	400	400	0
2PLE	0.362 (+- 0.808%)	27.598 (+- 0.830%)	400	400	0
2PLU	0.048 (+- 1.979%)	210.614 (+- 2.029%)	400	39	360
A2PLU	0.401 (+- 3.876%)	25.061 (+- 3.789%)	400	258	141
BTO	0.012 (+- ---)	1639.983 (+- ---)	400	8	391
ABTO	0.387 (+- 2.342%)	25.857 (+- 2.444%)	400	232	167
SV	0.073 (+- 2.242%)	137.865 (+- 2.235%)	400	51	348

Table A3.41: LC = 100%, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	19.816 (+- 0.132%)	1.178 (+- 0.190%)	0.787 (+- 0.362%)
PRE	25.668 (+- 0.330%)	1.322 (+- 0.214%)	1.111 (+- 0.407%)
2PLE	27.598 (+- 0.830%)	1.274 (+- 0.481%)	1.195 (+- 0.615%)
2PLU	26.243 (+- 1.075%)	1.192 (+- 0.540%)	0.833 (+- 1.126%)
A2PLU	7.067 (+- 3.647%)	1.193 (+- 0.147%)	0.896 (+- 0.396%)
BTO	15.609 (+-21.828%)	1.060 (+-20.405%)	0.662 (+-20.480%)
ABTO	3.041 (+- 1.429%)	1.163 (+- 0.238%)	0.800 (+- 0.551%)
SV	17.513 (+- 0.745%)	1.167 (+- 0.593%)	0.692 (+- 0.752%)

Table A3.42: LC = 100%, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	18.062 (+- 0.193%)	1.202 (+- 0.275%)	0.734 (+- 0.417%)
A2PLU	11.998 (+- 7.574%)	1.209 (+- 0.264%)	0.733 (+- 0.579%)
BTO	16.814 (+- 0.383%)	1.175 (+- 0.081%)	0.661 (+- 0.226%)
ABTO	12.151 (+- 9.002%)	1.164 (+- 0.218%)	0.678 (+- 0.488%)
SV	15.757 (+- 0.123%)	1.156 (+- 0.232%)	0.568 (+- 0.536%)

Table A3.43: LC = 100%, Wasted Work Costs

Summary Tables for LC = 0%, 20%, 50%, 80%, 100%, T_TYP = get-replace

ET: LC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
0%	108	108	109	110	106	106	105
20%	108	108	121	113	121	108	114
50%	106	109	174	113	204	107	158
80%	113	117	335	124	509	118	289
100%	130	139	1063	126	8276	130	696
CPU: LC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
0%	102	102	103	103	101	101	101
20%	102	103	114	108	116	106	111
50%	105	104	169	127	203	127	163
80%	109	106	332	156	522	157	310
100%	112	108	1038	158	8740	170	764
IO: LC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
0%	116	116	118	119	113	113	112
20%	117	116	130	124	126	116	117
50%	124	126	178	139	201	132	150
80%	136	140	323	163	476	158	253
100%	141	152	963	165	7344	164	577

Table A3.44: Overhead Summary (%)

ET: LC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
0%	100	100	100	100	100	100	100
20%	100	100	90	92	87	91	91
50%	100	100	62	63	51	59	63
80%	100	100	33	28	20	25	33
100%	100	100	12	28	2	12	13
CPU: LC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
0%	100	100	100	100	100	100	100
20%	100	100	90	95	87	95	91
50%	100	100	61	80	50	80	62
80%	100	100	31	65	19	64	32
100%	100	100	10	64	2	58	13
IO: LC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
0%	100	100	100	100	100	100	100
20%	100	100	92	96	89	96	94
50%	100	100	66	83	55	82	70
80%	100	100	35	69	23	68	40
100%	100	100	11	69	2	62	15

Table A3.45: Efficiency Summary (%)

Experiment 1 Results: (GRAN = 1, T_TYP = select-replace)

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.734 (+- 0.701%)	13.633 (+- 0.706%)	400	400	0
PRE	0.668 (+- 0.304%)	14.960 (+- 0.303%)	400	400	0
2PLE	0.664 (+- 0.294%)	15.051 (+- 0.294%)	400	400	0
2PLU	0.633 (+- 0.289%)	15.792 (+- 0.289%)	400	400	0
A2PLU	0.634 (+- 0.352%)	15.784 (+- 0.353%)	400	400	0
BTO	0.670 (+- 0.206%)	14.923 (+- 0.205%)	400	400	0
ABTO	0.670 (+- 0.442%)	14.932 (+- 0.441%)	400	400	0
SV	0.689 (+- 0.234%)	14.516 (+- 0.233%)	400	400	0

Table A3.46: T_REQ = 1, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	13.633 (+- 0.706%)	0.801 (+- 0.060%)	0.553 (+- 0.406%)
PRE	14.960 (+- 0.303%)	0.819 (+- 0.229%)	0.662 (+- 0.130%)
2PLE	15.051 (+- 0.294%)	0.822 (+- 0.209%)	0.663 (+- 0.362%)
2PLU	15.792 (+- 0.289%)	0.849 (+- 0.229%)	0.700 (+- 0.194%)
A2PLU	15.784 (+- 0.353%)	0.847 (+- 0.199%)	0.702 (+- 0.209%)
BTO	14.923 (+- 0.205%)	0.830 (+- 0.466%)	0.638 (+- 0.659%)
ABTO	14.932 (+- 0.441%)	0.825 (+- 0.403%)	0.642 (+- 0.372%)
SV	14.516 (+- 0.233%)	0.845 (+- 0.388%)	0.592 (+- 0.575%)

Table A3.47: T_REQ = 1, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
A2PLU	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
BTO	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
ABTO	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
SV	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)

Table A3.48: T_REQ = 1, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.721 (+- 0.483%)	13.869 (+- 0.484%)	400	400	0
PRE	0.625 (+- 0.284%)	15.992 (+- 0.283%)	400	400	0
2PLE	0.624 (+- 0.268%)	16.030 (+- 0.268%)	400	400	0
2PLU	0.553 (+- 0.366%)	18.086 (+- 0.366%)	400	380	20
A2PLU	0.564 (+- 1.397%)	17.725 (+- 1.379%)	400	378	21
BTO	0.622 (+- 0.992%)	16.080 (+- 1.004%)	400	384	15
ABTO	0.625 (+- 0.690%)	16.003 (+- 0.689%)	400	384	16
SV	0.627 (+- 1.837%)	15.951 (+- 1.814%)	400	367	33

Table A3.49: T_REQ = 2, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	13.869 (+- 0.484%)	0.813 (+- 0.290%)	0.567 (+- 0.590%)
PRE	15.992 (+- 0.283%)	0.853 (+- 0.359%)	0.725 (+- 0.721%)
2PLE	16.024 (+- 0.229%)	0.848 (+- 0.400%)	0.721 (+- 0.342%)
2PLU	17.176 (+- 0.390%)	0.880 (+- 0.274%)	0.799 (+- 0.508%)
A2PLU	15.966 (+- 0.507%)	0.864 (+- 0.299%)	0.795 (+- 0.330%)
BTO	15.393 (+- 0.054%)	0.854 (+- 0.264%)	0.692 (+- 0.301%)
ABTO	14.743 (+- 1.358%)	0.851 (+- 0.435%)	0.696 (+- 0.672%)
SV	14.553 (+- 0.728%)	0.851 (+- 0.423%)	0.607 (+- 0.484%)

Table A3.50: T_REQ = 2, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	15.500 (+- 6.119%)	0.889 (+- 1.787%)	0.736 (+- 2.110%)
A2PLU	14.257 (+- 5.027%)	0.858 (+- 1.612%)	0.739 (+- 1.464%)
BTO	14.905 (+- 1.863%)	0.844 (+- 1.024%)	0.662 (+- 2.033%)
ABTO	14.356 (+- 3.262%)	0.847 (+- 2.381%)	0.662 (+- 2.923%)
SV	13.373 (+- 2.365%)	0.820 (+- 1.035%)	0.521 (+- 1.644%)

Table A3.51: T_REQ = 2, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.680 (+- 0.137%)	14.705 (+- 0.137%)	400	400	0
PRE	0.476 (+- 1.034%)	20.990 (+- 1.033%)	400	400	0
2PLE	0.521 (+- 0.515%)	19.194 (+- 0.513%)	400	400	0
2PLU	0.276 (+- 2.126%)	36.294 (+- 2.122%)	400	232	167
A2PLU	0.386 (+- 0.423%)	25.889 (+- 0.423%)	400	281	119
BTO	0.334 (+- 2.502%)	29.986 (+- 2.454%)	400	239	160
ABTO	0.396 (+- 2.223%)	25.264 (+- 2.201%)	400	276	124
SV	0.384 (+- 1.254%)	26.073 (+- 1.260%)	400	234	166

Table A3.52: T_REQ = 5, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	14.705 (+- 0.137%)	0.849 (+- 0.285%)	0.611 (+- 0.745%)
PRE	20.990 (+- 1.033%)	1.004 (+- 0.737%)	1.057 (+- 1.546%)
2PLE	19.194 (+- 0.513%)	0.959 (+- 0.190%)	0.908 (+- 0.502%)
2PLU	21.657 (+- 0.645%)	0.981 (+- 0.310%)	1.111 (+- 0.803%)
A2PLU	12.507 (+- 3.861%)	0.968 (+- 0.286%)	1.065 (+- 0.204%)
BTO	17.556 (+- 0.747%)	0.936 (+- 0.518%)	0.854 (+- 0.470%)
ABTO	11.800 (+- 2.330%)	0.932 (+- 0.376%)	0.848 (+- 0.226%)
SV	14.797 (+- 1.022%)	0.910 (+- 0.351%)	0.654 (+- 0.878%)

Table A3.53: T_REQ = 5, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	18.326 (+- 2.090%)	0.975 (+- 0.522%)	0.976 (+- 1.255%)
A2PLU	11.192 (+- 2.567%)	0.960 (+- 0.495%)	0.940 (+- 0.988%)
BTO	16.500 (+- 0.366%)	0.919 (+- 0.295%)	0.780 (+- 0.640%)
ABTO	12.148 (+- 3.782%)	0.920 (+- 1.277%)	0.786 (+- 1.072%)
SV	13.860 (+- 1.546%)	0.870 (+- 0.542%)	0.554 (+- 0.847%)

Table A3.54: T_REQ = 5, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.630 (+- 0.773%)	15.874 (+- 0.775%)	400	400	0
PRE	0.296 (+- 1.205%)	33.829 (+- 1.196%)	400	400	0
2PLE	0.262 (+- 4.289%)	38.223 (+- 4.298%)	400	251	149
2PLU	0.059 (+-11.997%)	172.926 (+-11.603%)	400	62	337
A2PLU	0.178 (+- 1.608%)	56.257 (+- 1.620%)	400	185	215
BTO	0.051 (+- 6.182%)	197.779 (+- 6.463%)	400	41	359
ABTO	0.315 (+- 8.110%)	31.960 (+- 7.630%)	400	219	180
SV	0.203 (+- 4.232%)	49.347 (+- 4.265%)	400	132	267

Table A3.55: T_REQ = 10, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	15.874 (+- 0.775%)	0.917 (+- 0.184%)	0.662 (+- 1.177%)
PRE	33.829 (+- 1.196%)	1.358 (+- 0.861%)	1.959 (+- 1.467%)
2PLE	24.502 (+- 1.556%)	1.133 (+- 0.464%)	1.277 (+- 0.572%)
2PLU	33.451 (+- 4.427%)	1.191 (+- 0.522%)	1.675 (+- 2.162%)
A2PLU	10.932 (+- 4.873%)	1.094 (+- 0.835%)	1.497 (+- 0.682%)
BTO	20.732 (+- 1.485%)	1.089 (+- 0.687%)	1.072 (+- 0.455%)
ABTO	5.088 (+-11.751%)	1.048 (+- 0.963%)	1.144 (+- 2.127%)
SV	16.034 (+- 1.017%)	1.012 (+- 0.620%)	0.738 (+- 0.962%)

Table A3.56: T_REQ = 10, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	20.989 (+- 2.990%)	1.039 (+- 1.036%)	0.968 (+- 2.140%)
2PLU	23.384 (+- 0.740%)	1.129 (+- 0.322%)	1.298 (+- 0.714%)
A2PLU	13.086 (+- 2.383%)	1.088 (+- 0.379%)	1.210 (+- 0.744%)
BTO	18.248 (+- 0.447%)	1.037 (+- 0.189%)	0.927 (+- 0.536%)
ABTO	10.798 (+-19.478%)	1.030 (+- 0.503%)	0.945 (+- 2.304%)
SV	14.479 (+- 0.888%)	0.944 (+- 0.167%)	0.593 (+- 0.681%)

Table A3.57: T_REQ = 10, Wasted Work Costs

Summary Tables for GRAN = 1, T_REQ = 1, 2, 5, 10, T_TYP = select-replace

ET: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	110	110	116	116	109	110	106
2	115	116	130	128	116	115	115
5	143	131	247	176	204	172	177
10	213	241	1089	354	1246	201	311
CPU: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	102	103	106	106	104	103	105
2	105	104	114	112	109	109	114
5	118	113	198	162	183	159	180
10	148	191	807	257	1108	207	318
IO: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	120	120	127	127	115	116	107
2	128	127	148	148	127	128	115
5	173	149	297	240	226	197	172
10	296	280	1331	439	1386	291	293

Table A3.58: Overhead Summary (%)

ET: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	100	100	100	100	100	100	100
2	100	100	95	90	96	92	91
5	100	100	60	48	59	47	57
10	100	64	20	19	10	16	32
CPU: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	100	100	100	100	100	100	100
2	100	100	95	95	96	96	92
5	100	100	58	70	60	69	60
10	100	65	16	46	11	55	35
IO: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	100	100	100	100	100	100	100
2	100	100	95	95	96	96	93
5	100	100	61	73	62	71	62
10	100	69	19	52	12	60	38

Table A3.59: Efficiency Summary (%)

Experiment 1 Results: (GRAN = 10, T_TYP = select-replace)

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.734 (+- 0.701%)	13.633 (+- 0.706%)	400	400	0
PRE	0.627 (+- 0.258%)	15.947 (+- 0.257%)	400	400	0
2PLE	0.656 (+- 0.375%)	15.254 (+- 0.374%)	400	400	0
2PLU	0.517 (+- 1.175%)	19.335 (+- 1.167%)	400	329	71
A2PLU	0.570 (+- 2.054%)	17.539 (+- 2.056%)	400	349	50
BTO	0.555 (+- 1.436%)	18.030 (+- 1.452%)	400	330	70
ABTO	0.619 (+- 1.205%)	16.145 (+- 1.201%)	400	355	45
SV	0.669 (+- 0.696%)	14.937 (+- 0.690%)	400	400	0

Table A3.60: T_REQ = 1, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	13.633 (+- 0.706%)	0.801 (+- 0.060%)	0.553 (+- 0.406%)
PRE	15.947 (+- 0.257%)	0.841 (+- 0.245%)	0.729 (+- 0.368%)
2PLE	15.254 (+- 0.374%)	0.819 (+- 0.319%)	0.681 (+- 0.293%)
2PLU	15.886 (+- 0.533%)	0.846 (+- 0.532%)	0.720 (+- 0.640%)
A2PLU	13.567 (+- 1.540%)	0.836 (+- 0.350%)	0.718 (+- 0.422%)
BTO	14.490 (+- 0.373%)	0.826 (+- 0.315%)	0.644 (+- 0.477%)
ABTO	12.660 (+- 1.563%)	0.819 (+- 0.254%)	0.652 (+- 0.271%)
SV	14.937 (+- 0.690%)	0.845 (+- 0.208%)	0.628 (+- 0.218%)

Table A3.61: T_REQ = 1, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	13.978 (+- 2.647%)	0.838 (+- 0.677%)	0.667 (+- 0.601%)
A2PLU	11.878 (+- 3.994%)	0.843 (+- 0.873%)	0.664 (+- 1.642%)
BTO	14.669 (+- 0.825%)	0.820 (+- 1.130%)	0.631 (+- 1.610%)
ABTO	12.782 (+- 1.953%)	0.815 (+- 0.962%)	0.637 (+- 0.837%)
SV	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)

Table A3.62: T_REQ = 1, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.721 (+- 0.483%)	13.869 (+- 0.484%)	400	400	0
PRE	0.538 (+- 0.897%)	18.588 (+- 0.894%)	400	400	0
2PLE	0.562 (+- 1.555%)	17.783 (+- 1.539%)	400	385	15
2PLU	0.232 (+- 6.171%)	43.269 (+- 6.382%)	400	167	233
A2PLU	0.375 (+- 2.261%)	26.679 (+- 2.276%)	400	251	149
BTO	0.318 (+- 4.702%)	31.556 (+- 4.767%)	400	195	205
ABTO	0.485 (+- 2.429%)	20.613 (+- 2.431%)	400	271	128
SV	0.610 (+- 1.540%)	16.406 (+- 1.547%)	400	365	35

Table A3.63: T_REQ = 2, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	13.869 (+- 0.484%)	0.813 (+- 0.290%)	0.567 (+- 0.590%)
PRE	18.588 (+- 0.894%)	0.909 (+- 0.280%)	0.906 (+- 0.632%)
2PLE	17.135 (+- 0.505%)	0.865 (+- 0.726%)	0.790 (+- 0.616%)
2PLU	18.700 (+- 0.860%)	0.899 (+- 0.150%)	0.874 (+- 0.542%)
A2PLU	10.483 (+- 1.227%)	0.875 (+- 0.433%)	0.818 (+- 0.336%)
BTO	14.482 (+- 0.617%)	0.836 (+- 0.616%)	0.706 (+- 0.851%)
ABTO	8.764 (+- 2.166%)	0.839 (+- 0.508%)	0.704 (+- 0.594%)
SV	14.854 (+- 0.435%)	0.849 (+- 0.188%)	0.645 (+- 0.267%)

Table A3.64: T_REQ = 2, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	14.812 (+- 5.950%)	0.839 (+- 1.812%)	0.666 (+- 1.526%)
2PLU	15.473 (+- 1.283%)	0.891 (+- 0.610%)	0.786 (+- 0.594%)
A2PLU	10.349 (+- 3.640%)	0.872 (+- 0.253%)	0.758 (+- 1.024%)
BTO	14.188 (+- 0.345%)	0.827 (+- 0.479%)	0.670 (+- 0.488%)
ABTO	9.564 (+- 2.116%)	0.829 (+- 0.394%)	0.671 (+- 0.570%)
SV	14.124 (+- 3.705%)	0.823 (+- 0.823%)	0.535 (+- 1.610%)

Table A3.65: T_REQ = 2, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.680 (+- 0.137%)	14.705 (+- 0.137%)	400	400	0
PRE	0.398 (+- 1.180%)	25.135 (+- 1.181%)	400	400	0
2PLE	0.487 (+- 0.331%)	20.542 (+- 0.330%)	400	400	0
2PLU	0.030 (+- 6.420%)	331.516 (+- 6.902%)	400	26	374
A2PLU	0.252 (+-10.989%)	40.105 (+-11.417%)	400	196	204
BTO	0.056 (+-15.367%)	181.048 (+-14.179%)	400	38	362
ABTO	0.303 (+- 3.983%)	33.018 (+- 3.971%)	400	202	197
SV	0.368 (+- 1.462%)	27.187 (+- 1.470%)	400	228	172

Table A3.66: T_REQ = 5, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	14.705 (+- 0.137%)	0.849 (+- 0.285%)	0.611 (+- 0.745%)
PRE	25.135 (+- 1.181%)	1.076 (+- 0.410%)	1.344 (+- 1.864%)
2PLE	20.542 (+- 0.330%)	0.976 (+- 0.334%)	0.996 (+- 0.363%)
2PLU	24.533 (+- 3.807%)	1.013 (+- 1.210%)	1.188 (+- 1.747%)
A2PLU	5.247 (+-13.830%)	0.929 (+- 0.743%)	1.130 (+- 1.081%)
BTO	16.835 (+- 1.376%)	0.901 (+- 1.300%)	0.850 (+- 1.810%)
ABTO	3.983 (+- 3.920%)	0.882 (+- 0.492%)	0.935 (+- 0.437%)
SV	15.207 (+- 1.247%)	0.912 (+- 0.456%)	0.691 (+- 0.831%)

Table A3.67: T_REQ = 5, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	19.084 (+- 1.065%)	0.983 (+- 0.288%)	0.990 (+- 1.005%)
A2PLU	12.741 (+-17.137%)	0.936 (+- 0.461%)	0.932 (+- 0.653%)
BTO	14.975 (+- 0.547%)	0.881 (+- 0.170%)	0.770 (+- 0.567%)
ABTO	11.555 (+- 8.344%)	0.868 (+- 0.559%)	0.773 (+- 0.851%)
SV	13.877 (+- 1.956%)	0.873 (+- 0.471%)	0.558 (+- 0.913%)

Table A3.68: T_REQ = 5, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.630 (+- 0.773%)	15.874 (+- 0.775%)	400	400	0
PRE	0.355 (+- 0.564%)	28.135 (+- 0.565%)	400	400	0
2PLE	0.045 (+-18.368%)	230.411 (+-17.225%)	400	36	364
2PLU	0.001 (+- ---)	3749.384 (+- ---)	400	1	399
A2PLU	0.106 (+-87.316%)	34.199 (+-87.299%)	400	106	293
BTO	0.003 (+- ---)	4404.053 (+- ---)	400	2	397
ABTO	0.322 (+- 3.219%)	31.076 (+- 3.224%)	400	218	181
SV	0.099 (+- 4.446%)	101.289 (+- 4.249%)	400	62	338

Table A3.69: T_REQ = 10, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	15.874 (+- 0.775%)	0.917 (+- 0.184%)	0.662 (+- 1.177%)
PRE	28.135 (+- 0.565%)	1.177 (+- 0.267%)	1.499 (+- 0.310%)
2PLE	24.251 (+- 2.619%)	1.081 (+- 1.269%)	1.321 (+- 2.182%)
2PLU	19.160 (+- ---)	0.662 (+- ---)	0.990 (+- ---)
A2PLU	3.274 (+- ---)	0.619 (+- ---)	0.916 (+- ---)
BTO	21.041 (+-16.633%)	0.929 (+- 5.951%)	1.123 (+- 5.805%)
ABTO	3.319 (+- 3.171%)	0.963 (+- 0.594%)	1.201 (+- 0.710%)
SV	15.961 (+- 1.962%)	0.958 (+- 0.851%)	0.739 (+- 1.292%)

Table A3.70: T_REQ = 10, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	17.854 (+- 0.616%)	0.941 (+- 0.495%)	0.860 (+- 0.719%)
2PLU	21.478 (+- 0.461%)	1.043 (+- 0.479%)	1.161 (+- 0.525%)
A2PLU	18.544 (+-21.481%)	1.032 (+- 1.251%)	1.133 (+- 3.380%)
BTO	16.621 (+- 0.594%)	0.933 (+- 0.272%)	0.885 (+- 0.442%)
ABTO	11.732 (+- 4.125%)	0.931 (+- 0.489%)	0.902 (+- 0.913%)
SV	13.622 (+- 0.764%)	0.924 (+- 0.327%)	0.592 (+- 0.695%)

Table A3.71: T_REQ = 10, Wasted Work Costs

Summary Tables for GRAN = 10, T_REQ = 1, 2, 5, 10, T_TYP = select-replace

ET: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	117	112	142	129	132	118	110
2	134	128	312	192	228	149	118
5	171	140	2254	273	1231	225	185
10	177	1451	23620	215	27744	196	638
CPU: T_REQ	**PRE**	**2PLE**	**2PLU**	**A2PLU**	**BTO**	**ABTO**	**SV**
1	105	102	128	120	125	115	105
2	112	110	265	171	210	152	114
5	127	115	1804	225	1109	204	185
10	128	1184	18198	151	23991	189	655
IO: T_REQ	**PRE**	**2PLE**	**2PLU**	**A2PLU**	**BTO**	**ABTO**	**SV**
1	132	123	156	147	141	132	114
2	160	144	349	224	249	180	123
5	220	163	2554	345	1358	277	182
10	226	1550	28076	264	31665	295	600

Table A3.72: Overhead Summary (%)

ET: T_REQ	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
1	100	100	82	78	80	78	100
2	100	96	43	39	46	43	91
5	100	100	7	13	9	12	56
10	100	11	0	3	1	11	16
CPU: T_REQ	**PRE**	**2PLE**	**2PLU**	**A2PLU**	**BTO**	**ABTO**	**SV**
1	100	100	82	87	83	89	100
2	100	96	42	63	49	68	91
5	100	100	7	49	10	51	58
10	100	10	0	18	0	55	16
IO: T_REQ	**PRE**	**2PLE**	**2PLU**	**A2PLU**	**BTO**	**ABTO**	**SV**
1	100	100	83	88	83	89	100
2	100	97	44	65	50	69	93
5	100	100	8	54	10	55	62
10	100	13	0	23	1	62	19

Table A3.73: Efficiency Summary (%)

Experiment 2 Results: (T_TYP = select-replace)

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.501 (+- 0.109%)	19.949 (+- 0.109%)	400	400	0
PRE	0.420 (+- 0.287%)	23.813 (+- 0.287%)	400	400	0
2PLE	0.425 (+- 0.229%)	23.531 (+- 0.229%)	400	400	0
2PLU	0.419 (+- 0.140%)	23.849 (+- 0.140%)	400	400	0
A2PLU	0.425 (+- 0.339%)	23.511 (+- 0.339%)	400	400	0
BTO	0.439 (+- 0.144%)	22.773 (+- 0.144%)	400	400	0
ABTO	0.441 (+- 0.172%)	22.695 (+- 0.172%)	400	400	0
SV	0.445 (+- 0.408%)	22.491 (+- 0.408%)	400	400	0

Table A3.74: LC = 0%, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	19.949 (+- 0.109%)	1.331 (+- 0.313%)	0.640 (+- 0.644%)
PRE	23.813 (+- 0.287%)	1.389 (+- 0.260%)	0.962 (+- 0.574%)
2PLE	23.531 (+- 0.229%)	1.390 (+- 0.301%)	0.933 (+- 0.337%)
2PLU	23.849 (+- 0.140%)	1.388 (+- 0.272%)	0.960 (+- 0.415%)
A2PLU	23.511 (+- 0.339%)	1.371 (+- 0.238%)	0.947 (+- 0.702%)
BTO	22.773 (+- 0.144%)	1.362 (+- 0.257%)	0.883 (+- 0.449%)
ABTO	22.695 (+- 0.172%)	1.362 (+- 0.253%)	0.876 (+- 0.340%)
SV	22.491 (+- 0.408%)	1.361 (+- 0.206%)	0.864 (+- 0.296%)

Table A3.75: LC = 0%, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
A2PLU	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
BTO	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
ABTO	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
SV	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)

Table A3.76: LC = 0%, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.501 (+- 0.109%)	19.949 (+- 0.109%)	400	400	0
PRE	0.417 (+- 0.210%)	23.999 (+- 0.210%)	400	400	0
2PLE	0.425 (+- 0.179%)	23.515 (+- 0.180%)	400	400	0
2PLU	0.381 (+- 0.387%)	26.215 (+- 0.386%)	400	361	39
A2PLU	0.408 (+- 0.672%)	24.504 (+- 0.668%)	400	381	19
BTO	0.384 (+- 0.515%)	26.042 (+- 0.519%)	400	344	56
ABTO	0.428 (+- 0.382%)	23.359 (+- 0.382%)	400	380	20
SV	0.410 (+- 0.846%)	24.363 (+- 0.845%)	400	362	38

Table A3.77: LC = 20%, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	19.949 (+- 0.109%)	1.331 (+- 0.313%)	0.640 (+- 0.644%)
PRE	23.999 (+- 0.210%)	1.397 (+- 0.290%)	0.971 (+- 0.315%)
2PLE	23.515 (+- 0.180%)	1.390 (+- 0.160%)	0.934 (+- 0.447%)
2PLU	23.707 (+- 0.141%)	1.386 (+- 0.373%)	0.955 (+- 0.614%)
A2PLU	22.311 (+- 0.408%)	1.382 (+- 0.305%)	0.951 (+- 0.572%)
BTO	22.523 (+- 0.196%)	1.358 (+- 0.327%)	0.875 (+- 0.375%)
ABTO	21.093 (+- 0.073%)	1.357 (+- 0.179%)	0.871 (+- 0.353%)
SV	22.057 (+- 0.306%)	1.362 (+- 0.347%)	0.845 (+- 0.625%)

Table A3.78: LC = 20%, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	21.207 (+- 1.412%)	1.383 (+- 1.032%)	0.815 (+- 1.791%)
A2PLU	20.160 (+- 1.574%)	1.399 (+- 1.228%)	0.757 (+- 3.277%)
BTO	19.609 (+- 0.873%)	1.349 (+- 0.782%)	0.713 (+- 1.675%)
ABTO	20.332 (+- 2.787%)	1.339 (+- 1.035%)	0.800 (+- 1.525%)
SV	20.104 (+- 1.983%)	1.346 (+- 0.848%)	0.598 (+- 2.657%)

Table A3.79: LC = 20%, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.501 (+- 0.109%)	19.949 (+- 0.109%)	400	400	0
PRE	0.426 (+- 0.975%)	23.470 (+- 0.971%)	400	400	0
2PLE	0.426 (+- 1.200%)	23.454 (+- 1.186%)	400	400	0
2PLU	0.262 (+- 0.482%)	38.187 (+- 0.480%)	400	241	159
A2PLU	0.393 (+- 1.288%)	25.447 (+- 1.270%)	400	315	84
BTO	0.226 (+- 1.550%)	44.225 (+- 1.559%)	400	196	204
ABTO	0.419 (+- 2.175%)	23.882 (+- 2.162%)	400	313	87
SV	0.291 (+- 1.078%)	34.340 (+- 1.089%)	400	244	155

Table A3.80: LC = 50%, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	19.949 (+- 0.109%)	1.331 (+- 0.313%)	0.640 (+- 0.644%)
PRE	23.470 (+- 0.971%)	1.436 (+- 0.210%)	1.057 (+- 0.643%)
2PLE	23.454 (+- 1.186%)	1.414 (+- 0.371%)	1.013 (+- 0.596%)
2PLU	23.415 (+- 0.517%)	1.378 (+- 0.253%)	0.945 (+- 0.825%)
A2PLU	15.615 (+- 0.886%)	1.376 (+- 0.247%)	0.942 (+- 0.230%)
BTO	22.314 (+- 0.317%)	1.363 (+- 0.249%)	0.876 (+- 0.460%)
ABTO	13.823 (+- 1.518%)	1.353 (+- 0.197%)	0.842 (+- 0.647%)
SV	21.440 (+- 0.738%)	1.357 (+- 0.241%)	0.820 (+- 0.321%)

Table A3.81: LC = 50%, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	20.387 (+- 1.137%)	1.384 (+- 0.627%)	0.759 (+- 1.242%)
A2PLU	13.476 (+- 4.109%)	1.382 (+- 0.621%)	0.748 (+- 0.850%)
BTO	19.000 (+- 0.645%)	1.345 (+- 0.244%)	0.679 (+- 0.550%)
ABTO	14.137 (+- 4.806%)	1.337 (+- 0.630%)	0.717 (+- 2.103%)
SV	18.301 (+- 0.552%)	1.338 (+- 0.233%)	0.584 (+- 0.421%)

Table A3.82: LC = 50%, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.501 (+- 0.109%)	19.949 (+- 0.109%)	400	400	0
PRE	0.405 (+- 0.546%)	24.681 (+- 0.543%)	400	400	0
2PLE	0.400 (+- 0.673%)	25.018 (+- 0.676%)	400	400	0
2PLU	0.138 (+- 2.031%)	72.269 (+- 2.050%)	400	124	275
A2PLU	0.337 (+- 7.443%)	29.858 (+- 7.488%)	400	244	156
BTO	0.093 (+- 1.418%)	106.995 (+- 1.436%)	400	78	322
ABTO	0.365 (+- 2.885%)	27.387 (+- 2.866%)	400	246	153
SV	0.153 (+- 2.314%)	65.502 (+- 2.297%)	400	122	277

Table A3.83: LC = 80%, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	19.949 (+- 0.109%)	1.331 (+- 0.313%)	0.640 (+- 0.644%)
PRE	24.681 (+- 0.543%)	1.497 (+- 0.262%)	1.130 (+- 0.537%)
2PLE	25.018 (+- 0.676%)	1.436 (+- 0.274%)	1.125 (+- 1.520%)
2PLU	23.647 (+- 1.049%)	1.384 (+- 0.528%)	0.924 (+- 1.543%)
A2PLU	8.250 (+- 5.556%)	1.367 (+- 0.277%)	0.915 (+- 0.685%)
BTO	21.729 (+- 0.997%)	1.346 (+- 0.731%)	0.851 (+- 0.939%)
ABTO	6.366 (+- 1.490%)	1.341 (+- 0.270%)	0.838 (+- 0.981%)
SV	20.890 (+- 0.901%)	1.348 (+- 0.179%)	0.778 (+- 1.376%)

Table A3.84: LC = 80%, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	19.934 (+- 0.683%)	1.381 (+- 0.587%)	0.754 (+- 0.767%)
A2PLU	11.042 (+-23.524%)	1.377 (+- 0.109%)	0.740 (+- 0.680%)
BTO	18.585 (+- 0.241%)	1.340 (+- 0.308%)	0.671 (+- 0.891%)
ABTO	11.960 (+-10.951%)	1.336 (+- 0.471%)	0.696 (+- 1.464%)
SV	17.658 (+- 0.522%)	1.334 (+- 0.110%)	0.573 (+- 0.395%)

Table A3.85: LC = 80%, Wasted Work Costs

CCA	T	ET	RUNS	COMM	RSRT
NO CC	0.501 (+- 0.109%)	19.949 (+- 0.109%)	400	400	0
PRE	0.343 (+- 1.520%)	29.183 (+- 1.509%)	400	400	0
2PLE	0.338 (+- 0.720%)	29.597 (+- 0.717%)	400	400	0
2PLU	0.047 (+- 2.152%)	213.192 (+- 2.132%)	400	42	358
A2PLU	0.309 (+- 9.635%)	32.597 (+- 9.708%)	400	230	170
BTO	0.003 (+- ---)	3132.785 (+- ---)	400	2	398
ABTO	0.316 (+- 8.848%)	31.805 (+- 8.422%)	400	220	180
SV	0.059 (+- 3.990%)	168.482 (+- 3.927%)	400	46	354

Table A3.86: LC = 100%, Throughput, ET and run counts

CCA	ET	CPU	IO
NO CC	19.949 (+- 0.109%)	1.331 (+- 0.313%)	0.640 (+- 0.644%)
PRE	29.183 (+- 1.509%)	1.534 (+- 0.255%)	1.226 (+- 1.405%)
2PLE	29.597 (+- 0.717%)	1.454 (+- 0.123%)	1.225 (+- 1.166%)
2PLU	25.846 (+- 5.437%)	1.357 (+- 0.800%)	0.849 (+- 1.563%)
A2PLU	4.352 (+- 7.127%)	1.337 (+- 0.705%)	0.890 (+- 0.949%)
BTO	15.254 (+- ---)	1.066 (+- ---)	0.577 (+- ---)
ABTO	3.435 (+- 2.747%)	1.332 (+- 0.263%)	0.816 (+- 0.640%)
SV	18.710 (+- 1.331%)	1.330 (+- 0.356%)	0.694 (+- 1.372%)

Table A3.87: LC = 100%, Useful Work Costs

CCA	ET	CPU	IO
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	19.854 (+- 0.637%)	1.367 (+- 0.459%)	0.745 (+- 0.653%)
A2PLU	15.725 (+- 4.318%)	1.366 (+- 0.222%)	0.732 (+- 0.735%)
BTO	18.429 (+- 0.188%)	1.339 (+- 0.258%)	0.665 (+- 0.490%)
ABTO	13.160 (+- 5.501%)	1.331 (+- 0.317%)	0.687 (+- 0.771%)
SV	17.527 (+- 0.231%)	1.329 (+- 0.186%)	0.569 (+- 0.481%)

Table A3.88: LC = 100%, Wasted Work Costs

Summary Tables for LC = 0%, 20%, 50%, 80%, 100%, T_TYP = select-replace

ET: LC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
0%	119	118	120	118	114	114	113
20%	120	118	131	123	131	117	122
50%	118	118	191	128	222	120	172
80%	124	125	362	150	536	137	328
100%	146	148	1069	163	15704	159	845
CPU: LC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
0%	104	104	104	103	102	102	102
20%	105	104	115	109	119	107	113
50%	108	106	172	131	208	130	166
80%	112	108	334	169	518	163	329
100%	115	109	982	177	15452	182	866
IO: LC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
0%	150	146	150	148	138	137	135
20%	152	146	163	155	155	143	142
50%	165	158	226	179	248	163	186
80%	177	176	406	217	567	199	325
100%	192	191	1131	224	15877	216	790

Table A3.89: Overhead Summary (%)

ET: LC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
0%	100	100	100	100	100	100	100
20%	100	100	90	91	86	90	90
50%	100	100	61	61	51	58	62
80%	100	100	33	28	20	23	32
100%	100	100	12	13	0	11	11
CPU: LC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
0%	100	100	100	100	100	100	100
20%	100	100	90	95	86	95	91
50%	100	100	60	79	49	78	61
80%	100	100	31	61	20	62	31
100%	100	100	10	57	0	55	11
IO: LC	PRE	2PLE	2PLU	A2PLU	BTO	ABTO	SV
0%	100	100	100	100	100	100	100
20%	100	100	92	96	88	95	93
50%	100	100	65	83	55	81	69
80%	100	100	36	66	23	66	37
100%	100	100	12	62	0	59	14

Table A3.90: Efficiency Summary (%)

Appendix 4

SIMULATION RESULTS

Experiment 1: Results

CCA	T	ET	COMM	BLCK	RSRT
NO CC	20.000 (+- 0.000%)	0.500	2000	0	0
PRE	17.800 (+- 0.030%)	0.562	1780	94	0
2PLE	17.760 (+- 0.059%)	0.563	1776	125	0
2PLU	16.270 (+- 0.270%)	0.615	1626	102	52
A2PLU	16.410 (+- 0.232%)	0.609	1641	114	58
BTO	16.650 (+- 0.015%)	0.601	1664	0	55
XBTO	16.650 (+- 0.097%)	0.601	1665	0	55
ABTO	16.640 (+- 0.100%)	0.601	1664	0	58
SV	18.460 (+- 0.080%)	0.542	1846	0	76

Table A4.1: GRAN = 1, T_REQ = 1, Throughput and run counts

CCA	T	ET	COMM	BLCK	RSRT
NO CC	10.000 (+- 0.000%)	1.000	1000	0	0
PRE	9.069 (+- 0.083%)	1.103	906	197	0
2PLE	8.769 (+- 0.113%)	1.140	876	260	0
2PLU	7.786 (+- 0.658%)	1.284	778	187	63
A2PLU	7.869 (+- 0.716%)	1.271	786	202	70
BTO	8.390 (+- 0.038%)	1.192	839	0	67
XBTO	8.365 (+- 0.205%)	1.195	836	0	73
ABTO	8.381 (+- 0.040%)	1.193	838	0	70
SV	9.052 (+- 0.164%)	1.105	905	0	127

Table A4.2: GRAN = 1, T_REQ = 2, Throughput and run counts

CCA	T	ET	COMM	BLCK	RSRT
NO CC	4.000 (+- 0.000%)	2.500	400	0	0
PRE	3.425 (+- 0.418%)	2.920	342	309	0
2PLE	1.976 (+- 0.827%)	5.061	197	414	12
2PLU	1.952 (+- 0.936%)	5.123	195	337	63
A2PLU	2.008 (+- 2.643%)	4.980	200	314	58
BTO	3.077 (+- 0.360%)	3.250	307	0	114
XBTO	3.017 (+- 0.705%)	3.315	301	0	131
ABTO	3.074 (+- 0.141%)	3.253	307	0	114
SV	2.974 (+- 0.526%)	3.362	297	0	193

Table A4.3: GRAN = 1, T_REQ = 5, Throughput and run counts

CCA	T	ET	COMM	BLCK	RSRT
NO CC	2.000 (+- 0.000%)	5.000	200	0	0
PRE	1.397 (+- 0.218%)	7.158	139	377	0
2PLE	0.481 (+- 3.338%)	20.812	48	421	40
2PLU	0.442 (+- 4.241%)	22.599	44	393	62
A2PLU	0.536 (+- 3.474%)	18.657	53	295	48
BTO	1.074 (+- 1.582%)	9.311	107	0	193
XBTO	0.860 (+- 0.637%)	11.621	86	0	256
ABTO	1.140 (+- 1.657%)	8.772	113	0	163
SV	0.980 (+- 1.101%)	10.204	98	0	200

Table A4.4: GRAN = 1, T_REQ = 10, Throughput and run counts

CCA	T	ET	COMM	BLCK	RSRT
NO CC	20.000 (+- 0.000%)	0.500	2000	0	0
PRE	15.770 (+- 0.532%)	0.634	1577	343	0
2PLE	8.456 (+- 1.617%)	1.183	845	437	0
2PLU	6.613 (+- 0.592%)	1.512	661	283	156
A2PLU	6.688 (+- 1.616%)	1.495	668	363	203
BTO	14.560 (+- 0.070%)	0.687	1455	0	268
XBTO	14.530 (+- 0.072%)	0.688	1452	0	389
ABTO	14.120 (+- 0.290%)	0.708	1412	0	520
SV	16.570 (+- 0.079%)	0.604	1657	0	311

Table A4.5: GRAN = 10, T_REQ = 1, Throughput and run counts

CCA	T	ET	COMM	BLCK	RSRT
NO CC	10.000 (+- 0.000%)	1.000	1000	0	0
PRE	6.775 (+- 0.158%)	1.476	677	399	0
2PLE	1.685 (+- 3.234%)	5.935	168	466	13
2PLU	1.521 (+- 0.788%)	6.575	152	342	131
A2PLU	1.494 (+- 0.670%)	6.693	149	331	125
BTO	6.690 (+- 0.533%)	1.495	669	0	299
XBTO	6.354 (+- 0.396%)	1.574	635	0	472
ABTO	6.010 (+- 0.521%)	1.664	601	0	591
SV	7.239 (+- 0.103%)	1.381	723	0	348

Table A4.6: GRAN = 10, T_REQ = 2, Throughput and run counts

CCA	T	ET	COMM	BLCK	RSRT
NO CC	4.000 (+- 0.000%)	2.500	400	0	0
PRE	2.173 (+- 0.447%)	4.602	217	432	0
2PLE	0.234 (+- 6.497%)	42.827	23	423	65
2PLU	0.169 (+- 2.144%)	58.997	16	368	117
A2PLU	0.255 (+- 1.538%)	39.139	25	266	91
BTO	1.687 (+- 1.138%)	5.928	168	0	359
XBTO	1.218 (+- 1.835%)	8.210	121	0	624
ABTO	1.192 (+- 0.555%)	8.389	119	0	655
SV	2.005 (+- 0.561%)	4.988	200	0	346

Table A4.7: GRAN = 10, T_REQ = 5, Throughput and run counts

CCA	T	ET	COMM	BLCK	RSRT
NO CC	2.000 (+- 0.000%)	5.000	200	0	0
PRE	1.045 (+- 0.190%)	9.569	104	440	0
2PLE	0.032 (+-22.720%)	312.500	3	408	81
2PLU	0.024 (+-19.480%)	408.163	2	369	118
A2PLU	0.044 (+-28.560%)	227.273	4	324	111
BTO	0.458 (+- 1.709%)	21.858	45	0	383
XBTO	0.045 (+- 5.656%)	222.222	4	0	675
ABTO	0.370 (+-11.650%)	26.991	37	0	538
SV	0.649 (+- 2.502%)	15.420	64	0	263

Table A4.8: GRAN = 10, T_REQ = 10, Throughput and run counts

Summary Tables for GRAN = 1, T_REQ = 1, 2, 5, 10

CCA	1	2	5
NO CC	0.500 (+- 0.000%)	1.000 (+- 0.000%)	2.500 (+- 0.000%)
PRE	0.562 (+- 0.094%)	1.103 (+- 0.152%)	2.920 (+- 0.536%)
2PLE	0.563 (+- 0.149%)	1.139 (+- 0.552%)	4.276 (+- 2.645%)
2PLU	0.522 (+- 0.121%)	0.992 (+- 0.907%)	3.161 (+- 0.775%)
A2PLU	0.560 (+- 0.364%)	1.029 (+- 0.147%)	3.013 (+- 2.427%)
BTO	0.523 (+- 0.102%)	0.986 (+- 0.159%)	2.098 (+- 0.155%)
XBTO	0.555 (+- 0.347%)	1.054 (+- 0.464%)	2.339 (+- 0.777%)
ABTO	0.569 (+- 0.168%)	1.055 (+- 0.096%)	2.073 (+- 0.250%)
SV	0.449 (+- 0.659%)	0.765 (+- 0.787%)	1.530 (+- 0.951%)

CCA	10
NO CC	5.000 (+- 0.000%)
PRE	7.165 (+- 1.553%)
2PLE	5.547 (+- 1.140%)
2PLU	8.615 (+- 0.916%)
A2PLU	6.791 (+- 3.619%)
BTO	3.029 (+- 1.504%)
XBTO	3.506 (+- 1.305%)
ABTO	2.790 (+- 1.313%)
SV	2.958 (+- 0.869%)

Table A4.9: Useful Work Costs ET

CCA	1	2	5
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.506 (+-14.070%)
2PLU	0.851 (+- 5.009%)	1.615 (+- 1.114%)	4.116 (+- 0.980%)
A2PLU	0.833 (+- 4.679%)	1.689 (+- 3.332%)	3.905 (+- 4.041%)
BTO	0.015 (+-17.740%)	0.051 (+- 8.738%)	0.302 (+- 0.746%)
XBTO	0.017 (+- 5.547%)	0.051 (+- 3.108%)	0.374 (+- 0.965%)
ABTO	0.342 (+- 0.228%)	0.609 (+- 0.429%)	1.089 (+- 1.542%)
SV	0.239 (+- 0.868%)	0.409 (+- 0.975%)	0.822 (+- 1.052%)

CCA	10
NO CC	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)
2PLE	4.859 (+- 2.737%)
2PLU	7.916 (+- 2.734%)
A2PLU	6.625 (+- 4.628%)
BTO	0.955 (+- 1.784%)
XBTO	1.277 (+- 0.993%)
ABTO	1.446 (+- 1.313%)
SV	1.543 (+- 0.162%)

Table A4.10: Wasted Work Costs ET

Summary Tables for GRAN = 10, T_REQ = 1, 2, 5, 10

CCA	1	2	5
NO CC	0.500 (+- 0.000%)	1.000 (+- 0.000%)	2.500 (+- 0.000%)
PRE	0.634 (+- 0.583%)	1.476 (+- 0.670%)	4.627 (+- 1.861%)
2PLE	1.185 (+- 1.641%)	4.884 (+- 4.156%)	3.056 (+-14.490%)
2PLU	0.635 (+- 0.858%)	2.271 (+- 0.659%)	7.231 (+- 5.633%)
A2PLU	0.720 (+- 1.272%)	2.202 (+- 0.704%)	5.292 (+- 3.563%)
BTO	0.284 (+- 0.178%)	0.473 (+- 0.892%)	0.803 (+- 0.256%)
XBTO	0.359 (+- 0.842%)	0.581 (+- 0.780%)	0.901 (+- 0.978%)
ABTO	0.442 (+- 0.340%)	0.655 (+- 0.494%)	0.922 (+- 0.209%)
SV	0.204 (+- 0.267%)	0.323 (+- 0.808%)	0.760 (+- 0.547%)

CCA	10
NO CC	5.000 (+- 0.000%)
PRE	9.557 (+- 6.319%)
2PLE	0.565 (+-20.810%)
2PLU	9.743 (+- 3.689%)
A2PLU	6.955 (+- 5.983%)
BTO	1.226 (+- 1.092%)
XBTO	1.518 (+- 2.321%)
ABTO	1.336 (+- 0.248%)
SV	2.208 (+- 2.871%)

Table A4.11: Useful Work Costs ET

CCA	1	2	5
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.650 (+-10.060%)	6.860 (+- 5.181%)
2PLU	1.715 (+- 0.438%)	3.013 (+- 0.738%)	5.472 (+- 0.686%)
A2PLU	1.822 (+- 0.882%)	3.086 (+- 2.994%)	4.639 (+- 2.914%)
BTO	0.029 (+- 1.967%)	0.089 (+- 1.904%)	0.274 (+- 1.285%)
XBTO	0.049 (+- 0.304%)	0.144 (+- 0.908%)	0.356 (+- 0.237%)
ABTO	0.277 (+- 0.488%)	0.369 (+- 0.175%)	0.433 (+- 0.878%)
SV	0.126 (+- 0.231%)	0.197 (+- 1.208%)	0.443 (+- 0.990%)

CCA	10
NO CC	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)
2PLE	9.449 (+- 0.863%)
2PLU	6.218 (+- 0.746%)
A2PLU	5.654 (+- 1.035%)
BTO	0.409 (+- 0.311%)
XBTO	0.467 (+- 0.249%)
ABTO	0.470 (+- 0.933%)
SV	1.249 (+- 0.153%)

Table A4.12: Wasted Work Costs ET

CCA	T	ET	COMM	BLCK	RSRT
NO CC	20.000 (+- 0.000%)	0.500	2000	0	0
PRE	17.860 (+- 0.064%)	0.560	1785	0	0
2PLE	17.860 (+- 0.064%)	0.560	1785	0	0
2PLU	16.950 (+- 0.046%)	0.590	1695	0	0
A2PLU	16.950 (+- 0.046%)	0.590	1695	0	0
BTO	16.950 (+- 0.046%)	0.590	1695	0	0
XBTO	16.950 (+- 0.046%)	0.590	1695	0	0
ABTO	16.950 (+- 0.046%)	0.590	1695	0	0
SV	18.870 (+- 0.051%)	0.530	1886	0	0

Table A4.13: LC = 0%, Throughput and run counts

CCA	T	ET	COMM	BLCK	RSRT
NO CC	20.000 (+- 0.000%)	0.500	2000	0	0
PRE	17.830 (+- 0.050%)	0.561	1783	45	0
2PLE	17.820 (+- 0.007%)	0.561	1781	76	0
2PLU	16.770 (+- 0.027%)	0.596	1677	43	29
A2PLU	16.720 (+- 0.033%)	0.598	1671	69	37
BTO	16.730 (+- 0.018%)	0.598	1672	0	42
XBTO	16.950 (+- 0.070%)	0.590	1695	0	0
ABTO	16.950 (+- 0.009%)	0.590	1694	0	0
SV	18.640 (+- 0.060%)	0.536	1864	0	42

Table A4.14: LC = 20%, Throughput and run counts

CCA	T	ET	COMM	BLCK	RSRT
NO CC	20.000 (+- 0.000%)	0.500	2000	0	0
PRE	17.720 (+- 0.015%)	0.564	1771	191	0
2PLE	17.710 (+- 0.022%)	0.565	1770	231	0
2PLU	15.950 (+- 0.040%)	0.627	1595	127	98
A2PLU	15.600 (+- 0.042%)	0.641	1560	186	162
BTO	15.870 (+- 0.041%)	0.630	1587	0	176
XBTO	15.130 (+- 0.046%)	0.661	1512	0	328
ABTO	13.530 (+- 0.017%)	0.739	1352	0	626
SV	17.920 (+- 0.028%)	0.558	1792	0	178

Table A4.15: LC = 50%, Throughput and run counts

CCA	T	ET	COMM	BLCK	RSRT
NO CC	20.000 (+- 0.000%)	0.500	2000	0	0
PRE	17.670 (+- 0.003%)	0.566	1767	343	0
2PLE	14.060 (+- 0.050%)	0.711	1405	388	0
2PLU	12.370 (+- 0.234%)	0.808	1236	210	172
A2PLU	11.950 (+- 0.030%)	0.837	1194	333	299
BTO	14.240 (+- 0.027%)	0.702	1423	0	301
XBTO	12.660 (+- 0.148%)	0.790	1265	0	598
ABTO	8.857 (+- 0.500%)	1.129	442	0	703
SV	17.110 (+- 0.019%)	0.584	1711	0	332

Table A4.16: LC = 80%, Throughput and run counts

CCA	T	ET	COMM	BLCK	RSRT
NO CC	20.000 (+- 0.000%)	0.500	2000	0	0
PRE	9.961 (+- 0.002%)	1.004	996	448	0
2PLE	0.820 (+- 0.684%)	12.195	82	492	0
2PLU	0.434 (+- 0.448%)	23.041	43	43	390
A2PLU	0.475 (+- 0.236%)	21.053	47	259	218
BTO	7.835 (+- 0.246%)	1.276	783	0	436
XBTO	7.378 (+- 0.170%)	1.355	737	0	831
ABTO	0.554 (+-16.470%)	18.051	27	0	1392
SV	9.995 (+- 0.027%)	1.001	999	0	428

Table A4.17: LC = 100%, Throughput and run counts

Summary Tables for LC = 0%, 20%, 50%, 80%, 100%

CCA	0%	20%	50%
NO CC	0.500 (+- 0.000%)	0.500 (+- 0.000%)	0.500 (+- 0.000%)
PRE	0.560 (+- 0.000%)	0.561 (+- 0.114%)	0.564 (+- 0.508%)
2PLE	0.560 (+- 0.000%)	0.561 (+- 0.043%)	0.565 (+- 0.186%)
2PLU	0.590 (+- 0.001%)	0.537 (+- 0.056%)	0.379 (+- 0.154%)
A2PLU	0.590 (+- 0.001%)	0.539 (+- 0.044%)	0.385 (+- 0.065%)
BTO	0.590 (+- 0.001%)	0.539 (+- 0.020%)	0.378 (+- 0.006%)
XBTO	0.590 (+- 0.001%)	0.590 (+- 0.001%)	0.393 (+- 0.023%)
ABTO	0.590 (+- 0.001%)	0.590 (+- 0.003%)	0.421 (+- 0.012%)
SV	0.530 (+- 0.000%)	0.484 (+- 0.005%)	0.338 (+- 0.015%)

CCA	80%	100%
NO CC	0.500 (+- 0.000%)	0.500 (+- 0.000%)
PRE	0.170 (+- 0.000%)	0.301 (+- 0.212%)
2PLE	0.711 (+- 0.348%)	12.190 (+- 0.371%)
2PLU	0.249 (+- 0.340%)	2.330 (+- 0.000%)
A2PLU	0.255 (+- 0.108%)	2.188 (+- 0.692%)
BTO	0.210 (+- 0.030%)	0.115 (+- 0.027%)
XBTO	0.233 (+- 0.046%)	0.125 (+- 0.050%)
ABTO	0.300 (+- 0.229%)	0.150 (+- 2.296%)
SV	0.175 (+- 0.018%)	0.102 (+- 0.025%)

Table A4.18: Useful Work Costs ET

CCA	0%	20%	50%
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	0.000 (+- 0.000%)	1.339 (+- 0.970%)	1.999 (+- 1.890%)
A2PLU	0.000 (+- 0.000%)	2.100 (+- 0.460%)	2.071 (+- 0.508%)
BTO	0.000 (+- 0.000%)	0.015 (+- 4.042%)	0.028 (+- 0.827%)
XBTO	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.043 (+- 0.823%)
ABTO	0.000 (+- 0.000%)	0.000 (+- 0.000%)	0.264 (+- 0.113%)
SV	0.000 (+- 0.000%)	0.283 (+- 0.031%)	0.211 (+- 0.013%)

CCA	80%	100%
NO CC	0.000 (+- 0.000%)	0.000 (+- 0.000%)
PRE	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLE	0.000 (+- 0.000%)	0.000 (+- 0.000%)
2PLU	2.019 (+- 1.081%)	0.300 (+- 0.000%)
A2PLU	2.064 (+- 0.222%)	1.991 (+- 0.235%)
BTO	0.057 (+- 0.737%)	0.029 (+- 0.645%)
XBTO	0.057 (+- 0.342%)	0.048 (+- 0.976%)
ABTO	0.218 (+- 1.402%)	0.141 (+- 1.207%)
SV	0.104 (+- 0.001%)	0.096 (+- 0.008%)

Table A4.19: Wasted Work Costs ET

Comparative Tables for Adaptive Restarts

	GRAN = 1 (100 Granules)			
T_REQ	2PLU	<A2PLU	=A2PLU	>A2PLU
1	16.270	16.420	16.410	16.130
2	7.786	7.971	7.869	7.536
5	1.952	1.911	2.008	2.109
10	0.442	0.397	0.536	0.756
	GRAN = 10 (10 Granules)			
T_REQ	2PLU	<A2PLU	=A2PLU	>A2PLU
1	6.613	6.787	6.688	6.523
2	1.521	1.303	1.494	2.047
5	0.169	0.138	0.255	0.661
10	0.024	0.021	0.044	0.442
LC	2PLU	<A2PLU	=A2PLU	>A2PLU
0%	16.950	16.950	16.950	16.950
20%	16.770	16.710	16.720	16.850
50%	15.950	15.580	15.600	16.120
80%	12.370	11.910	11.950	12.520
100%	0.434	0.480	0.475	1.033

Table A4.20: Adaptive Restart 2PLU: T

	GRAN = 1 (100 Granules)			
T_REQ	2PLU	<A2PLU	=A2PLU	>A2PLU
1	52 (106)	62 (122)	58 (114)	46 (87)
2	63 (187)	74 (224)	70 (202)	44 (129)
5	63 (337)	73 (398)	58 (314)	30 (143)
10	62 (393)	67 (448)	48 (295)	20 (113)
	GRAN = 10 (10 Granules)			
T_REQ	2PLU	<A2PLU	=A2PLU	>A2PLU
1	156 (283)	239 (429)	203 (363)	106 (191)
2	131 (342)	170 (465)	125 (331)	61 (151)
5	117 (368)	137 (455)	91 (266)	41 (103)
10	118 (369)	131 (407)	111 (324)	35 (74)
LC	2PLU	<A2PLU	=A2PLU	>A2PLU
0%	0 (0)	0 (0)	0 (0)	0 (0)
20%	29 (43)	37 (79)	37 (69)	16 (19)
50%	98 (127)	164 (215)	162 (186)	67 (105)
80%	172 (210)	313 (366)	299 (333)	157 (195)
100%	390 (43)	412 (446)	218 (254)	87 (119)

Table A4.21: Adaptive Restart 2PLU: RC (BC)

	GRAN = 1 (100 Granules)				
T_REQ	BTO	XBTO	<ABTO	=ABTO	>ABTO
1	16.650	16.650	16.660	16.640	16.620
2	8.390	8.365	8.327	8.381	8.376
5	3.077	3.017	2.903	3.074	3.318
10	1.074	0.860	0.521	1.140	1.300
	GRAN = 10 (10 Granules)				
T_REQ	BTO	XBTO	<ABTO	=ABTO	>ABTO
1	14.560	14.530	13.400	14.120	14.380
2	6.690	6.354	4.307	6.010	6.516
5	1.687	1.218	0.116	1.192	1.973
10	0.458	0.045	0.023	0.370	0.877
LC	BTO	XBTO	<ABTO	=ABTO	>ABTO
0%	16.950	16.950	16.950	16.950	16.950
20%	16.730	16.950	16.950	16.950	16.950
50%	15.870	15.130	10.150	13.350	16.200
80%	14.240	12.660	4.373	8.857	13.870
100%	7.835	7.378	0.190	0.554	7.389

Table A4.22: Adaptive Restart BTO : T

	GRAN = 1 (100 Granules)				
T_REQ	BTO	XBTO	<ABTO	=ABTO	>ABTO
1	55	55	54	58	47
2	67	73	81	70	47
5	114	131	161	114	47
10	193	256	388	163	45
	GRAN = 10 (10 Granules)				
T_REQ	BTO	XBTO	<ABTO	=ABTO	>ABTO
1	268	389	675	520	228
2	299	472	1065	591	184
5	359	624	1364	655	127
10	383	675	843	538	83
LC	BTO	XBTO	<ABTO	=ABTO	>ABTO
0%	0	0	0	0	0
20%	42	0	0	0	0
50%	176	328	1292	626	100
80%	301	598	2369	1403*	307
100%	436	831	3042	2784*	694

Table A4.23: Adaptive Restart BTO : RC

* Estimated from a run with batch length of 50000 simulation units.

	GRAN = 1 (100 Granules)			
T_REQ	2PLU	<A2PLU	=A2PLU	>A2PLU
1	0.615	0.609	0.609	0.620
2	1.284	1.255	1.271	1.327
5	5.123	5.233	4.980	4.742
10	22.599	25.189	18.657	13.228
	GRAN = 10 (10 Granules)			
T_REQ	2PLU	<A2PLU	=A2PLU	>A2PLU
1	1.512	1.473	1.495	1.533
2	6.575	7.675	6.693	4.885
5	58.997	72.464	39.139	15.129
10	408.163	476.190	227.273	22.624
LC	2PLU	<A2PLU	=A2PLU	>A2PLU
0%	0.590	0.590	0.590	0.590
20%	0.596	0.598	0.598	0.593
50%	0.627	0.642	0.641	0.620
80%	0.808	0.840	0.837	0.799
100%	23.041	20.833	21.053	9.681

Table A4.24: Adaptive Restart 2PLU: ET

	GRAN = 1 (100 Granules)				
T_REQ	BTO	XBTO	<ABTO	=ABTO	>ABTO
1	0.601	0.601	0.601	0.601	0.602
2	1.192	1.195	1.201	1.193	1.194
5	3.250	3.315	3.445	3.253	3.014
10	9.311	11.621	19.194	8.772	7.692
	GRAN = 10 (10 Granules)				
T_REQ	BTO	XBTO	<ABTO	=ABTO	>ABTO
1	0.687	0.688	0.746	0.708	0.695
2	1.495	1.574	2.322	1.664	1.535
5	5.928	8.210	8.621	8.389	5.068
10	21.858	222.222	434.783	26.991	11.403
LC	BTO	XBTO	<ABTO	=ABTO	>ABTO
0%	0.590	0.590	0.590	0.590	0.590
20%	0.598	0.590	0.590	0.590	0.590
50%	0.630	0.661	0.985	0.739	0.617
80%	0.702	0.790	2.287	1.129	0.721
100%	1.276	1.355	52.632	18.051	1.354

Table A4.25: Adaptive Restart BTO : ET

Appendix 5

EXPERIMENTAL TECHNIQUES

In this appendix, the O/S Environment and the experimental and statistical techniques used in this study are described. The UNIX utility functions and commands used for the benchmark and prototype experiments are described and some assumptions about the file system are noted.

5.1. Operating System Environment

The benchmark tests and Prototype experiments were carried out on a VAX/750 running UNIX 4.2 bsd. The system had two disks; the system disk, a DEC RA81 with UDA50 Disk Controller [38,39]; and the user disk, A Fujitsu Eagle M2351A/AF Mini Disk Drive [45] with an Emulex SC41/MS SMD disk controller [41]. The RA81 specifications were, disk capacity: 456MB (Winchester double density), average access time: 28ms (maximum 50ms), and data transfer rate: 17.4 Megabits/second. The Fujitsu Eagle specifications were, disk capacity: 474MB, average access time: 18ms (maximum 35ms), and data transfer rate: 1.859 MB/second.

The file system was as described by McKusick [72]. Our approach in this study has been not to attempt changes to either the file system or the kernel, but to take a "macro" view as far as both were concerned. Further we have assumed, since the benchmarks and prototype experiments were run in single-user mode, that the file sytem is benevolent in its behaviour.

In the proptotype expriments of Chapter 4, a scheduler was written which read a file containing a script of the transactions to be run. To run each transaction as a process, the scheduler "fork"ed a child process which then "exec"ed the transaction [116]. Both the scheduler and transactions ran at the same priority and no attempt was made to increase the priority of the scheduler.

For the CC information, advisory file locking ("flock") was used in exclusive mode to simulate a semaphore [117, 72]. This was controlled by the DBMS CCA functions and hidden from the user. Block and restart delays were implemented using the UNIX "sleep" command [117].

5.2. Chapter 3: Benchmark Tests

The benchmarks for the prototype system were designed to pinpoint potential bottlenecks in TDBS/C or regions of the experiments in which unexpected behaviour such as thrashing might occur. The UNIX commands used for the benchmarks were "time" [117] and "gprof" [118].

Using gprof, the greatest percentage of time spent during transaction execution was found to be in the string manipulation and IDLIST functions. This amounted to less than 10% of the total transaction time. The time spent in all other subparts of the system was found to be less than 2-3% of the total time. That the IDLIST functions are the most expensive in terms of cost is not surprising since all the other database access primitive functions must look up relavant information in the IDLIST.

The UNIX "time" command provides three measures:

1) The elapsed time of the command (in seconds)

2) Time spent in the system (measured to 0.001 second)

3) Time spent in the execution of the command (measured to 0.001 second)

Here the "time spent in the system" is taken as a measure of the IO activity of the transaction and the "time spent in execution of the command" as the CPU time of the transaction. The three measures are labelled ET, IO and CPU in the tables of results.

Each benchmark transaction was run 110 times, the first ten results discarded and the remaining 100 sets of values used to calculate point estimators of the mean values for ET, CPU and IO, assuming a normal distribution. The results are presented in Appendix 2. For each point estimator, the standard deviation and variance were calculated. In general the standard deviation was in the order of 7% of the mean value and the variance in the order of <1%. The error measurement was the 90% confidence

interval which is expressed below as a percentage of the point estimator.

Since the measurement of each transaction, including the overhead, is independent, a measure of "database work" done by the transaction was estimated using (Time - Overhead) for each transaction. Examples are given below in Table A5.1 for the "get-repl" transaction in Table 3.8 and the "grs/gru" transactions of Table A2.13 (both cases accessing a database of 1000 tuples, with a 100 * 1K buffer).

	transaction			overhead			(transaction- overhead)		
	ET	CPU	IO	ET	CPU	IO	ET	CPU	IO
get-repl	0.04	0.08	1.02	1.05	1.80	1.95	0.05	0.08	1.27
grs 1	1.06	1.30	1.59	"	"	"	4.84	5.08	6.54
gru 1	1.01	1.53	1.64	"	"	"	3.90	5.35	6.45
grs 2	0.53	1.66	1.68	"	"	"	3.03	5.26	6.72
gru 2	0.24	1.66	1.73	"	"	"	2.40	4.93	6.67
grs 5	0.16	1.35	1.54	"	"	"	1.66	2.87	5.46
gru 5	0.18	1.15	1.49	"	"	"	1.95	3.43	6.37
grs 10	0.08	1.05	1.64	"	"	"	1.41	2.16	6.47
gru 10	0.17	1.06	1.47	"	"	"	1.16	2.07	4.73
grs 100	0.06	0.24	1.41	"	"	"	0.24	0.31	3.72
gru 100	0.18	0.34	1.05	"	"	"	0.25	0.40	1.51

Table A5.1: Benchmark Error Estimations (90% CI)

As a first order approximation, the approach described above was felt to be reasonable. The most variable measure is IO with a standard deviation of up to 10% of the point estimator value. The "grs/gru" transactions are very short as is the "overhead" transaction and the error becomes much less for a larger transaction such as the "get-repl". The accuracy however, depends on that of the "time" command where the ET is measured to the second and the sum of the CPU and IO values may exceed the ET value by one second. This raised questions about the use of the "time" command for the prototype experiments.

5.3. Chapter 4: Prototype Experiments

From the benchmarks of Chapter 3, the ETs of the transactions were determined and by assuming a perfect linear system, the requirements for the prototype experiments could be estimated, simply by scaling up the benchmark results. For example, to take 50 timings from each of 10 concurrently running transactions of 2 seconds ET, would give a batch length of 1000 seconds. Taking, say 10 batches, to derive point estimators, gives a run time of just under 3 hours. There is clearly a severe practical limit

placed on the size of transaction which can be tested. From the results in Appendix 2, a 2 second ET limit would indicate a transaction which accesses at most 10 tuples. In previous studies, a limited number of tuples have been accessed. These two points led to the choice of T_REQ for the prototype experiments.

As a further result of the benchmark experiments, the method of timing was changed. Using the UNIX "time" command, the ET is expressed as the difference between the start and finish times of the process. Inspecting the kernel code for the "time" command revealed that CPU and IO are measured using "gettimeofday" and "getrusage" commands [117]. The "gettimeofday" command returns a time in seconds and microseconds (although the BUGS comment declares that the time is never correct enough to believe the microsecond values). The "getrusage" returns the CPU and IO times as well as information on paging, swapping, block input/output operations and voluntary and involuntary context switches. The voluntary context switch value gives an indication of the number of switches to await the availability of a resource; here this is interpreted as mainly waiting for IO resources. The involuntary context switch value gives an indication of the number of switches caused by exceeding the time slice. The results showed that there was no swapping, most page swaps did require IO activity (as oppose to reclaiming the page from the list of pages awaiting reallocation) and the ratio of voluntary to involuntary context switches was of the order of 1:3, indicating that the transactions were making efficient use of available time.

For the prototype experiments, the timing was embedded within the transaction and ET measured over the period during which the database accesses were performed. This is illustrated in Figure A5.1

Each batch in the prototype experiments consisted of 500 transaction runs with MPL = 10. The first 10 values were discarded to minimise startup transient effects and the next 400 values used to derive the results presented in Appendix 3. Within the batch, ET, CPU and IO values were assumed to be normally distributed. To derive a point estimator for T, 10 batches were run. The mean values for T, ET, CPU and IO from each batch were used to derive point estimators with a 90% confidence interval using a Student-T distribution with 9 degrees of freedom [122]. In the prototype experiments, the CCA overhead is being measured. The baseline used was the same transaction run without concurrency control. Measures of average ET, CPU and IO are presented for both committed and restarted

transactions. The latter is used to measure resources wasted by restarts. A measure of T is taken as being statistically significant if 50 or more transactions have committed. The results are presented in Appendix 3.

StartProcess
gettimeofday
OpenDatabase
a_1
a_2
...
a_n
CloseDatabase
getrusage
gettimeofday
StopProcess

Figure A5.1: Transaction Timing for Prototype Experiments

In Experiment 1, primary key values are randomly generated using the UNIX "random" function [177].

5.4. Chapter 5: Simulation Experiments

The simulation experiments were run using the method of batched means [32,96,43,76] where each simulation run is divided into n batches, the first being discarded to avoid transient effects on starting the run. Here, 21 batches were run, each of length 100000 simulation units (1 simulation unit being interpreted as 1 millisecond for discussion). The main problem in using batched means is in determining the independence of the batches. Generally the batches are not independent of each other, but in practice, given reasonably large sub-runs, the dependence can be assumed to extend over a small number of sub-runs [76]. In this study, only adjacent batches were assumed to be dependent and non-adjacent batches were assumed to be independent, hence with a covariance value of zero [32]. The batch size of 100000 simulation units was chosen to support the assumption about the independence of non-adjacent batches. The remaining 20 batches were analysed using a Student-T distribution with 19 degrees of freedom. Normally, the covariance for adjacent batches is likely to be positive [76] but since the estimator from the experimental results is itself a random variable, a negative covariance value is possible [23]. Such a negative estimate indicates a tendency for the variables (from adjacent batches) to move in opposite directions [122]. In the context of batched means simulation experiments, this may be

interpreted as indicating that the covariance is neglible. In fact, for our experiments, the covariance estimates, whether positive or negative, were small as shown in Tables A5.2 and A5.3.

CCA	avg	90% CI	var	covar	SD
NO CC	4.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
PRE	3.425E+00	1.433E-02	1.373E-03	-8.625E-03	3.706E-02
2PLE	1.976E+00	1.635E-02	1.788E-03	9.210E-02	4.228E-02
2PLU	1.952E+00	1.826E-02	2.230E-03	9.974E-02	4.723E-02
A2PLU	2.008E+00	5.306E-02	1.884E-02	-2.804E-02	1.373E-01
BTO	3.077E+00	1.109E-02	8.221E-04	-1.880E-03	2.867E-02
XBTO	3.017E+00	2.125E-02	3.022E-03	-6.960E-03	5.497E-02
ABTO	3.074E+00	4.329E-03	1.250E-04	2.400E-03	1.120E-02
SV	2.974E+00	1.564E-02	1.636E-03	-8.600E-03	4.044E-02

Table A5.2: (GRAN = 1, T_REQ = 5): Example Experimental Values (T)

CCA	avg	90% CI	var	covar	SD
NO CC	2.500E+00	2.455E-06	4.031E-11	7.987E-09	6.349E-06
PRE	2.920E+00	1.566E-02	1.640E-03	-8.845E-03	4.050E-02
2PLE	4.276E+00	1.131E-01	8.557E-02	-5.095E-01	2.925E-01
2PLU	3.161E+00	2.451E-02	4.018E-03	2.933E-01	6.339E-02
A2PLU	3.013E+00	7.311E-02	3.576E-02	-1.187E-01	1.891E-01
BTO	2.098E+00	3.246E-03	7.049E-05	1.235E-04	8.396E-03
XBTO	2.339E+00	1.817E-02	2.210E-03	-2.710E-03	4.701E-02
ABTO	2.073E+00	5.183E-03	1.800E-04	5.490E-03	1.341E-02
SV	1.530E+00	1.456E-02	1.418E-03	-3.826E-03	3.765E-02

Table A5.3: (GRAN = 1, T_REQ = 5): Example Experimental Values (ET)

Two random number generators were used in the simulation. The first was based on the method used in DEMOS [40, 16] and was used to generate seeds for the second, the standard SIMULA procedure RANDINT [15, 76] which provides an integer in a given range with equal probability. This was done to avoid generating the same sequence of pseudo-random numbers for each transaction.

REFERENCES

1. Agrawal, R., Carey, M.J., and DeWitt, D.J., "Deadlock Detection is Cheap," Memo No. UCB/ERL M83/5, University of Berkeley (UCB), California, USA, (January 1983).

2. Agrawal, R., Carey, M.J., and Livny, M., "Models for Studying Concurrency Control Performance: Alternatives and Implications," *Proceedings SIGMOD-85 (International Conference on Management of Data)*, pp. 108-121, Austin, Texas, USA, (May 1985).

3. Agrawal, R., Carey, M.J., and Livny, M., "Concurrency Control Performance Modeling: Alternatives and Implications," *ACM TODS* , **Vol. 12**, (4), pp. 609-654, (December 1987).

4. Agrawal, R., Carey, M.J., and McVoy, L.W., "The Performance of Alternative Strategies for Dealing with Deadlocks in Database Management Systems," *IEEE Transactions on Software Engineering*, (1988). To Be Published .

5. Amble, T. and et, al, "Draft Report on the Programming Language ASTRAL," ASTRA-notat no. 31, RUNIT, Trondheim, Norway, (May 1981).

6. Babb, E., "Implementing a Relational Database by Means of Specialized Hardware," *ACM TODS*, **Vol. 4**, (1), pp. 1-29, (March 1979).

7. Bayer, R., Heller, H., and Reiser, A., "Parallelism and Recovery in Database Systems," *ACM TODS*, **Vol. 5**, (2), pp. 139-156, (June 1980).

8. Bell, D.A. and Deen, S.M., "Hash Trees versus B-Trees," *The Computer Journal*, **Vol. 27**, (3), pp. 218-224, The British Computer Society, (1984).

9. Bernstein, P.A., Rothnie, J.B., Goodman, N., and Papadimitriou, C., "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases," *IEEE Transactions on Software Engineering*, **Vol. SE-4**, (4), pp. 154-168, (May 1978).

10. Bernstein, P.A., Shipman, D.W., and Wong, W.S., "Formal Aspects of Serializability in Database Concurrency Control," *IEEE Transactions on Software Engineering*, **Vol. SE-5**, (3), pp. 203-216, (May 1979).

11. Bernstein, P.A. and Shipman, D.W., "The Correctness of Concurrency Control Mechanisms in a System for Distributed Databases (SDD-1)," *ACM TODS*, **Vol. 5**, (1), pp. 52-68, (March 1980).

12. Bernstein, P.A. and Goodman, N., "Concurrency Control in Distributed Database Systems," *Computer Surveys*, **Vol. 13**, (2), pp. 185-221, (June 1981).

13. Bernstein, P.A., "A Sophisticate's Introduction to Distributed Database Concurrency Control," *Proceedings of the 8th VLDB*, pp. 62-76, Mexico City, Mexico, (September 1982).

14. Bernstein, P.A., Goodman, N., and Hadzilacos, V., "Recovery Algorithms for Database Systems," Technical Report, TR 10-83, Aiken Computing Lab., Harvard University, Massachusets, USA, (1983).

15. Birtwistle, G.M., Dahl, O-J., Myhrhaug, B., and Nygaard, K., *Simula Begin*, Auerbach, Philadelphia, USA, (1973).

16. Birtwistle, G.M., "DEMOS - Discrete Event Simulation for Beginners," *SIMULA Newsletter*, **Vol. 7**, (1), pp. 4-7, (February 1979).

17. Bitton, D., DeWitt, D.J., and Turbyfill, C., "Benchmarking Database Systems: A Systematic Approach," *Proceedings of the 9th VLDB*, pp. 8-19, Florence, Italy, (1983).

18. Bitton, D. and Turbyfill, C., "Design and Analysis of Multi-User Benchmarks for Database Systems," Technical Report TR 84-598, Dept Computer Science, Cornell Univ, Ithaca, N.Y., USA, (January 1984).

19. Boorman, B.J., "CAFS-ISP - A UK Context Addressable File Store," in *Database Machines*, ed. A.K. Sood, NATO ASI Series, Vol. F24, Springer-Verlag, (1986).

20. Boral, H., "A Methodology for Database System Performance Evaluation," *Proceedings SIGMOD-84 (International Conference on Management of Data)*, pp. 176-185, Boston, Massachusets, USA, (January 1984).

21. Boral, H. and Gold, I., "Towards a Self-Adapting Centralized Concurrency Control Algorithm," *Proceedings SIGMOD-85 (International Conference on Management of Data)*, pp. 18-31, Austin, Texas, USA, (May 1985).

22. Buer, D.J. Van, Gates, R.O., and Lund, E.O., "A Comparison of Performance of Similar Queries on Similar Databases on Several Relational Systems: Hardware and Software," in *Database Machines*, ed. A.K. Sood, NATO ASI Series, Vol. F24, Springer-Verlag, (1986).

23. Carey, M.J., "Modeling and Evaluation of Database Concurrency Control Algorithms," Ph.D. Thesis (Memo. No. UCB/ERL 83/56), University of Berkeley (UCB), California, USA, (September 1983).

24. Carey, M.J., "An Abstract Model of Database Concurrency Control Algorithms," *SIGMOD Record (Proceedings of SIGMOD-83)*, **Vol. 13**, (4), pp. 97-107, San Jose, California, USA, (May 1983).

25. Carey, M.J. and Stonebraker, M.R., "The Performance of Concurrency Control Algorithms for Database Management Systems," *Proceedings of the 10th VLDB*, pp. 107-118, Singapore, (August 1984).

26. Chesnais, A., Gelenbe, E., and Mitrani, I., "On the Modelling of Parallel Access to Shared Data," *Communications of the ACM*, **Vol. 26**, (3), pp. 196-202, (March 1983).

27. Chou, H.T., DeWitt, D.J., Katz, R.H., and Klug, A.C., "The Design and Implementation of the Wisconsin Storage System," Computer Science Technical Report #524, University of Wisconsin-Madison, USA, (November 1983).

28. Cockshot, W.P., Atkinson, M.P., Chisholm, K.J., Baily, P.J., and Morrison, R., "Persistent Object Management System," *Software Practice & Experience*, **Vol. 14**, (1), pp. 49-71, (January 1984).

29. Codd, E.F., "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, **Vol. 13**, (6), pp. 377-387, (June 1970).

30. Codd, E.F., "Extending the Database Relational Model to Capture More Meaning," *ACM TODS*, **Vol. 4**, (4), pp. 397-434, (December 1979).

31. Codd, E.F., "Relational Database: A Practical Foundation for Productivity (The 1981 Turing Award Lecture)," *Communications of the ACM*, **Vol. 25**, (2), pp. 109-117, (February 1982).

32. Conway, R.W., "Some Tactical Problems in Digital Simulation," *Management Science*, **Vol. 10**, (1), pp. 47-61, (October 1963).

33. Date, C.J., *An Introduction to Database Systems (3rd Ed.)*, 1, Addison-Wesley, (1981).

34. Date, C.J., *An Introduction to Database Systems (Vol. 2)*, 2, Addison-Wesley, (1983).

35. Date, C.J., *An Introduction to Database Systems (4th Ed.)*, 1, Addison-Wesley, (1986).

36. Deen, S.M., Nikodem, D., and Vashishta, A., "The Design of a Canonical Database System (PRECI)," *The Computer Journal*, **Vol. 24**, (3), pp. 200-209, The British Computer Society, (August 1981).

37. Deen, S.M., Edgar, J.A., Nikodem, D., and Vashishta, A., "Run-Time Management in a Canonical DBMS (PRECI)," *Proceedings of BNCOD-2*, pp. 218-239, J. Wiley, (July 1982).

38. Digital Ltd.Digital,, *RA81 Disk Drive User Guide (ER-ORA81-UG-001)*, (1982).

39. Digital Ltd.Digital,, *UDA50 User Guide (EK-UDA50-UG-002)*, (October 1982).

40. Downham, D.Y. and Roberts, F.D.K., "Multiplicative Congruential Pseudo-random Number Generators," *The Computer Journal*, **Vol. 10**, (11), pp. 74-77, The British Computer Society, (1967).

41. Emulex Ltd.Emulex,, *SC41/MD SMD Disk Controller Technical Manual*, 3545 Harbor Boulevard, Costa Mesa, California 92626, USA, (October 1985).

42. Eswaran, K.P., Gray, J.N., Lorie, R.A., and Traiger, I.L., "The Notions of Consistency and Predicate Locks in a Database System," *Communications of the ACM*, **Vol. 19**, (11), pp. 624-633, (November 1976).

43. Fishman, G.S., *Principles of Discrete Event Simulation*, John Wiley & Sons, (1978).

44. Franascek, P. and Robinson, J.T., "Limitation of Concurrency in Transaction Processing," *ACM TODS*, **Vol. 10**, (1), pp. 1-28, (March 1985).

45. Fujitsu Ltd.Fujitsu,, *M2351A/AF Mini-Disc Drive (Eagle) Manual*, (1981).

46. Galler, B.I., "Concurrency Control Performance Issues," Ph.D. Thesis (Technical Report CSRG-147), University of Toronto, Ontario, Canada, (September 1982).

47. Garcia-Molina, H., Germano, F., and Kohler, W.H., "Architectural Overview of a Distributed Software Testbed," *Proceedings of the 16th Hawaii International Conference on System Sciences*, pp. 310-319, (1983).

48. Gardarin, G. and Melkanoff, M., "Concurrency Control Principles in Distributed and Centralised Databases," *INRIA Rapports de Recherche No. 113*, 78153 Le Chesnay, France, (January 1982).

49. Gerritsen, R., Cortes, R., Ribiero, J., and Zowader, R., *WAND Users' Guide*, Dept. of Decision Sciences, The Wharton School, Univ. of Pennsylvania, Pennsylvania, USA, (1976).

50. Graham, S.L., Kessler, P.B., and McKusick, M.K., "gprof: a Call Graph Execution Profiler," *SIGPLAN Notices (Proceedings of the ACM SIGPLAN '82 Symposium on Compiler Conctruction)*, **Vol. 17**, (6), pp. 120-126, (June 1982).

51. Gray, J., "Notes on Data Base Operating Systems," *IBM Research Lab. Research Report RJ2188 (30001) 2/23/78*, (February 1978).

52. Gray, J. and , al et, "The Recovery Manager of the System R Database Manager," *Computer Surveys*, **Vol. 13**, pp. 223-242, (June 1981).

53. Gray, J., "A Straw Man Analysis of Probability of Waiting and Deadlock," *IBM Research Lab. Research Report RJ3066 (38112) 2/26/81*, (February 1981).

54. Griffeth, N.D. and Morsi, M., *SORCERER: A Distributed Database Testbed and Simulation Tool*, School of Inf & Comp Sci, Georgia Inst of Technology, Atlanta, Georgia, USA, (February 1983).

55. Holt,, "Some Deadlock Properties of Computer Systems," *Computer Surveys*, **Vol. 4**, pp. 79-195, (December 1972).

56. Jipping, M.J. and Ford, R., "Predicting Performance of Concurrency Control Designs," *Proceedings of the 1987 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 132-140, Banff, Alberta, Canada, (May 1987).

57. Kedem, Z. and Silberschatz, A., "Non Two-Phase Locking Protocols with Shared and Exlusive Locks," *Proceedings of the 6th VLDB*, pp. 309-317, (1980).

58. Kedem, Z., Mohan, C., and Silberschatz, A., "An Efficient Deadlock Removal Scheme for Non Two-Phase Locking Protocols," *Proceedings of the 8th VLDB*, pp. 91-97, Mexico City, Mexico, (September 1982).

59. Kent, J. and Garcia-Molina, H., "Performance Evaluation of Crash Recovery Systems," Technical Report 329, Princeton University, Princeton, N.J., USA, (November 1984).

60. Kent, J., Garcia-Molina, H., and Chung, J., "An Experimental Evaluation of Crash Recovery Mechanisms," *Proceedings of the 4th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pp. 113-122, Portland, Oregon, USA, (March 1985).

61. Kernighan, B.W. and Ritchie, D.M., *The C Programming Language*, Prentice Hall, (1978).

62. Kersten, M. and Wasserman, A.I., "The Architecture of the PLAIN Database Handler," *Software Practice & Experience*, **Vol. 11**, (2), pp. 175-186, (February 1981).

63. Kersten, M. and Tebra, H., "Application of an Optimistic Concurrency Control Method," *Software Practice & Experience*, **Vol. 14**, (2), pp. 153-168, (February 1984).

64. Kersten, M., Wasserman, A.I., and Riet, R.P. van der, *Troll/USE Ref. Manual; Troll/USE Library Ref. Manual; RAPID/USE Ref. Manual*, 2, Wiskundig Seminarium, Vrije Universiteit, De Boelaan 1081 HV, Amsterdam, The Netherlands, (May 1984).

65. Kohler, W.H., "A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems," *Computer Surveys*, **Vol. 13**, (2), pp. 149-183, (June 1981).

66. Kohler, W.H., Wilner, K.C., and Stankovic, J.A., "An Experimental Comparison of Locking Policies," *SIGMOD Record (Proceedings of SIGMOD-83)*, **Vol. 13**, (4), San Jose, California, USA, (May 1983).

67. Kung, H.T. and Robinson, J.T., "On Optimistic Methods for Concurrency Control," *ACM TODS*, **Vol. 6**, (2), pp. 213-226, (June 1981).

68. Lin, W.K. and Nolte, J., "Performance of Two Phase Locking," *Proceedings of the 6th Berkeley Workshop on Distributed Database Management and Computer Networks*, (February 1982).

69. Lin, W.K. and Nolte, J., *Basic Timestamp, Multiple Version Timestamp, and Two Phase Locking*, Computer Corporation of America, Four Cambridge Center, Cambridge, MA 02142, USA, (1983). Internal Report

70. Mackert, L.F. and Lohman, G.M., "R* Optimizer Validation and Performance Evaluation for Local Queries," *Proceedings SIGMOD-86 (International Conference on Management of Data)*, pp. 84-95, Washington DC, USA, (May 1986).

71. Madelaine, J., "Evaluation d'Algorithmes de Controle de Concurrence," *INRIA Rapports de Recherche No. 164*, 78153 Le Chesnay, France, (October 1982).

72. McKusick, M.K., Joy, W.N., Leffler, S.J., and Fabry, R.S., "A Fast File System for UNIX," *ACM Transactions on Computer Systems*, **Vol. 2**, (3), pp. 181-197, (August 1984).

73. Menasce, D.A. and Muntz, R.R., "Locking and Deadlock Detection in Distributed Databases," *IEEE Transactions Software Engineering*, **Vol. SE-5**, (3), pp. 195-202, (May 1979).

74. Menasce, D.A. and Nakanishi, T., "Optimistic vs Pessimistic Concurrency Control Mechanisms," *Information Systems*, **Vol. 7**, (1), pp. 13-27, (1982).

75. Milenkovic, M., *Update Synchronization in Multiaccess Systems*, U.M.I. Press, Ann Arbor, Michigan, USA, (1981).

76. Mitrani, I., *Simulation techniques for discrete event systems*, CUP, (1982).

77. Mohan, C., "Strategies for Enhancing Concurrency and Managing Deadlocks in Data Base Locking Protocols," Ph.D. Thesis, Univ of Texas at Austin, Texas, USA, (December 1981).

78. Mohan, C., Fussel, D., and Silberschatz, A., "A Biased Non Two Phased Locking Protocol," Improving Database Usability and Responsiveness, Academic Press, (1982).

79. Mohan, C., Fussel, D., Kedem, Z., and Silberschatz, A., "Lock Conversions in Non-Two Phase Locking," *IEEE Transactions on Software Engineering*, **Vol. SE-11**, (1), pp. 15-22, (January 1985).

80. Palmer, E.F., *INGRES/ORACLE Benchmark Tests (Internal Report)*, Impell Corporation, Georgia, USA. Private Communication

81. Papadimitriou, C., "The Serializability of Concurrent Database Updates," *Journal of the ACM*, **Vol. 26**, (4), pp. 631-653, (October 1979).

82. Peinl, P. and Reuter, A., "Empirical Comparison of Database Concurrency Control Schemes," *Proceedings of the 9th VLDB*, pp. 97-119, Florence, Italy, (November 1983).

83. Petri, G. and Oehman, M., "Utvaerdering av Systemutvecklingsverktyg foer Struktur-90," (Evaluation of Systems Development Tools for Struktur-90), Stockholm, Sweden. Private Communication .

84. Potier, D. and Leblanc, Ph., "Analysis of Locking Policies in Database Management Systems," *Communications of the ACM*, **Vol. 23**, (10), pp. 584-593, (October 1980).

85. Ries, D.R. and Stonebraker, M., "Effects of Locking Granularity in a Database Management System," *ACM TODS*, **Vol. 2**, (3), pp. 233-246, (September 1977).

86. Ries, D.R., "The Effects of Concurrency Control on Database Mangement System Performance," Ph.D. Thesis, Computer Science Department, University Berkeley, California, USA, (April 1979).

87. Ries, D.R. and Stonebraker, M., "Locking Granularity Revisited," *ACM TODS*, **Vol. 4**, (2), pp. 210-227, (June 1979).

88. Riet, R.P. van der, Wasserman, A.I., Kersten, M., and Jonge, W. De, "High Level Programming Features for Improving the Efficiency of a Relational Database System," *ACM TODS* , **Vol. 6**, (3), pp. 464-485, (September 1981).

89. Ritchie, D.M. and Thompson, K., "The UNIX Time-Sharing System," *Communications of the ACM*, (7), pp. 365-375, (July 1974).

90. Rosenkrantz, D.J., Stearns, R.E., and Lewis, P.M. III, "System Level Concurrency Control for Distributed Database Systems," *ACM TODS*, **Vol. 3**, (2), pp. 178-198, (June 1978).

91. Ross, D.F., "Database Research Project Documentation," Collection of Notes from PRECI collaboration, Dept. of Computer Science, Heriot-Watt University, Edinburgh, Scotland, UK, (April 1985).

92. Ross, D.F., "An Introduction to TDBS/C: Testbed Database in C," Technical Report TR/87-03, Dept. of Computer Science, Heriot-Watt University, Edinburgh, Scotland, UK, (April 1987).

93. Rubenstein, W.B., Kubicar, M.S., and Cattell, R.G.G., "Benchmarking Simple Database Operations," *Proceedings SIGMOD-87 (International Conference on Management of Data)*, pp. 387-394, San Francisco, California, USA, (May 1987).

94. SCOT,, "Detection et Prevention des Inter-Blocages dans un Systeme Transactionel Centralise," *Rapports de Recherche SCOT No. 17*, IMAG, Grenoble, France, (April 1981).

95. SQL,, *Database Language SQL (Final Draft ISO 9075-1987(F)), ISO TC97 Sc21 WG3-DBL RENO-2*, IBM Corp., J67 B47, 555 Bailey Ave.,San Jose, California, 95150, USA, (November 1987).

96. Sargent, R., "Stastical Analysis of Simulation Output Data," *Proceedings of the 4th Annual Symposium on the Simulation of Computer Systems*, Boulder, Colorado, USA, (July 1976).

97. Schmidt, J.W., "Some High Level Language Constructs for Data of Type Relation," *ACM TODS* , **Vol. 2**, (3), pp. 247-261, (September 1977).

98. Shannon, R.E., *Systems Simulation: the art and science*, Prentice-Hall, (1975).

99. Stearns, R.E., Lewis, P.M., and Rosenkrantz, D.J., "Concurrency Control for Database Systems," *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, pp. 19-32, (1976).

100. Stonebraker, M., "Retrospection on a Database System," *ACM TODS*, **Vol. 5**, (2), pp. 225-240, (June 1980).

101. Stonebraker, M., "Operating Systems Support for Database Management," *Communications of the ACM*, **Vol. 24**, (7), pp. 412-418, (July 1981).

102. Stonebraker, M., Woodfill, J., Ranstrom, J., Murphy, M., Meyer, M., and Allman, E., "Performance Enhancements to a Relational Database System," *ACM TODS*, **Vol. 8**, (2), pp. 167-185, (June 1983).

103. Tay, Y.C., Goodman, N., and Suri, R., "Performance Evaluation of Locking in Databases: A Survey," Technical Report, TR 17-84, Aiken Computing Lab., Harvard University, Massachusets, USA, (February 1984).

104. Tay, Y.C., "A Mean Value Performance Model for Locking in Databases," Ph.D. Thesis (CRCT Report TR-04-84), Harvard University, Massachusets, USA, (1984).

105. Tay, Y.C., Suri, R., and Goodman, N., "A Mean Value Performance Model for Locking in Databases: The No-Waiting Case," *Journal of the ACM*, **Vol. 32**, (3), pp. 618-651, (July 1985).

106. Tay, Y.C., Goodman, N., and Suri, R., "Locking Performance in a Centralized Database," *ACM TODS*, **Vol. 10**, (4), pp. 415-462, (December 1985).

107. Tay, Y.C., *Locking Performance in Centralized Databases*, Academic Press, (1987).

108. Taylor, R.W., "User Experience with the Bitton Lee IDM 500," in *Database Machines*, ed. A.K. Sood, NATO ASI Series, Vol. F24, Springer-Verlag, (1986).

109. Tebra, J.R.B., *Transaction Management in the PLAIN Database Handler*, Wiskundig Seminarium, Vrije Universiteit, De Boelaan 1081 HV, Amsterdam, The Netherlands, (January 1982).

110. Thanos, C., Carlesi, C., and Bertino, E., "Performance Evaluation of Two-Phase Locking Algorithms in a System for Distributed Databases," *3rd Symp on Reliability in Distributed Software*, pp. 57-69, Florida, USA, (October 1983).

111. Thanos, C., Carlesi, C., and Bertino, E., "Performance Evaluation of Timestamp Ordering Algorithms in a System for Distributed Databases," *Convencio Informatica Llatina*, pp. 305-319, Barcelona, Spain, (June 1983).

112. Thomas, R.H., "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases," *ACM TODS*, **Vol. 4**, (2), pp. 180-209, (June 1979).

113. Thomasian, A., "Performance Evaluation of Centralized Databases with Static Locking," *IEEE Transactions on Software Engineering*, **Vol. SE-11**, (4), pp. 346-355, (April 1985).

114. Tsichritzis, D. and Klug, A., "The ANSI/X3/SPARC DBMS Framework: Report of the Study Group on DBMS," *Informations Systems*, **Vol. 3**, (1978).

115. Tsichritzis, D.C. and Lochovsky, F.H., *Data Models*, Prentice Hall, (1982).

116. UNIX,, *UNIX Programmer's Manual, 4.2 Berkeley Software Distribution, 7th Edition*, 2B, Computer Science Division, Detp. of Electrical Engineering and Computer Science, University of California, Berkeley, California 94720, USA, (January 1979).

117. UNIX,, *UNIX Programmer's Manual, 4.2 Berkeley Software Distribution, Virtual VAX 11 Version*, 1a and 1b, Computer Science Division, Detp. of Electrical Engineering and Computer Science, University of California, Berkeley, California 94720, USA, (August 1983).

118. Ullman, J.D., *Principles of Database Systems*, Pitman, (1980).

119. Vashishta, A., "An Implementation of Schemas and a Relational Facility for a Canonical Data Model," Ph.D. Thesis, Aberdeen University, Aberdeen, Scotland, UK, (December 1982).

120. Wasserman, A.I., Sheretz, D.D., Kersten, M.L., Riet, R.P. van der, and Dippe, M., "Revised Report on the Programming Language PLAIN," *ACM SIGPLAN Notices*, **Vol. 16**, (5), pp. 59-80, (1981).

121. Wasserman, A.I. and Kersten, M.L., *Troll/USE Tutorial; TBE/Tutorial; RAPID/USE Tutorial*, 1, Wiskundig Seminarium, Vrije Universiteit, De Boelaan 1081 HV, Amsterdam, The Netherlands, (May 1984).

122. Wonnacot, R.J. and Wonnacot, T.H., *Introductory Statistics*, J. Wiley, (1969).

123. Zipf, J.K., *Human Communication and The Principle of Least Effort*, Addison-Wesley, (1949).

124. Zloof, M.M., "Query-By-Example: a Database Language," *IBM System Journal*, (4), (1977).